FRANÇOIS RIOUX

CONCEPTION ET MISE EN ŒUVRE DE MULTICHRONIA, UN CADRE CONCEPTUEL DE SIMULATION VISUELLE INTERACTIVE

Thèse présentée
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de doctorat en génie électrique
pour l'obtention du grade de Philosophiae Doctor (Ph.D.)

FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

2009

Résumé

Cette thèse présente *Multichronia*, un cadre conceptuel de simulation interactive fournissant une représentation visuelle du cheminement d'un utilisateur dans l'exploration des simulations d'un système complexe. En complément aux méthodes formelles d'analyse, Multichronia vise à aider ses utilisateurs à comprendre un système sous étude en fournissant quatre boucles interactives. La boucle d'exploration de l'espace des paramètres permet à un utilisateur de modifier des paramètres de simulation afin de tester des hypothèses. La boucle d'exploration de l'espace des simulations lui permet de manipuler les données correspondant à des instances de simulation. Notamment, elle rend disponible des opérations de sélection et d'alignement via une interface graphique. La boucle d'exploration de l'espace des données lui permet de transformer les flots de données. Finalement, la boucle d'exploration de l'espace visuel lui permet d'afficher des données et de manipuler leur aspect visuel.

Afin de représenter le cheminement d'un utilisateur dans son exploration de l'espace des paramètres, une interface graphique a été développée. Il s'agit de l'*arbre multichronique*, une vue formelle donnant une représentation informative de l'état de l'analyse d'un problème ainsi que la possibilité d'exécuter une foule d'opérations interactives. D'autre part, le cadre conceptuel Multichronia forme un pipeline de données générique allant d'un simulateur jusqu'à un logiciel d'analyse. Un modèle conceptuel peut être extrait de ce pipeline de même que le flux de données correspondant. Dans cette thèse, il a été spécialisé avec la technologie XML. Cette dernière permet entre autres de définir une méthodologie de conception du modèle de données associé à un simulateur.

La mise en œuvre de Multichronia a permis de vérifier la validité des concepts proposés. L'architecture logicielle adoptée est un cadre d'application, de sorte que de nouveaux simulateurs puissent être facilement exploités. Deux applications concrètes ont été implantées, soit la simulation tactique et stratégique de l'attaque de convois militaires. Des modifications mineures aux simulateurs ont été nécessaires afin qu'ils rencontrent certains critères établis dans cette thèse. Somme toute, ces applications ont montré que Multichronia peut être déployé pour des applications quelconques.

Abstract

This thesis introduces *Multichronia*, a conceptual framework for interactive simulation that provides a visual representation of the work accomplished by a user in exploring a complex system. As a complement to formal analysis, Multichronia aims at helping its users in understanding the system under study. For doing so, it provides four interactive exploration loops. The parameter space exploration loop allows for hypothesis testing via the modification of simulation parameters. The simulation space exploration loop allows for the manipulation of data related to each simulation instance. Notably, an original user interface allows for the selection and alignment of simulation instances. The data space exploration loop allows for the transformation of data flows. Finally, the visual space exploration loop displays data in the form of an image and allows for the manipulation of their visual aspects.

An original user interface was developed in order to represent the parameter space exploration work accomplished by a user. In fact, the *multichronic tree* is a formal representation that provides an informative visualization for the current state of the analysis while allowing several interactive operations. On the other hand, the Multihronia conceptual framework acts as a data pipeline spanning from a simulator to a visual analysis software. A data conceptual model can be extracted from that pipeline, as well as a generic data flow. This thesis proposes a specialization of the conceptual model with the XML technology, which results in the definition of a design methodology for development of the data model associated with a simulator.

The implementation of Multichronia allowed verifying the validity of the proposed concepts. A software framework was developed such that additional simulators can be exploited in a given application. Two concrete applications were implemented, that is a tactical and a strategic simulation of military convoys. In order to meet established conventions in Multichronia, minor modifications were needed on the simulators. However, these applications demonstrate that the framework can be deployed in more general contexts.

Avant-propos

J'aimerais tout d'abord remercier mon directeur de thèse, Dr. Denis Laurendeau, pour m'avoir supervisé durant ces trois années. Son efficacité remarquable et son esprit critique ont été particulièrement utiles pour la révision de manuscrits.

J'aimerais également remercier M. Guy Turcotte, chef de la section « Systèmes de Systèmes » à RDDC Valcartier, pour m'avoir hébergé au laboratoire d'immersion virtuelle (LIV) durant ces trois années. J'ai ainsi eu accès à de l'équipement sophistiqué de réalité virtuelle qui m'a permis de parfaire mes connaissances dans ce domaine. Merci à toute l'équipe du LIV (Marielle, Éric, François et Fred) pour ces beaux moments passés en votre compagnie et aux stagiaires/étudiants qui ont agrémenté ces étés tranquilles à RDDC. Un merci particulier à Fred, mon partenaire de covoiturage.

Je remercie M. Michel Lizotte pour m'avoir donné la chance de contribuer au projet IMAGE et ainsi m'assurer que le logiciel développé pendant mes études doctorales sera utilisé « pour vrai » dans des expérimentations de psychologie cognitive.

Un merci particulier à Dr. François Bernier, mon partenaire dans le développement de l'aspect simulation pour le projet IMAGE. Plusieurs concepts rapportés dans cette thèse sont le fruit de nombreuses discussions que nous avons eues ensemble. Son imagination et sa fougue sont pour moi une source d'inspiration.

Merci au CRSNG, au FQRNT et à RDDC Valcartier pour leur soutien financier.

Merci à mes parents pour m'avoir transmis de si bonnes valeurs. Merci à mes amis et à mes frères pour votre support.

Finalement, merci à Marie-Ève, mon amour, pour ton soutien et ton écoute.

Table des matières

Résumé ii

Abstract iii

Avant-propos iv

Table des matières v

Liste des tableaux viii

Table des figures ix

1 Introduction 1
- 1.1 Mise en contexte . . . 1
- 1.2 Objectifs . . . 2
- 1.3 Problèmes abordés . . . 4
- 1.4 Problèmes non-abordés . . . 5
- 1.5 Contributions . . . 6
- 1.6 Résultats . . . 7
- 1.7 Structure de la thèse . . . 8

2 Revue de littérature 10
- 2.1 Systèmes complexes . . . 10
  - 2.1.1 Caractéristiques des systèmes complexes . . . 11
  - 2.1.2 Aspects cognitifs reliés à la complexité . . . 12
  - 2.1.3 Modélisation des systèmes complexes . . . 13
- 2.2 Simulation . . . 14
  - 2.2.1 Avantages, désavantages et difficultés reliés à la simulation . . . 15
  - 2.2.2 Expériences par ordinateur ou études computationnelles . . . 18
  - 2.2.3 Design d'expériences par ordinateur . . . 20
  - 2.2.4 « Data farming » . . . 24
- 2.3 Simulation interactive . . . 26
  - 2.3.1 Simulation interactive visuelle (VIS) . . . 28

2.3.2 Pilotage de simulations (CS) 30
2.3.3 Fonctionnalités des systèmes de simulation interactive existants 36
2.4 Vers un espace de travail intégré 37
2.4.1 Environnements de résolution de problèmes 38
2.4.2 Autres utilisations de la simulation interactive 39

3 Multichronia – principes de base 40
3.1 Cadre conceptuel Multichronia 41
3.2 Pipeline de données de Multichronia 43
3.2.1 Simulation 43
3.2.2 Alignement et sélection des flots de données 44
3.2.3 Traitement des flots de données 44
3.2.4 Rendu visuel 45
3.2.5 Utilisateur 45
3.3 Boucles d'interaction de Multichronia 45
3.3.1 Exploration de l'espace des paramètres de simulation 46
3.3.2 Exploration de l'espace des simulations 47
3.3.3 Exploration de l'espace des données de simulation 47
3.3.4 Exploration de l'espace visuel 48
3.4 Gestionnaire Multichronia 48

4 Arbre multichronique 50
4.1 Revue des interfaces de contrôle de simulation 51
4.2 Design de l'arbre multichronique 55
4.3 Opérations supportées par l'arbre multichronique 59
4.4 Intégration de l'arbre multichronique dans le cadre conceptuel Multichronia 63
4.4.1 Contraintes imposées au simulateur 64
4.4.2 Automatisation d'opérations dans l'arbre multichronique 66

5 Pipeline de données générique du cadre conceptuel Multichronia 68
5.1 Revue de la modélisation d'un pipeline de données 69
5.2 Modèle de données conceptuel, indépendant de la technologie 70
5.3 Utilisation du XML comme technologie unificatrice 75
5.3.1 Modèle de données conceptuel spécifique au XML 77
5.3.2 Méthodologie de modélisation des données 78
5.3.3 Flux de données basé sur le XML 84

6 Scénario d'expérimentation de Multichronia 86
6.1 Contexte général de l'attaque de convois militaires 86
6.2 Modélisation tactique de l'attaque d'un convoi militaire 88
6.3 Modélisation stratégique d'attaques de convois militaires par coévolution 91

6.4 Rôle de l'utilisateur et utilisations de Multichronia 94

7 Résultats et discussion **97**
7.1 Architecture de base du cadre d'application Multichronia 97
7.1.1 Paquetage « context » 98
7.1.2 Paquetage « ui » 99
7.1.3 Paquetage « rules » 100
7.1.4 Paquetage « simulation » 100
7.1.5 Paquetage « instrumentation » 105
7.1.6 Paquetage « data » 108
7.2 Implantation de Multichronia 111
7.2.1 Affichage et interaction de l'arbre multichronique 112
7.2.2 Exploitation des processeurs multi-cœurs 115
7.2.3 Fonctionnalités additionnelles 117
7.3 Discussion sur les contextes de simulation implantés dans Multichronia 121
7.3.1 Simulation tactique d'attaque de convois militaires 121
7.3.2 Simulation stratégique d'attaque de convois militaires 127
7.4 Conclusion 132

8 Conclusion **133**
8.1 Retour sur les objectifs 134
8.2 Limitations 135
8.3 Perspectives 136

Bibliographie **139**

A Dynamique du scénario d'attaque de convoi militaire **152**

Liste des tableaux

4.1 Fonctionnalités associées aux interfaces d'exploration de l'espace des paramètres et de l'espace des simulations existant dans la littérature . . . 56
4.2 Contraintes imposées au simulateur par l'arbre multichronique 64

7.1 Description des fonctionnalités qu'un simulateur exploité par Multichronia doit implanter ou émuler 102
7.2 Caractéristiques déterminant la priorité des exécutions de simulations du service de gestion du temps processeur 104
7.3 Principaux fils d'exécution de Multichronia et leur description 115

Table des figures

2.1 Processus de modélisation, exécution et analyse de simulation 14
2.2 Boucle d'apprentissage idéalisée. L'apprentissage implique une expérimentation continue entre le monde réel et le monde virtuel (simulation) [Sterman, 1994] 17
2.3 Processus de création d'un métamodèle et liens avec le modèle de simulation et le problème étudié (tirée de [Kleijnen et Sargent, 2000]) 20
2.4 Designs d'expérience par ordinateur recommandés en fonction du nombre de facteurs considérés et de la complexité de la surface de réponse [Kleijnen *et al.*, 2005] 22
2.5 Exemple de design d'expérience par ordinateur pour sept facteurs basé sur les hypercubes latins presque orthogonaux [Sanchez, 2009] 23
2.6 Processus du *data farming* [Horne, 2001] 25
2.7 Processus générique de simulation interactive (adaptée de [Wright, 2004]) 27

3.1 Exemple typique du cadre d'application Multichronia. L'utilisateur interagit avec la simulation (gauche) et observe les données associées (droite) 41
3.2 Cadre conceptuel Multichronia montrant les quatre boucles d'interaction et les unités de transformation formant le pipeline de données 42

4.1 Arbre d'historique de simulation de GRASPARC [Brodlie *et al.*, 1993] . 52
4.2 Architecture de HyperScribe [Wright et Walton, 1996] 53
4.3 Arbre d'historique de simulation dans TRICEPS [Pickles *et al.*, 2004a] 54
4.4 Arbre d'expérience bâti en pausant, clonant, modifiant des paramètres, et redémarrant des simulations [Fischer *et al.*, 2007] 55
4.5 Représentation conceptuelle de l'arbre multichronique et définitions correspondantes 58

5.1 Modèle de données conceptuel, indépendant de la technologie 72
5.2 Flux de données complet, indépendant de la technologie 74
5.3 Modèle de données conceptuel spécialisé pour le format de données XML. Les boîtes bleues montrent une méthodologie à suivre tandis que les boîtes jaunes identifient un pipeline de données 79

5.4 Méthodologie de modélisation (haut) et pipeline de données conceptuel unifié XML (bas) 80
5.5 Exemple de diagramme UML des classes de l'état de la simulation . . . 80
5.6 Schéma XML résultant de la transformation d'un diagramme UML de classes des entités d'une simulation 82
5.7 Résultat de la compilation d'un schéma XML représentant les entités d'une simulation 83
5.8 Flux de données concret unifié XML 85

6.1 Terrain sur lequel évoluent les véhicules du convoi, les agents IED et les agents RPG 89
6.2 Détails du processus de coévolution dans le contexte d'une simulation stratégique 92
6.3 Rôle de l'utilisateur dans le contexte de simulation stratégique, inspiré de [Bernier et Rioux, 2008] 95

7.1 Principaux paquetages logiciels du cadre d'application Multichronia et leurs dépendances 98
7.2 Diagramme de séquence d'une interaction de l'utilisateur menant à l'exécution d'une action dans Multichronia 101
7.3 Diagramme de déploiement de Multichronia et interconnexions. Le même ordinateur peut héberger Multichronia et les simulateurs 105
7.4 Interface de configuration du filtre de données imposé au mécanisme de sérialisation d'un simulateur 107
7.5 Axe du temps montrant l'attente nécessaire pour l'obtention de nouvelles données suite à la création d'un point de divergence (moment « Interaction ») sans (en rouge) et avec (en vert) le mécanisme de points de sauvegarde périodiques (« CP »). La simulation est rendue à un point ultérieur 108
7.6 Diagramme de séquence montrant une instance de simulation qui se connecte au gestionnaire de données entrantes 109
7.7 Diagramme de séquence montrant les traitements possibles lors de l'envoi de données à partir d'une instance de simulation 110
7.8 Diagramme de séquence montrant les traitements possibles lors de la lecture d'un cours d'action, opération qui est activée par l'utilisateur . 111
7.9 Représentation visuelle d'un document XML incluse dans une boîte de dialogue permettant le changement de paramètres de simulation 113
7.10 Capture d'écran de Multichronia montrant les fonctionnalités interactives offertes à l'utilisateur pour le contexte de simulation tactique 114

7.11 Représentation des différents fils d'exécution faisant partie de Multichronia et leurs interactions. Les actions numérotées indiquent la séquence des actions accomplies à l'interne suite à la création d'un point de divergence 118
7.12 Photo du « SpacePilot ». La rondelle sur le dessus du périphérique sert à manipuler des objets virtuels avec six degrés de liberté 119
7.13 Capture d'écran d'un scénario typique implanté dans Pythagoras. Les agents sont représentés par les pastilles bleues et rouges 122
7.14 Flot de données original dans Pythagoras 123
7.15 Flot de données de Pythagoras qui inclut les modifications qui ont été apportées afin de l'exploiter dans Multichronia 125
7.16 Flot de données et architecture du simulateur de convoi stratégique . . 129
7.17 Capture d'écran d'un arbre multichronique dans le contexte de simulation stratégique d'attaque de convois militaires 130
7.18 Gauche : boîte de dialogue permettant de modifier des paramètres de simulation stratégique (encore au stade de prototype). Droite : boîte de dialogue permettant de modifier l'allocation initiale d'un cours d'action 131

Chapitre 1

Introduction

1.1 Mise en contexte

L'humain a toujours voulu comprendre le fonctionnement du monde qui l'entoure. Depuis des siècles, les hommes de science ne cessent de réaliser des découvertes nous en apprenant un peu plus sur l'univers, les êtres vivants et la société. Les systèmes construits par les humains ont jadis été très simples, mais sont désormais compliqués. La barrière qui existe aujourd'hui, empêchant les scientifiques de mieux comprendre les systèmes qu'ils étudient, se nomme la « complexité » [Poussart, 2006]. Les systèmes complexes, évoluant souvent au bord du chaos, présentent plusieurs caractéristiques qui rendent leur comportement difficilement prédictible, notamment la dépendance par rapport à leur historique et la manifestation de comportements émergents [Sterman, 2004]. Les humains ont donc de la difficulté à comprendre ces systèmes à l'aide d'outils d'analyse traditionnels [Dörner, 1996].

La simulation par ordinateur fait partie des outils modernes employés par les scientifiques et les ingénieurs afin d'analyser une grande variété de problèmes dont l'amplitude et la complexité varient sur un très large spectre. La première étape de la méthodologie associée à cet outil est la modélisation, qui consiste à créer des modèles simplifiés du problème sous étude. Par la suite, ces modèles sont exécutés selon un design d'expérience (*Design of Experiment*, DoE) afin d'assurer la validité statistique des résultats obtenus. Un DoE spécifie un sous-ensemble de l'espace des paramètres associé aux modèles de simulation ; un échantillonnage complet est rarement envisagé puisqu'il est généralement impossible d'exécuter toutes les combinaisons de paramètres dans un temps raisonnable [Kleijnen *et al.*, 2005].

Un système complexe dont l'état final dépend de ceux par lesquels il est passé pose particulièrement problème pour les DoE, surtout lorsque des décisions doivent être prises *pendant* l'exécution d'une simulation. En effet, le nombre de paramètres pouvant être modifiés dans une simulation varie exponentiellement avec le nombre de décisions possibles. Ceci rend même un scénario des plus simples très complexe relativement au nombre de simulations à exécuter afin d'échantillonner l'espace des paramètres correctement. Puisqu'il n'existe pas de solution formelle à ce problème, cette thèse fait l'hypothèse que *l'ajout d'un humain dans la boucle de simulation* permettrait *une augmentation de ses performances* en termes de vitesse de compréhension d'un système complexe, pourvu qu'il dispose d'un *ensemble d'outils appropriés.* Par exemple, un humain serait en mesure de prendre des décisions intuitivement et en temps opportun. Ainsi, des simulations qui tiennent compte d'une combinaison de paramètres n'ayant aucun sens physique ou qui n'apportent rien à la solution d'un problème n'auraient pas à être exécutées.

Ce genre d'analyse informelle, aussi appelée « What-if ? », est parfois caricaturé comme étant du simple « essai et erreur » par les puristes du domaine de l'analyse de données résultant de l'exécution de DoE. Cet argument est probablement vrai lorsqu'un utilisateur n'est pas familier avec le système qu'il étudie. Par contre, l'intuition d'experts en la matière possédant plusieurs années d'expérience dans l'exploitation de modèles, appuyée d'outils interactifs appropriés, est fort probablement plus efficace qu'une méthode formelle, en particulier pour des systèmes complexes.

Bien que la présence d'un humain dans la boucle de simulation amène plusieurs points positifs de par son intuition et sa capacité à reconnaître des patrons visuels, il n'en demeure pas moins qu'elle peut également devenir problématique en raison des biais cognitifs auxquels il est soumis. Les outils interactifs fournis à l'utilisateur ont ainsi un grand rôle à jouer dans la minimisation de l'effet des biais cognitifs et dans le succès de l'humain à comprendre le système sous étude. Il est d'ailleurs essentiel de procéder à des expérimentations avec les sujets humains afin de mesurer la contribution de tels outils.

1.2 Objectifs

Les objectifs de cette thèse se divisent en trois principaux regroupements. Le premier est de *concevoir l'architecture d'un environnement de simulation interactive générique pouvant être utilisé dans plusieurs contextes.* Cet objectif comprend celui de bâtir une architecture logicielle capable de s'adapter à divers simulateurs et de définir les critères

assurant la compatibilité d'un simulateur avec le logiciel construit. Un environnement de simulation interactive générique devrait également inclure un pipeline permettant aux flots de données de voyager d'un simulateur jusqu'à un logiciel de visualisation tout en étant traités adéquatement par les unités de transformation intermédiaires. Finalement, le logiciel développé devra laisser à ses utilisateurs une liberté totale quant à la configuration des différentes facettes des contextes d'exploitation. Une revue complète des systèmes existants permettra d'identifier les points faibles bénéficiant d'un traitement particulier lors de l'élaboration de l'architecture, et contribuera à la définition de fonctionnalités essentielles que devrait inclure l'environnement développé.

Le deuxième objectif principal de cette thèse est de *concevoir une interface usager de simulation interactive en utilisant des techniques modernes d'interaction.* En effet, les premiers systèmes de simulation interactive ont été construits il y a plus de trois décennies alors que la qualité graphique des interfaces était plutôt primaire et que les possibilités d'interaction étaient très limitées. Les systèmes de simulation interactive de cette époque, qui n'ont pas beaucoup évolué depuis, ont donc été évalués et critiqués alors que la technologie n'était pas encore à un stade assez avancé pour que les systèmes interactifs soient dignes de ce nom [Bell et O'Keefe, 1995]. La technologie disponible aujourd'hui offre beaucoup plus de possibilités au niveau de l'affichage et de l'interaction de l'utilisateur avec l'ordinateur. Cette thèse vise donc à bâtir une interface de simulation interactive répondant à des besoins répertoriés dans les systèmes existants en plus d'inclure des fonctionnalités originales dont l'utilité devra être évaluée par des expérimentations sur des sujets humains.

Le troisième objectif principal de cette thèse est de *réaliser au moins deux applications concrètes* pouvant être utilisées dans des expérimentations de psychologie cognitive dont le but est de mesurer la compréhension du système complexe étudié par le participant. Le logiciel développé comprendra donc des fonctionnalités telles qu'une instrumentation visant à recueillir un historique des événements provenant des actions de l'utilisateur. Ces applications concrètes feront également intervenir des scénarios possédant plusieurs des caractéristiques des systèmes complexes, notamment la prise de décisions *pendant* l'exécution d'une simulation. Il est important de souligner que le but de cette thèse n'est pas d'évaluer expérimentalement les performances de participants dans l'utilisation du logiciel développé par rapport à des techniques classiques. Les résultats attendus sont plutôt une architecture générique d'environnement de simulation interactive appuyée par sa mise en œuvre complète et le développement d'applications concrètes montrant la flexibilité de l'architecture réalisée tant au niveau de l'interface graphique qu'au niveau du transport des données.

1.3 Problèmes abordés

En lien avec les objectifs ci-dessus, plusieurs problèmes seront abordés dans cette thèse, la plupart étant reliés aux notions fondamentales de la simulation interactive. Le premier problème est celui de l'*exploration des paramètres de simulation.* Concrètement, il s'agit de déterminer une manière générique capable d'exploiter un simulateur afin d'effectuer des changements de paramètres pendant l'exécution d'une simulation. De plus, le développement d'une interface graphique via laquelle un utilisateur a accès aux paramètres disponibles pour modifications est une tâche essentielle menant à une solution au problème. Cette interface lui permettrait donc de tester des hypothèses sur les modèles de simulation formulées à partir d'une analyse des résultats. Les modules développés devront également être intégrés dans un espace de travail permettant à l'utilisateur d'exécuter plusieurs opérations dans une application unique dérivant d'une architecture générique. Le *développement de cette architecture* constitue un problème en soi puisqu'elle devra être en mesure de s'adapter à plusieurs contextes et par le fait même exploiter plusieurs simulateurs, chacun possédant des caractéristiques particulières.

Un autre problème abordé concerne les notions reliées au *contrôle des simulations* pendant la phase d'exploration des simulations. La littérature relative à ce problème est très rare. Une solution originale, dont le but est de permettre le contrôle interactif de simulations et de représenter la progression d'un utilisateur dans l'étude d'un problème complexe, sera donc développée. En lien avec le problème précédent, celui de *transporter des données de manière générique* à partir d'un simulateur vers un logiciel d'analyse est un très grand défi. En effet, il faudra développer un modèle conceptuel de données pouvant s'adapter à plusieurs architectures de simulateurs chacune possédant une manière spécifique de structurer ses données. De même, le développement conceptuel du pipeline de données devra assurer une grande flexibilité quant aux transformations effectuées sur les données et les conversions disponibles pour que les données analysées soient pertinentes et dans un format adéquat.

Finalement, le dernier problème abordé concerne la *mise en œuvre de l'architecture* qui intègre les solutions aux problèmes mentionnés dans les paragraphes précédents. Le développement d'un tel logiciel est une lourde tâche de programmation puisque l'architecture doit être générique, donc adaptable à plusieurs situations concrètes. De plus, elle devra comporter des fonctionnalités interactives provenant de divers modules mis en correspondance à l'aide de mécanismes sophistiqués. Il sera également important de *définir un contexte d'utilisation de l'application concrète* afin d'assurer sa compatibilité avec les expérimentations sur les sujets humains.

1.4 Problèmes non-abordés

Bien que plusieurs problèmes soient abordés dans l'élaboration de cette thèse, certains, même s'ils sont intimement reliés, ne seront qu'effleurés ou tout simplement ignorés. En effet, des solutions existantes seront utilisées afin de solutionner certains problèmes alors que d'autres feront partie des travaux futurs. Les problèmes non-abordés se retrouvent dans la liste suivante :

- *Le développement d'un module de visualisation de données provenant de la simulation de systèmes complexes en vue d'analyse.* Les outils de visualisation utilisés seront commerciaux ou disponibles gratuitement. Un environnement de visualisation générique pouvant être exploité par le logiciel développé dans cette thèse est envisagé dans les travaux futurs.
- *Le développement de simulateurs.* Dans le cadre de cette thèse, des simulateurs existants sont exploités. Il se peut que certaines modifications soient nécessaires, mais celles-ci se situeront seulement au niveau de la collecte des données et du contrôle de la simulation.
- *Le développement de modèles de simulation.* En lien avec le point précédent, le but de cette thèse est d'utiliser des modèles existants et non de les développer. Par contre, la configuration de modèles afin de réaliser un scénario de simulation pourrait être un travail à effectuer pour la réalisation des objectifs.
- *L'exploitation de simulateurs de phénomènes physiques qui nécessitent de plusieurs heures à plusieurs jours à s'exécuter et qui génèrent des téraoctets de données.* Les simulateurs exploités sont plutôt ceux qui fournissent des données exploitables pour l'analyse, que ce soit des données intermédiaires ou des données résultantes, dans un temps de l'ordre de la seconde à la minute. Ce critère est nécessaire afin que le système développé demeure interactif.
- *L'exploitation de simulations stochastiques.* Puisque des décisions doivent être prises pendant l'exécution, il est impossible de gérer plusieurs réplications simultanément. Ainsi, seulement les simulations déterministes sont considérées dans cette thèse.
- *L'exécution de simulations sur une grille ou une ferme de calcul.* Dans cette thèse, des ordinateurs dotés de plusieurs processeurs seront utilisés. Il est assumé que ces ordinateurs possèdent une puissance de calcul suffisante pour les besoins courants.
- *L'évaluation du logiciel de simulation interactive développé sur des sujets humains.* Bien que la mise en œuvre du logiciel de simulation interactive soit complétée et que le développement d'applications concrètes soit envisagé, l'évaluation sur des sujets humains des outils interactifs originaux ne fait pas partie de cette thèse.

1.5 Contributions

Cette section dresse une liste des contributions apportées à la science par les travaux de cette thèse :

- *Développement d'un cadre conceptuel de simulation interactive, Multichronia, permettant une exploration de différents espaces via quatre boucles d'interaction.* Basé sur des besoins répertoriés dans la littérature et sur des besoins originaux, le cadre conceptuel Multichronia (des racines grecques « *multi* » pour « plusieurs » et « *chronia* » pour « temps ») offre plusieurs fonctionnalités qui ne font pas partie des systèmes existants. La boucle interactive d'exploration de l'espace des simulations en est le meilleur exemple. En effet, elle permet à l'utilisateur, via l'interface graphique associée, d'effectuer plusieurs opérations qui ne sont pas offertes par des systèmes semblables.
- *Formalisation de la représentation visuelle et des opérations interactives associées aux boucles d'exploration des paramètres et des simulations.* L'interface graphique qui supporte Multichronia, l'arbre multichronique, offre à l'utilisateur une représentation formelle de son travail d'analyse. Basé sur une représentation retrouvée dans la littérature, l'arbre multichronique offre davantage de fonctionnalités interactives. De plus, il a été conçu afin que son utilisation soit intuitive et que sa représentation visuelle soit informative.
- *Modèle conceptuel définissant une méthodologie employée lors de la construction du modèle de données d'un simulateur visant à être exploité par Multichronia.* Dans bien des cas, les modèles conceptuels de données pour les systèmes de simulation interactive sont *ad hoc*, c'est-à-dire qu'ils ne sont pas standards pour toutes les applications d'un système. Dans d'autres cas, les données sont transférées dans un format propriétaire. Il est alors impossible de les utiliser avec un logiciel de visualisation externe. D'autre part, l'instrumentation d'un simulateur, une opération visant à récupérer les données de simulation à l'aide d'appels de fonctions insérés dans le code du simulateur, nécessite typiquement énormément de travail. Dans Multichronia, le modèle conceptuel de données est générique et il utilise un standard couramment employé dans plusieurs domaines. De plus, il propose une méthodologie quasi-automatisée pour l'instrumentation d'un simulateur.
- *Mise en œuvre d'une application test pouvant être utilisée lors d'expérimentations de psychologie cognitive sur des humains afin de déterminer l'impact de divers outils de simulation interactive sur la compréhension d'un système complexe.* Puisque les dernières expérimentations sur des sujets humains qui évaluent l'impact d'outils de simulation interactive sur leur performance à solutionner un problème datent de plus de dix ans, la mise en œuvre d'une application test afin

de réaliser de telles expérimentations constitue une contribution certaine. Il s'agit également du premier logiciel de simulation interactive employant des techniques modernes d'interaction à être évalué expérimentalement.

1.6 Résultats

L'implantation du logiciel développé pour cette thèse sera partie intégrante de la première version de IMAGE, une suite d'outils visant à améliorer la performance de ses utilisateurs dans la compréhension de systèmes complexes [Lizotte *et al.*, 2008]. Le projet IMAGE, géré par Recherche et Développement Défense Canada (RDDC) Valcartier, a guidé le développement des applications concrètes proposées dans cette thèse. Les résultats présentés dans cette thèse sont les suivants :

- L'architecture générique du cadre conceptuel Multichronia, dont le but est de fournir une base flexible pour des applications concrètes de simulation interactive.
- L'implantation d'un prototype du cadre conceptuel Multichronia, incluant une implantation de l'arbre multichronique qui est essentiel afin de représenter visuellement le cheminement de l'utilisateur dans son processus d'exploration du problème.
- Deux applications concrètes dérivées du cadre conceptuel Multichronia qui seront utilisées afin d'évaluer expérimentalement la performance des utilisateurs face aux outils interactifs proposés.
  - Un contexte de simulation tactique d'attaque de convois militaires par des insurgés qui utilise le simulateur multiagent Pythagoras.
  - Un contexte de simulation stratégique d'attaque de convois militaires par des insurgés qui utilise un simulateur de coévolution entre les deux clans.
- Des résultats qualitatifs venant renforcer des choix de design effectués lors du développement de Multichronia.

Les concepts et résultats présentés dans cette thèse ont également été le fruit de communications effectuées lors d'ateliers de travail ou de conférences nationales et internationales, énumérées ci-dessous :

- F. Bernier et F. Rioux. *Convoy Scenario for Complexity Study - Co-evolution for strategic red-teaming*, Rapport technique DRDC Valcartier, 2008
- F. Rioux, F. Bernier et D. Laurendeau, Multichronia – a Generic Parameter, Simulation, Data, and Visual Space Exploration Framework, Interservice/Industry Training, Simulation & Education Conference, Orlando, Décembre 2008

- F. Rioux, F. Bernier et D. Laurendeau, Design and Implementation of an XML-Based, Technology-Unified Data Pipeline for Interactive Simulation, Winter Simulation Conference, Miami, Décembre 2008
- F. Rioux, F. Bernier et D. Laurendeau, Visualizing and Interacting with Multiple Simulations Using the Multichronic Tree, Modeling, Simulation and Visualization Conference, Las Vegas, Juillet 2008
- F. Rioux, Visual Interactive Simulation and Data Farming : How Can They Co-Exist ?, International Data Farming Workshop 16, Monterey, Avril 2008
- F. Rioux, R. Drouin et D. Laurendeau, A Generic Framework for Simulation Parameter Space Exploration, Colloque REPARTI, Montréal, 2007 (Poster)

1.7 Structure de la thèse

Le Chapitre 2 présente une revue exhaustive de la littérature sur les sujets pertinents à cette thèse. Notamment, les principales caractéristiques des systèmes complexes sont énumérées, suivi d'une description du processus de résolution de problèmes par simulation, soit la séquence « modélisation, exécution et analyse ». Ensuite, les designs d'expérience sont introduits et critiqués. Un exemple montrant pourquoi les méthodes classiques de résolution de problèmes ne peuvent pas être employées dans l'étude des problèmes complexes est exposé. La simulation interactive est présentée comme une solution complémentaire. Des logiciels de simulation interactive sont ensuite énumérés, et le chapitre se termine avec une revue des environnements de travail intégrés.

Le Chapitre 3 présente le cadre conceptuel Multichronia de manière exhaustive. Il s'attarde notamment sur les différentes unités de transformation formant le pipeline de données qui s'étend d'un simulateur jusqu'à un logiciel de visualisation. Par la suite, les différentes boucles d'exploration sont présentées, soit une boucle d'exploration pour l'espace des paramètres, une autre pour l'espace des simulations, une autre pour l'espace des données et une dernière pour l'espace visuel. Chacune de ces boucles est décrite en détails. Finalement, le gestionnaire Multichronia est présenté. Ce dernier implante des fonctionnalités telles que la tenue de livre des simulations instanciées et le transfert de métadonnées entre les unités de transformation.

Le Chapitre 4 débute par une revue des interfaces de contrôle de simulation qui utilisent un arbre afin de représenter le travail accompli par un utilisateur. L'arbre multichronique est ensuite introduit, accompagné de toutes les opérations qu'il supporte. Une discussion sur son intégration dans le cadre conceptuel Multichronia suit, accompagnée de quelques contraintes imposées sur un simulateur afin qu'il soit exploité

par Multichronia. Ce chapitre se termine par une discussion sur l'automatisation de certaines opérations facilitant le travail d'un utilisateur lors de son analyse.

Le Chapitre 5 traite du pipeline de données générique qui est inclus dans le cadre conceptuel Multichronia. Un modèle conceptuel de données est tout d'abord présenté, suivi du flot de données associé. Ce modèle est par la suite spécialisé afin que le XML soit le format de données principal circulant dans le flot de données associé. Une méthodologie de conception du modèle de données d'un simulateur est finalement présentée.

Le Chapitre 6 introduit le scénario d'expérimentation de Multichronia qui sera utilisé dans le projet IMAGE. Des généralités sur les attaques de convois militaires par des insurgés sont tout d'abord présentées, suivi du scénario tactique proprement dit pour lequel les détails de la dynamique sont disponibles à l'Annexe A. Le scénario stratégique est ensuite introduit, menant à une description du rôle de l'utilisateur dans l'exploitation de ces deux scénarios.

Le Chapitre 7 relate les résultats qu'a produits cette thèse et en discute. Une vue globale des paquetages de l'architecture est tout d'abord présentée, décrivant plus en détails comment elle a été découpée. Une description de l'implantation, centrée sur les opérations de base et les fonctionnalités supplémentaires qui ont été développées afin d'améliorer l'expérience interactive d'un utilisateur, est par la suite exposée. Le chapitre se poursuit avec une discussion sur l'implantation des deux contextes concrets. Celle relative au contexte de simulation tactique de convois militaires traite entre autre des modifications qui ont été réalisées dans Pythagoras afin qu'il puisse être exploité par Multichronia. Pour sa part, la discussion sur l'implantation du scénario stratégique d'attaque de convoi discute d'un simulateur développé spécifiquement afin d'être exploité par Multichronia. Quelques mesures qualitatives justifiant certains critères de design de Multichronia terminent le chapitre.

Le Chapitre 8 conclut la thèse et présente quelques travaux futurs dont certains sont déjà en cours de réalisation.

Chapitre 2

Revue de littérature

Ce chapitre présente une revue des travaux pertinents pour l'élaboration de la thèse. Premièrement, les systèmes à l'étude dans les travaux actuels seront présentés. Ces systèmes dit « complexes » sont courants dans plusieurs champs de recherche et sont typiquement difficiles à comprendre et à gérer pour les humains. Deuxièmement, le domaine de la simulation sera présenté avec une emphase sur la simulation des systèmes complexes. S'en suivra une discussion sur la simulation interactive, qui consiste à intégrer un humain dans le processus d'exploration basé sur la simulation. Finalement, différents espaces de travail intégrés seront revus ainsi que quelques méthodes d'analyse facilitant le travail du concepteur de modèles de systèmes complexes et l'intégration de différents outils.

2.1 Systèmes complexes

Les systèmes complexes sont présents dans plusieurs sphères d'activités en raison des moyens technologiques qui permettent de connecter plusieurs systèmes jadis indépendants. C'est pourquoi l'étude de la complexité engendrée par les systèmes complexes est l'objet de nombreuses recherches. Par contre, les systèmes complexes ont toujours fait partie de la réalité des humains quand on considère par exemple les systèmes biologiques et les sociétés. Ces deux derniers exemples font partie des systèmes les plus complexes connus à ce jour, mais à deux échelles totalement différentes. L'un possède des milliards de cellules qui interagissent pour donner un être vivant tandis que l'autre est composé de personnes qui interagissent également entre elles et qui sont à la source de différents phénomènes sociaux (e.g. guerres civiles, manifestations pacifistes, économie).

2.1.1 Caractéristiques des systèmes complexes

Bien qu'ils soient en apparence totalement différents, les systèmes biologiques et les sociétés possèdent des caractéristiques propres aux systèmes complexes [Poussart, 2006]. En voici quelques exemples :

- Plusieurs composantes inter-reliées.
- Liens non-linéaires entres les composantes.
- Présence de boucles de rétroaction entre les composantes.
- Présence de délais dans le système (et dans les boucles de rétroaction).

Ces caractéristiques donnent lieu à des manifestations propres aux systèmes complexes :

- Émergence de comportements.
- Effet « papillon ».
- Équilibre fragile.
- Auto-organisation.

La vie en soi est un exemple de comportement émergent provenant d'un système trop complexe pour qu'il soit entièrement compris encore aujourd'hui. Il existe aussi plusieurs manifestations de l'effet papillon au niveau de la société. Par exemple, une simple « discussion de salon de coiffure » a donné lieu à un scandale politique et à la démission d'un ministre ! L'équilibre fragile des différents écosystèmes présents sur notre planète démontre à quel point ils peuvent être complexes. Finalement, les organismes vivants sont des exemples typiques d'auto-organisation et d'adaptation.

Il est important de faire la distinction entre *complexe* et *compliqué*. Un système compliqué peut être formé de milliers de sous-systèmes sans pour autant être considéré comme complexe [Edmonds, 1999]. Par exemple, l'ordinateur sur lequel le présent document a été conçu est un système compliqué pour un individu ayant des notions de base d'électronique, de physique et de thermodynamique. Par contre, probablement que la même personne aurait du mal à comprendre le fonctionnement d'une bactérie présente sur le clavier de ce même ordinateur. Il est donc important de considérer les connaissances *a priori* sur un sujet qui entrent en ligne de compte dans la description des systèmes complexes ; quelque chose de complexe pour l'un peut seulement être compliqué ou encore simple pour l'autre [Hübler, 2007]. Il existe en réalité un continuum caractérisant les systèmes allant de simple jusqu'à chaotique. Un système chaotique en est un pour lequel il est impossible de prédire le comportement résultant seulement en regardant ses paramètres initiaux [Johnson, 2007]. Les résultats peuvent être stables ou

instables pour une faible variation des paramètres initiaux. Il s'agit d'une manifestation extrême de l'effet papillon.

2.1.2 Aspects cognitifs reliés à la complexité

Il est en général relativement facile d'identifier des systèmes complexes avec lesquels la plupart des individus auront de la difficulté. Dörner, dans son livre « The Logic of Failure – Recognizing and Avoiding Error in Complex Systems », le fait très bien en proposant des systèmes non linéaires et couplés en apparence simples mais comportant un grand nombre d'éléments en interaction. Ainsi, la plupart des participants de son expérimentation ont eu énormément de difficulté à les gérer [Dörner, 1996]. Plusieurs chercheurs, notamment en psychologie cognitive, se sont demandés pourquoi les humains avaient tant de difficulté à gérer de tels systèmes. La réponse à cette question vient en plusieurs volets. Premièrement, quand il s'agit de gérer un nouveau système avec lequel un individu n'est pas familier, les biais cognitifs dominent souvent sa manière de penser lors de la prise de décisions [Arnott, 1998]. Par exemple, la personne exposée à un système complexe peut nier certaines informations qui ne s'apparentent pas à ce à quoi elle s'attend. De plus, les individus ont plutôt tendance à s'attarder sur un aspect particulier du système plutôt que de l'étudier dans son ensemble. Il s'agit d'un comportement bien connu des humains qui ont tendance à voir un système de manière réductionniste plutôt que de manière holistique [Dörner, 1980]. Or, une vision holistique d'un système complexe est essentielle afin de prendre en compte tous les aspects du problème et afin de parvenir à une meilleure compréhension de l'ensemble [Poussart, 2006].

Étant donné la piètre performance des humains à gérer un système complexe, il est essentiel de spécifier le but visé par la personne qui étudie un tel système. On retrouve habituellement quatre principaux buts, soit apprendre (acquérir une meilleure compréhension à l'aide d'une rétroaction des actions portées) sur le fonctionnement d'un système complexe, résoudre des problèmes impliquant un système complexe, prendre des décisions impliquant un système complexe et formuler des politiques (décisions robustes pouvant être utilisées dans plus d'un contexte) impliquant un système complexe [Spector, 2008]. Tout dépendant du but, il sera plus efficace de manipuler directement le système (*experiential* ou *procedural learning*) avec ou sans but spécifique ou de seulement observer les actions effectuées par une autre personne dont le but était de comprendre le système [Osman, 2008]. Ces procédures requièrent toutefois le développement et la validation de méthodes permettant la représentation et la modélisation des systèmes complexes [Spector, 2008].

2.1.3 Modélisation des systèmes complexes

Quelques techniques de modélisation existent afin de représenter un système complexe par un modèle, c'est-à-dire une représentation standard comprise par un grand nombre de personnes. Premièrement, la méthode de la dynamique des systèmes (*System Dynamics*) de Sterman permet d'encapsuler sous la forme d'un diagramme de boucles causales (*Causal Loop Diagram*) les relations entre plusieurs entités [Sterman, 2004]. Parmi ces relations, on retrouve les boucles de rétroaction (positive et négative) et les délais, deux éléments propres aux systèmes complexes. Les diagrammes de boucles causales les plus simples peuvent être résolus analytiquement par des équations différentielles. Cependant, il est en général difficile de résoudre des systèmes complexes décrits par des diagrammes de boucles causales en raison de la pénible tâche qui est de résoudre analytiquement les équations associées.

Un deuxième type de modélisation de systèmes complexes fréquemment utilisé dans la littérature est la modélisation par des systèmes multiagents [Iba, 2004]. Un *agent* est une entité ayant la capacité de percevoir son environnement, de raisonner à propos de cet environnement et de poser des actions afin de modifier son état ou afin de modifier l'environnement [Jennings et Wooldridge, 2000]. Dans un système multiagents, plusieurs agents font partie du même environnement et interagissent donc les uns avec les autres. La présence d'interactions diverses entre plusieurs agents rend un tel système complexe. Un agent peut, par exemple, modéliser un organisme vivant comme une fourmi, ou encore un soldat sur le champ de bataille. Contrairement à la modélisation par dynamique de systèmes, il est très difficile, voire impossible, de résoudre un système multiagent analytiquement [Drogoul *et al.*, 2002]. Toutefois, il serait possible de construire des agents réels de type robots, de les placer dans un environnement physique et de les laisser interagir afin d'observer leur comportement.

La technique la plus utilisée quand vient le temps de résoudre des systèmes complexes et de calculer les résultats face à des changements dans les paramètres des modèles est la simulation par ordinateur [Johnson, 2007]. Dans le cas des systèmes complexes modélisés par la dynamique des systèmes, la simulation par ordinateur consiste à calculer, à l'aide de méthodes d'intégration numériques, les valeurs associées à chacun des éléments du modèle et à faire avancer le temps. Dans le cas d'un système multiagents, l'environnement de même que les agents sont modélisés de manière numérique. La section suivante décrit plus en détails les avantages reliés à la simulation, de même que les techniques d'analyse typiquement utilisées par les modélisateurs de systèmes complexes dont le but est de comprendre les systèmes sous étude.

2.2 Simulation

De plus en plus de problèmes sont étudiés à l'aide d'outils de simulation par ordinateur. Les scientifiques qui étudient les systèmes complexes bénéficient particulièrement de la simulation par ordinateur pour l'exploration de leur problème afin de les aider à prendre de meilleures décisions lorsque des choix se présentent [Robinson, 2004]. Les scientifiques qui travaillent avec des systèmes physiques ont également recours à la simulation par ordinateur afin de modéliser et explorer leurs problèmes. De plus en plus de chercheurs dans des domaines tels que la médecine et la biologie profitent des avantages reliés à la simulation afin de modéliser leurs objets d'étude, de les simuler et d'en retirer de précieuses informations avant de procéder à des expériences concrètes [Cohen et Harel, 2007; Lloyd *et al.*, 2007].

La Figure 2.1 montre le processus classique sur lequel les utilisateurs de simulation s'appuient généralement [Fishwick, 1997]. Les tâches correspondant aux modules (rectangles courbés de la Figure 2.1) peuvent être réalisées soit individuellement, en équipe ou de manière automatique tout dépendant de la complexité du problème étudié. Dans un premier temps, il est essentiel de passer par le module de *modélisation de la simulation* afin de modéliser le problème sous étude à l'aide d'un formalisme ou d'un langage approprié (e.g. Arena [Rockwell Automation Inc., 2009], Flexsim [Flexsim, 2009], GPSS/H [Henriksen et Crain, 1989], Simul8 [Hauge et Paige, 2001]). Le but de la modélisation est de construire une abstraction du problème étudié pouvant être utilisée afin d'expérimenter différentes variations du modèle [Robinson *et al.*, 2004]. La résultante du module de modélisation comporte typiquement deux éléments : le modèle et le scénario. Dans la littérature, on confond parfois le modèle et le scénario [Zeigler *et al.*, 2000], mais dans la présente thèse ils représentent deux entités distinctes [Harrison *et al.*, 2005]. Le modèle renferme des équations mathématiques affectées au calcul de l'état de la simulation à partir de l'état précédent et des structures de données stockant l'état courant d'une simulation. Pour sa part, le scénario comprend les paramètres qui décrivent l'état initial de la simulation ainsi que toutes les données venant perturber

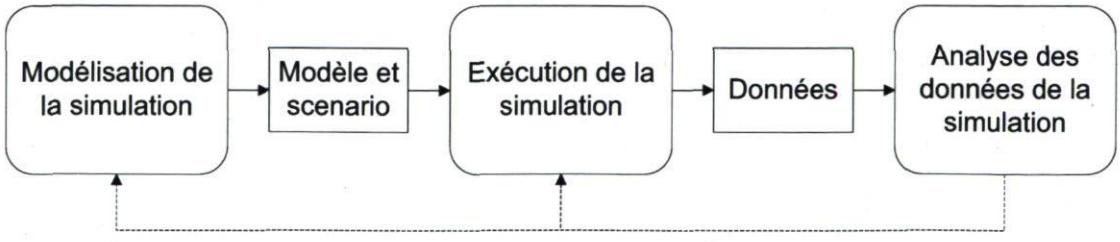


FIGURE 2.1 – Processus de modélisation, exécution et analyse de simulation

les paramètres du modèle pendant son exécution. Dans la présente thèse, nous supposons que les modèles étudiés sont disponibles mais qu'il est possible d'apporter des modifications au scénario et aux paramètres associés aux modèles.

Le module d'*exécution de la simulation* accepte en entrée le modèle de simulation ainsi que le scénario. Il est essentiel à ce moment que les modules de modélisation et d'exécution soient compatibles en termes d'échange de données. Le module d'exécution de la simulation se charge d'exécuter les calculs inclus dans le modèle de simulation et ajuste périodiquement l'état de la simulation. Le moment auquel survient cet ajustement dépend du formalisme de simulation utilisé. Dans le cas d'une *simulation basée sur le temps continu* (*continuous time simulation*), l'état de la simulation est ajusté à chaque pas de temps, dont la valeur (Δt) dépend de plusieurs facteurs [Zeigler *et al.*, 2000]. Par contre, dans le cas d'une *simulation basée sur les événements discrets* (*discrete event-based simulation*), l'état de la simulation est ajusté lorsqu'un nouvel événement survient. Ces deux méthodes présentent des avantages et des inconvénients qui sont discutés dans le livre de Zeigler *et al.* (2000). Les résultats du module d'exécution de la simulation sont des données envoyées au module suivant soit périodiquement ou lorsque la simulation est terminée.

Le module d'*analyse des données de simulation* accepte les données produites par le module d'exécution de la simulation afin qu'elles soient étudiées, analysées, décortiquées, etc. par un utilisateur. Typiquement, l'utilisateur visualisera les données avec un logiciel approprié de visualisation interactive [Johnson *et al.*, 1999]. Suite à cette analyse, l'utilisateur aura la possibilité de revenir sur la modélisation ou encore d'effectuer des exécutions additionnelles (lignes pointillées de la Figure 2.1).

La section suivante montre que la simulation présente des avantages marqués face à d'autres techniques d'analyse. Par la suite, une revue critique des techniques d'expériences basées sur des simulations et d'analyse de données de simulations sera présentée.

2.2.1 Avantages, désavantages et difficultés reliés à la simulation

La simulation présente plusieurs avantages par rapport à l'utilisation de systèmes physiques. En effet, historiquement, la première simulation a eu lieu durant la deuxième guerre mondiale dans le cadre du projet Manhattan afin d'étudier le processus de détonation de la bombe atomique [Wikipedia, 2009a]. L'avantage évident de modéliser et de simuler ce problème, par rapport à construire et faire exploser une vraie bombe, est

le danger extrême associé au système réel, et également le coût de fabrication élevé. Ce dernier avantage est crucial pour les scientifiques qui développent de nouvelles puces électroniques. Réaliser chacun des prototypes serait beaucoup trop dispendieux et rendrait le coût du design prohibitif. Ainsi, des logiciels de simulation très performants qui tiennent compte de tous les effets physiques intervenant dans les circuits électroniques sont utilisés pour simuler les puces et leur interaction avec les autres circuits bien avant le processus de fabrication [Denk, 2007]. De même, la simulation d'un système permet dans certains cas d'obtenir des résultats plus rapidement qu'avec le système réel (e.g. simulation des changements climatiques, simulations économiques). Dans d'autres cas, les résultats de simulation parviennent à l'utilisateur sur une échelle de temps plus longue par rapport au système réel (e.g. processus chimiques, explosions). La simulation permet de fournir des données qui ne seraient pas disponibles en utilisant les instruments de mesure actuels (e.g. écoulement de fluides, tensions dans un matériau). Finalement, Sterman (1994) propose que des outils de simulation (monde virtuel, *virtual world*) puissent être utilisés afin d'en apprendre davantage sur ce monde réel (Figure 2.2). Le monde virtuel permet aux utilisateurs de tenir des *expériences contrôlées* souvent impossibles à réaliser dans le monde réel, facilitant ainsi l'apprentissage du système sous étude, un processus interactif et itératif. Des tests d'hypothèses du type « qu'arrive-t-il si ? » (*What-if ?*) peuvent également être accomplis [Philippakis, 1988].

Bien que la simulation comporte plusieurs avantages par rapport au système réel, il n'en demeure pas moins que les utilisateurs se doivent de rester vigilants et de garder leur esprit critique face aux résultats obtenus suite à l'exécution d'une simulation [Robinson, 2004]. En effet, les modèles établis dans l'étape de modélisation de la simulation ne sont que des approximations du monde réel. Ils sont plus ou moins fidèles à la réalité, ce qui signifie que les données qui en résultent ne sont pas nécessairement correctes. Il existe par contre une foule de processus de validation et de vérification qui sortent du cadre de cette thèse [Law, 2006].

Les aspects probabilistes reliés aux modèles de simulation stochastiques ne sont pas à négliger lors d'une étude visant à extraire des données quantitatives d'un modèle [Law, 2006]. En effet, les données résultant de l'exécution d'une simulation stochastique ne sont qu'un échantillon de la fonction de distribution de probabilité des données de sortie. Pour obtenir des données statistiquement valides, il est impératif d'exécuter plusieurs réplications d'une simulation stochastique afin de calculer des statistiques sur les données de sortie et ainsi obtenir une courbe de distribution des probabilités. Dans la présente thèse, nous considérons seulement les simulations déterministes, c'est-à-dire celles ne comportant pas de variables incertaines. Par contre, les travaux pourraient être étendus aux simulations stochastiques comme il est discuté dans la section traitant des travaux futurs.

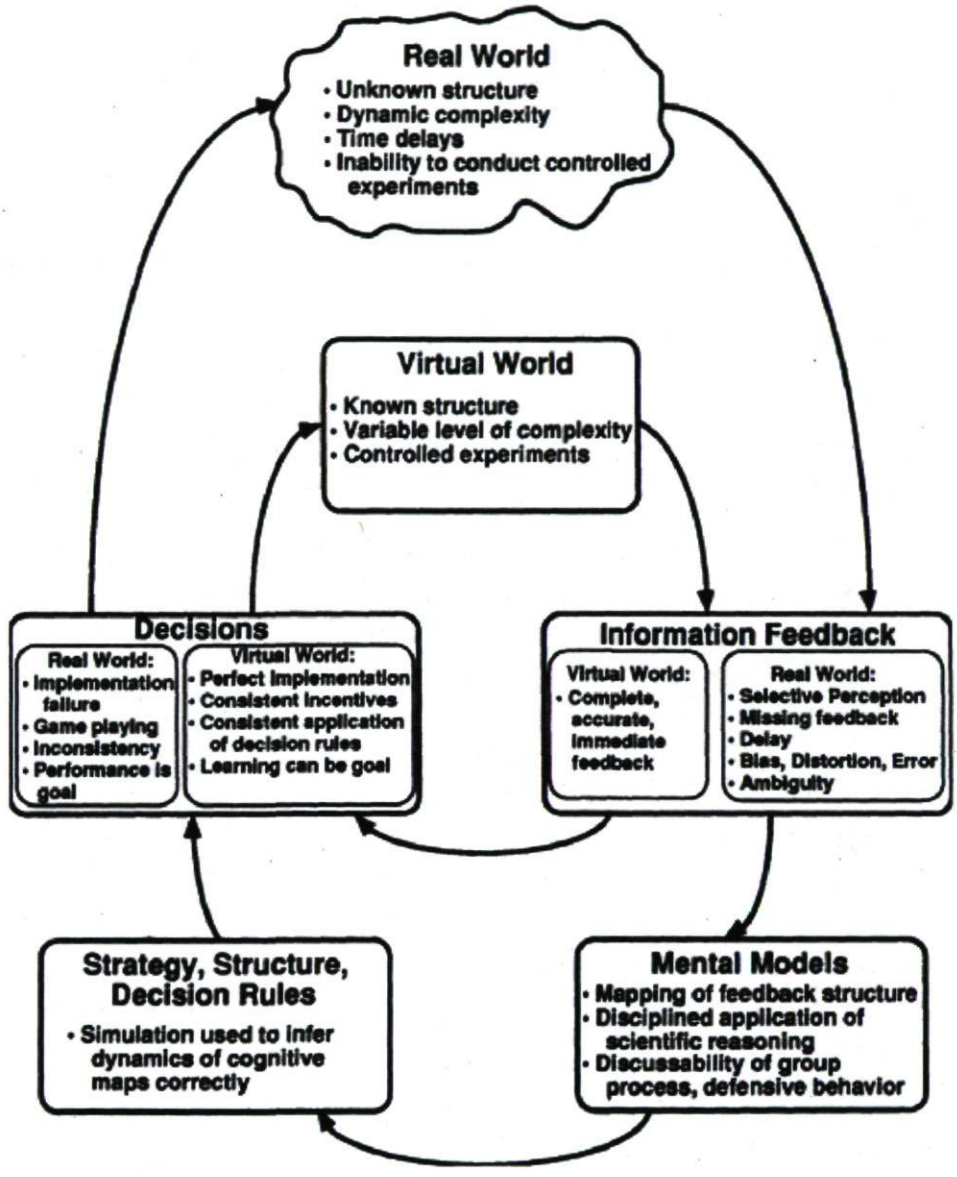


FIGURE 2.2 – Boucle d'apprentissage idéalisée. L'apprentissage implique une expérimentation continue entre le monde réel et le monde virtuel (simulation) [Sterman, 1994]

2.2.2 Expériences par ordinateur ou études computationnelles

Tout comme pour les expériences impliquant des systèmes réels, il est essentiel en simulation de procéder, suite à une modélisation adéquate, à la cueillette des données. Le processus de génération des données porte plusieurs noms dans la littérature, dont « expérience par ordinateur », « simulation en lot » et « expérience *in silico* » (pour faire contraste aux expériences traditionnelles *in vivo* ou *in vitro*). Un *design d'expérience* est la procédure formelle à suivre afin de générer des données statistiquement valides à l'aide des modèles de simulation [Santner *et al.*, 2003]. Dans un design d'expérience, les paramètres d'entrée sont appelés *facteurs* tandis que les données de sortie sont appelées *réponses*. La quantité de données nécessaires pour réaliser l'analyse dépend de plusieurs paramètres, dont le nombre de facteurs influençant le modèle exécuté pendant la simulation, la variabilité du modèle, de même que la précision requise pour l'analyse de la réponse.

Le design d'expériences pour les systèmes réels est bien connu dans la littérature [Montgomery, 2001], avec plusieurs succès dans divers champs d'intérêts (e.g. agriculture, médecine, industrie). Plusieurs techniques existent afin de minimiser la quantité d'échantillons à recueillir dans l'espace des facteurs. Les techniques d'analyse associées à ces designs d'expériences sont également bien connues. Par contre, étant donné que la simulation diffère en plusieurs points des systèmes réels, des techniques particulières ont été mises au point afin de prendre en compte ces différences tout en fournissant des designs d'expériences optimisés en termes de charge de calcul et de temps nécessaire pour l'exécution. Comme proposé par Kleijnen *et al.* (2005), il existe plusieurs différences entre les designs d'expériences classiques et les designs d'expériences par ordinateur.

- **Nombre de facteurs potentiels** : dans le cas d'une expérience sur un système réel, le nombre de facteurs considérés est limité. Cette limite est imposée par le nombre d'échantillons nécessaires afin que les résultats soient statistiquement valides et parce qu'il est impossible de tout contrôler dans le monde réel ; la valeur de certains facteurs sera le fruit du hasard, d'autres seront tout simplement impossibles à ajuster vers une valeur souhaitée. Dans le cas d'une expérience par ordinateur, le nombre de facteurs est typiquement beaucoup plus élevé, allant de quelques-uns à quelques dizaines voire même des centaines. La limite au nombre de facteurs considérés est imposée par le temps de calcul requis afin de simuler toutes les combinaisons de facteurs nécessaires pour un design d'expérience. De plus, étant donné la nature mathématique des modèles, le concepteur de l'expérience a un contrôle absolu sur la valeur de chacun des facteurs.
- **Choix de la mesure de performance** : dans le cas d'une expérience sur un système réel, la mesure de performance est déterminée dès le début de l'expérience.

Il en est ainsi parce qu'il y a un coût associé à la mesure des réponses dans les systèmes réels en termes de capteurs ou collecte de données sur le terrain. Par contre, étant donné la facilité avec laquelle les données sont accessibles dans les systèmes simulés, il devient parfois difficile pour un concepteur d'expérience de choisir une mesure qui viendrait traduire un succès ou un échec. De plus, les résultats souhaités peuvent être des mesures relatives d'un ensemble de facteurs par rapport à un autre et non pas des résultats en valeur absolue.

- **Complexité de la réponse** : dans le cas d'une expérience sur un système réel, la réponse est habituellement interprétée comme étant une combinaison polynomiale des facteurs. Or, dans le cas de simulations de systèmes complexes avec plusieurs paramètres, la réponse de la simulation peut ne pas être obtenue à l'aide de simples régressions dû à la complexité de la réponse et au grand nombre de facteurs. Il est donc essentiel pour un concepteur d'expérience par ordinateur de considérer quelle sera la méthode d'analyse appropriée compte tenu des résultats envisagés.

Ces différences par rapport aux expériences sur des systèmes réels imposent aux concepteurs d'expériences de simulation une vision particulière et une manière différente d'aborder un problème. Le choix de l'expérience par ordinateur sera aussi modulé par son but. Kleijnen *et al.* (2005) énumèrent les trois principaux buts d'une expérience par ordinateur : 1) Développer une connaissance de base sur un modèle, 2) trouver des politiques ou décisions robustes aux variations dans les scénarios de simulation, 3) comparer les mérites de différentes décisions ou politiques. Il est clair que ces buts font intervenir des résultats qualitatifs plutôt que quantitatifs. Par exemple, une politique robuste sera qualifiée de « satisfaisante » (*satisficing*) au lieu d'optimale. Ces études qualitatives sont nécessaires entre autres en raison des résultats associés à la simulation des systèmes complexes qui rendent l'analyse quantitative beaucoup plus difficile [Johnson, 2007].

De son côté, Law (2006) propose que l'objectif premier d'une expérience par ordinateur soit d'effectuer une *épuration des paramètres* ou une *analyse de sensibilité* afin de garder seulement les facteurs ayant un effet significatif sur la réponse [Law, 2006]. Par la suite, quand les principaux facteurs et leurs effets sont connus, il est possible de construire un *métamodèle de simulation* ou une *surface de réponse* qui vient approximer la réponse d'un simulateur pour un ensemble de facteurs qualifiés d'importants [Kleijnen et Sargent, 2000]. Un métamodèle de simulation est un moyen de prédire la réponse d'un modèle pour une configuration de facteurs n'ayant pas été simulée. Il est également possible, à l'aide d'un métamodèle, de trouver la combinaison de facteurs qui optimise (maximise ou minimise) la réponse du modèle. La Figure 2.3 montre la relation entre le métamodèle et les autres entités dans le processus de simulation. Il est clair que le métamodèle sert à analyser le problème étudié de même qu'à valider à la fois le modèle

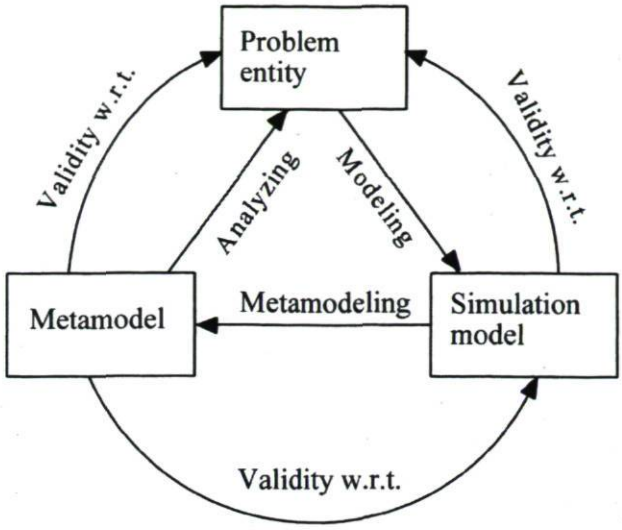


FIGURE 2.3 – Processus de création d'un métamodèle et liens avec le modèle de simulation et le problème étudié (tirée de [Kleijnen et Sargent, 2000])

de simulation et le problème étudié.

Les métamodèles de simulation sont habituellement calculés à l'aide de régressions linéaires qui relient les réponses du modèle à un facteur [Kleijnen, 1992]. Il existe des techniques permettant d'estimer des régressions pour des réponses non linéaires par rapport aux facteurs [Dos Santos, 2007]. Le Kriging est également utilisé comme remplacement des régressions linéaires [Kleijnen, 2007]. Les réseaux de neurones artificiels sont parfois utilisés afin de former le métamodèle d'une simulation. Finalement, Wang *et al.* (2007) présentent une revue complète des méthodes utilisées pour le calcul d'un métamodèle de simulation [Wang et Shan, 2007].

2.2.3 Design d'expériences par ordinateur

Après avoir fixé le but de l'expérience par ordinateur, il est essentiel de procéder au design de l'expérience en question. Ces expériences formelles reposent sur des principes statistiques qui ont fait leurs preuves pour les designs d'expérience de systèmes réels. D'ailleurs, la plupart des techniques associées aux designs d'expériences classiques (e.g. hypercube latin, 2^k factoriel, 2^{k-p} factoriel fractionné) peuvent être utilisées dans des expériences par ordinateur [Sanchez, 2005]. Par contre, tel que mentionné plus haut, étant donné les particularités des modèles étudiés, les designs classiques d'expérience doivent être adaptés à de nouveaux contextes [Sanchez, 2005]. Les principes de base du

design d'expériences sur des systèmes réels sont les suivants : contrôle, réplication et randomisation. Le *contrôle* signifie que l'expérience sera menée de manière systématique et non pas guidée par une approche d'essais et erreurs lors de la collecte des résultats. La *réplication* signifie que suffisamment de données seront recueillies afin d'obtenir des résultats fournissant de faibles intervalles de confiance et des tests d'hypothèse puissants. La *randomisation* assure que les principales sources de biais seront éliminées des données. Bien que ces trois principes soient essentiels dans les expériences sur des systèmes réels, il en est tout autrement dans le cas des expériences par ordinateur. En effet, la randomisation n'est pas nécessaire parce que tous les paramètres sont contrôlables lors de l'exécution d'une simulation [Santner *et al.*, 2003]. De plus, la contrôlabilité de tous les paramètres a donné lieu à de nouveaux designs d'expériences qui seraient difficilement réalisables dans le monde réel. Un bref survol des techniques les plus utilisées dans la communauté des expériences par ordinateur sera effectué. Le choix d'un design dépendra entre autres du nombre de facteurs à étudier, du nombre d'exécutions de simulations pouvant être complétées dans le temps alloué à l'étude et du type de métamodèle anticipé [Cioppa et Lucas, 2007]. La Figure 2.4 présente les designs d'expérience suggérés par Kleijnen *et al.* (2005) pour un nombre de facteurs donné ainsi qu'une complexité de surface de réponse anticipée. Pour un faible nombre de facteurs et une réponse de surface de faible complexité, Kleijnen *et al.* (2005) suggèrent un design d'expérience factoriel à deux niveaux. Pour un grand nombre de facteurs, ils suggèrent la bifurcation séquentielle, qui s'apparente à une recherche binaire qui élimine séquentiellement les facteurs ayant le moins d'influence sur la réponse. Lorsque la réponse anticipée est complexe, Kleijnen *et al.* (2005) suggèrent d'utiliser les hypercubes latins lorsque le nombre de facteurs est grand et un design factoriel à plusieurs niveaux lorsque le nombre de facteurs est faible.

Le design des *hypercubes latins presque orthogonaux* (*nearly orthogonal latin hypercubes*, NOLH) est un design d'expérience méritant une attention particulière étant donné la performance accrue qu'il offre comparé aux techniques classiques [De Rainville *et al.*, 2009; Cioppa, 2002]. En effet, pour un faible nombre d'exécutions, le NOLH permet d'explorer un espace de paramètres à plusieurs dimensions de manière presque orthogonale (peu de corrélation entre deux colonnes de la matrice du design d'expérience) tout en explorant adéquatement la totalité de l'espace. Cette exploration permet ainsi d'identifier les facteurs critiques, les interactions les plus importantes entre ces facteurs et la plage sur laquelle ces effets surviennent. De plus, cette technique impose peu de contraintes par rapport à la surface de réponse anticipée. Cioppa (2002) propose donc une méthodologie générale de design d'expérience pour un nombre de facteurs allant jusqu'à 22. La Figure 2.5 montre un exemple des valeurs de facteurs employées pour un design de deux à sept facteurs. Il s'agit d'une feuille Excel dans laquelle le concepteur du design doit remplir, pour chacun des facteurs considérés, la valeur minimale, la valeur

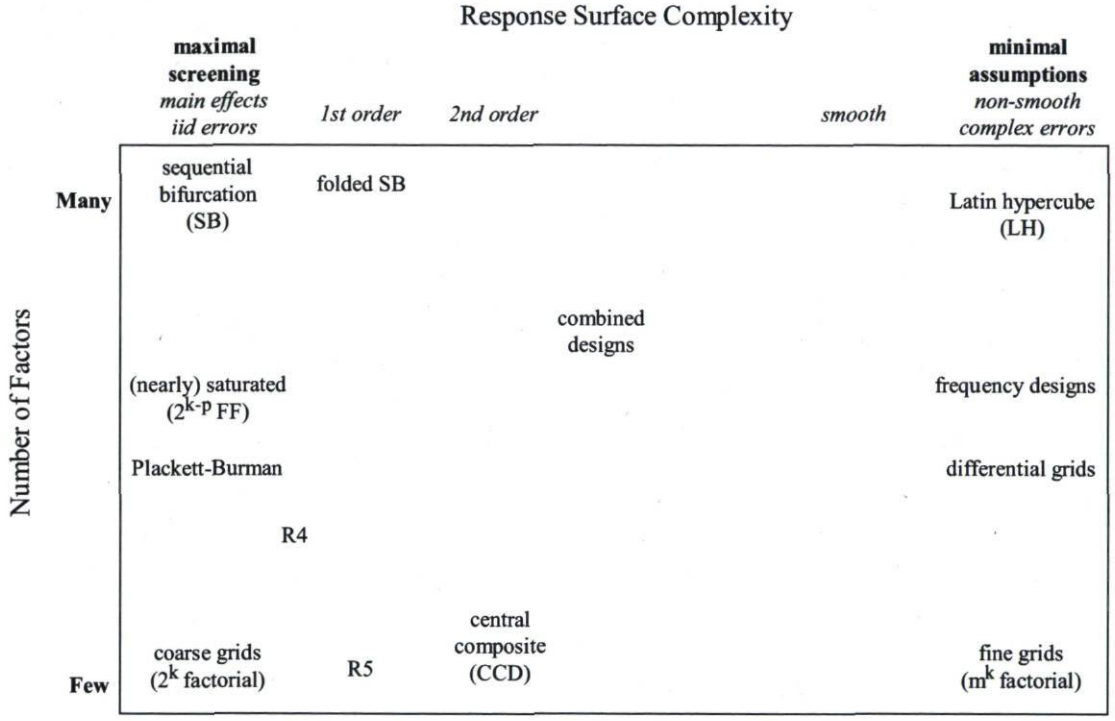


FIGURE 2.4 – Designs d'expérience par ordinateur recommandés en fonction du nombre de facteurs considérés et de la complexité de la surface de réponse [Kleijnen *et al.*, 2005]

		Factors						
	low level	1	1	1	1	1	1	1
	high level	17	17	17	17	17	17	17
	decimals	0	0	0	0	0	0	0
	factor name							
Experiment runs		6	17	14	7	5	16	10
		2	5	15	10	1	6	11
		3	8	2	5	11	14	17
		4	11	6	17	10	3	13
		13	16	8	3	6	1	14
		17	6	7	14	2	13	15
		11	4	17	6	15	8	16
		10	15	13	16	14	11	12
		9	9	9	9	9	9	9
		12	1	4	11	13	2	8
		16	13	3	8	17	12	7
		15	10	16	13	7	4	1
		14	7	12	1	8	15	5
		5	2	10	15	12	17	4
		1	12	11	4	16	5	3
		7	14	1	12	3	10	2
		8	3	5	2	4	7	6

FIGURE 2.5 – Exemple de design d'expérience par ordinateur pour sept facteurs basé sur les hypercubes latins presque orthogonaux [Sanchez, 2009]

maximale et le nombre de décimales souhaitées. En exécutant 17 simulations, le NOLH garantit que l'espace des facteurs sera exploré de manière presque orthogonale tout en couvrant la totalité de l'espace des facteurs. De même, pour 22 facteurs, 129 exécutions suffisent pour couvrir l'espace des paramètres alors qu'il aurait été nécessaire d'en exécuter 2^{22} (en supposant un facteur à deux niveaux), soit plus de 4 millions d'exécutions, dans le cas d'un design d'expérience factoriel ; il est clair que le gain en temps de calcul est considérable.

Les designs d'*expériences basées sur les fréquences* (*frequency-based experiments*, FBE) sont également intéressants étant donné la manière non traditionnelle d'aborder le problème [Sanchez *et al.*, 2006]. En effet, cette technique développée par Schruben *et al.* (1987) nécessite peu d'exécutions de simulations, mais requiert qu'une simulation atteigne un régime permanent [Schruben et Cogliano, 1987]. Une FBE consiste à varier les facteurs pendant l'exécution, chacun selon une courbe sinusoïdale oscillant à une fréquence particulière. Il suffit alors d'appliquer une transformée de Fourier sur la réponse afin de trouver le spectre de la réponse. Les facteurs ayant le plus d'influence sur la réponse sont ceux dont la valeur dans le spectre est la plus élevée. De plus, les interactions entre les facteurs deviennent visibles dans le spectre de la réponse [Schruben et Cogliano, 1981]. Il n'en demeure pas moins que plusieurs simulations ne peuvent bénéficier de cette technique étant donné qu'elles se terminent à un instant ou un autre

sans atteindre de régime permanent. Plus récemment, Wu (2002) a proposé une manière d'utiliser les designs FBE grâce à la puissance de calcul accrue des ordinateurs [Wu, 2002]. En effet, la valeur des facteurs est ajustée selon une fréquence sinusoïdale non pas pendant l'exécution de la simulation, mais au début de chacune des exécutions. Cette méthode requiert donc l'exécution d'un plus grand nombre de simulations, mais permet à un analyste de trouver les facteurs ayant une grande influence sur la réponse en analysant le spectre formé par cette réponse.

Finalement, comme le montre la Figure 2.4, il existe plusieurs autres designs d'expériences par ordinateur dans la littérature. Or, le but de cette thèse n'est pas de proposer une nouvelle méthode formelle d'expérience par ordinateur, mais bien une méthode informelle. Pour plus de détails concernant les méthodes formelles d'expériences par ordinateur, voir l'excellente revue de Kleijnen *et al.* (2005).

2.2.4 « Data farming »

La technique du *data farming* (DF) consiste à exploiter une architecture de calcul à haute performance afin d'exécuter un modèle simple des centaines, voire des milliers de fois dans le but d'explorer au maximum l'espace des paramètres de la simulation [Brandstein et Horne, 1998]. Le résultat du DF consiste en un terrain de possibilités qui est analysé afin de découvrir des tendances, des anomalies, ou, tout simplement, afin d'en arriver à une meilleure compréhension du problème sous étude. Le DF tire avantage des designs d'expériences par ordinateur afin de pouvoir explorer le plus grand sous-ensemble possible de l'espace des paramètres pour un même nombre d'exécutions de simulations. De plus, étant donné la quantité énorme de données générées, les analystes ne sont pas seulement intéressés par les tendances, mais également par les observations aberrantes (*outliers*) qui se manifestent rarement, mais qui donnent beaucoup d'information sur les conditions de succès ou d'échec d'un scénario donné.

La Figure 2.6 montre le processus du DF par lequel un analyste doit passer pour parvenir à extraire des informations du modèle. La première étape consiste à construire itérativement le scénario sous étude à l'aide de modèles appelés « distillations » (*scenario building loop*). Une *distillation* est un modèle très simple de basse fidélité, facilement configurable, qui s'exécute rapidement et qui a la capacité de fournir seulement l'essence du problème sans rentrer dans les détails [Horne et Meyer, 2004]. La deuxième étape du processus consiste à préparer l'expérience par ordinateur qui servira à générer toutes les données. Par la suite, le design d'expérience est exécuté et les données sont recueillies puis stockées dans une base de données (*scenario run space execution loop*). Enfin, les données sont analysées et visualisées de manière interactive. Si les résultats

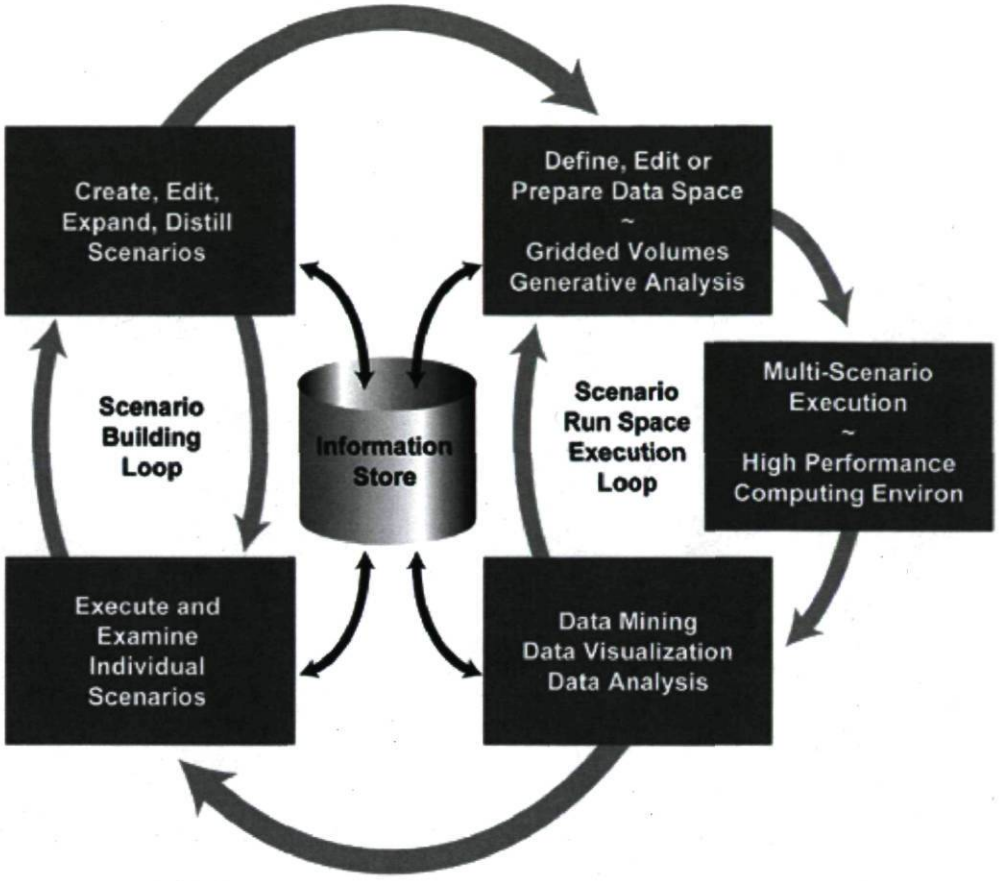


FIGURE 2.6 – Processus du *data farming* [Horne, 2001]

sont satisfaisants, le processus se termine. Par contre, il se peut que le scénario doive être retravaillé, de même que le design d'expérience qui sera éventuellement ajusté suite aux trouvailles de l'analyste.

Un modèle basé sur des agents sera typiquement utilisé afin de modéliser les problèmes étudiés dans un contexte de DF. Dans le cadre de « Project Albert », le projet duquel est né le DF, plusieurs modèles d'agents ont été développés. ISAAC [Ilachinski, 1997], MANA [Lauren et Stephen, 2002] et Pythagoras [Bitinas, 2002] en sont quelques exemples. Ces modèles sont très simples, mais permettent de capturer l'essence du problème. Ils sont également très flexibles, permettant ainsi aux analystes de modéliser un très grand nombre de scénarios seulement en ajustant quelques paramètres. Ils ne sont pas nécessairement fidèles à la réalité parce que la fidélité n'est pas une finalité du DF. La puissance du DF vient donc du fait que le modèle d'agent sera exécuté un très grand nombre de fois. En effet, un grand nombre d'exécutions permet à l'analyste d'identifier « le » cas où il s'est passé quelque chose d'imprévu dans le scénario. Les résultats obtenus sont plutôt qualitatifs que quantitatifs, améliorant ainsi la connaissance intuitive

de l'analyste [Horne et Meyer, 2004].

Un des principes de base du DF et des designs d'expérience par ordinateur est de « laisser faire le travail par le modèle ». Autrement dit, l'analyste construit un modèle de son problème, trouve un design d'expérience satisfaisant, exécute ce design de manière passive, et enfin analyse les données résultant de l'exécution du design d'expérience. L'analyse des résultats permet à l'analyste d'augmenter sa connaissance par rapport à son modèle, de même que son intuition, qui est en quelque sorte acquise avec l'expérience [Patton, 2003; Dane et Pratt, 2007].

Le DF gère avec succès les problèmes comportant quelques dizaines voire quelques centaines de paramètres. Par contre, il en est tout autrement pour les scénarios dans lesquels des décisions doivent être prises pendant l'exécution. En effet, le nombre de facteurs est exponentiel en fonction du nombre de décisions prises puisque chaque décision donne la possibilité de modifier à nouveau tout paramètre. Le nombre de combinaisons à étudier devient alors trop grand pour qu'une technique telle que le DF puisse être utilisée, même pour un modèle comportant un faible nombre de facteurs.

Dans cette thèse, une solution complémentaire au DF est proposée. Le principe de base est d'ajouter un analyste dans la boucle d'exécution du scénario afin qu'il puisse profiter de son intuition pour explorer lui-même le modèle. Donc, au lieu de laisser faire le travail par le modèle, cette thèse propose de *laisser l'analyste et le modèle travailler de pair.*

2.3 Simulation interactive

La simulation interactive implique l'interaction de l'utilisateur avec un modèle de simulation *pendant* son exécution [Standridge *et al.*, 1990]. Plutôt que de laisser le modèle travailler tout seul, l'utilisateur a la possibilité d'altérer des paramètres de simulation et de percevoir les effets résultants immédiatement. Dans le domaine de la simulation interactive, *paramètre* est employé au lieu de *facteur* afin de désigner un champ variable d'un modèle de simulation. La Figure 2.7 montre le processus générique de simulation interactive au centre duquel se trouve l'utilisateur du système. Ce dernier peut interagir avec le système tout en ayant un but précis (e.g. optimisation du système, recherche du lien entre deux paramètres spécifiques) ou sans but spécifique, seulement pour améliorer sa compréhension du modèle via l'exploration [Osman, 2008]. Le processus de simulation interactive commence par l'établissement de conditions initiales et d'un scénario de simulation. S'en suit la simulation de ce scénario qui génère des données brutes de

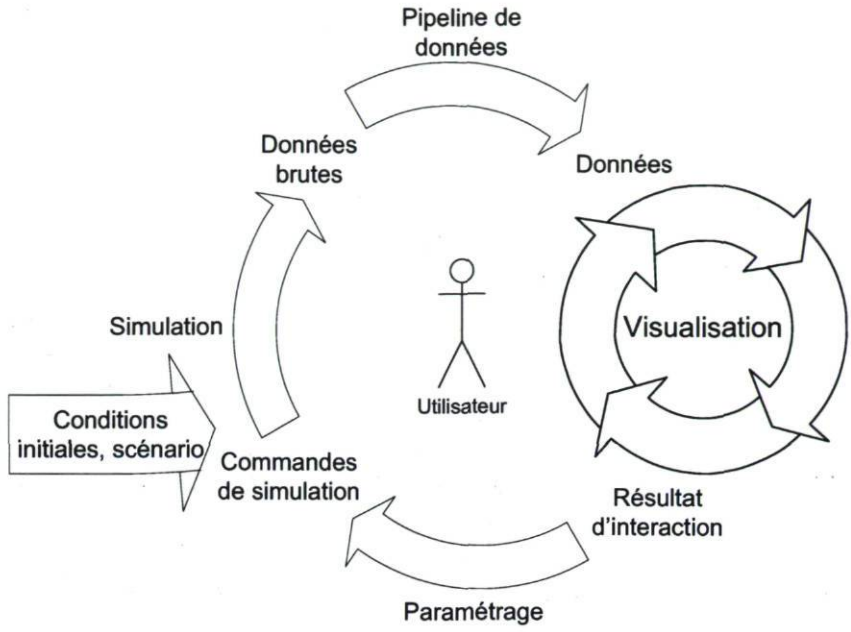


FIGURE 2.7 – Processus générique de simulation interactive (adaptée de [Wright, 2004])

simulation. Ces données passent par un pipeline de données amenant celles-ci du simulateur vers les modules d'analyse. Les données sont ensuite visualisées et analysées à l'aide d'un système de visualisation approprié [Johnson *et al.*, 1999] (e.g. représentation 2D/3D, colonnes de chiffres). À ce moment, l'utilisateur a la possibilité de modifier le cours de la simulation via une interface. Les résultats de cette interaction sont traduits par un processus de paramétrage, puis envoyés au simulateur. Cette boucle est répétée jusqu'à ce que l'utilisateur soit satisfait des résultats obtenus ou que son but soit atteint. Il est donc clair que la simulation interactive est un processus itératif et que l'apprentissage de l'utilisateur, si tel est le but de l'exploration, se fait de manière incrémentale. Cette approche rejoint également celle généralement utilisée par les humains dans leur processus d'apprentissage.

Dans la littérature, il existe quelques domaines de recherche distincts pour lesquels la notion de simulation interactive est importante. Premièrement, en recherche opérationnelle, on parle de « simulation interactive visuelle » (*Visual Interactive Simulation*, VIS) [Hurrion, 1976]. La VIS consiste à altérer des paramètres pendant l'exécution de la simulation de manière à imiter un opérateur/décideur qui viendrait prendre une décision par rapport à la poursuite temporelle de la simulation. Deuxièmement, en simulation scientifique, on parle de « pilotage de simulation » (*Computational Steering*, CS) [Brooks, 1988]. Le CS consiste, tout comme la VIS, à varier des paramètres de simulation pendant l'exécution. La différence entre la VIS et le CS réside dans les applications pour lesquelles ces deux techniques sont utilisées et dans les types de sys-

tèmes de visualisation utilisés. Troisièmement les mondes virtuels incluent souvent de la simulation interactive, dite « temps réel » [Codella *et al.*, 1992]. La simulation doit s'exécuter assez rapidement pour que l'utilisateur puisse interagir avec le monde virtuel. Les simulateurs de vol et les jeux « à la première personne » (*first person shooter*) pour lesquels les mondes sont dotés de comportement physiques en sont quelques exemples. Toutefois, ce domaine de recherche ne sera pas détaillé dans cette thèse en raison des buts différents des mondes virtuels par rapport à ceux de cette thèse.

2.3.1 Simulation interactive visuelle (VIS)

Historiquement, la VIS a vu le jour avant le CS. En effet, c'est Hurrion (1976) qui a exprimé le besoin d'introduire la prise de décision par des humains en cours de simulation étant donné que la modélisation de systèmes experts pouvant remplacer de vrais humains était très difficile dans les années 1970 [Hurrion, 1976]. Le système conçu par Hurrion permettait à un utilisateur de démarrer une simulation en spécifiant les paramètres initiaux, de visualiser le cours de la simulation, d'interrompre la simulation à un moment donné, de changer des paramètres et de reprendre la simulation en tenant compte des changements effectués. L'approche de Hurrion a été généralisée et quelques produits commerciaux comme SEE-WHY, WITNESS et GENETIK ont vu le jour [Gilman et Billingham, 1989; Concannon et Becker, 1990]. Bell et O'Keefe expliquent bien l'ampleur qu'a prise l'adoption des systèmes de VIS par quelques industries manufacturières dans les années 1980, surtout en Angleterre [Bell et O'Keefe, 1987]. Encore aujourd'hui, plusieurs logiciels de simulation commerciaux (e.g. Anylogic [X. J. Technologies, 2009] et WITNESS [Lanner Group, 2009]) possèdent des fonctionnalités de simulation interactive visuelle [Robinson, 2004]. Par contre, l'engouement des chercheurs pour les systèmes de simulation interactive a diminué vers le début des années 1990 en raison des nombreuses critiques émises par les puristes de l'analyse des données de simulation [Standridge *et al.*, 1990] de même que par des résultats expérimentaux faisant croire que la simulation interactive ne devrait pas être utilisée dans tous les contextes [Bell et O'Keefe, 1995]. Les avantages et désavantages reliés à la simulation interactive visuelle sont énumérés dans les paragraphes suivants.

L'un des avantages de la VIS est que les résultats de la simulation sont immédiatement disponibles pour l'utilisateur. À mesure que les données sont générées, il est possible de visualiser leur évolution sous forme d'animations [Bell et O'Keefe, 1995]. Par rapport à une simple analyse des résultats finaux, il est plus facile avec des animations de suivre le cours de la simulation et de détecter des évènements nécessitant une intervention de l'utilisateur [Robinson, 2004]. De plus, ce dernier a la possibilité de tester des hypothèses sur le champ, sans avoir à redémarrer la simulation à partir

du début. Cette technique de test d'hypothèse est aussi appelée « What-If? ». Dans la littérature, la VIS est également reconnue pour être utile dans le support à la prise de décisions, afin d'aider à la compréhension du système sous étude et pour aider à la communication des résultats entre le personnel technique et les décideurs [Huyet et Pierreval, 2008].

Bien que la VIS présente plusieurs avantages par rapport à la simulation en lot, il n'en demeure par moins que plusieurs ont critiqué le fait que les décisions survenant au cours de la simulation sont prises par des humains qui n'ont pas nécessairement le bagage nécessaire pour la compréhension des effets de leurs décisions [Standridge *et al.*, 1990; Kelton et Barton, 2003]. Le résultat potentiel est une exploration déformée par l'utilisateur qui possède plusieurs biais cognitifs par rapport à ce qu'est la réalité et ce qu'il perçoit, surtout en présence de systèmes complexes [Dörner, 1980]. Davis *et al.* (1991) notent également qu'il existe un certain danger à formuler des hypothèses et à en tirer des conclusions immédiates à partir des résultats observés [Davis *et al.*, 1991]. Il s'agit en effet d'un biais cognitif répandu qui se nomme « l'illusion de contrôle » parce que, souvent, les changements dans les résultats observés ne sont que le fruit d'interactions entre les paramètres du modèle et non pas une conséquence directe du changement apporté. Les utilisateurs de systèmes permettant le test d'hypothèses ont donc tendance à surestimer la contribution de ce genre de technologie dans leur prise de décision [Davis et Kottemann, 1994]. Toutefois, dans un autre contexte, Bell *et al.* (1995) ont pu démontrer que les utilisateurs d'un système de VIS obtenaient de meilleurs résultats dans l'accomplissement de leur tâche par rapport à la solution intuitive qu'ils proposaient au début de l'expérience [Bell et O'Keefe, 1995]. La solution trouvée n'était par contre pas la meilleure solution obtenue par une analyse formelle. Il est donc essentiel d'aborder avec prudence la valeur ajoutée de la VIS par rapport aux expériences formelles, et la méthodologie à adopter ainsi que les outils à fournir aux utilisateurs de systèmes de VIS [Bell et O'Keefe, 1995].

Une autre critique face à la VIS est qu'en présence de systèmes stochastiques, la résultante d'un changement de paramètre n'est qu'un échantillon tiré d'une distribution de probabilités parmi tant d'autres [Bell, 1989; Standridge *et al.*, 1990]. En effet, il est difficile de tirer une conclusion sur le résultat final d'une simulation en tenant compte d'un échantillon seulement. Comme Bell (1989) le propose, il est toutefois possible d'obtenir des informations sur ce qui a mené au résultat, c'est-à-dire sur la période transitoire pendant laquelle les modèles ont convergé vers la solution finale. Il est donc essentiel pour un utilisateur de VIS d'être vigilant par rapport aux conclusions tirées de l'exploration d'un modèle. Bell *et al.* (1995) font aussi remarquer que la VIS et les expériences formelles de simulation partagent des objectifs différents [Bell et O'Keefe, 1995]. En effet, quand les objectifs d'un problème sont difficiles à exprimer et que

les critères de performance sont difficiles à quantifier, la génération d'alternatives peut seulement parvenir d'un expert en la matière ; une expérience formelle ne saurait trouver une solution à un problème mal défini. En contrepartie, dans le cas d'un problème d'optimisation, il serait insensé de demander à un utilisateur d'exécuter la tâche. C'est pourquoi cette thèse propose qu'un logiciel de simulation interactive tel qu'un VIS soit utilisé conjointement avec des méthodes formelles d'analyse. De plus, les contextes d'utilisation des travaux proposés dans cette thèse sont orientés vers la prise de décision *pendant* l'exécution de la simulation, qui nécessite l'intervention d'un utilisateur étant donné l'impossibilité de développer des algorithmes de prise en charge automatique des décisions dû à la grande complexité des systèmes envisagés.

2.3.2 Pilotage de simulations (CS)

Le besoin de pouvoir piloter une simulation a été énoncé par McCormick *et al.* (1987) dans un contexte où des scientifiques devaient être en mesure de manipuler leur modèle, ou encore d'intervenir sur le déroulement d'une simulation scientifique [McCormick *et al.*, 1987].

« Scientists not only want to analyze data that results from super-computations; they also want to interpret what is happening to the data during super-computations. Researchers want to steer calculations in close-to-real-time; they want to be able to change parameters, resolution or representation, and see the effects. They want to drive the scientific discovery process; they want to interact with their data. »

Brooks (1988) a suggéré le développement d'interfaces et d'outils servant au pilotage de simulations, et que l'utilisateur expert d'une interface de pilotage de simulation serait plus performant qu'un ordinateur pour guider le processus d'exploration du modèle de simulation vers des domaines de l'espace des paramètres qui valent la peine d'être observés [Brooks, 1988]. Il n'a fallu que quelques années pour que des applications concrètes de pilotage de simulation voient le jour. En 1990, Marshall *et al.* ont réalisé une étude portant sur l'écoulement des fluides dans le Lac Érié [Marshall *et al.*, 1990]. Ils ont trouvé que le pilotage de simulation augmente la capacité de repérer des problèmes dans les modèles étant donné que le temps entre le changement d'un paramètre et la perception de son effet est très court. De plus, le CS permet une meilleure compréhension de ce qui se passe dans une simulation parce que les utilisateurs peuvent se concentrer sur l'observation des résultats et sur le choix des paramètre à varier, et ce, dans une seule et même application.

Suite aux travaux de Marshall *et al.* qui étaient très spécifiques à leur application,

d'autres systèmes plus généraux ont vu le jour. Ces environnements de CS ont été développés afin de résoudre des problèmes propres à des domaines spécifiques, pour la plupart de nature scientifique et à caractère très complexe. Il semble toutefois que cette approche de simulation interactive ait été épargnée par les critiques parce que, contrairement aux systèmes de VIS qui datent des années 1980, certains systèmes de CS sont très récents. Ils ont trouvé leur niche dans les applications scientifiques qui demandent une grande puissance de calcul. Les paragraphes suivants présentent une revue chronologique des environnements de pilotage de simulation.

GRASPARC

GRASPARC (*GRAphical environment for Supporting PARallel Computing*) a été développé au début des années 1990 par une équipe britannique [Brodlie *et al.*, 1993]. Il s'agit plus d'un environnement de résolution de problèmes que de pilotage de simulation, mais il possède des caractéristiques qui valent la peine d'être mentionnées.

GRASPARC est effectivement le premier environnement de simulation interactive permettant de sauvegarder l'état d'une simulation à un moment précis, de poursuivre l'exécution, puis de revenir en arrière afin de repartir une simulation à partir d'un point de sauvegarde. Ses auteurs ont conclu, d'après leur expérience sur le processus de résolution de problèmes, que l'établissement d'un arbre affichant l'historique de la simulation pouvait contribuer à la compréhension du problème considéré.

VASE

Un des premiers environnements de pilotage de simulation est VASE [Jablonowski *et al.*, 1993], développé à l'Université de l'Illinois par Jablonowski au début des années 1990. VASE, pour *Visualization and Application Steering Environment*, est un environnement de visualisation et de pilotage utilisé pour les applications scientifiques ou d'ingénierie.

Un utilisateur de VASE doit posséder le code source de l'application à piloter. Il doit tout d'abord insérer des points d'arrêt (*breakpoints*) pendant lesquels il sera possible de visualiser et piloter la simulation. Il s'agit de l'étape de la pré-compilation. En passant à l'étape de la post-compilation, l'utilisateur exécute le programme VASE qui comprend maintenant du code instrumenté. Lorsqu'un point d'arrêt est rencontré, l'utilisateur a la possibilité de piloter sa simulation ou de la poursuivre. La structure du programme

est donc entièrement prédéfinie par VASE et l'usager doit s'en contenter.

Les limitations de VASE sont nombreuses. Bien qu'il soit un logiciel très flexible, il est impossible d'exécuter une simulation entièrement interactive avec VASE. En effet, l'usager peut interagir avec la simulation seulement lorsqu'un point d'arrêt est rencontré. Il est également impossible d'utiliser un simulateur en sources fermées, car les points d'arrêt doivent absolument être insérés dans le code source. La nécessité de modifier le code source implique également le travail d'un informaticien qui n'est pas nécessairement qualifié par rapport à l'application.

FALCON

Falcon [Gu et Eisenhauer, 1994], qui est lui aussi un des premiers environnements de pilotage de simulation, a été développé à Georgia Tech au début des années 1990, après VASE. Falcon est différent de VASE car il met l'emphase sur les applications de simulation parallèle distribuée sur plusieurs ordinateurs.

L'architecture de Falcon implique, tout comme VASE, une instrumentation du code source de l'application à piloter, ce qui est parfois loin d'être trivial. La nouveauté par rapport à VASE est l'ajout d'un module de monitoring qui envoie à la fois des informations au client de pilotage et au module d'analyse. Bien que Falcon réponde à bien des besoins primaires du pilotage de simulation, son utilisation n'est pas très intuitive car un expert en informatique doit se charger de l'instrumentation.

CUMULVS

CUMULVS [Geist *et al.*, 1996] a été développé au Oak Ridge National Laboratory au milieu des années 1990. Ce système se démarque des environnements décrits jusqu'ici par son architecture. En effet, les systèmes précédents étaient des logiciels outils (*toolkits*), alors que CUMULVS est un intergiciel (*middleware*), ou, autrement dit, un logiciel qui agit comme une passerelle de communication entre d'autres logiciels. CUMULVS permet aux programmeurs d'incorporer facilement des notions de pilotage de simulation dans des programmes existants. Sans aucune modification au logiciel de simulation original, il est possible d'effectuer un pilotage. Une condition doit toutefois être satisfaite : le logiciel de simulation doit utiliser PVM comme protocole de communication interprocessus.

Une fonctionnalité intéressante dans CUMULVS est la possibilité de sauvegarder

l'état de la simulation à un instant donné [Kohl et Papadopoulas, 1998]. Ceci permet à un utilisateur de revenir en arrière si le besoin se fait sentir et de recommencer la simulation à un point antérieur. Il est également possible d'avancer dans le temps, si, bien sûr, un point de sauvegarde avait été créé.

CSE

CSE (*Computational Steering Environment*) a été développé au milieu des années 1990 au centre de recherche en informatique d'Amsterdam [van Wijk et van Liere, 1994; van Liere *et al.*, 1996; Mulder, 1998]. Ses auteurs sont parmi les premiers à implanter un environnement intégré de visualisation et de pilotage de simulation. Leur architecture se démarque des autres par l'utilisation d'un gestionnaire de données qui communique avec des processus satellites. En effet, le gestionnaire de données crée, écrit, et lit des variables utilisées par tous les modules satellites qui sont, par exemple, le moteur de simulation, l'environnement de visualisation, et d'autres modules satellites. Donc, contrairement aux architectures précédentes, l'engin de simulation n'est plus le cœur de l'environnement logiciel, mais bien un client comme tous les autres. Cette approche est intéressante, car l'architecture n'est pas dépendante du simulateur. C'est au simulateur à s'adapter à CSE, et non l'inverse.

Afin de configurer l'environnement, des objets graphiques paramétrables (PGOs) sont utilisés. Ces objets possèdent des degrés de liberté et peuvent être combinés afin de représenter des objets plus complexes. Le scénario de simulation ainsi que tous les paramètres associés sont donc configurés à l'aide d'une interface graphique. Un point faible de cette approche est la quantité de travail nécessaire afin d'implémenter sous la forme de PGOs un scénario complexe existant dans un autre format. Par contre, cette approche permet la réutilisabilité des éléments programmés.

SCIRun

SCIRun a été développé à l'Université de l'Utah vers la fin des années 1990 [Parker *et al.*, 1997; Johnson *et al.*, 1999; Parker, 1999]. Il s'agit d'un environnement intégré de résolution de problèmes avec lequel il est possible de piloter des simulations et de visualiser les résultats simultanément. L'architecture est modulaire et très flexible. Le point fort de SCIRun est la possibilité de configurer le flot de données de manière visuelle, d'où le nom de programmation visuelle. En effet, une interface graphique permet de connecter des modules ensemble et de visualiser la manière dont les données seront

traitées. Elle permet également de programmer le flot de données de simulation.

SCIRun se rapproche d'un environnement de travail intégré qui permet de réaliser le processus scientifique en entier. Par contre, tout comme avec CSE, une configuration entièrement graphique pour des simulations complexes peut s'avérer fastidieux. C'est la raison pour laquelle un tel système devrait selon nous permettre l'utilisation de plusieurs niveaux de modules graphiques, des plus complexes aux plus simples.

CAVEStudy

CAVEStudy a été développé au début des années 2000 à l'Université de Vrije [Renambot *et al.*, 2000; Germans *et al.*, 2001]. Il s'agit d'un système intégré de pilotage de simulation et de visualisation dans un environnement immersif. Il n'est pas nécessaire de modifier le code source du simulateur afin de piloter une simulation, ce qui présente un avantage par rapport aux systèmes précédents. Toutefois, les applications sont limitées aux simulateurs qui fournissent des résultats intermédiaires et le pilotage n'est en fait qu'un suivi de la simulation via ses résultats intermédiaires.

Au niveau de l'architecture, l'usager communique avec la simulation par le biais d'un proxy et d'un serveur de simulation. Il est possible de piloter en tout temps la représentation visuelle des éléments de la simulation. La force de CAVEStudy est l'utilisation de métaphores d'interaction courantes dans les environnements immersifs afin d'interagir avec la simulation. Un choix judicieux de ces métaphores peut résulter en une immersion plus importante des scientifiques et des utilisateurs d'environnements de pilotage de simulations.

gViz

gViz a été développé au début des années 2000 dans le cadre du projet britannique e-Science, qui vise à donner aux scientifiques des outils informatiques pour le pilotage de simulations et la visualisation dans des environnements de type grille de calcul (*computational grid*) [Wood *et al.*, 2003; Brodlie *et al.*, 2004]. Les environnements de type grille de calcul sont caractérisés par une grande distribution physique des nœuds de calcul et une exécution parallèle de la simulation.

gViz permet d'interfacer des logiciels de visualisation comme IRIS Explorer ou VTK à une simulation qui s'exécute sur une grille de calcul. Ses auteurs ont développé le

langage skML afin de standardiser la représentation du flot de données provenant de la simulation et les interactions avec l'utilisateur. gViz est donc un intergiciel semblable à CUMULVS, à l'exception qu'il emploie des outils plus récents et que le langage de communication entre la simulation et la visualisation est formel.

Reality Grid

Reality Grid a été développé par un groupe britannique au début des années 2000 et est encore en développement [Brooke *et al.*, 2003; Pickles *et al.*, 2004b; Kalawsky *et al.*, 2005]. Tout comme gViz, Reality Grid se veut un environnement de pilotage de simulation destiné aux calculs scientifiques exécutés sur une grille de calcul. L'architecture de Reality Grid est basée sur une librairie de pilotage, dont les appels au code sont insérés dans l'application visée. Cette dernière communique ensuite avec des clients de pilotage qui envoient des commandes par le biais d'une interface, c'est-à-dire en passant par la librairie de pilotage. Cette dernière transporte également les données afin de les diriger vers un client de visualisation.

Reality Grid présente beaucoup de potentiel seulement pour des applications dont les sources sont ouvertes en raison du code d'instrumentation qui doit être inséré. L'implantation semble intéressante et les applications très diversifiées. Comme pour quelques systèmes similaires, il est possible de sauvegarder une simulation à l'aide de points de contrôle, et de revenir en arrière au besoin. Il n'y a toutefois aucune sauvegarde automatique de l'historique de simulation.

EPSN

EPSN (Environnement pour le Pilotage de Simulations Numériques) a été développé à l'INRIA au début des années 2000 [Esnard *et al.*, 2004; Esnard, 2005]. Il s'agit d'un environnement de pilotage de simulation qui se concentre sur la simulation parallèle couplée à la visualisation parallèle. En effet, la simulation est exécutée de manière distribuée, ce qui complique la gestion des données du pilotage. Le couplage avec un système de visualisation distribuée comme Parallel-VTK complique le tout davantage.

EPSN repose sur une modélisation rigoureuse de l'exécution de la simulation, la modélisation hiérarchique en tâches (MHT). De cette manière, le client de pilotage de simulation sait à tout moment où il lui sera possible d'insérer des commandes et où il pourra obtenir des données pour la visualisation. La MHT est également utilisée

pour l'instrumentation du code source de la simulation car elle définit les fonctions nécessitant une insertion, de même que leur emplacement dans la simulation. La communication entre les différentes entités, c'est-à-dire la transmission des données ainsi que les commandes de pilotage, s'effectue à l'aide de CORBA.

Un point faible d'EPSN est le manque d'intégration de ses différents composants. En effet, ses concepteurs n'ont pas mis l'emphase sur l'intégration d'un client de visualisation et de pilotage. Les auteurs ont d'ailleurs mentionné quelques limitations liées à cet aspect. Pour pallier à cette faiblesse, ils prévoient implanter un module qui stocke l'historique de la simulation afin de pouvoir la répéter *a posteriori*.

2.3.3 Fonctionnalités des systèmes de simulation interactive existants

La revue des systèmes existants de CS montre que chacun possède plusieurs fonctionnalités propres aux systèmes de simulation interactive : interface avec le simulateur, intégration du système de visualisation, interface usager de pilotage et maintien d'un historique de pilotage. Ces fonctionnalités sont tout aussi pertinentes pour les systèmes de VIS et devront par conséquent faire partie du système développé dans le cadre de cette thèse.

La plupart des systèmes actuels nécessitent une *instrumentation du code* de simulation afin d'insérer, aux endroits propices, des appels de fonctions permettant soit d'introduire des nouvelles données correspondant aux changements de paramètres, soit d'en retirer des informations pertinentes pour l'utilisateur. Cette approche est limitée en termes de flexibilité étant donné la nécessité d'ajouter du code d'instrumentation à chaque fois qu'un changement survient dans le modèle de simulation. D'autres systèmes proposent d'*intercepter les messages* passés entre les différents modules de simulation afin de modifier des paramètres sur le champ. Cette approche est donc limitée aux systèmes distribués qui ne seront pas considérés pour les présents travaux. Cette thèse favorise plutôt une approche flexible en intégrant l'instrumentation du code avec une autre fonctionnalité qui est celle de la sauvegarde de la simulation à un temps donné.

L'*intégration des systèmes de pilotage de simulation avec la visualisation* est réussie dans la plupart des systèmes revus. Certains sont toutefois fortement couplés avec un système de visualisation spécifique. Étant donné que les systèmes de CS sont pour la plupart orientés vers des applications scientifiques, il est incontestable que le format des données ainsi que les fonctionnalités de visualisation offertes doivent être parti-

culièrement pointus, donc difficiles à généraliser. Dans la présente thèse, l'intégration du simulateur avec un système de visualisation se fera de manière transparente et générique. Le système de pilotage ne sera donc pas fortement couplé à un logiciel de visualisation en particulier, ni à un logiciel de simulation. Ainsi, le format des données qui circuleront dans le pipeline de données se devra d'être le plus générique et flexible possible.

Les *interfaces usager des systèmes de pilotage de simulation* revus sont pour la plupart des interfaces classiques spécifiques à une application. Des logiciels comme SCIRun et CSE permettent toutefois de configurer entièrement le pipeline de données associé aux modèles sous la forme de composantes. À nouveau par souci de flexibilité, cette thèse devra considérer une approche générique pour le développement de l'interface de pilotage de simulation.

Finalement, peu de systèmes offrent la possibilité de *maintenir un historique de pilotage des simulations.* Certains systèmes comme Reality Grid, GRASPARC et CUMULVS offrent la possibilité de sauvegarder la simulation en des instants donnés. Par contre, seulement GRASPARC propose une représentation de ces points de sauvegarde sous la forme d'un arbre d'historique de simulation. Cette approche est intéressante parce que l'arbre d'historique permet à l'utilisateur de voir en un coup d'œil par quel chemin il est passé pour en arriver à un résultat particulier. Cette thèse propose donc d'étendre cette approche de visualisation du processus de pilotage de simulation en rendant l'arbre d'historique de simulation plus interactif et en y ajoutant des fonctionnalités de visualisation.

2.4 Vers un espace de travail intégré

Tel que mentionné dans la section précédente, la technique de la simulation interactive a été intégrée dans plusieurs logiciels de recherche et dans des logiciels commerciaux. Or, la simulation interactive fait appel à plusieurs notions techniques qui rendent difficile son intégration dans un logiciel existant. En effet, il est essentiel d'assurer que la simulation se déroule comme elle se doit, c'est-à-dire en exécutant les modèles correctement et dans un temps raisonnable de manière à permettre l'interaction avec l'utilisateur [Kalawsky *et al.*, 2005]. De plus, l'utilisateur doit observer des données correctes et bien présentées. Bref, l'espace de travail de l'utilisateur est dit *intégré*, ce qui signifie que plusieurs outils sont à sa disposition afin qu'il puisse résoudre un problème particulier ou tenter de comprendre le système sous étude, tout dépendant du but visé. Dans le cadre de cette thèse, le but des utilisateurs visés est d'améliorer leur compréhension des

systèmes complexes sous étude.

L'intégration de plusieurs outils logiciels dans un même espace de travail nécessite une méthodologie de conception ainsi que l'utilisation de formats de données permettant aux divers modules d'être interopérables [Harrison *et al.*, 2005]. Tandis que les travaux de Harrison *et al.* (2005) s'orientent vers la réutilisabilité des modèles de simulation, les présents travaux portent sur l'intégration de différents outils menant à un cadre d'application permettant à l'utilisateur d'exécuter des simulations de manière interactive.

2.4.1 Environnements de résolution de problèmes

Le principe d'environnement de résolution de problèmes (*Problem Solving Environments*, PSE) existe depuis le début des années 1960. Cependant, peu de PSEs ont été conçus avant les années 1990 en raison de la faible puissance de calcul disponible à l'époque [Houstis et Rice, 2000]. Un *environnement de résolution de problèmes* est un système computationnel qui fournit aux utilisateurs un ensemble complet et adéquat d'outils de haut niveau afin de résoudre des problèmes dans un domaine spécifique [Rice et Boisvert, 1996]. Parmi les outils proposés, on retrouve typiquement, en plus des outils de simulation interactive, des systèmes de visualisation scientifique avancés, des outils « intelligents » dont le but est d'aider l'utilisateur dans sa recherche de solutions, des modules d'optimisation, des modules fournissant des recommandations à l'utilisateur quant aux opportunités d'exploration et des systèmes experts [Rice et Boisvert, 1996]. Les environnements de résolution de problèmes existants sont très nombreux. GRASPARC [Brodlie *et al.*, 1993], SCIRun [Parker, 1999], Matlab [The Mathworks, 2009], Cactus Code [Allen *et al.*, 2000] et Proteus [Cannataro *et al.*, 2004] en sont quelques exemples. Ils répondent pour la plupart à la définition de base d'un PSE [Rice et Boisvert, 1996], soit

PSE = interface usager + librairies + base de connaissances + intégration

Il n'en demeure pas moins que les environnements de résolution de problèmes sont souvent spécifiques à un domaine d'application. Les outils visant à aider l'utilisateur sont difficilement généralisables. De plus, peu d'efforts ont été mis sur le développement d'interfaces graphiques conviviales et utiles aux utilisateurs. En ne considérant que les outils de simulation interactive, cette thèse vise précisément à résoudre ces problèmes, c'est-à-dire de construire un cadre conceptuel général de simulation interactive et de développer une interface graphique de simulation interactive exploitant ce cadre conceptuel et ce, à l'aide une implantation la plus générique possible.

2.4.2 Autres utilisations de la simulation interactive

En plus d'être intégrés dans des environnements de résolution de problèmes, des modules de simulation interactive sont utilisés dans des *systèmes d'aide à la décision basés sur les modèles* (*Model-Driven Decision Support Systems*, DSS) [Power et Sharda, 2007]. Ces systèmes viennent plutôt en support aux décideurs dans leur prise de décision afin de tester des hypothèses (analyse de type « What if ? ») et de visualiser les résultats simulés immédiatement. Power *et al.* (2007) font une excellente revue des systèmes existants en plus d'énumérer des questions de recherche qui seront déterminantes pour les années à venir, parmi lesquelles se trouve l'*efficacité de la simulation interactive dans les DSS*. Les travaux menés par Hurrion sur la VIS proposaient déjà dans les années 1980 l'utilisation de la simulation interactive comme une solution pour aider les décideurs à mieux analyser des problèmes dits « semi-structurés » pour lesquels il n'y a pas d'objectif clair, mais plutôt un large spectre de possibilités [Bell et O'Keefe, 1994]. Des expérimentations impliquant des sujets humains ont également été réalisées, prouvant que l'utilisation de la VIS dans un système d'aide à la décision pouvait améliorer la qualité des décisions [Chau et Bell, 1995]. Cependant, tout comme la majorité des PSEs, les DSSs sont très spécifiques au contexte d'application. Par exemple, on retrouve dans la littérature des applications de gestion de l'irrigation pour les agriculteurs [Le Bars et Le Grusse, 2008].

Pour sa part, l'*environnement d'études computationnelles* SimX vise à exploiter l'expérience de l'utilisateur afin qu'il guide l'exploration de l'espace des paramètres d'un problème complexe donné [Yau *et al.*, 2006]. L'approche de Yau *et al.* permet donc à l'utilisateur de piloter une expérience par ordinateur à l'aide de contrôles de haut niveau au lieu de considérer chacune des simulations individuellement. Cette technique d'interaction demeure de la simulation interactive dans la mesure où la tâche d'exploration de l'espace des paramètres est exécutée de manière automatique, en laissant la possibilité à l'utilisateur d'intervenir afin de guider l'exploration.

Chapitre 3

Multichronia – principes de base

Ce chapitre vise à présenter Multichronia, le cadre conceptuel (*conceptual framework*) qui fait l'objet des travaux de cette thèse et dont l'architecture flexible et générique peut s'adapter à plusieurs types de problèmes/contextes qui requièrent des prises de décision de la part de l'utilisateur en cours de simulation. Multichronia vient des racines grecques « *multi* » pour « plusieurs » et « *chronos* » (« *chronia* ») pour « temps ». Autrement dit, il s'agit d'un cadre conceptuel conçu dans le but de permettre à un utilisateur d'accomplir simultanément et de manière interactive des opérations sur plusieurs flots de données résultant de l'exécution de simulations. Multichronia est donc une base solide fournissant des outils interactifs de simulation et de visualisation flexibles et génériques. La Figure 3.1 présente une vue globale du cadre conceptuel pour une application spécifique. On remarque la présence de l'utilisateur, au centre de la boucle, qui a la capacité d'interagir avec l'interface de contrôle des simulations (gauche de la figure), de même que la possibilité d'observer des données de simulation (droite de la figure). Contrairement aux méthodes formelles d'analyse de simulation qui s'exécutent de manière autonome, Multichronia vise à exploiter l'expertise et l'intuition de l'utilisateur afin qu'il puisse tester des hypothèses sur les modèles de simulation et ainsi obtenir de manière informelle une meilleure compréhension du problème sous étude, c'est-à-dire sans avoir recours à des techniques d'analyse statistique des résultats. Multichronia ne vise pas à remplacer les méthodes formelles d'analyse de simulation, mais plutôt à complémenter ces méthodes en exploitant plus efficacement les capacités de l'utilisateur.

Le *cadre d'application* (*software framework*), un logiciel comprenant des modules génériques devant être spécialisés pour une application spécifique, a été préféré pour la mise en œuvre de Multichronia à d'autres approches de design logiciel comme les boîtes à outils (*toolbox*) parce que le système envisagé est centré sur l'utilisateur plutôt que

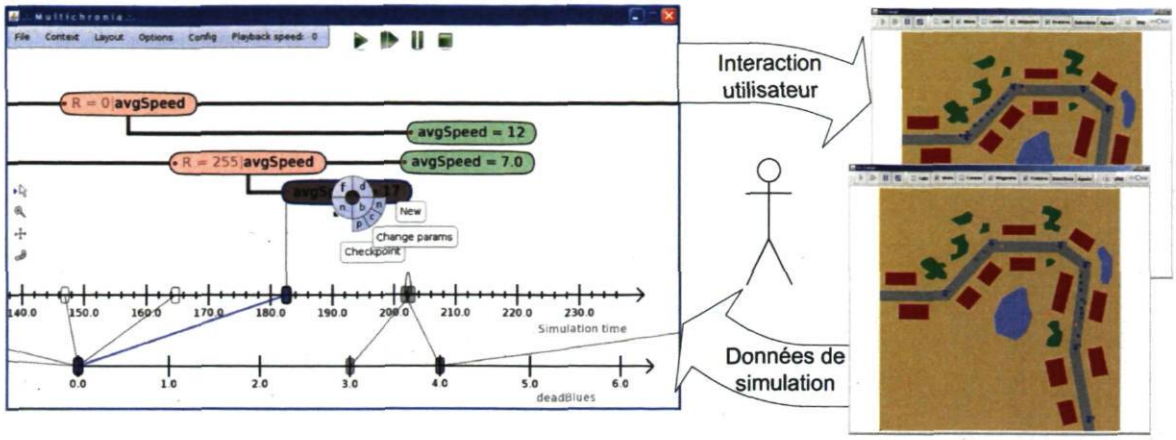


FIGURE 3.1 – Exemple typique du cadre d'application Multichronia. L'utilisateur interagit avec la simulation (gauche) et observe les données associées (droite)

sur le modèle de simulation [Riehle, 2000]. En effet, des applications telles que Reality Grid offrent des fonctionnalités semblables de contrôle interactif de simulation, mais considèrent le modèle comme étant l'élément critique du système [Brooke *et al.*, 2003]. Dans le contexte de cette thèse, les modèles de simulation sont considérés comme étant disponibles à l'utilisateur et ce dernier est l'élément central du système. Son but est de comprendre les modèles de simulation à l'aide des outils interactifs proposés par l'environnement. Ainsi, les applications construites à partir de cadres d'applications offriront à leurs utilisateurs un environnement de travail adéquat et bien structuré qui comprendra des fonctionnalités leur permettant d'exploiter efficacement leur expertise et leur intuition dans la compréhension des modèles de simulation. Un cadre d'application implique toutefois qu'une spécialisation soit dérivée pour chacune des applications développées. Ce chapitre décrit les différents types d'interactions possibles dans Multichronia ainsi que les aspects généraux liés au développement du cadre conceptuel.

3.1 Cadre conceptuel Multichronia

Le cadre conceptuel Multichronia comprend plusieurs éléments montrés à la Figure 3.2. On retrouve principalement deux types d'éléments : les unités de transformation du pipeline de données (rectangles) et les points d'interaction de l'utilisateur avec Multichronia (rectangles arrondis). Ce cadre conceptuel est inspiré à la fois des travaux de Kalawsky *et al.* (2005) et de Wright (2004) qui ont présenté des systèmes complets de pilotage de simulations dans lesquels la visualisation joue un rôle capital. Dans cette thèse, le point central d'intérêt est l'interaction directe de l'utilisateur avec les simu-

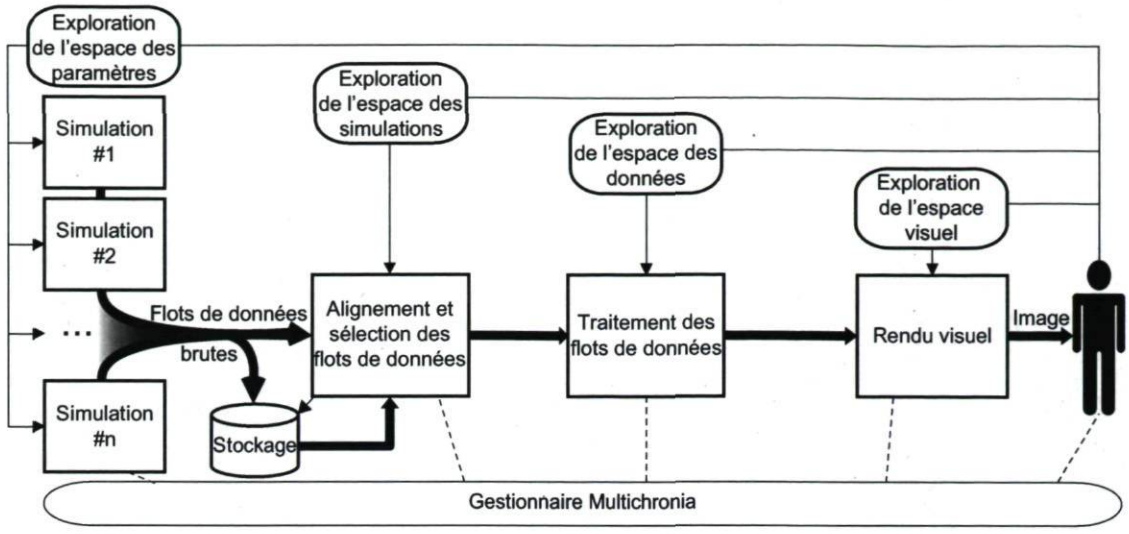


FIGURE 3.2 – Cadre conceptuel Multichronia montrant les quatre boucles d'interaction et les unités de transformation formant le pipeline de données

lations, sans pour autant négliger les aspects reliés au cheminement des données du module de simulation au module de visualisation. La justification de placer l'emphase sur l'interaction de l'utilisateur et des exécutions de simulation réside dans le fait que peu de travaux ont été réalisés en ce sens et se prête bien à des contributions originales. Les méthodes formelles évitent la présence d'un utilisateur dans la boucle de simulation [Kleijnen *et al.*, 2005], tandis que les méthodes informelles actuelles se concentrent typiquement sur les changements apportés sur une simulation à la fois [Brodlie et Wood, 2007].

Les principes de base de la simulation interactive, fondamentaux dans le développement de Multichronia, sont bien connus (voir Figure 2.7) : l'utilisateur démarre une simulation en considérant un ensemble de paramètres initiaux, puis les données associées à cette exécution de simulation (traits gras) sont envoyées vers un module de visualisation. L'utilisateur perçoit les données de simulation et a la possibilité d'interagir avec le modèle de simulation pour tester des hypothèses dans le but d'améliorer sa compréhension du système. Ce processus interactif est également itératif, ce qui donne l'opportunité à l'utilisateur de revisiter cette boucle à maintes reprises.

L'une des particularités de Multichronia par rapport à la plupart des systèmes répertoriés dans la littérature est sa capacité de prendre en charge l'exécution et l'analyse des données de plusieurs simulations simultanément. Ainsi, le pipeline de données associé à Multichronia comprend un module supplémentaire à celui proposé par Kalawsky *et al.* (2005). Il s'agit de l'unité d'alignement et de sélection des flots de données de simulation.

De même, les fonctionnalités interactives comportent une boucle d'interaction supplémentaire, soit celle de l'exploration de l'espace des simulations. Les sections suivantes décrivent respectivement le pipeline de données de Multichronia et les fonctionnalités interactives associées aux quatre boucles d'interaction faisant intervenir l'intuition et l'expertise de l'utilisateur dans l'analyse d'un problème donné.

3.2 Pipeline de données de Multichronia

Le pipeline de données de Multichronia est composé de quatre unités de transformation. Les données sont générées par des instances de simulation et parviennent jusqu'à l'utilisateur après avoir été sélectionnées, alignées, traitées, filtrées, puis rendues sous forme d'images animées. Les paragraphes suivants décrivent en détails les fonctions respectives de chaque unité de transformation ainsi que les différents formats de données qui voyagent d'un bout à l'autre du pipeline.

3.2.1 Simulation

L'unité de transformation *simulation* est à l'origine du pipeline de données. Conceptuellement, il s'agit d'une source de données pouvant être paramétrée par un utilisateur. Une simulation est concrètement associée à l'exécution de modèles par un simulateur qui consomme des ressources matérielles de l'ordinateur sur lequel l'exécution se déroule. Les paramètres relatifs à une unité de transformation *simulation* sont les paramètres des modèles ainsi que les paramètres d'exécution de la simulation (e.g. pas de temps, temps de simulation maximal). La sortie d'une unité de transformation *simulation* est un flot de données brutes de simulation, c'est-à-dire que ces données sont dans un format propre au simulateur. Afin d'être propagés vers le reste du pipeline de données, les flots de données brutes sont convertis dans un format compatible avec les autres unités de transformation. Le Chapitre 5 traite entièrement de ce problème qui a été solutionné à l'aide d'une méthode entièrement générique basée sur l'ingénierie des modèles.

Le cadre conceptuel Multichronia fournit à ses utilisateurs une base solide visant à interfacer au pipeline de données un grand nombre de simulateurs répondant à un nombre minimal de critères. Le premier de ces critères, associé à l'unité de transformation *simulation*, est que le simulateur doit permettre la récupération d'un flot de données correspondant aux données de la simulation. Il est également important de noter que, comme le montre la Figure 3.2, plusieurs unités de transformation *simulation*

peuvent faire partie du même pipeline de données. Des mécanismes sont mis en place afin d'assurer un partage équitable des ressources entre les instances de simulateurs et afin d'éviter le plus possible la création d'un goulot d'étranglement dans le pipeline de données. De plus, un *module de stockage* des flots de données assure la persistance des données générées par les unités de transformation *simulation* en archivant les données générées par toutes les simulations en temps réel.

3.2.2 Alignement et sélection des flots de données

L'unité de transformation *alignement et sélection des flots de données* (AS) est l'élément original des travaux courants par rapport à ceux que l'on retrouve dans la littérature. En effet, compte tenu que Multichronia offre la possibilité d'exécuter plusieurs simulations simultanément, il est essentiel d'inclure dans le pipeline de données une unité de transformation servant à aligner et à sélectionner les flots de données se propageant dans le reste du pipeline. L'*alignement* consiste à synchroniser deux (ou plusieurs) flots de données sur une condition définie par l'utilisateur ou par un algorithme approprié. La *sélection* consiste à choisir un (ou plusieurs) flot de données parmi ceux qui sont disponibles afin de le propager plus loin dans le pipeline de données. Un flot de données sélectionné peut donc être traité par les unités de transformation subséquentes.

L'unité de transformation AS accepte en entrée des flots de données provenant des unités de transformation *simulation* ou du module de stockage, puis rend disponibles en sortie des flots de données sélectionnés et alignés selon la volonté de l'utilisateur. Il va sans dire que l'interface usager correspondant à l'unité de transformation AS est d'une importance capitale dans l'implantation du cadre conceptuel Multichronia. C'est pourquoi le Chapitre 4 lui est entièrement dédié.

3.2.3 Traitement des flots de données

L'unité de transformation *traitement des flots de données* est essentielle afin d'exécuter des calculs/traitements sur les données provenant de la simulation (e.g. moyenne, fenêtre de données, transformée de Fourier). Cette unité de transformation accepte en entrée les flots de données alignés et sélectionnés par l'utilisateur et offre en sortie des flots de données modifiés et/ou enrichis de nouvelles données structurées, adaptées pour la visualisation. Les traitements sur les données sont des plus génériques et sont implantés en respectant le format de données standard circulant à travers le pipeline de données.

3.2.4 Rendu visuel

L'unité de transformation *rendu visuel* accepte en entrée les flots de données structurées générés par l'unité de transformation précédente à partir desquelles une séquence d'images est créée. Le rendu visuel des données les rend disponibles à l'utilisateur. Plusieurs options sont configurables, dont l'aspect visuel des données et la possibilité de rendre la totalité des données ou seulement un sous-ensemble.

3.2.5 Utilisateur

L'utilisateur, bien qu'il soit situé au centre de la boucle de simulation interactive, se trouve au bout du pipeline de données dont l'origine est l'unité de transformation *simulation*. Le rôle de l'utilisateur est d'observer les images produites par le module de rendu visuel, de formuler des hypothèses et de les tester à l'aide des outils mis à sa disposition. L'utilisateur est également responsable de la configuration des unités de transformation formant le pipeline de données. La section suivante traite en détails des fonctionnalités associées à chacune des boucles d'interaction faisant partie du cadre conceptuel Multichronia.

3.3 Boucles d'interaction de Multichronia

Le cadre conceptuel Multichronia permet à un utilisateur d'explorer quatre espaces conceptuels : l'espace des paramètres de simulation, l'espace des simulations, l'espace des données de simulation et l'espace visuel. Une boucle d'interaction est associée à chacun de ces espaces (rectangles arrondis de la Figure 3.2), ce qui permet à un utilisateur d'exécuter des actions ayant un effet sur les unités de transformation correspondantes du pipeline de données. Les flots de données y circulant sont par conséquent modifiés selon la volonté de l'utilisateur. Seulement trois de ces boucles interactives d'exploration se retrouvent dans la littérature [Kalawsky *et al.*, 2005]. L'exploration interactive de l'espace des simulations est une contribution de cette thèse. Les paragraphes suivants décrivent brièvement les actions pouvant être entreprises par un utilisateur travaillant avec une application basée sur le cadre conceptuel Multichronia. Il est important de noter que, conceptuellement, ces boucles d'interaction sont disjointes. Par contre, tout dépendant de la mise en œuvre, elles seront potentiellement intégrées dans une ou plusieurs interfaces usager distinctes. La transition entre les boucles d'exploration interactive sera également transparente pour l'utilisateur.

3.3.1 Exploration de l'espace des paramètres de simulation

Un utilisateur peut *tester des hypothèses* sur un modèle de simulation par le biais de la boucle d'exploration de l'espace des paramètres de simulation. Un *paramètre* est défini comme « une variable d'entrée d'une simulation pouvant être modifiée dans d'autres instances de simulation ». Un paramètre n'est pas limité à une valeur numérique, mais il peut avoir un sens conceptuel comme la présence ou l'absence d'une caractéristique. Avant de démarrer l'exécution d'une simulation, l'utilisateur doit créer un scénario dont le but est de spécifier l'ensemble initial des paramètres. Ensuite, la boucle d'exploration de l'espace des paramètres de simulation permet à un utilisateur de changer la valeur de ceux-ci en cours d'exécution. Un *changement de paramètre* correspond donc à l'opération de modifier une valeur numérique ou conceptuelle ayant un effet marqué ou pas d'effet du tout sur la réponse de la simulation.

Dans l'implantation de base du cadre conceptuel Multichronia, l'utilisateur est responsable de la pertinence du changement d'un paramètre. Un tel changement peut avoir un sens physique ou logique (e.g. varier la météo pendant une opération militaire, prendre la décision stratégique d'investir une certaine somme d'argent à un moment donné) ou ne pas avoir de sens réel (e.g. varier le blindage d'un véhicule en cours de route, téléporter un agent d'une position initiale vers une nouvelle position). Même si les variations de paramètres peuvent ne pas avoir de sens, il peut être intéressant pour un utilisateur de tout simplement « jouer » avec le modèle et d'observer les effets des changements de paramètres impossibles à accomplir dans la réalité, simplement dans le but d'obtenir une compréhension intuitive sur la réponse du modèle à ces modifications. En exécutant de tels tests d'hypothèse, l'utilisateur doit rester vigilant face aux conclusions à tirer du modèle [Kelton et Barton, 2003]. En effet, des conclusions hâtives de cause à effet d'un changement de paramètre sur une réponse sont très faciles à formuler. L'illusion du contrôle confond parfois un utilisateur sur sa capacité réelle à influencer la réponse de la simulation par rapport à l'effet des facteurs de bruit (*noise factors*) [Kottemann *et al.*, 1994].

Le cadre conceptuel Multichronia offre à un utilisateur la possibilité de générer simultanément autant de simulations qu'il le veut. Ainsi, il lui est possible de tester plusieurs hypothèses concurremment en poursuivant l'exécution des simulations originales. La justification de cette fonctionnalité vient du fait que lorsqu'il formule une hypothèse dans le but de comprendre le modèle, un utilisateur peut demeurer intéressé par ce qui se déroule dans la simulation originale. Dans les systèmes de simulation interactive répertoriés dans la littérature, il est impossible pour un utilisateur de revenir facilement en arrière puis de retourner en avant sans se soucier de sauvegarder l'état courant d'une simulation (*checkpoint*) [Pickles *et al.*, 2004b]. L'exemple qui suit illustre

encore mieux l'avantage qu'offre Multichronia par rapport aux systèmes actuels. Dans un contexte de jeu vidéo de stratégie, il arrive que le joueur désire employer une nouvelle stratégie à un moment dans la partie dans le but de vaincre l'ennemi. Étant donné que la stratégie est nouvelle et que le joueur n'en connaît pas *a priori* le résultat, un comportement bien normal est de sauvegarder l'état actuel de la partie afin de pouvoir éventuellement la recharger si les choses tournent mal. Le joueur essaiera une nouvelle stratégie puis rechargera la version sauvegardée au besoin jusqu'à ce que son but soit atteint. L'objectif de la boucle d'interaction avec l'espace des paramètres est précisément de permettre à l'utilisateur d'essayer de nouvelles combinaisons de paramètres et de revenir dans le passé au besoin. Dans Multichronia, l'utilisateur n'a pas à gérer les détails tels que la sauvegarde et la gestion des exécutions de simulations puisque de telles opérations sont prises en charge par le cadre d'application.

3.3.2 Exploration de l'espace des simulations

Afin de gérer la grande quantité de données de simulation disponibles pour l'analyse, un utilisateur du cadre conceptuel Multichronia dispose de la boucle d'exploration de l'espace des simulations. Cette boucle d'interaction est associée à l'unité de transformation *alignement et sélection des flots de données*. Sa présence est justifiée par le fait que, contrairement aux systèmes répertoriés dans la littérature, Multichronia permet à ses utilisateurs d'explorer plusieurs simulations à la fois. Ainsi, la boucle d'exploration de l'espace des simulations fournit des fonctionnalités permettant à l'utilisateur de naviguer facilement dans les multiples flots de données de simulation disponibles. Plus précisément, cette boucle d'exploration propose à l'utilisateur un *arbre multichronique* qui est une représentation visuelle de toutes les simulations en cours et qui lui offre la possibilité de les manipuler. Les deux principales fonctionnalités offertes par cette interface sont l'alignement des simulations (i.e. possibilité de retourner en arrière et d'avancer) ainsi que leur sélection. L'arbre multichronique affiche également des informations de haut niveau permettant ainsi à un utilisateur d'obtenir de l'information sur les simulations en cours d'exécution. Plus de détails sont disponibles au Chapitre 4 qui est spécifiquement consacré à la description des fonctionnalités offertes par l'arbre multichronique et à l'élaboration d'une terminologie associée.

3.3.3 Exploration de l'espace des données de simulation

La boucle d'exploration de l'espace des données de simulation permet à un utilisateur de configurer l'unité de transformation de traitement des données. En effet, c'est

en interagissant avec cette boucle d'exploration que l'utilisateur décide de la nature des traitements effectués sur les flots de données de même que de ceux qui seront influencés par ces traitements. Dans les travaux de Kalawsky *et al.* (2005), cette boucle d'exploration correspond à la préparation des données pour la visualisation. À l'aide de cette boucle d'interaction, l'utilisateur choisit également le sous-ensemble de données qui est envoyé au module de rendu visuel. Le cadre conceptuel Multichronia n'impose aucune contrainte sur les traitements pouvant être effectués lors de l'exploration des données ni sur les technologies pouvant être utilisées. Le chapitre traitant de la mise en œuvre de Multichronia décrira plus en détails les mécanismes spécifiques qui ont été mis en place pour assurer ces fonctionnalités. Toutefois, peu d'emphase a été portée sur cette boucle d'interaction étant donné la grande disponibilité de systèmes semblables dans la littérature (e.g. Aurora [Abadi *et al.*, 2003] et Borealis [Abadi *et al.*, 2005]).

3.3.4 Exploration de l'espace visuel

La boucle d'exploration de l'espace visuel implique une interface qui présente à l'usager les données provenant des simulations et qui lui offre la possibilité de configurer le format du rendu. Une fonctionnalité importante de cette boucle d'interaction est la possibilité de comparer une ou plusieurs simulations à l'aide d'une métaphore visuelle appropriée. De plus, la vue associée aux données peut être configurée de manière à changer l'aspect visuel de leur représentation sans en affecter le contenu (e.g. translation, rotation, changement d'échelle, changement de couleur). Les travaux relatifs à l'exploration de l'espace visuel ne sont pas partie intégrante de cette thèse. C'est pourquoi peu de détails sont disponibles sur cet aspect de recherche dans les chapitres suivants. Par contre, des travaux intéressants et pertinents pour la présente thèse ont récemment été réalisés par divers groupes [Jankun-Kelly *et al.*, 2007; Aigner *et al.*, 2008].

3.4 Gestionnaire Multichronia

Le *gestionnaire Multichronia* assure la cohérence entre tous les modules du cadre conceptuel. Ce module transmet des métadonnées entre les unités de traitement afin que les informations concernant les flots de données se propagent d'un bout à l'autre du pipeline de données. De plus, le gestionnaire assure que les actions de l'utilisateur seront communiquées à tous les modules concernés. Ces actions sont également archivées afin de rendre possible une analyse *a posteriori* des démarches de l'utilisateur. L'archivage des actions se déroule également à plusieurs niveaux selon les besoins du concepteur

de l'expérimentation, soit en conservant les actions de très bas niveau comme le déplacement et les clics de souris ou les actions de haut niveau comme le changement d'un paramètre à un instant donné.

Le gestionnaire Multichronia assure également la cohérence entre les différents contextes de simulation présents dans Multichronia. En effet, le cadre logiciel permet à plusieurs niveaux de simulations de communiquer entre eux. Par exemple, un utilisateur peut exécuter une simulation qui se déroule à un niveau stratégique (beaucoup de planification, peu de tâches opérationnelles), mais il peut également s'intéresser à ce qui se passe au niveau tactique (peu de planification, beaucoup de tâches opérationnelles). Dans ce cas, le gestionnaire Multichronia permettrait d'assurer la cohérence entre ces différents contextes en modifiant les opérations disponibles et en affichant correctement les interfaces graphiques associées.

Ce chapitre a décrit brièvement les modules qui font partie du cadre conceptuel Multichronia. Le pipeline de données est formé de différentes unités de transformation de flots de données paramétrables par l'utilisateur qui, lui, interagit avec Multichronia via quatre boucles d'interaction visant à explorer quatre espaces conceptuels distincts. L'exploration de l'espace des paramètres de simulation est bien connue dans le domaine de la simulation interactive. Le chapitre suivant traite d'un concept original développé dans le cadre de cette thèse qui permet à l'utilisateur d'explorer l'espace des simulations. Les boucles interactives de l'exploration de l'espace des données et de l'espace visuel ne seront pas plus détaillées. Par contre, une solution originale en ce qui a trait à l'échange des données provenant des simulations jusqu'au module de rendu visuel sera décrite en détails dans le Chapitre 5. Finalement, la mise en œuvre d'une application de Multichronia sera présentée au Chapitre 7.

Chapitre 4

Arbre multichronique

Ce chapitre présente un concept original développé dans le cadre de cette thèse, l'arbre multichronique, qui est l'interface graphique avec laquelle l'utilisateur d'une application basée sur le cadre conceptuel Multichronia explore l'espace des paramètres et l'espace des simulations. Cette interface est donc capitale dans le succès d'un utilisateur à comprendre le système complexe qui l'intéresse. Dans la conception de l'arbre multichronique, les principes clés de l'élaboration des interfaces homme-machine (IHM) devront être suivis rigoureusement afin d'assurer qu'il soit utilisable et utile [Norman et Draper, 1986]. Une telle interface devrait donc être intuitive et naturelle, en plus d'exploiter la capacité perceptuelle de l'humain à reconnaître des patrons dans les représentations d'information graphique [Sips *et al.*, 2007].

La structure de ce chapitre est la suivante : en premier lieu, une revue des interfaces graphiques de simulation interactive est présentée, suivie de l'énumération des critères de design qui ont été retenus pour le développement de la nouvelle interface graphique. Ensuite, la représentation originale, un arbre multichronique, est décrite ainsi que tous les termes nécessaires à la compréhension des concepts rattachés. S'en suit une description de toutes les actions que peut poser l'utilisateur sur l'arbre multichronique, accompagnée d'exemples concrets venant justifier leur développement. Une discussion sur l'intégration de l'arbre multichronique dans le cadre conceptuel Multichronia, comprenant une énumération des contraintes imposées sur un simulateur et un survol de l'automatisation de certaines opérations, termine le chapitre.

4.1 Revue des interfaces de contrôle de simulation

Plusieurs logiciels de simulation interactive ne possèdent aucun moyen de visualiser le cheminement d'un utilisateur dans l'exploration des paramètres d'un modèle de simulation. En effet, les paramètres de simulation sont souvent modifiés par l'utilisateur grâce à des boîtes de dialogue ou des fenêtres de commande, ce qui limite la traçabilité des résultats. Une solution à ce problème consiste à présenter à l'utilisateur, dans un espace à plusieurs dimensions (trois dimensions spatiales + effets visuels), les points de l'espace des paramètres qu'il a visités pendant son exploration [Winfield, 1998], lui permettant ainsi d'observer les lacunes au niveau de la couverture de l'espace. Cette technique est toutefois limitée au nombre de paramètres qu'il est possible de positionner sur un graphique à N dimensions [Grinstein *et al.*, 2001].

Pour certains types de simulations, l'absence d'un historique des paramètres visités *pendant l'exécution de la simulation* ne cause pas de problèmes. Les simulations dont les états futurs ne dépendent pas des états passés, c'est-à-dire celles pour lesquelles il est toujours possible de retourner dans un état passé en ajustant l'ensemble des paramètres de simulation à une valeur déterminée, en sont un exemple. Par contre, les systèmes complexes considérés dans le cadre de cette thèse ne possèdent pas cette caractéristique. En effet, un système complexe est dépendant de son historique. En d'autres mots, un changement dans le passé affecte l'état futur d'un système complexe. Il est donc essentiel de tenir un historique des paramètres visités pendant l'exécution afin que l'utilisateur connaisse à tout moment l'état précis du système et ses antécédents. De plus, dans les systèmes étudiés, non seulement l'état final de la simulation est intéressant, mais également tout le *cours d'action* qui se déroule entre le début et la fin de la simulation. Il peut survenir des évènements particuliers à tout moment et, en conséquence, la compréhension du système en dépend.

D'après une recherche exhaustive de la littérature des environnements de simulation interactive, GRASPARC est le premier outil permettant à un utilisateur de représenter graphiquement les points de variation de paramètres à mesure que le temps de simulation avance [Brodlie *et al.*, 1993]. La Figure 4.1 montre une capture d'écran de cette interface appelée « arbre d'historique de simulation » avec laquelle l'utilisateur contrôle les changements de paramètres de simulation et les flots de données résultants. Les branches verticales de l'arbre sont des instances de simulation qui ont été considérées par l'utilisateur. Chaque cercle noir correspond à une sauvegarde instantanée de l'état de la simulation (*snapshot* ou *checkpoint*), de laquelle l'utilisateur peut changer des paramètres avant de repartir la simulation qui tiendra compte des changements effectués. L'arbre d'historique de simulation permet donc à l'utilisateur d'exécuter une

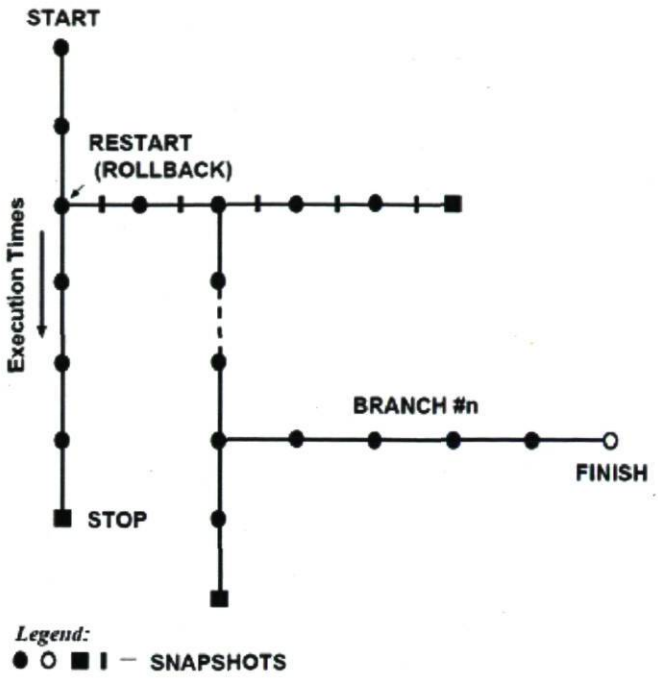


FIGURE 4.1 – Arbre d'historique de simulation de GRASPARC [Brodlie *et al.*, 1993]

simulation et de revenir dans le passé s'il observe une situation qui mérite une attention particulière, tout en modifiant des paramètres de simulation afin de générer des données supplémentaires pouvant mener à une meilleure compréhension du phénomène étudié.

S'inspirant des travaux sur GRASPARC de Brodlie *et al.* (1993), Wright *et al.* (1996) ont développé HyperScribe, un arbre d'historique de simulation intégré dans IRIS Explorer, un logiciel de visualisation modulaire [Wright et Walton, 1996]. Dans HyperScribe, l'arbre d'historique de simulation perd son statut d'interface principale de contrôle des paramètres de la simulation et des flots de données. Comme le montre la Figure 4.2, l'arbre d'historique de simulation devient plutôt une interface reliée à une base de données qui stocke les données provenant du module *simulation* et qui les rend disponibles pour être visualisées plus tard ou pour repartir une simulation en considérant un ensemble de paramètres différents. Les fonctionnalités disponibles dans HyperScribe sont donc identiques à celles offertes par GRASPARC, mais la manière d'interagir est fort différente. Plutôt que d'être l'interface utilisateur principale de l'environnement de résolution de problèmes, l'arbre d'historique de simulation est utilisé comme tout autre module dans IRIS Explorer. Le focus de HyperScribe est donc porté sur la visualisation de données plutôt que sur la simulation. Dans le cadre de cette thèse, l'aspect « simulation » est aussi important que l'aspect « visualisation ». C'est pourquoi les boucles d'interaction du cadre conceptuel Multichronia sont toutes situées au même niveau. L'arbre multichronique n'est donc pas relayé au second degré d'interaction,

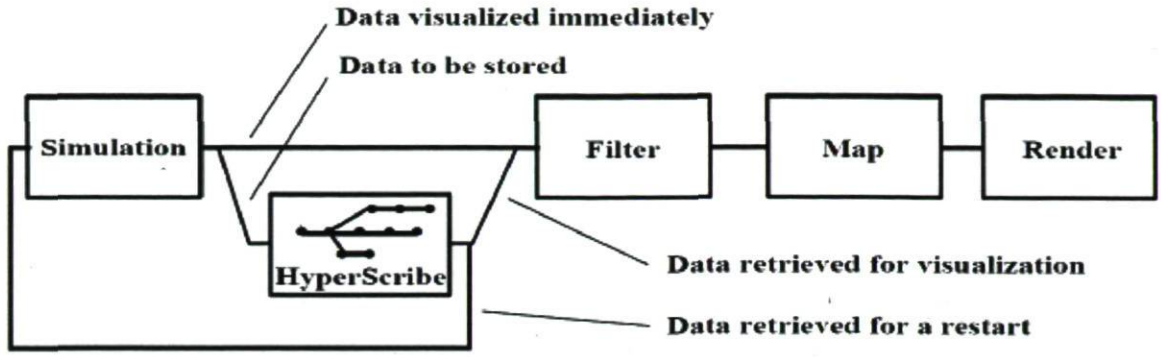


FIGURE 4.2 – Architecture de HyperScribe [Wright et Walton, 1996]

mais constitue plutôt un élément principal d'une application basée sur Multichronia. Par ailleurs, les fonctionnalités offertes par l'arbre d'historique de simulation sont très limitées. Étant donné que ces logiciels datent de plus de 10 ans, il est possible que la puissance de calcul disponible à l'époque ait nui au développement de fonctionnalités supplémentaires. L'arbre multichronique fournira donc des fonctionnalités beaucoup plus avancées en termes d'interaction et de rendu visuel.

Les travaux de Pickles *et al.* (2004) sur une interface de contrôle de simulation interactive, également inspirés par GRASPARC, s'orientent dans le sens proposé par cette thèse. Le système de pilotage de simulation TRICEPS offre une qualité de rendu et une présentation d'information pertinente de loin supérieures aux systèmes précédents. Comme le montre la Figure 4.3, les nœuds de l'arbre d'historique de simulation représentent dans une vue 3D l'état de la simulation à chaque instant où l'utilisateur a décidé de sauvegarder une simulation. Cependant, Pickles *et al.* (2004) ne détaillent pas davantage leur système, laissant croire que les fonctionnalités interactives sont limitées. Comme dans les systèmes précédents, il est néanmoins possible de sauvegarder l'état d'une simulation en un temps donné et de redémarrer une simulation à partir d'une sauvegarde. Pour sa part, l'arbre multichronique offre davantage de fonctionnalités interactives telle que la manipulation directe des nœuds de l'arbre, tout en permettant à un utilisateur de s'informer sur l'état de son cheminement à travers l'espace des paramètres par un simple coup d'œil.

De leur côté, Fischer *et al.* (2007) ont récemment présenté une interface qui s'apparente à celle de GRASPARC, sans pour autant la citer comme référence (voir Figure 4.4). Il s'agit encore d'un arbre montrant l'historique des simulations étudiées, toutefois baptisé « arbre d'expérience » [Fischer *et al.*, 2007]. Ce dernier est construit de manière similaire à GRASPARC, sauf qu'il ne semble pas possible de retourner dans le passé afin de démarrer une simulation à partir d'un point de sauvegarde. La création d'une

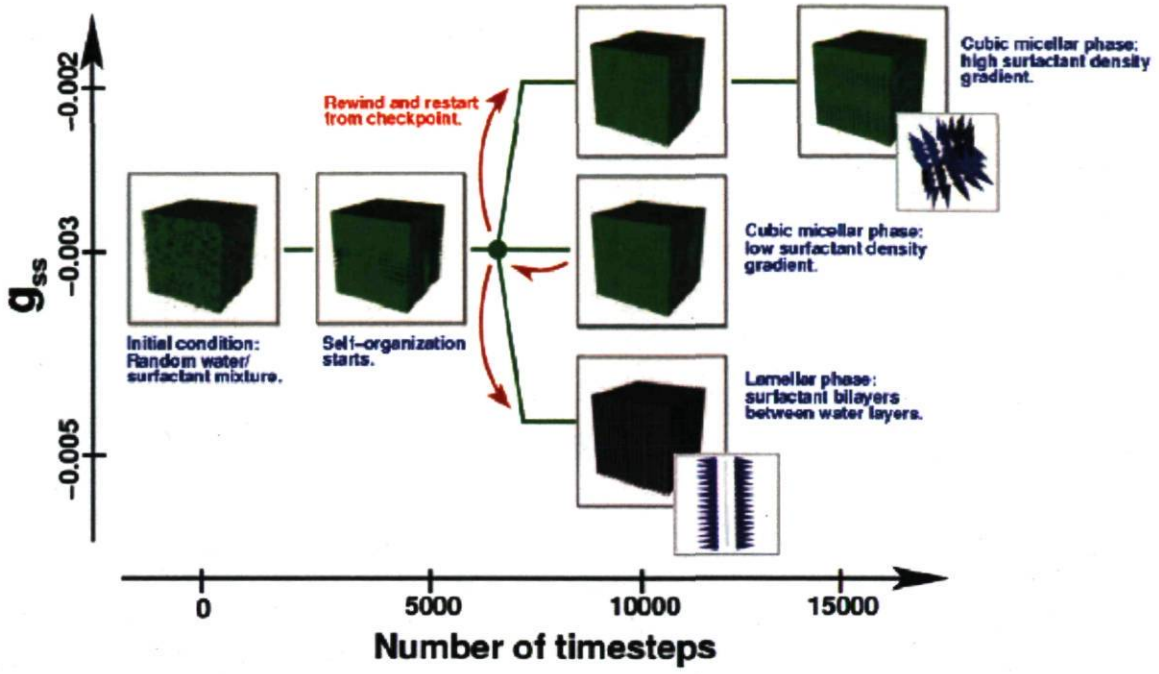


FIGURE 4.3 – Arbre d'historique de simulation dans TRICEPS [Pickles *et al.*, 2004a]

nouvelle branche de simulation se déroule comme suit : à un moment jugé opportun par l'utilisateur, toutes les simulations sont stoppées. Ce dernier peut ensuite cloner une ou plusieurs simulations selon ses besoins, modifier des paramètres et charger le contenu modifié dans une nouvelle instance de simulation. Le fait de cloner une simulation évite la ré-exécution des modèles à partir de zéro [Hybinette et Fujimoto, 2001]. Dans l'application considérée par Fischer *et al.* (2007), les simulations consomment beaucoup de temps de calcul, ce qui implique une allocation de ressources dans une ferme de calcul pour toute nouvelle instance de simulation. Lorsque l'allocation est complétée, les simulations sont redémarrées et, au besoin, des algorithmes d'équilibrage de calcul assurent que toutes les simulations s'exécutent également. L'utilisateur peut observer plusieurs résultats de simulation en parallèle. Il s'agit d'une caractéristique intéressante pour la comparaison des résultats de simulation après la modification de paramètres. Toutefois, la comparaison s'effectue en observant des résultats semblables dans plusieurs vues et non pas en observant les résultats intégrés dans une vue unique. Les auteurs ont toutefois identifié cette fonctionnalité comme faisant partie des travaux futurs.

En se basant sur les systèmes revus dans les paragraphes précédents, il semble qu'un arbre soit une représentation reconnue pour afficher l'état de l'exploration des paramètres d'un modèle de simulation et afin d'établir une correspondance entre les instances de simulation. Le Tableau 4.1 présente un résumé des fonctionnalités offertes par les systèmes existants en termes d'interaction avec l'espace des simulations. Il est

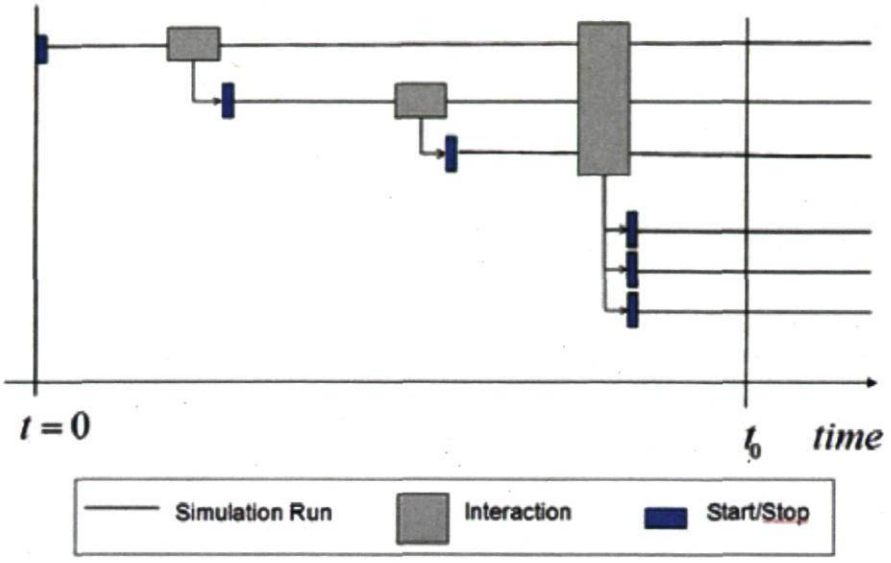


FIGURE 4.4 – Arbre d'expérience bâti en pausant, clonant, modifiant des paramètres, et redémarrant des simulations [Fischer *et al.*, 2007]

clair que chacun offre des fonctionnalités intéressantes sans pour autant offrir la totalité des fonctionnalités énumérées. Plusieurs autres fonctionnalités dont l'utilisateur bénéficierait sont également identifiables. L'arbre multichronique a donc été conçu en considérant l'ensemble des fonctionnalités existantes et originales.

4.2 Design de l'arbre multichronique

Dans un contexte où plusieurs simulations sont nécessaires afin d'explorer un problème complexe et dans lequel des décisions peuvent être prises à tout moment pendant l'exécution d'une simulation, il est facile d'identifier plusieurs limitations aux systèmes décrits dans la section précédente. Premièrement, tous les systèmes ne permettent que la sauvegarde ponctuelle de l'état de la simulation. Autrement dit, il n'est pas possible de *sauvegarder une simulation en continu.* Un utilisateur peut recharger les données correspondant à un point de sauvegarde et modifier les paramètres de simulation uniquement à cet endroit. La gestion des points de sauvegarde peut donc devenir une tâche difficile pour l'utilisateur lorsque l'arbre en contient un grand nombre. De plus, tous les systèmes sauf TRICEPS n'offrent aucun moyen de déterminer quel paramètre a été modifié et quelle est sa nouvelle valeur. L'approche adoptée dans TRICEPS ne permet

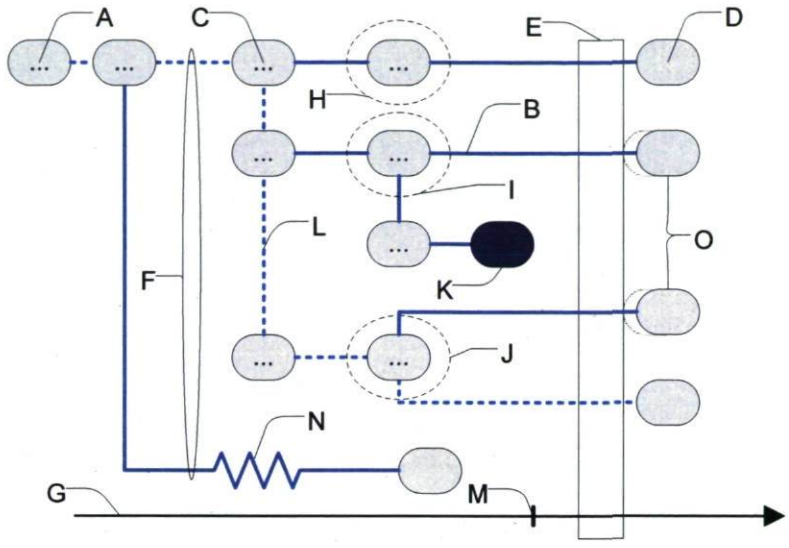
TABLEAU 4.1 – Fonctionnalités associées aux interfaces d'exploration de l'espace des paramètres et de l'espace des simulations existant dans la littérature

Fonctionnalité	**GRAS-PARC**	**Hyper-Scribe**	**TRICEPS**	**Arbre d'expérience**
Représentation de l'état courant de l'historique des simulations	X	X	X	X
Création d'un point de sauvegarde	X	X	X	X
Démarrage d'une simulation à partir d'un point de sauvegarde	X	X	X	X
Démarrage d'une simulation à partir d'un point de sauvegarde antérieur	X	X		
Comparaison des résultats de plusieurs simulations			X	X
Exécution de plusieurs simulations en parallèle				X
Information/visualisation pertinente dans les nœuds de l'arbre			X	
Sauvegarde de l'arbre pour utilisation future		X		

toutefois qu'à un seul paramètre d'être varié étant donné que sa valeur correspond à la position de la branche correspondante sur l'axe vertical de l'arbre (voir Figure 4.3). *Inclure davantage d'information dans l'arbre d'historique de simulation*, permettant à l'utilisateur de s'orienter plus facilement dans l'espace des paramètres étudiés, est une caractéristique essentielle dont l'arbre multichronique bénéficie. D'après la documentation disponible sur les systèmes actuels, il n'est pas facile de déterminer si l'*interface usager fournit de la rétroaction face aux actions posées par l'utilisateur*. Il s'agit d'un principe de base du design des IHM qui est pris en compte dans le développement de l'arbre multichronique. Prévenir et gérer les erreurs, être cohérent dans les termes et les représentations et minimiser l'effort cognitif de l'utilisateur sont d'autres critères de design importants. De plus, l'interface usager conçue dans le cadre de cette thèse exploite la puissance de calcul des ordinateurs modernes et leur capacité à rendre des graphiques remplis d'effets visuels qui produisent une représentation davantage informative tout en étant esthétique et entièrement interactive.

La Figure 4.5 montre la représentation originale qui est proposée dans cette thèse. Elle s'inspire des systèmes précédents [Brodlie *et al.*, 1993; Wright et Walton, 1996; Pickles *et al.*, 2004a; Fischer *et al.*, 2007] tout en formalisant les termes et les actions possibles sur l'arbre et en permettant beaucoup plus d'interaction. Concrètement, le but de l'arbre multichronique est de représenter le chemin parcouru par l'utilisateur dans l'espace des paramètres. La forme que prend l'arbre est en quelque sorte déterminée par les actions de l'utilisateur qui explore l'espace des paramètres, et ainsi crée l'espace des simulations. L'arbre multichronique permet donc à l'utilisateur d'interagir avec l'espace des simulations.

Conformément à la Figure 4.5, l'arbre multichronique origine d'un nœud racine (A) qui fournit le *scénario initial* ainsi que l'ensemble des paramètres de départ des modèles sous étude. Les nœuds enfant (C) correspondent à des *points de divergence*, appelés ainsi parce que, typiquement, les résultats de deux (ou plusieurs) simulations ayant le même ensemble de paramètres de départ commencent à diverger à partir du moment où il y a modification d'un paramètre. Dans la littérature, on réfère plutôt à « point de décision » ; dans la présente thèse, un point de divergence peut être créé soit par l'utilisateur, soit par des mécanismes automatiques. Le terme « divergence » a donc été préféré à « décision » parce que l'utilisateur n'est pas nécessairement responsable de la *création d'un point de divergence* (I et J). Selon la volonté de l'utilisateur, la *modification d'un paramètre* peut ne pas résulter en la création d'une nouvelle branche (H). Par contre, dans les conditions où il y a *création d'une nouvelle branche*, la branche originale demeure disponible afin que l'utilisateur puisse *comparer les résultats* d'un changement de paramètre par rapport à aucune intervention de sa part. Le *changement de paramètres en continu* (N) est également envisageable dans la définition conceptuelle



A. Racine de l'arbre : scénario initial (conditions initiales)
B. Branche de l'arbre : cours d'action partiel
C. Nœud de l'arbre : point de divergence
D. Feuille de l'arbre : instance de simulation
E. Segments parallèles : fenêtre de comparaison
F. Branches parallèles : points alignés
G. Axe horizontal : axe de référence
H. Modification d'un paramètre de simulation
I. Création d'un point de divergence
J. Modification de plusieurs paramètres de simulation
K. Simulation sélectionnée
L. Chemin de la racine à une feuille : cours d'action
M. Position sur l'axe de référence : point
N. Signe de zigzag : changement de paramètre continu
O. Feuilles en mouvement : simulation en cours de lecture

FIGURE 4.5 – Représentation conceptuelle de l'arbre multichronique et définitions correspondantes

de l'arbre multichronique. Dans ce cas, tous les changements continus de paramètre s'effectuent sur la même branche.

La structure de l'arbre multichronique est telle que chaque rangée, qui se termine par une feuille de l'arbre, correspond à une *instance de simulation* (D). Un chemin de la racine à la feuille est un *cours d'action* (L), alors qu'une branche entre deux nœuds est un *cours d'action partiel* (B). L'*espace des simulations* est donc défini comme étant « l'ensemble des cours d'action partiels faisant partie d'un arbre multichronique ». L'interaction de l'utilisateur avec l'arbre multichronique donne également lieu à des *animations et à des changements d'apparence* pour les nœuds potentiellement affectés par les actions de l'utilisateur (K).

La Figure 4.5 montre des instances de *simulation en cours de lecture* (O), c'est-à-dire que les données correspondant à ces simulations sont en synchronisme avec celles affichées dans l'unité de transformation de rendu visuel. En effet, à chaque branche de l'arbre (cours d'action partiel) correspond un flot de données ordonné dans le temps de simulation. Chaque nœud de l'arbre (points de divergence et simulations), auquel correspond un instant dans le temps de simulation, est aligné sur un *axe de référence* (G), l'axe horizontal de l'arbre multichronique. Une *variable de référence* reliée à un axe de référence sert à ordonner horizontalement les nœuds de l'arbre. La position sur un axe de référence (M) correspond donc à une valeur de la variable de référence. Typiquement, le « temps de simulation » sera considéré comme la variable de référence, mais dans certains cas l'utilisateur pourrait ordonner les nœuds de l'arbre selon une autre variable qui est contrainte à augmenter ou diminuer de façon monotone. Il est aussi possible d'*aligner des cours d'action de manière ponctuelle* (F) ou les *évaluer sur une fenêtre de comparaison* (E). Finalement, tous les nœuds contenus dans l'arbre multichronique possèdent une étiquette, omise sur la Figure 4.5, renseignant l'utilisateur sur les paramètres qui ont été changés au point de divergence correspondant, ou fournissant toute autre information jugée pertinente dans un contexte donné.

4.3 Opérations supportées par l'arbre multichronique

L'arbre multichronique supporte l'exploration de l'espace des paramètres et de l'espace des simulations en permettant à l'utilisateur d'exécuter diverses opérations interactives. Des opérations interactives appartenant aux deux boucles d'interaction correspondantes seront donc décrites dans les paragraphes suivants. Les tâches interactives

répertoriées dans la littérature (Tableau 4.1) sont entièrement supportées et seront redéfinies dans le contexte de l'arbre multichronique. La liste suivante décrit chacune des opérations possibles en donnant un exemple d'utilisation et des précisions sur le résultat de l'exécution d'une telle opération.

- **Créer/Détruire un point de divergence** : à un temps de simulation arbitraire (peut-être dans le passé), un utilisateur peut décider de modifier un paramètre de simulation afin de tester une hypothèse ou afin de prendre une décision qui influencera le reste du cours d'action. De plus, un algorithme de surveillance de données de simulation peut, selon des critères définis à l'avance, demander au gestionnaire Multichronia de modifier le cours d'une simulation. Que cette opération soit effectuée manuellement par l'utilisateur ou automatiquement par un algorithme approprié, elle consiste à créer un point de divergence. La simulation initiale continue de s'exécuter comme si rien ne s'était passé après s'être clonée au moment opportun, alors que le clone s'exécute en tenant compte de l'ensemble de paramètres modifié. L'utilisateur peut également faire abstraction de la simulation initiale en ne considérant que l'instance s'exécutant avec l'ensemble des paramètres altérés. De la même manière, un utilisateur ou un algorithme de surveillance peut décider qu'un cours d'action n'est plus intéressant et le retirer de l'arbre. La destruction d'un point de divergence entraîne avec lui tous ses nœuds fils.
- **Varier un paramètre de manière continue** : un utilisateur peut être intéressé à varier un paramètre de simulation de manière continue, comme il le ferait avec un logiciel de pilotage de simulation. L'arbre multichronique fournit donc un mécanisme qui permet à cette opération de se réaliser. Concrètement, le succès de la mise en œuvre de cette opération dépendra fortement du simulateur utilisé.
- **Dupliquer les actions portées sur un cours d'action** : un utilisateur peut être intéressé à dupliquer les opérations portées sur un cours d'action et les appliquer sur un autre afin d'observer les résultats. L'utilisateur doit cependant être très vigilant dans l'utilisation de cette opération étant donné que des incohérences temporelles peuvent apparaître dans le cours d'action sur lequel les opérations dupliquées sont appliquées. Par exemple, un couplage caché entre deux paramètres peut invalider une modification pourtant valable dans le cours d'action initial. Cette invalidité est causée par une différence initiale entre les deux cours d'action qui rendrait la duplication d'un cours d'action inutile si elle était inexistante.
- **Supporter visuellement la lecture (*playback*) des cours d'action** : l'utilisateur doit observer les données résultant d'une simulation dans un logiciel approprié. Il s'agit d'une opération disponible nativement dans le cadre conceptuel Multichronia. Par contre, la commande et la gestion des données affichées par l'unité de transformation de rendu visuel est en partie réalisée dans l'arbre multichroni-

que. En effet, l'opération de sélection des données implique pour l'utilisateur le choix des flots de données qui seront affichés.

- **Simuler et faire la lecture de plusieurs cours d'action simultanément** : un utilisateur peut vouloir démarrer plusieurs simulations simultanément en explorant plusieurs ensembles de paramètres et, en même temps, faire la lecture d'un cours d'action différent. Cette opération implique donc une gestion asynchrone de la sauvegarde et de la lecture des flots de données de simulation.
- **Pauser/Reprendre/Arrêter la lecture d'un cours d'action** : un utilisateur peut vouloir pauser la lecture d'une simulation afin de se concentrer sur un détail perçu parmi les données et par la suite reprendre la lecture. De plus, l'utilisateur peut stopper la lecture d'un cours d'action, ce qui équivaut à le détruire.
- **Avancer/Reculer dans le temps de simulation** : un utilisateur peut être intéressé par ce qui s'est passé plus tôt dans le temps de simulation. Avec les systèmes disponibles dans la littérature, il est possible de revenir aux points de sauvegarde, mais non à un temps de simulation arbitraire. Ce manque de flexibilité est considéré comme une limitation et est corrigé dans cette thèse. L'utilisateur a donc la possibilité de sauter d'un temps de simulation à un autre instantanément. Ainsi, reculer le temps de simulation permet de revisiter les cours d'action passés alors qu'avancer dans le temps de simulation permet à l'utilisateur de se projeter dans les cours d'action futurs. L'utilisateur peut se promener dans le temps de simulation à sa guise, comme il le ferait avec un magnétoscope. L'interaction envisagée consiste à manipuler directement les feuilles de l'arbre, qui sont des instances de simulation, en les glissant horizontalement avec un périphérique approprié. Si la variable de référence n'est pas le temps de simulation, les cours d'action se déplaceront selon les valeurs prises par leur position horizontale sur l'axe de référence.
- **Varier la vitesse de lecture d'un cours d'action** : à nouveau selon la métaphore du magnétoscope, un utilisateur peut être intéressé à varier la vitesse de lecture d'un cours d'action. Il est ainsi possible de ralentir la lecture d'un cours d'action afin de mettre l'emphase sur un évènement en particulier ou encore sauter des segments inintéressants en augmentant la vitesse de lecture.
- **Varier le niveau de détails de lecture d'un cours d'action** : un utilisateur peut, par exemple, vouloir s'attarder seulement sur les évènements de haut niveau survenus pendant le cours d'action alors que dans d'autres cas il peut désirer étudier toutes les données et s'attarder sur les détails de bas niveau. Selon l'implantation, cette opération peut fortement dépendre du système de visualisation employé, ou encore être directement accessible via l'interface de l'arbre multichronique.
- **Changer la variable de référence** : un utilisateur peut vouloir comparer des cours d'action selon d'autres variables/paramètres que le temps de simulation.

Changer la variable de référence est donc une opération essentielle à la réalisation de ce besoin.

- **Synchroniser plusieurs lectures de cours d'action** : un utilisateur peut vouloir comparer des évènements survenant dans plus d'un cours d'action. Par exemple, le bris d'un équipement peut survenir dans deux cours d'action différents, mais pas au même temps. L'utilisateur peut être intéressé à savoir ce qui a causé le bris (temps de simulation avant l'évènement) et les conséquences de ce bris (temps de simulation après l'évènement), et ainsi comparer les deux cours d'action. Afin d'accomplir cette tâche, ces derniers doivent être synchronisés par un mécanisme supporté par l'arbre multichronique. Lorsque deux ou plusieurs cours d'action sont synchronisés, leur lecture se fait de manière simultanée. Par exemple, les opérations d'avance/recul rapide dans le temps s'exécutent de manière identique pour les deux lectures de cours d'action.
- **Réduire/Développer une arborescence** : quand un arbre multichronique devient un fouillis en raison de la trop grande quantité de points de divergence créés par l'utilisateur, il peut être intéressant de masquer des parties de l'arbre. C'est pourquoi il est possible de réduire une arborescence et de la développer pour y revenir. Ainsi, l'utilisateur évite de se débarrasser de nœuds qui seraient potentiellement intéressants dans les analyses futures tout en visualisant un arbre montrant une quantité d'information adéquate.
- **Changer la disposition des nœuds d'un arbre** : l'algorithme de disposition des nœuds d'un arbre, qui aligne les nœuds horizontalement et verticalement dans l'espace de travail, peut être personnalisé comme le souhaite un utilisateur. Cette opération lui permet de visualiser sa session de travail sous différentes perspectives.
- **Assigner des étiquettes pertinentes aux nœuds et aux branches** : afin de se retrouver dans l'arbre multichronique, l'utilisateur a besoin d'indices lui permettant de déterminer quel nœud correspond à quelle modification de paramètre. Une étiquette associée à un nœud et une infobulle fournissant des renseignements sur un cours d'action partiel ou un point de divergence sont tout indiqués pour remplir cette tâche.
- **Ajouter un signet/Prendre une note dans l'arbre** : afin de documenter son exploration, un utilisateur peut être intéressé à prendre des notes de ce qu'il observe. L'arbre multichronique permet la prise de telles notes ou encore la sauvegarde d'une configuration particulière sous la forme d'un signet.
- **Sauvegarder/Charger une session de travail** : étant donné que l'exploration de modèles de simulation peut durer plusieurs heures et s'échelonner sur plusieurs jours, l'utilisateur peut être intéressé à sauvegarder sa session de travail afin d'y retourner plus tard ou afin de partager les fruits de son travail avec des collègues. L'arbre multichronique permet donc sa sérialisation dans un fichier.

- **Garder une trace des actions de l'utilisateur** : une interface utilisateur telle que l'arbre multichronique doit être évaluée par des expériences sur les sujets humains afin de vérifier sa validité par rapport à des interfaces plus traditionnelles. Afin d'analyser le comportement des individus participant à l'étude, il est possible d'enregistrer les actions de bas et de haut niveau exécutées par un utilisateur et des les rejouer sur demande.

Les paragraphes précédents donnent une liste exhaustive des fonctionnalités interactives qu'une implémentation complète de l'arbre multichronique serait en mesure d'offrir. Il est clair que toutes les fonctionnalités offertes par les systèmes revus dans la littérature sont également disponibles dans l'arbre multichronique. Avec l'ajout de fonctionnalités originales dont le but est de faciliter le travail de l'utilisateur dans son exploration/analyse de modèles de simulation complexes, l'arbre multichronique se démarque donc de GRASPARC et ses successeurs.

4.4 Intégration de l'arbre multichronique dans le cadre conceptuel Multichronia

Afin d'être exploité correctement, l'arbre multichronique doit être en symbiose avec les autres modules formant le cadre conceptuel Multichronia. Du point de vue de l'interface utilisateur, il est clair que l'arbre multichronique occupe une place importante lors de l'exploration de l'espace des paramètres, mais aussi pendant l'analyse des données (exploration de l'espace des simulations, des données et visuel). Un arrangement approprié sur les moniteurs d'une station de travail serait donc de placer l'interface associée à l'arbre multichronique sur un premier moniteur et les autres modules d'analyse sur un deuxième. Ainsi, l'utilisateur serait libre d'utiliser l'arbre multichronique à sa guise tout en observant les données sélectionnées dans le logiciel de visualisation situé sur un deuxième moniteur.

D'un point de vue technique, il est clair que l'interface de l'arbre multichronique doit être compatible avec les autres éléments du cadre conceptuel. Concrètement, le logiciel de bas niveau de l'arbre multichronique est associé à l'unité de transformation *d'alignement et de sélection de flots de données.* Les données circulant dans cette unité de transformation doivent être compatibles avec celles provenant du simulateur. Le chapitre suivant est entièrement consacré à ce problème. De plus, l'arbre multichronique doit communiquer avec les instances de simulateur (e.g. spécifier les paramètres à modifier et leur valeur, pauser/reprendre l'exécution) selon un protocole bien défini.

Certaines opérations énumérées plus haut requièrent que des fonctionnalités spécifiques soient implantées dans le simulateur. Finalement, il est essentiel pour l'arbre multichronique de communiquer avec le gestionnaire Multichronia afin de passer/recevoir des métadonnées aux/des autres modules, ce qui permet d'activer des opérations automatiquement à l'aide de règles prédéfinies.

4.4.1 Contraintes imposées au simulateur

Les opérations qu'un utilisateur peut effectuer sur un arbre multichronique sont nombreuses et devraient idéalement faire partie de toute implantation concrète. Par contre, certaines d'entre elles dépendent du simulateur qui est exploité lors de la génération des données. Si le simulateur ne répond pas à certaines contraintes très spécifiques, des fonctionnalités peuvent ne pas être disponibles pour l'utilisateur, enlevant ainsi certains avantages que possède l'arbre multichronique par rapport à d'autres interfaces/techniques d'exploration. Le Tableau 4.2 résume les contraintes imposées à un simulateur en les associant avec les opérations correspondantes de l'arbre multichronique. L'hypothèse est faite que certaines fonctionnalités de base sont présentes pour tout simulateur, soit charger un scénario initial et pauser/reprendre une simulation.

TABLEAU 4.2 – Contraintes imposées au simulateur par l'arbre multichronique

Opérations sur l'arbre multichronique	**Contraintes sur le simulateur**	**Notes**
Créer un point de divergence[1] et varier un paramètre en continu	Permettre le changement direct de paramètre et le clonage de simulation	L'opération de créer un point de divergence implique de 1) cloner la simulation originale et 2) modifier directement le paramètre sur le clone
Créer un point de divergence[1]	Permettre la sauvegarde ponctuelle de la simulation et le chargement d'un fichier de sauvegarde (optionnel : en tout temps)	L'opération de créer un point de divergence implique de 1) effectuer la sauvegarde ponctuelle d'une simulation, 2) modifier le fichier résultant et 3) recharger la version modifiée dans une nouvelle instance du simulateur

Suite à la page suivante

TABLEAU 4.2 – suite de la page précédente

Opérations sur l'arbre multichronique	**Contraintes sur le simulateur**	**Notes**
Supporter visuellement la lecture des cours d'action	Envoyer les données de simulation vers des modules logiciels externes	Un simulateur devrait être en mesure d'envoyer les données de simulation de manière asynchrone à l'arbre multichronique pour qu'il puisse faire la lecture des cours d'action
	Quantité de données traitable en temps réel	L'arbre multichronique et l'environnement de lecture de cours d'action devraient être en mesure de traiter les données en temps réel, sans que des délais soient perceptibles par l'utilisateur
	Environnement visuel de lecture de cours d'action disponible	Afin que les utilisateurs puissent observer ce qui se passe dans la simulation, un environnement de lecture de cours d'action est essentiel. Sinon, un logiciel de visualisation sur mesure doit être développé
Simuler et faire la lecture de plusieurs cours d'action simultanément	Instances multiples du simulateur capables de s'exécuter sur un ordinateur	Le logiciel devrait être en mesure d'exploiter efficacement un ordinateur doté de plusieurs processeurs ou encore une ferme de calcul
Générique	Disponibilité des données de simulation dans un temps raisonnable	L'utilisateur ne devrait pas attendre plus que quelque secondes avant que des données de simulation lui soient disponibles, sinon l'interactivité est perdue
	Sources ouvertes (souhaitable)	L'ouverture du code source rend les modifications à apporter au simulateur beaucoup plus faciles pour un développeur

[1]L'une ou l'autre de ces deux contraintes doit être satisfaite afin que cette opération soit disponible

Il est clair qu'un simulateur doit répondre à une quantité minimale de critères afin qu'un utilisateur puisse profiter d'une fonctionnalité de base de l'arbre multichronique, soit la création de points de divergence. La technique de clonage de simulation proposée par Fischer *et al.* (2007), similaire à celle décrite dans le Tableau 4.2, est la plus commune (sauvegarde ponctuelle, modification de paramètre et chargement dans une nouvelle instance de simulateur). La modification de paramètres en continu est une technique empruntée au pilotage de simulation, et donc peu répandue dans les simulateurs qui n'exploitent pas les grilles de calcul. Comme suggéré dans le Tableau 4.2, la quantité de données à traiter par l'arbre multichronique ne devrait pas être trop grande. En effet, seulement les données essentielles aux modules subséquents du pipeline de données devraient poursuivre leur route après avoir été générées par le simulateur. Dans le présent contexte, une base de données stocke les données potentiellement utiles, puis des mécanismes de filtrage discriminent les différents types de données. Plus de détails sur ce processus sont disponibles au chapitre suivant.

L'existence de ces contraintes retire certains types de simulateurs de la liste de ceux potentiellement exploitables par un arbre multichronique. Premièrement, la plupart des simulateurs destinés à des applications scientifiques génèrent des données beaucoup trop lentement et en trop grande quantité pour qu'ils puissent être utilisés de manière interactive. En effet, une simulation d'écoulement de fluides, par exemple, peut nécessiter plusieurs heures voire plusieurs jours en temps de calcul [Kreylos *et al.*, 2002]. L'utilisation de fermes de calcul ou de grilles computationnelles, comme le fait la communauté du pilotage de simulation, diminue les temps de calcul afin qu'une interaction soit possible [Pickles *et al.*, 2005]. L'emploi de telles infrastructures sort du cadre de cette thèse, mais pourrait éventuellement faire partie de travaux futurs. Deuxièmement, il est fort à parier que la plupart des simulateurs à sources fermées ne répondront pas à toutes les contraintes énoncées plus haut. Compte tenu de la fermeture des sources, il sera impossible de les modifier afin de programmer des fonctionnalités supplémentaires sauf en faisant appel au concepteur, ce qui demande du temps et habituellement beaucoup d'argent.

4.4.2 Automatisation d'opérations dans l'arbre multichronique

Le déclenchement automatique d'opérations, activées par des règles, devrait diminuer la quantité de travail à accomplir par l'utilisateur dans l'exécution de tâches simples et répétitives sur l'arbre multichronique. Par exemple, un utilisateur pourrait avoir le désir de créer systématiquement un point de divergence lorsque la valeur d'une variable

dépasse un seuil donné. Le déclenchement automatique de cette opération permettrait à l'utilisateur de se concentrer sur d'autres tâches que celle de surveiller la valeur de la variable pendant l'exploration de l'espace des paramètres et des simulations.

Conceptuellement, une règle est gérée par un module de l'arbre multichronique qui agit comme surveillant des données provenant des simulations, des actions de l'utilisateur et des métadonnées provenant du gestionnaire Multichronia. Une règle est composée de trois parties : une condition, une conclusion et une (ou des) action. Une condition est évaluée comme étant soit « vraie » ou « fausse », tout dépendant des données dont elle a la charge de surveiller. Quand la condition est évaluée « vraie », la conclusion correspondante est activée, ce qui mène à l'exécution d'une ou de plusieurs actions en séquence ou simultanément, tout dépendant de la configuration de la conclusion. Il existe deux types de règles, soit les règles automatiques et les règles manuelles. La condition d'une règle automatique est typiquement associée à des données provenant de l'extérieur de l'arbre multichronique (e.g. d'une simulation, d'une autre unité de transformation), tandis que les règles manuelles sont activées par des actions de l'utilisateur. Ainsi, une condition associée à une règle manuelle est toujours évaluée « vraie ».

Il est important de noter qu'à chaque activation de règle, une entrée est ajoutée à la trace des évènements haut-niveau survenus dans l'interface de l'arbre multichronique. L'analyse *a posteriori* des actions de l'utilisateur est donc facilitée par rapport à la tâche d'analyser des mouvements/clics de souris et des touches de clavier tapées. Finalement, le développement de règles intelligentes permettant une exploration automatisée de l'espace des paramètres sort du cadre de cette thèse et est à la base des travaux de Richard Drouin, un étudiant au doctorat au LVSN de l'Université Laval. Il est prévu d'intégrer les résultats de ses travaux dans le cadre conceptuel Multichronia.

Ce chapitre a présenté l'interface utilisateur qui contribue aux deux principales boucles d'interaction du cadre conceptuel Multichronia, l'exploration de l'espace des paramètres et de l'espace des simulations. Basé sur la représentation d'arbre historique des simulations de GRASPARC et ses successeurs, l'arbre multichronique offre une représentation originale du travail effectué par un utilisateur pendant l'exploration. Les composantes visuelles de l'arbre sont clairement définies, de même que les fonctionnalités interactives qu'il permet. Par contre, afin d'utiliser l'arbre multichronique à sa pleine capacité, le simulateur exploité doit répondre à certaines contraintes et, de manière générale, doit être livré avec le code source ouvert. Finalement, les opérations répétitives exécutées par l'utilisateur peuvent devenir automatiques par l'utilisation de règles activées selon des conditions sur les données ou sur les actions de l'utilisateur.

Chapitre 5

Pipeline de données générique du cadre conceptuel Multichronia

Ce chapitre traite du pipeline de données établi pour le cadre conceptuel Multichronia. Il possède plusieurs caractéristiques, dont celles d'être générique et indépendant de la technologie utilisée. Par ailleurs le pipeline rend chaque unité de transformation indépendante des autres puisqu'elles sont seulement reliées par leurs données d'entrée et de sortie. Une unité peut donc être remplacée par une version alternative offrant des fonctionnalités semblables de manière transparente, ce qui est une excellente qualité d'un logiciel flexible. Les unités de transformation sont dans l'ordre la simulation, l'alignement et la sélection des flots de données, le traitement des données et le rendu visuel. Conformément à la Figure 2.1, l'unité de transformation *simulation* fait partie de l'étape d'exécution de la simulation tandis que les autres unités de transformation font partie de l'étape de l'analyse des données de la simulation. Dans ce chapitre, l'étape de la modélisation de la simulation est également considérée comme faisant partie du modèle de données du cadre conceptuel Multichronia.

Bien que les étapes du processus de simulation interactive soient clairement définies et précisées dans le contexte du cadre conceptuel Multichronia, il en est tout autrement pour le format des données et les transformations qui sont nécessaires afin de les rendre compatibles avec les unités de transformation. Il est convenu que le résultat de l'étape de modélisation de la simulation est un scénario accompagné des modèles de simulation correspondants. L'étape de l'exécution de la simulation accepte le scénario et les modèles afin de générer des données de simulation qui sont envoyées au module d'analyse des données. Donc, le format des données intermédiaires se doit d'être le plus générique possible afin d'être indépendant des modules de traitement.

L'objectif de ce chapitre est d'établir un modèle conceptuel des données impliquées dans le processus de simulation afin que tous les modules travaillent en symbiose et une méthodologie visant à aider les développeurs de systèmes de simulation interactive dans l'intégration de nouveaux outils. Des travaux reliés seront tout d'abord présentés, suivis d'un modèle de données conceptuel et d'un flot de données indépendants de la technologie du simulateur. Par la suite, une spécialisation du modèle de données générique sera présentée, montrant que l'utilisation du *Extensible Markup Language* (XML) facilite le développement d'un pipeline de données répondant à plusieurs critères de design relatifs à la simulation interactive.

5.1 Revue de la modélisation d'un pipeline de données

Un recensement de la littérature fait état de quelques pipelines de données, principalement mis en place pour des applications de visualisation [Haber et McNabb, 1990]. Haber et McNabb (1990) ont en effet proposé un pipeline de données pour la visualisation scientifique qui comporte quatre étapes de traitement. À partir de l'état brut, les données sont structurées, filtrées, associées à des primitives visuelles et rendues graphiquement dans une image. Ce pipeline de données est implicite dans l'unité de transformation *rendu visuel* du cadre conceptuel Multichronia. En contrepartie, peu de travaux ont été réalisés en ce qui a trait au transfert de données de la simulation jusqu'au rendu visuel, et encore moins pour des simulations interactives.

Dans le domaine de la simulation, plusieurs travaux ont été réalisés sur l'interopérabilité entre les différents modèles d'un simulateur et afin d'assurer leur réutilisabilité. Par exemple, Harrison *et al.* (2005) suggèrent un processus appelé KARMA visant à capturer le savoir d'experts dans des modèles interopérables et persistants pour plusieurs applications [Harrison *et al.*, 2005]. Leur approche est fondée sur des concepts, outils et leçons apprises du génie logiciel qui guident les développeurs de modèles. Ces derniers développent, de manière modulaire, des configurations de faible granularité basées sur le XML. Ces travaux se comparent à ceux présentés dans cette thèse parce que Harrison *et al.* (2005) proposent une méthodologie de design qui est en quelque sorte une manière d'encadrer les développeurs dans le processus de modélisation et simulation. Par contre, la méthodologie proposée dans cette thèse porte sur les problématiques reliées à l'analyse des données de simulation et sur l'interaction avec l'utilisateur pendant l'exécution de la simulation plutôt que sur la modélisation et l'interopérabilité des modèles.

Dans le domaine de la simulation interactive, quelques travaux sont détaillés à propos du cheminement des données de simulation. Entre autres, Brooke *et al.* (2003) ont développé la librairie de pilotage de simulation « RealityGrid » [Brooke *et al.*, 2003]. Il s'agit d'une librairie fournissant des interfaces qui permettent d'instrumenter le code de simulation (insérer des appels de fonctions de la librairie) dans des endroits appropriés afin de mettre à jour des paramètres, récupérer des données de simulation et contrôler l'exécution de la simulation. Bien que l'approche par instrumentation soit très flexible, elle contraint le simulateur à communiquer seulement avec une librairie possédant un format de données fermé. Dans cette thèse, une approche basée sur les modèles est adoptée, ce qui signifie que le développement conceptuel des modèles de simulation contribuera en même temps à construire des portions du pipeline de données. De plus, certaines tâches s'exécuteront automatiquement, diminuant la possibilité d'erreurs commises par le programmeur qui instrumente le code source. L'approche par composantes adoptée par Parker *et al.* (1997) dans le développement de SCIRun est intéressante parce qu'elle assure la compatibilité des modèles en ce qui a trait à l'échange de données [Parker *et al.*, 1997]. Par contre, elle est difficilement utilisable pour un modèle de simulation existant parce qu'il doit être séparé en des composantes distinctes, chacune possédant des paramètres particuliers.

De leur côté, Röhl et Uhrmacher (2005) définissent un formalisme de simulation dans un fichier XML et assurent sa validité et son efficacité lors de l'exécution par la liaison des données XML (*XML data binding*) [Röhl et Uhrmacher, 2005]. Les présents travaux s'inspirent de cette technique, mais la généralisent afin qu'une autre technologie que le XML puisse être utilisée dans un contexte approprié. De plus, le pipeline de données proposé ne se limite pas aux étapes de modélisation et d'exécution de la simulation, mais définit des standards de données pour l'analyse et la transformation des flots de données.

5.2 Modèle de données conceptuel, indépendant de la technologie

Cette section présente le modèle de données conceptuel qui établit la base d'une méthodologie visant à faciliter le développement d'un logiciel de simulation interactive. Le but du modèle de données, à ne pas confondre avec le modèle de simulation, est d'assurer que les modules faisant partie du processus de simulation seront compatibles au niveau des données qu'ils échangent. La Figure 5.1 montre les principaux blocs du modèle de données indépendant de la technologie utilisée. Les grands rectangles

représentent des unités de stockage tandis que les rectangles arrondis représentent des unités de transformation. Les unités de stockage encapsulent l'état des données à une étape déterminée et les unités de transformation acceptent en entrée des données d'un certain type et fournissent en sortie des données d'un type défini. Les grandes flèches montrent la direction que prennent les données, c'est-à-dire à partir du scénario vers les données de simulation. Les petites flèches montrent la relation entre les éléments des modèles associés aux unités de stockage/transformation.

Le premier bloc du modèle de données conceptuel est le *scénario de simulation* qui capture l'état de tous les éléments faisant partie de la simulation (e.g. acteurs, modèles, interactions, évènements planifiés). Il peut s'agir soit de l'état initial qui correspond, lorsque chargé dans le simulateur, au démarrage d'une simulation au temps zéro, soit de l'état d'une simulation issu d'un point de sauvegarde, qui correspond au redémarrage d'une simulation à partir du temps où elle a été sauvegardée. Les détails du modèle de l'unité de stockage du scénario de simulation montrent plusieurs niveaux d'abstraction. Premièrement, le scénario proprement dit est un fichier modélisé par un *document.* Ce document lui-même est une instance d'un *modèle de document* et est associé à un *schéma de document* qui définit les éléments que le document peut contenir et qui sert à valider son contenu lors de l'analyse syntaxique. Le schéma du document est une instance d'un *modèle de schéma* qui établit la relation de validité avec le modèle de document. Cette représentation est issue des travaux de Kurtev (2005) qui a étudié les transformations entre les modèles de données [Kurtev et van den Berg, 2005]. Chaque couche horizontale est associée à un niveau de modélisation plus ou moins abstrait. Les relations verticales sont des relations de type « instanceDe », alors que les relations horizontales dépendent du contexte.

Avant d'être chargé dans le simulateur, un scénario doit tout d'abord être converti dans un format approprié afin de maintenir le flux de données indépendant du format de données du simulateur. Cette tâche est accomplie par un moteur de conversion qui accepte en entrée un document de scénario et fournit en sortie des objets directement accessibles par le simulateur, c'est-à-dire dans un langage natif (e.g. Java, C, C++). Ces objets natifs sont appelés *objets désérialisés.* Ils possèdent également plusieurs niveaux d'abstraction, triviaux pour les langages orientés objets et montrés dans le rectangle correspondant de la Figure 5.1. Premièrement, chaque objet désérialisé, ou *objet natif*, est l'instance d'une *classe native concrète* qui est l'abstraction d'un objet à l'aide d'une classe dans un langage natif approprié. Une classe native concrète est elle-même une instance ontologique d'un *modèle de classe*, modélisé dans un langage visuel approprié (e.g. UML). La relation « instance ontologique » est donc un moyen de relier un concept à un type dans un langage donné [Atkinson et Kuhne, 2003]. La même relation est présente entre le concept *instance de classe*, qui est employé au niveau conceptuel dans

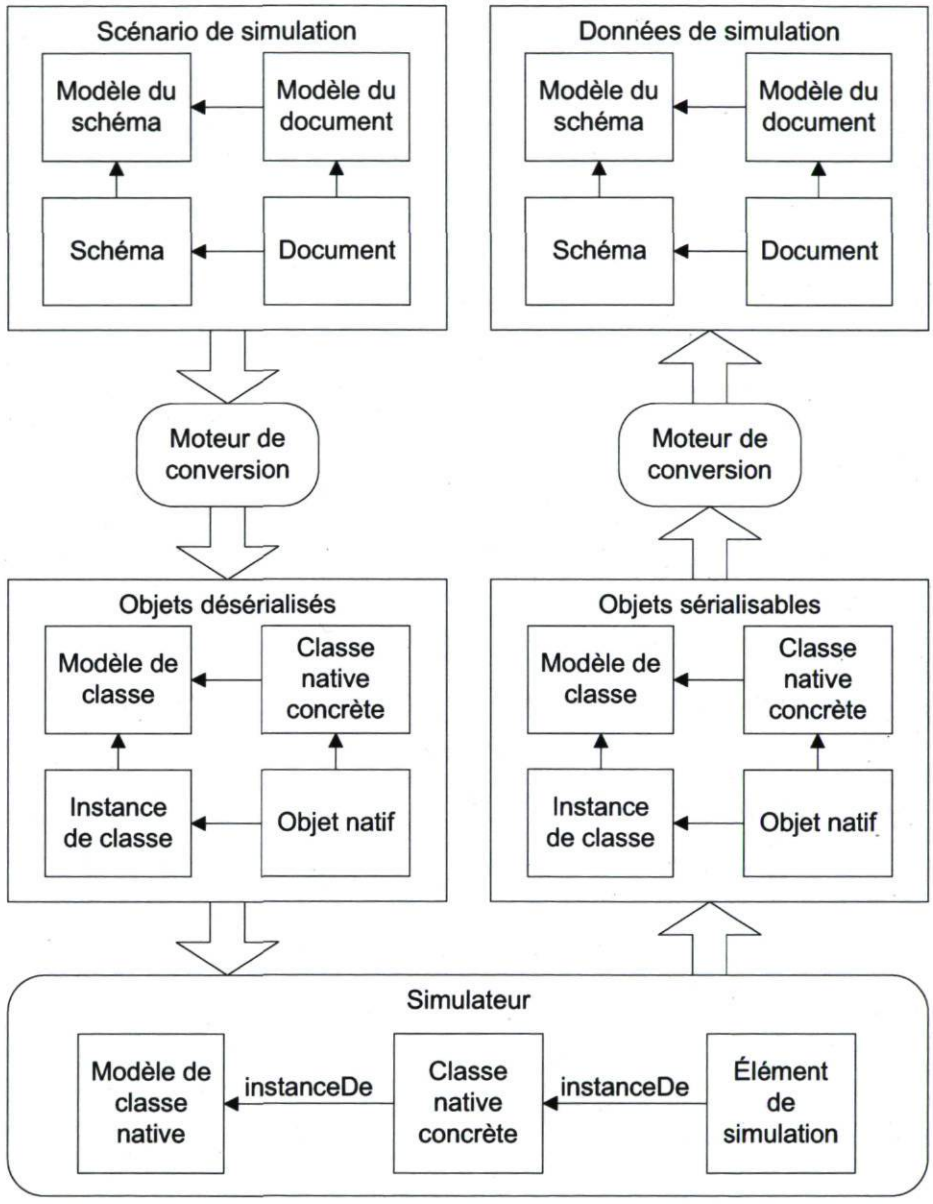


FIGURE 5.1 – Modèle de données conceptuel, indépendant de la technologie

des diagrammes appropriés, et l'objet natif.

Le transfert de données des objets désérialisés vers les éléments de simulation se fait par le biais d'appels natifs, qui sont beaucoup plus rapides que des appels sur des objets sérialisés dans un document. Ce modèle de données fait donc l'hypothèse que les éléments de simulation sont accessibles soit directement ou à l'aide d'une interface de programmation, ce qui est une fonctionnalité commune pour plusieurs simulateurs. Un *élément de simulation* est une instance de *classe native concrète* qui est elle-même une instance ontologique d'un *modèle de classe native*. Des mécanismes internes au simulateur exécutent les modèles de simulation et mettent à jour les éléments de simulation périodiquement (voir Figure 5.2). À chacune de ces mises à jour, l'état de la simulation, c'est-à-dire le contenu de tous les éléments de la simulation, est copié à l'aide d'appels natifs en totalité ou en partie vers des objets sérialisables. Ces derniers sont modélisés de la même manière que les objets désérialisés. Puis, finalement, les objets sérialisables sont sérialisés par un moteur de conversion et écrits dans un document au format correspondant aux données de simulation. Il est important de noter que les objets désérialisés et les objets sérialisables sont modélisés de la même manière, mais pas nécessairement par des modèles identiques. Il en est de même pour le scénario de simulation et les données de simulation.

La Figure 5.2 montre le flux de données de simulation interactive complet résultant de l'intégration du modèle de données dans les étapes du processus de simulation (modélisation, exécution, analyse). Il est important de noter que les unités de transformation du cadre conceptuel Multichronia se situent au niveau de l'exécution de la simulation et de l'analyse des données. Le processus de simulation interactive fait tout d'abord intervenir un utilisateur qui peut interagir avec le système lors de l'édition du scénario et/ou du fichier de sauvegarde et au niveau de l'analyse des données par le biais des interfaces disponibles dans le cadre conceptuel Multichronia. Suite à l'édition du scénario, les données initiales sont modifiées/converties/utilisées et transitent dans les unités de stockage appropriées, présentées à la Figure 5.1, soit dans l'ordre le document du scénario, les objets désérialisés, l'état de la simulation, les objets sérialisables et les données de simulation. Ces dernières sont traitées par les unités de transformation du pipeline d'analyse des données, jusqu'à l'unité de rendu visuel. La copie de l'état de la simulation vers les objets sérialisables s'effectue périodiquement afin de former un flot de données se dirigeant vers le module d'analyse des données. Un utilisateur a le contrôle sur le déroulement de la simulation et il peut modifier des paramètres directement dans le moteur de simulation (ligne pointillée) ou par le biais des objets sérialisables sauvegardés, puis rechargés dans le simulateur après avoir été modifiés. L'unité de transformation *alignement et sélection des flots de données* permet de plus l'exécution de plusieurs simulateurs simultanément, chacun fournissant en sortie un flot

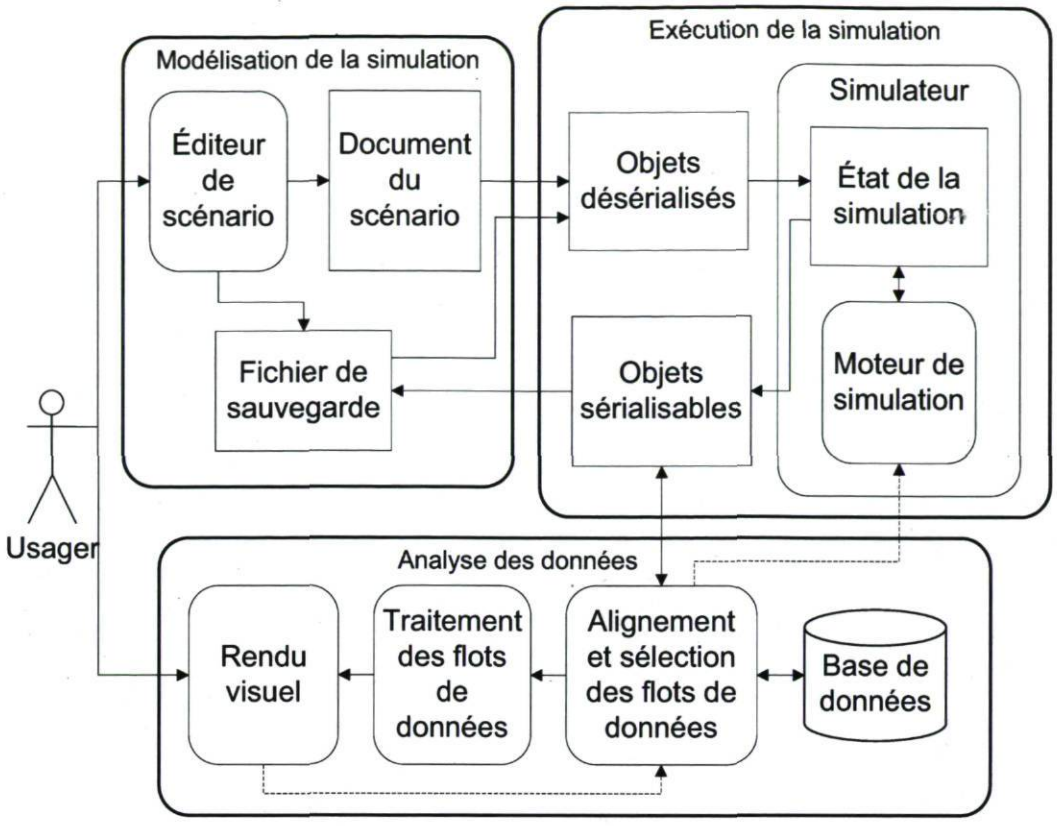


FIGURE 5.2 – Flux de données complet, indépendant de la technologie

d'objets sérialisés.

Le système complet montré à la Figure 5.2 est donc totalement indépendant de la technologie des modules qui le composent. Tout simulateur répondant aux contraintes de la section 4.4.1 peut donc être utilisé afin de générer des données. L'étape cruciale dans le développement d'un tel système est l'établissement d'un lien entre l'état de la simulation et les objets sérialisables/désérialisés, qui implique le développement de code natif. La/les technologie(s) utilisée(s) dans le flux de données sera/seront donc déterminante(s) pour le succès d'un système qui intègre facilement et correctement les modules qui le composent.

5.3 Utilisation du XML comme technologie unificatrice

Cette thèse suggère que le XML est un format de données bien adapté au modèle conceptuel présenté dans la section précédente. Un document XML peut effectivement contenir le scénario, et des technologies reliées peuvent s'occuper de la sérialisation/désérialisation du scénario ou des données de simulation aux objets natifs. Le modèle de données conceptuel spécialisé pour la technologie XML suggère de plus une méthodologie de design qui facilite le travail des développeurs de simulateurs par une approche d'ingénierie basée sur les modèles (*Model-Driven Engineering*, MDE). Cette méthodologie est valide autant pour les développeurs qui travaillent sur des simulateurs existants que pour ceux qui développent de nouveaux simulateurs et qui souhaitent que le moteur de simulation demeure indépendant des modèles de données associés aux étapes de la modélisation et de l'analyse.

Le XML est un langage de balisage générique développé comme un sous-ensemble du SGML (*Standard Generalized Markup Language*) et ayant comme première application le web [Wikipedia, 2009b]. Un *document* peut être représenté par un arbre, donc il contient des *nœuds*. Les nœuds qui portent un nom balisé (entre les caractères « < » et « > ») sont des *éléments*. Un élément comprend des *attributs nommés* et possède des nœuds enfant qui sont soit des éléments ou des *nœuds de texte*. Le *nœud racine* est le premier élément composant un document. Contrairement au HTML, dont le nom des éléments est prédéterminé, un utilisateur peut, en XML, définir ses propres balises. Ce langage est ainsi utilisé afin de décrire les données de plusieurs domaines d'application comme le web sémantique [Decker *et al.*, 2000], les mathématiques [W3C, 2009a], la simulation de systèmes biologiques [Lloyd *et al.*, 2004] et la prise de décision militaire [Hobbs, 2003]. Bien que le XML soit très flexible et qu'il soit utilisé à toutes les sauces, Boyno (2006) met en garde ses utilisateurs afin qu'ils l'utilisent seulement là où il se doit en fournissant des règles à suivre pour l'utilisation des outils le supportant [Boyno, 2006].

Il existe plusieurs catégories d'outils associés au XML. Les analyseurs syntaxiques (*parsers*) sont utilisés afin de lire un document et d'en construire une représentation informatique, tout en validant que son contenu est correct. L'étape de validation est optionnelle et dépend d'un modèle de document qui peut être exprimé en plusieurs formats. Les plus répandus sont le DTD (*Document Type Definition*) et le schéma XML. Ces deux formats permettent de spécifier le nom des balises, des attributs et des relations entre les éléments d'un document XML. Le schéma offre toutefois l'avantage, par rapport au DTD, de définir le type des données associées aux attributs et aux nœuds

de texte des éléments.

Parmi les autres outils relatifs au XML, on retrouve la liaison des données XML (*XML data binding*) qui permet un accès direct et bidirectionnel aux données contenues dans un document XML par le biais d'une représentation exprimée dans un langage natif [Wikipedia, 2009c]. La liaison des données XML implique le développement d'un schéma XML et sa compilation dans un langage natif approprié (e.g. Java, C++, Python, .NET). L'opération de sérialisation (*marshalling*) accepte les données en mémoire et compose le document XML correspondant en utilisant les mêmes noms de classes et d'attributs que dans l'objet natif. Pour sa part, l'opération de désérialisation (*unmarshalling*) accepte un fichier XML et construit en mémoire une structure correspondante à l'aide de classes portant les mêmes noms que les éléments du document XML. L'avantage de la liaison des données XML par rapport au DOM (*Document Object Model*), qui consiste à construire un arbre de données en mémoire à partir d'un document, est le maintien des types reliés aux attributs et aux nœuds de texte. En effet, DOM rend ces derniers disponibles en format texte seulement, laissant place à l'erreur dans le processus de conversion du texte en données natives.

Afin d'archiver des fichiers XML, il existe des bases de données XML qui stockent des documents, les indexent et permettent la recherche de manière similaire aux bases de données relationnelles [Wikipedia, 2009d]. La visualisation du contenu d'un document ou d'un schéma XML est également possible en utilisant des logiciels comme XMLSpy [Altova, 2009]. Finalement, des outils de requête et de transformation peuvent être utilisés afin de modifier le contenu d'un document XML. Le langage XQuery permet l'exécution de requêtes afin d'extraire des informations d'un document XML et de le reformuler en tenant compte des informations extraites [W3C, 2009b]. La syntaxe d'une requête XQuery (« FLWOR » pour *for, let, where, order, return*) est similaire à une requête SQL (*select-from-where*), c'est-à-dire qu'elle est formée d'une première partie qui extrait certaines données du document, d'une deuxième qui combine/filtre ces données et d'une troisième qui exprime le résultat dans un format approprié. Pour sa part, XSLT est le langage de transformation de documents XML le plus connu [W3C, 2009c]. Il permet la transformation d'un document XML en un nouveau document dont le contenu est différent, mais basé sur le contenu du document initial.

En plus d'être lisible en format texte, le XML est auto-descriptif, c'est-à-dire que la structure de données décrit son contenu. En utilisant des noms de balises significatifs, il est trivial pour un utilisateur non familier avec un schéma de découvrir la signification du contenu d'un fichier XML. L'intégration de métadonnées dans un flot de données XML permet l'ajout de détails pertinents à l'analyse des données comme la provenance ou le temps de création. Un document XML, comme la plupart des outils mentionnés

plus haut, est de plus portable sur différentes plateformes.

Bien qu'il soit très populaire et pratique dans le présent contexte, le XML possède plusieurs points négatifs. En effet, il s'agit d'un langage verbeux, c'est-à-dire que le rapport entre la quantité de données utiles et la quantité de données totales (données de structure + données utiles) est faible. Il en résulte un gaspillage de bande passante lors de la transmission de documents XML et une efficacité diminuée de l'analyse syntaxique. Par contre, dans le cas où la performance du XML standard est un facteur limitant pour un logiciel, il existe des formats XML binaires dont le contenu est moins redondant et dont le traitement est plus efficace [Geer, 2005]. Les formats XML binaires proposés doivent toutefois répondre à des standards bien précis afin de s'assurer que le XML demeure entièrement interopérable. Pour leur part, les formats purement binaires (e.g. HDF5, NetCDF) permettent de compresser les données de manière plus efficace et de garder les types associés aux éléments du document [McGrath, 2003]. Finalement, le XML n'est pas encore complètement adapté aux applications de flots de données. En effet, les applications web, qui correspondent à la majorité des utilisations du XML, emploient cette technologie en chargeant des documents complets et non pas en analysant les données aussitôt qu'elles sont arrivées. En contrepartie, les formats de données binaires permettent l'accès aléatoire au document. C'est la raison pour laquelle certaines applications scientifiques utilisent à la fois le XML et des formats binaires dans la même application [McGrath, 2003].

5.3.1 Modèle de données conceptuel spécifique au XML

Dans le cadre de cette thèse, le format de données XML a été choisi pour la démonstration du modèle de données proposé dû à la disponibilité des outils de conversion, de transformation et de manipulation. De plus, le XML permet de représenter un grand nombre de structures de données de manière beaucoup plus flexible que la plupart des formats binaires. Finalement, le XML est simple d'utilisation tant au niveau des outils disponibles que pour l'édition de documents.

La Figure 5.3 montre une spécialisation XML du modèle de données conceptuel de la Figure 5.1. Plus précisément, un scénario de simulation est stocké dans un *document XML*. Ce document XML est validé par le *schéma XML* associé et est une instance d'un *modèle de document XML* qui spécifie la syntaxe du format de données (e.g. balises, attributs, nœuds texte). De la même manière, le schéma XML est une instance du *modèle de schéma XML*, qui lui-même spécifie le lien de validité avec le modèle de document. Un développeur n'a cependant pas à spécifier les deux métaniveaux du langage parce qu'ils sont implicites. La conversion du scénario de simulation XML en

objets désérialisés natifs s'effectue par l'opération de désérialisation du processus de liaison des données XML. Un *objet natif* est l'instance d'une *classe native*, spécifiée à partir de l'étape de compilation du schéma XML dans le processus de liaison (haut de la Figure 5.4). Puisque ces classes sont générées automatiquement, leur spécification en UML est superflue.

La copie des données provenant du scénario vers les éléments de simulation s'effectue par des appels natifs des *objets des éléments de simulation* sur les objets désérialisés. Typiquement, une librairie de liaison des données XML génère des méthodes « get » et « set » permettant respectivement la lecture et l'écriture d'attributs nommés associés aux éléments d'un document XML. Pour la copie de l'état de la simulation vers les objets sérialisables, les attributs des objets éléments de simulation sont copiés vers les attributs correspondants des objets natifs sérialisables. Puis, l'étape de sérialisation du processus de liaison des données XML convertit tous les attributs en format texte pouvant être écrit dans un document ou dans un flot de données XML. Il est important de noter que les schémas XML des données de simulation et du scénario de simulation ne sont pas obligatoirement identiques. Cependant, dans le cas où ils le sont, des données de simulation peuvent être rechargées dans le simulateur de manière transparente, sans qu'aucune conversion n'ait lieu. Ainsi, le processus de simulation interactive est facilité parce que la sauvegarde ponctuelle d'une simulation et son chargement dans le simulateur deviennent triviaux. De plus, la modification de paramètres de simulation s'effectue tout simplement en modifiant le document XML sauvegardé lors de la sérialisation.

5.3.2 Méthodologie de modélisation des données

La Figure 5.3 met en surbrillance des boîtes bleues, reprises et ordonnées dans le haut de la Figure 5.4 afin de montrer une méthodologie de modélisation des données. La mise en œuvre complète de cette méthodologie permet à un développeur de simulateur de modéliser les éléments de la simulation et d'avoir automatiquement à sa disposition un format de fichier permettant 1) de charger un scénario de simulation et 2) de sauvegarder les données utiles résultant de l'exécution de la simulation ou son état complet en un instant donné.

La première étape de la méthodologie consiste à développer un modèle de classes UML des éléments de la simulation faisant partie de l'état de la simulation. La Figure 5.5 montre un exemple fictif d'un tel modèle dans lequel une entité de simulation est un agent qui évolue sur un terrain. La classe principale de l'état de la simulation, « Simulation », a comme enfant quatre éléments, soit « timeStep », « InitialScenario », « Random » et « simulationEntitiesList », qui correspondent respectivement au pas de

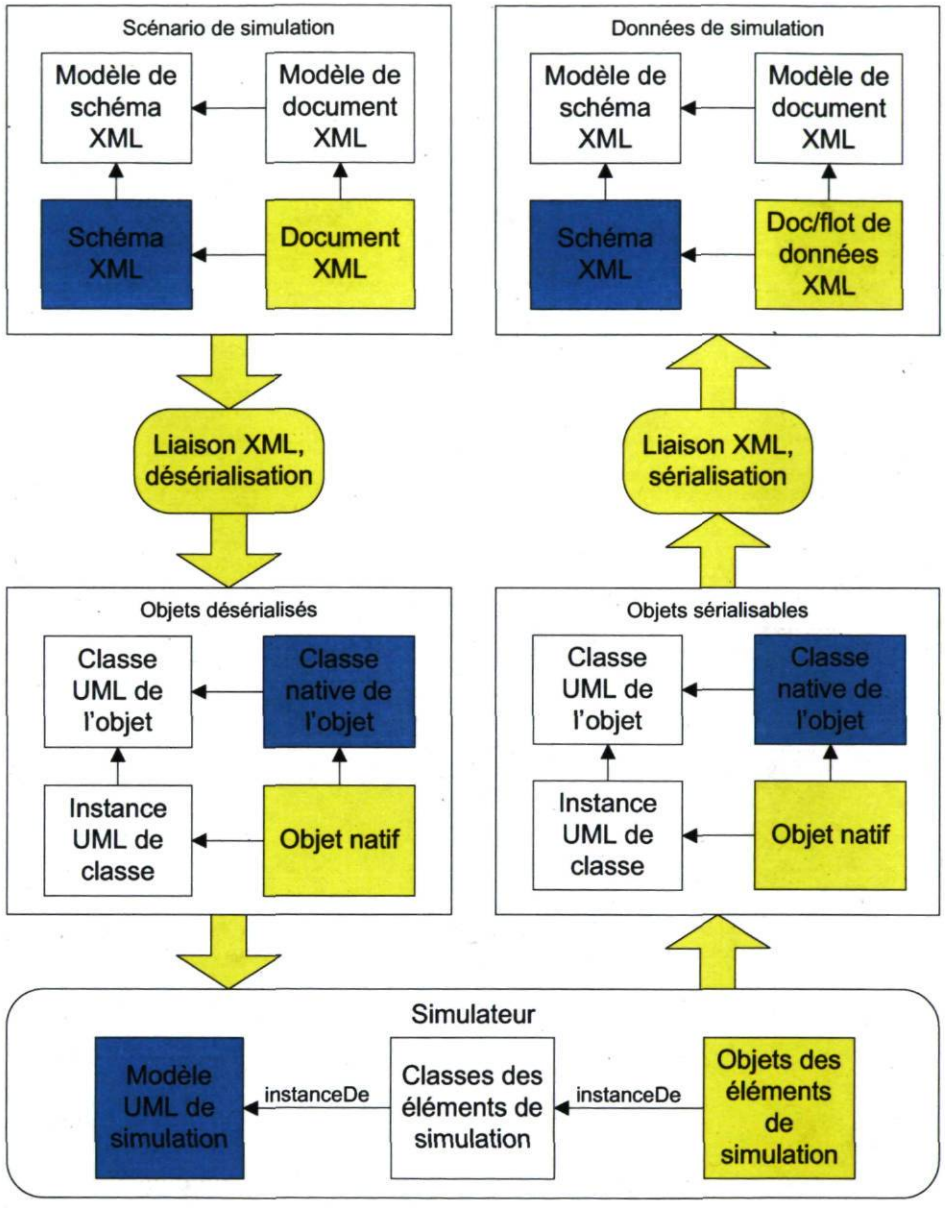


FIGURE 5.3 – Modèle de données conceptuel spécialisé pour le format de données XML. Les boîtes bleues montrent une méthodologie à suivre tandis que les boîtes jaunes identifient un pipeline de données

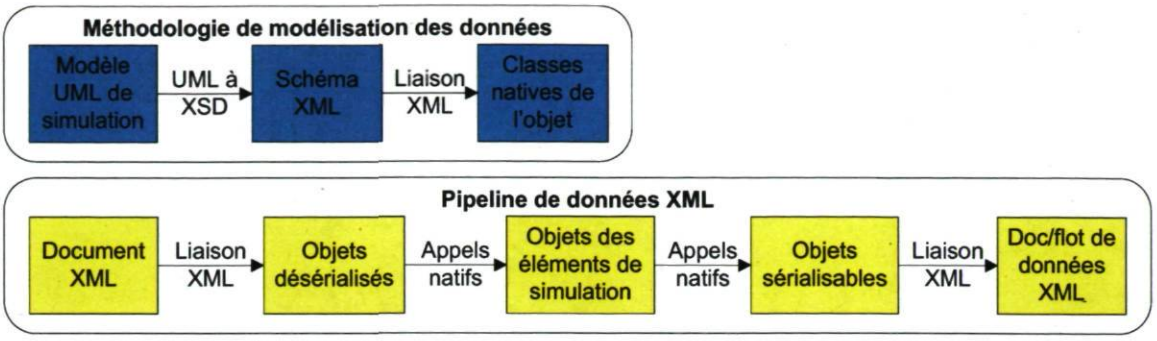


FIGURE 5.4 – Méthodologie de modélisation (haut) et pipeline de données conceptuel unifié XML (bas)

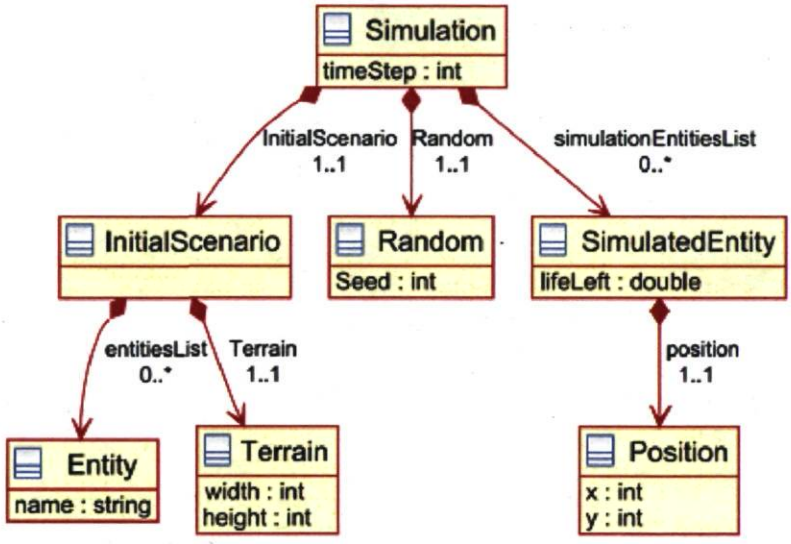


FIGURE 5.5 – Exemple de diagramme UML des classes de l'état de la simulation

temps de simulation courant, aux paramètres associés au scénario initial (e.g. définition des entités devant être instanciées et du terrain sur lequel elles vont évoluer), au générateur de nombres aléatoires et aux entités de simulation instanciées. Il est clair que lors du chargement initial du scénario, le moteur de simulation interprète tout d'abord le scénario initial, ce qui mène à l'instanciation des entités simulées et à l'ajustement de leurs paramètres initiaux. Par contre, pendant l'exécution, le scénario initial demeure intact. Les autres attributs sont alors mis à jour périodiquement ; dans le cas d'une simulation à temps continu à chaque pas de temps, et dans le cas d'une simulation à évènements discrets à chaque évènement.

L'étape du développement du modèle des classes de l'état de la simulation est celle qui demande le plus de travail et le plus de réflexion de la part du développeur. Toutefois,

s'il commet une erreur lors de cette étape, il peut y revenir sans grande incidence sur le reste de la méthodologie. En effet, les autres étapes consistent à transformer le modèle initial dans un autre langage à l'aide d'outils appropriés. Le modèle UML est tout d'abord transformé en schéma XML [Novotný, 2004; Carlson, 2006]. La présence d'un schéma en tant que format du scénario de simulation permet donc au développeur de valider les documents XML qu'il conçoit et entend charger dans l'état de la simulation. Étant donné qu'il y a correspondance entre les classes présentes dans le modèle UML de l'état de la simulation et les éléments définis dans le schéma, le transfert de données est relativement facile. Le résultat du transfert du modèle de la Figure 5.5 en schéma XML est montré à la Figure 5.6. Il est clair que toutes les classes modélisées sont définies par des éléments dans le schéma. De plus, toutes les relations entre les classes sont gardées.

La troisième étape de la méthodologie assure que le transfert de données du document XML jusqu'aux structures de l'état de la simulation se déroule sans faille. Il s'agit de la compilation du schéma XML par le processus de liaison de données XML. En effet, le résultat de la compilation du schéma est un ensemble de classes qui correspondent exactement aux éléments du schéma, et par le fait même aux éléments de l'état de la simulation. La Figure 5.7 montre le fichier Java produit par le compilateur JAXB [JAXB, 2009] pour la classe « SimulatedEntity ». Les éléments enfant sont clairement définis avec leurs méthodes *get* et *set*. De plus, des annotations Java sont ajoutées afin d'aider la librairie JAXB lors de la sérialisation et de la désérialisation à bien accomplir son travail.

En résumé, le développeur du simulateur n'intervient qu'une seule fois dans la méthodologie de modélisation des données présentée, soit dans le développement du modèle UML de l'état de la simulation. Les autres étapes s'exécutent automatiquement. De plus, en utilisant des outils de modélisation modernes comme Rational Rose [IBM, 2009] ou Together [Borland, 2009], le code source associé aux classes de l'état de la simulation est généré automatiquement. L'étape de la compilation du schéma XML en classes contenant des données peut alors paraître inutile. Cette thèse propose toutefois que cette étape est essentielle pour les deux raisons suivantes : 1) du point de vue de la programmation, l'analyse syntaxique d'un fichier XML est une opération dans laquelle des erreurs de programmation peuvent facilement être insérées. Par exemple, il peut y avoir erreur au niveau de l'interprétation des types de données chargés d'un document XML puisque ces derniers sont perdus en format texte. Aussi, le nom des éléments retrouvés dans l'analyseur syntaxique et dans le document XML doit être exactement le même que dans le schéma, sans quoi une erreur d'analyse survient. Finalement, l'analyseur syntaxique doit être mis à jour afin d'interpréter les nouveaux éléments pour tout ajout dans l'état de la simulation ; 2) du point de vue génie logiciel, l'ajout d'une couche supplémentaire entre le simulateur et les documents de scénario/de données de

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Entity" type="Entity"/>
  <xsd:element name="InitialScenario" type="InitialScenario"/>
  <xsd:element name="Position" type="Position"/>
  <xsd:element name="Random" type="Random"/>
  <xsd:element name="SimulatedEntity" type="SimulatedEntity"/>
  <xsd:element name="Simulation" type="Simulation"/>
  <xsd:element name="Terrain" type="Terrain"/>
  <xsd:complexType name="Entity">
    <xsd:sequence>
      <xsd:element name="name" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="InitialScenario">
    <xsd:sequence>
      <xsd:element ref="Terrain"/>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="entitiesList"
type="Entity"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Position">
    <xsd:sequence>
      <xsd:element name="x" type="xsd:int"/>
      <xsd:element name="y" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
  <x  <xsd:element name="Seed" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
  <xs sd:complexType name="Random">
    <xsd:sequence>
    d:complexType name="SimulatedEntity">
    <xsd:sequence>
      <xsd:element name="lifeLeft" type="xsd:double"/>
      <xsd:element name="position" type="Position"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Simulation">
    <xsd:sequence>
      <xsd:element name="timeStep" type="xsd:int"/>
      <xsd:element ref="Random"/>
      <xsd:element ref="InitialScenario"/>
      <xsd:element maxOccurs="unbounded" minOccurs="0" name="simulationEntitiesList"
type="SimulatedEntity"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="Terrain">
    <xsd:sequence>
      <xsd:element name="width" type="xsd:int"/>
      <xsd:element name="height" type="xsd:int"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

FIGURE 5.6 – Schéma XML résultant de la transformation d'un diagramme UML de classes des entités d'une simulation

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "SimulatedEntity", propOrder = { "lifeLeft", "position" })

public class SimulatedEntity
{
    protected double lifeLeft;
    @XmlElement(required = true)
    protected Position position;

    public double getLifeLeft()
    {
        return lifeLeft;
    }

    public void setLifeLeft(double value)
    {
        this.lifeLeft = value;
    }

    public Position getPosition()
    {
        return position;
    }

    public void setPosition(Position value)
    {
        this.position = value;
    }
}
```

FIGURE 5.7 – Résultat de la compilation d'un schéma XML représentant les entités d'une simulation

simulation le rend indépendant de la technologie utilisée. Si pour une raison quelconque le XML ne remplit plus les besoins de la simulation, il peut être remplacé par un autre langage de manière transparente au simulateur. Ce dernier continue tout simplement de lire/écrire les données de l'état de la simulation dans des classes intermédiaires dédiées au stockage de données.

Cette méthodologie de modélisation des données est donc destinée au développement de nouveaux simulateurs. Afin de l'utiliser pour des simulateurs existants, il faut la modifier légèrement. L'ajout d'une étape d'ingénierie inverse de l'état de la simulation avant le développement du modèle UML facilite l'application de la méthodologie. En effet, le modèle UML peut être généré automatiquement à partir de classes existantes. La conversion en schéma XML et la liaison des données se déroulent alors normalement. L'écriture de code source reliant les classes existantes de l'état de la simulation aux classes générées par l'étape de liaison des données XML est la seule tâche à accomplir manuellement. Par contre, dans le cas où l'ingénierie inverse n'est pas possible, il faut développer un nouveau modèle UML qui sera utilisé strictement pour la génération du schéma XML.

5.3.3 Flux de données basé sur le XML

Les boîtes jaunes mises en évidence à la Figure 5.3 indiquent les divers éléments d'un flux de données basé sur le XML, repris dans le bas de la Figure 5.4. Il débute par le scénario de simulation (ou point de sauvegarde) exprimé dans un document XML. La fonction de désérialisation de la liaison des données XML transforme le document en objets natifs. Les éléments de ces objets sont ensuite copiés vers les éléments de la simulation, également exprimés en objets natifs. Après avoir été modifiés par le moteur de simulation les éléments de la simulation sont périodiquement copiés dans d'autres objets natifs, cette fois-ci sérialisables. L'étape de sérialisation du processus de liaison transforme finalement ces objets en documents/flots XML pour analyse. La Figure 5.8 reprend ce flux de données, mais en ajoutant quelques détails au niveau de l'analyse. Les données de simulation sont ainsi stockées dans une base de données XML gérée par l'arbre multichronique. Les flots sélectionnés et alignés poursuivent leur route vers la visualisation en passant par une étape de filtrage. Ces flots de données XML peuvent donc être modifiés par les technologies associées, soit le XQuery ou le XSLT. Finalement, un logiciel de visualisation interactive approprié permet l'affichage des données. L'utilisation du XML et de la suite d'outils disponibles permet donc de faciliter l'intégration des différents composants et rend les modules interopérables tout en demeurant indépendants.

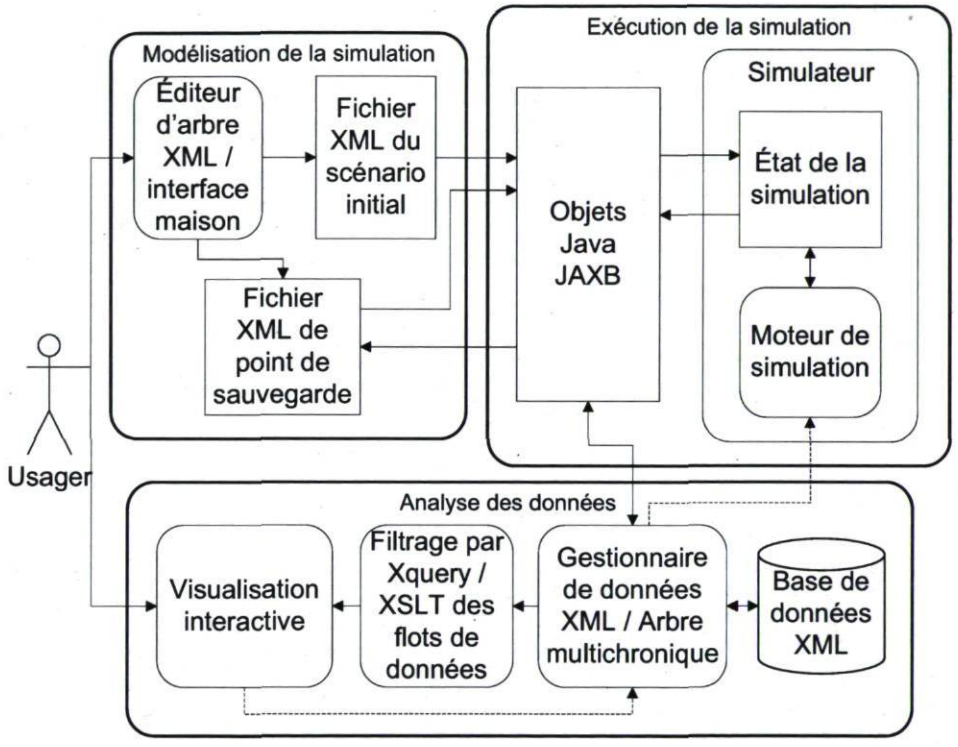


FIGURE 5.8 – Flux de données concret unifié XML

Chapitre 6

Scénario d'expérimentation de Multichronia

Ce chapitre présente le scénario d'expérimentation qui a été choisi afin de mettre en œuvre Multichronia. Il s'agit de l'étude de l'attaque de convois militaires, tels que ceux des Forces Canadiennes qui circulent présentement en Afghanistan, par des insurgés [The Canadian Press, 2008]. Ce scénario est utilisé afin de valider expérimentalement les composantes du projet IMAGE, réalisé à RDDC Valcartier [Lizotte *et al.*, 2007], dans lequel le cadre d'application Multichronia implante les fonctionnalités d'exploration de simulations. Pendant cette expérimentation, il sera demandé à des participants d'utiliser Multichronia afin de générer des données les aidant à comprendre les caractéristiques du système complexe qu'est le scénario d'attaque de convoi militaire. Un contexte général sera tout d'abord présenté, suivi d'une description de la modélisation basse fidélité d'une attaque de convoi. Par la suite une description de la modélisation stratégique des attaques de convoi sera décrite pour terminer avec le rôle de l'utilisateur et l'intégration de ce scénario dans Multichronia.

6.1 Contexte général de l'attaque de convois militaires

Les convois militaires sont généralement associés au transport de troupes, de munitions, de matériel ou de ravitaillement. Il existe typiquement deux types de convois qui partagent des buts différents : les convois de logistique et les convois d'attaque. Les convois de logistique sont déployés afin de ravitailler, par exemple, des postes avancés.

Le but principal d'un tel convoi est de transporter le maximum de charge utile (cargo) pour un minimum de déplacement. Pour ce qui est des convois d'attaque, ils sont déployés afin d'éliminer des cibles ennemies. Leur but est donc de se déplacer le plus rapidement possible afin d'atteindre l'objectif avant de retourner à la base. De plus, un but commun aux deux types de convois est la survie des soldats qui les composent. Un convoi militaire typique est composé de plusieurs types de véhicules dont des véhicules de transport et des véhicules pouvant assurer une défense en cas d'attaque des insurgés.

Pour ce qui est des insurgés, ils n'ont qu'un seul but : causer le maximum de pertes dans les convois. Les moyens qu'ils emploient pour arriver à leur but sont très primitifs comparés aux armes des soldats des Forces Canadiennes, mais ils réussissent tout de même à causer énormément de dommages dû à l'effet de surprise dont ils disposent. Le temps n'est pas une contrainte pour eux, donc ils peuvent attaquer à n'importe quelle heure du jour et à diverses fréquences. Les armes qu'ils possèdent sont entre autres des dispositifs explosifs improvisés (*Improvised Explosive Device*, IED) et des grenades propulsées par fusée (*Rocket-Propelled Grenade*, RPG). Un IED est un dispositif explosif fabriqué avec des matériaux courants en agriculture ou dans le domaine médical [Defense Update, 2004b]. Les types de IEDs sont variés, allant du kamikaze portant une ceinture d'explosifs au dispositif activé par téléphone cellulaire, en passant par les dispositifs activés à distance et branchés à un détonateur. Ces deux derniers types de IED sont enterrés sous la surface de la route sur laquelle circule le convoi. Pour ce qui est des RPGs, ils sont habituellement lancés par des insurgés embusqués qui se cachent derrière des bâtiments ou des civils [Defense Update, 2004a]. Il est reconnu qu'un IED cause le maximum de dommages en dessous du véhicule, pouvant éventuellement l'éventrer, alors qu'un RPG cause davantage de dommages sur le côté d'un véhicule.

Les stratèges militaires ont développé plusieurs tactiques afin de minimiser les pertes causées par les attaques de convoi [Defense Update, 2005]. En effet, les conducteurs de véhicules de convoi tentent de garder une certaine vitesse et ne s'arrêtent que très rarement afin d'éviter les embuscades. De plus, la configuration d'un convoi fait en sorte que des véhicules de protection se trouvent à l'avant et à l'arrière afin de faire feu sur tout suspect s'approchant trop près. Il existe aussi des moyens technologiques comme des brouilleurs d'ondes qui rendent inefficaces les déclenchements de IEDs par téléphone cellulaire ou des avions sans pilote (*Unmanned Aerial Vehicle*, UAV) qui détectent du haut des airs des portions de la route où un IED aurait pu être enterré. Par contre, un moyen très efficace qui sert à protéger les occupants d'un véhicule est d'y rajouter du blindage, ce qui augmente la masse du véhicule, donc diminue sa mobilité. L'ajout de blindage est donc avantageux au niveau de la survie des occupants, mais est désavantageux au niveau des buts respectifs des convois de logistique et d'attaque. En effet, l'ajout de blindage élimine une partie du cargo transportable pour un convoi de

logistique et ralentit un convoi d'attaque.

6.2 Modélisation tactique de l'attaque d'un convoi militaire

Le scénario de l'attaque de convois militaires par des insurgés a été modélisé au niveau tactique, c'est-à-dire que les opérations de déplacement du convoi, l'attaque des véhicules par les insurgés et le retour de feu du convoi répondent à des règles/équations préétablies. Le niveau de fidélité envisagé n'est pas très élevé puisque le but de l'expérimentation exploitant ce scénario n'est pas de retirer des données sur des nouvelles stratégies à adopter par les Forces Canadiennes. L'objectif est plutôt d'évaluer la capacité des participants à comprendre le système complexe qui leur est présenté et la plus-value des outils interactifs d'exploration tactique du scénario, dont Multichronia fait partie. Par contre, le modèle établi doit répondre à certaines caractéristiques des systèmes complexes afin que le but de l'expérimentation soit atteint.

Certains travaux visant à modéliser l'attaque de convois par des insurgés ont été réalisés [Hakola, 2004; Gill et Kalantzis, 2007]. Dans cette thèse, la modélisation par agent a été adoptée parce que cette technique permet de facilement modéliser des entités capables de prendre des décisions et de les faire interagir entre elles et avec leur environnement. Le convoi est ainsi composé de dix agents identiques alors que les insurgés sont modélisés par cinq agents de type IED ou RPG. Tous les agents évoluent sur le terrain montré à la Figure 6.1. Il s'agit d'un terrain d'une dimension de 700x700 unités sur lequel plusieurs entités de différents types sont positionnées. Le convoi démarre au point de départ et se rend à la destination finale en passant par les points de cheminement (croix bleues). Les agents IED sont placés sur le chemin en des endroits stratégiques. Les agents RPG sont placés près des bâtiments (rectangles rouges). Les bâtiments et les arbres (en vert sur la Figure 6.1) sont des mesures de camouflage pour les agents RPG parce qu'ils bloquent les senseurs des véhicules du convoi.

Les agents véhicules du convoi comprennent deux paramètres ajustables. Il s'agit de la quantité de blindage en-dessous du véhicule (UAT) ainsi que la quantité de blindage sur le côté du véhicule (SAT). Ces deux paramètres prennent des valeurs de 0 à 10 et sont ajustés avant le début de la simulation. Une faible valeur de blindage offre peu de protection au véhicule, mais lui permet davantage de mobilité. Une grande valeur de blindage offre beaucoup de protection au véhicule, mais moins de mobilité, le rendant plus vulnérable aux RPGs dont la précision de tir est supérieure sur un convoi se

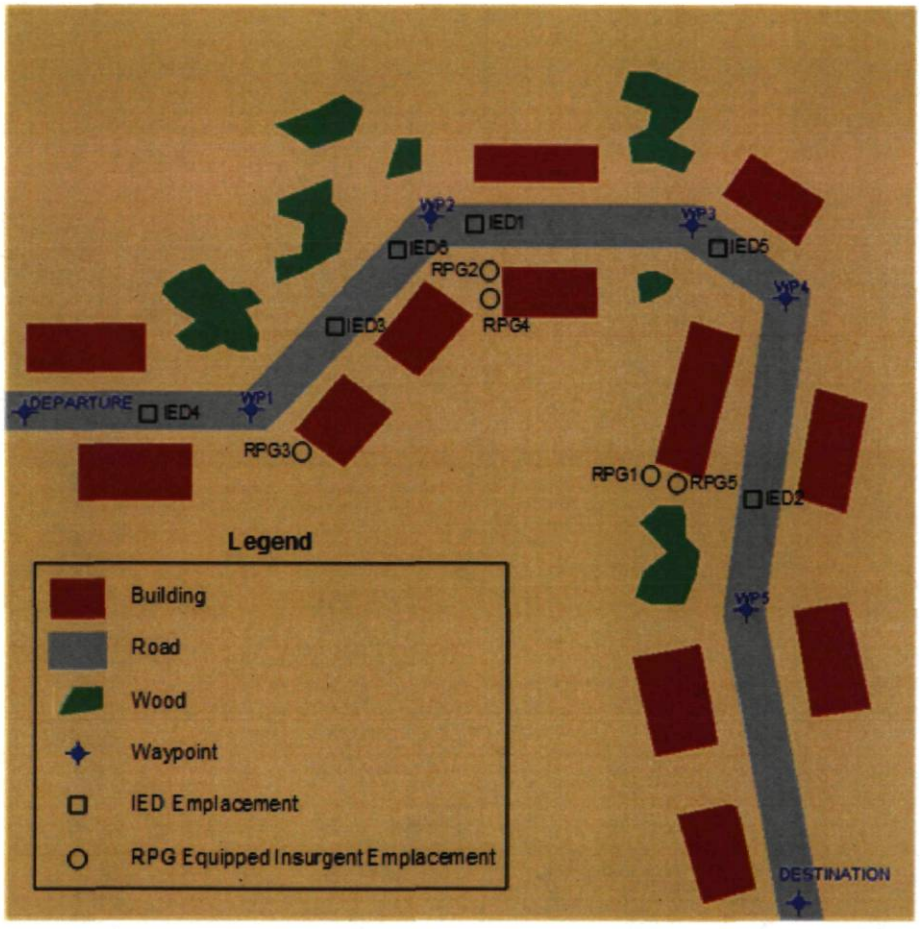


FIGURE 6.1 – Terrain sur lequel évoluent les véhicules du convoi, les agents IED et les agents RPG

déplaçant lentement. Pour ce qui est des insurgés, le seul paramètre variable est leur nombre, ajustable avant le début de la simulation. En effet, la somme des agents RPG et des agents IED présents sur le terrain est cinq. Ils sont disposés sur les emplacements disponibles, montrés à la Figure 6.1. Ainsi, lorsque trois agents IED sont présents sur le terrain, il n'y aura seulement que deux agents RPG.

Pour ce qui est de la dynamique de la simulation, lorsque le convoi avance sur la route, à une vitesse déterminée, et qu'un agent IED s'y trouve, ce dernier explose aussitôt qu'un véhicule s'approche de lui à moins de 20 unités. De même, aussitôt qu'un RPG perçoit un membre du convoi avec son senseur (la distance varie avec la vitesse du convoi), il tire dans sa direction. Les attaques des IEDs et des RPGs, dont le dommage est modulé respectivement par la quantité de blindage présente en-dessous et sur le côté du véhicule ont un effet sur la vie restante du véhicule touché. Un véhicule tombe dans l'état « mort » quand sa vie devient zéro. Les véhicules du convoi se défendent également après avoir été attaqués en répliquant aux RPGs. Par contre, en mode « défense », le convoi se déplace de deux unités par pas de temps, ce qui le ralentit considérablement. Il devient donc plus vulnérable aux RPGs restants. Plus de détails sont disponibles sur la dynamique d'un scénario typique à l'Annexe A.

Les données disponibles pour l'analyse d'une simulation tactique sont nombreuses. En effet, le cours d'action en entier peut être analysé. Ces données comprennent notamment la position de tous les agents, leur niveau de vie et leur état courant en fonction du temps. Des mesures d'efficacité (MoE) et des mesures de performance (MoP) peuvent également être extraites de l'état final du convoi et des insurgés. Il s'agit en effet de la compilation des données de simulation disponibles à l'état final comme la survie des véhicules du convoi, la quantité de cargo transporté et le temps mis pour arriver à destination. Pour plus de détails sur le calcul de ces MoEs et MoPs, voir le rapport technique de Bernier et Rioux (2008).

Certaines des caractéristiques du scénario d'attaque de convoi militaire correspondent à la définition même d'un système complexe [Poussart, 2006]. Par exemple, le fait que le modèle comprenne plusieurs entités interagissant entre elles laisse croire que des comportements émergents pourront se manifester. De plus, les interactions entre les entités ne sont pas nécessairement linéaires. En fait, elles dépendent de l'implantation des modèles dans le simulateur utilisé. L'exécution d'un scénario est dépendante de son historique, c'est-à-dire que le futur d'un convoi est dépendant de son passé, ce qui est une autre caractéristique des systèmes complexes. Par contre, un des problèmes du scénario tactique pour qu'il soit utilisé avec le cadre d'application Multichronia et qu'il puisse bénéficier de toutes les fonctionnalités offertes est l'impossibilité de prendre des décisions sur les paramètres d'intérêt pendant l'exécution du scénario. En fait, des

décisions peuvent être prises à tout moment, mais sans avoir un sens physique. Par exemple, un utilisateur pourrait changer le blindage d'un véhicule pendant qu'il se déplace et observer l'effet de ce changement. Par contre, il est peu probable que ce genre de test d'hypothèse soit réalisable dans la réalité.

Il est important de noter qu'aucun processus stochastique n'est présent dans le scénario actuel. Les senseurs ont donc des caractéristiques déterminées, de même que les armes qui ont un indice de dommage fixe. Les ressources de calcul limitées et le focus des travaux, portés davantage sur l'interaction avec l'utilisateur, ont forcé le développement d'une simulation déterministe. En effet, la présence d'éléments stochastiques dans une simulation nécessite plusieurs réplications d'un scénario possédant les mêmes paramètres initiaux, mais dont l'initialisation du générateur de nombres aléatoires est différente. Les données résultant des réplications sont donc compilées et des statistiques sur les résultats de la simulation sont alors recueillies. Ainsi, puisque plusieurs répétitions de simulation doivent être exécutées, une ferme de calcul est nécessaire afin que les données soient générées suffisamment rapidement pour maintenir une interaction avec l'utilisateur. De plus, la prise de décision pendant l'exécution n'est pas réalisable lorsque plusieurs réplications sont présentes parce qu'une décision peut être valide dans un cours d'action et invalide dans un autre. Il s'agit d'ailleurs d'un problème non résolu dans cette thèse.

6.3 Modélisation stratégique d'attaques de convois militaires par coévolution

Puisque le scénario tactique d'attaque de convoi militaire présenté à la section précédente ne permet pas d'exploiter entièrement la fonctionnalité de prise de décisions en cours de simulation de Multichronia, un scénario stratégique a été développé. Ce scénario est dit *stratégique* parce qu'il permet à l'utilisateur de prendre des décisions de haut niveau sur le comportement des agents formant le convoi lors de simulations tactiques futures. Le résultat d'une simulation stratégique est donc un enchaînement de simulations tactiques pour lesquelles les paramètres initiaux sont ajustés par une règle d'évolution en fonction des performances obtenues dans le passé (voir Figure 6.2). Il s'agit en d'autres mots d'un scénario de coévolution entre le convoi et les insurgés dans lequel l'utilisateur va être appelé à varier les paramètres d'évolution afin d'atteindre un objectif fixé.

La *coévolution* est un terme dont l'origine est la biologie évolutionnaire qui décrit

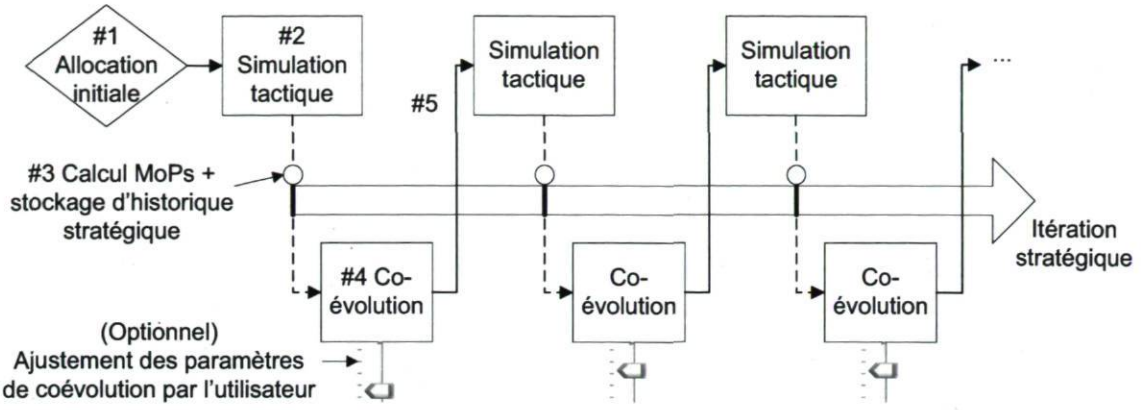


FIGURE 6.2 – Détails du processus de coévolution dans le contexte d'une simulation stratégique

le phénomène où deux espèces évoluent en même temps en réponse directe l'une de l'autre [McDonald et Upton, 2005]. Dans le cas présent, deux « clans » luttent l'un contre l'autre, soit les agents formant le convoi et les agents jouant le rôle des insurgés (RPG et IED). Les deux clans évoluent de manière à maximiser leur objectif à chaque itération. Ainsi, une simulation stratégique, ou coévolution, dans le contexte actuel consiste à (voir Figure 6.2) :

1. Réaliser l'allocation initiale des paramètres du scénario tactique pour les deux clans, soit le SAT et le UAT pour le convoi et le nombre de RPGs et de IEDs pour les insurgés ;
2. Exécuter une simulation tactique tenant compte de la valeur des paramètres à l'instant présent ;
3. Calculer des MoPs à partir de l'état final de la simulation tactique et les stockers dans l'historique de simulation stratégique ;
4. Calculer à l'aide d'une *règle d'évolution* de nouvelles valeurs de paramètres pour la prochaine simulation du scénario tactique en tenant compte des MoPs obtenues pour chacun des clans ;
5. Retourner à 2) jusqu'à ce que les résultats du scénario stratégique converge (i.e. exactement les mêmes paramètres pour le convoi et les insurgés qu'à un pas de temps précédent) ou qu'un nombre d'évolutions suffisant ait été généré.

En ce qui a trait aux règles d'évolution, les insurgés ont un seul objectif, soit celui de causer le maximum de pertes au convoi. La MoP considérée est la quantité totale de dommage causé au convoi pendant une simulation tactique. Une règle d'évolution typique est donc de choisir la prochaine allocation de RPGs et de IEDs qui causera le

maximum de dommage au convoi. De l'autre côté, le convoi possède plusieurs MoPs qui rendent la règle d'évolution un peu plus sophistiquée. Il s'agit de la capacité de survie du convoi, du flux de transport de cargo et du nombre de pas de temps pour parvenir à destination. Une pondération entre ces trois MoPs transformera donc le scénario de coévolution en un problème multi-objectif. Les objectifs du convoi sont donc variés et dépendront de l'implantation de la règle d'évolution. On remarque toutefois que les objectifs des deux types de convoi mentionnés plus haut, les convois de logistique et les convois d'attaque, correspondent à différentes combinaisons de ces mesures d'efficacité, soit respectivement une emphase sur la quantité de cargo transporté ou sur la vitesse d'exécution d'une mission.

Dans la littérature, une itération de coévolution consiste typiquement à simuler une grande quantité de scénarios tenant compte d'une variété d'ensembles de paramètres et à choisir le meilleur résultat [Choo *et al.*, 2007]. Ainsi, dans le présent contexte, des heuristiques permettront au convoi ainsi qu'aux insurgés d'évoluer de manière concurrente tout en minimisant la charge de calcul. De plus, des changements que l'on pourrait qualifier de « politiques » pourront être ajoutés dans les règles d'évolution afin de forcer les prises de décision de la part de l'utilisateur. Par exemple, il serait possible d'introduire une perturbation dans la règle d'évolution des insurgés afin de sortir la coévolution d'un point d'équilibre. Pour de plus amples détails, un rapport technique précisant l'implémentation des règles d'évolution dans IMAGEV0, le premier prototype du projet IMAGE, tant au niveau tactique qu'au niveau stratégique, a été rédigé à RDDC Valcartier [Bernier et Rioux, 2008].

Finalement, tout comme la simulation tactique, la simulation stratégique d'attaque de convoi militaire est un système complexe. En effet, des caractéristiques de systèmes complexes telles que la dépendance à l'historique sont facilement identifiables étant donné que les deux types d'agents évoluent séparément. Par contre, d'autres caractéristiques dépendent fortement de l'implantation des règles d'évolution associées au convoi et aux insurgés. Par exemple, il serait possible d'y introduire des délais ou des boucles de rétroaction, rendant la compréhension du scénario de simulation stratégique beaucoup plus difficile pour l'utilisateur. Des études pilotes impliquant des participants sauront trouver le bon niveau de complexité dans les règles d'évolution afin que les expérimentations sur les sujets humains soient valides et donnent des résultats significatifs.

6.4 Rôle de l'utilisateur et utilisations de Multichronia

Dans le cadre du projet IMAGE, le rôle de l'utilisateur est de tester des hypothèses sur un système complexe avec un outil d'exploration de simulations, d'explorer les résultats de simulation avec un logiciel de visualisation de données et de représenter la compréhension du système complexe avec un logiciel de représentation des connaissances. L'outil d'exploration de simulations est une spécialisation du cadre d'application Multichronia.

Les deux scénarios actuels (tactique et stratégique) peuvent être étudiés de trois façons dans Multichronia. Premièrement, l'utilisateur peut explorer des simulations strictement au niveau tactique (voir contexte tactique dans la Figure 6.3). Dans ce cas, il effectue l'allocation initiale des paramètres (SAT, UAT et disposition des insurgés sur le terrain) pour par la suite exécuter la simulation correspondante. Il répète ces opérations jusqu'à ce qu'il soit satisfait des résultats obtenus, c'est-à-dire qu'il ait bien compris comment se déroule le scénario. Dans ce contexte, Multichronia est utilisé strictement pour réaliser l'allocation initiale avant le début de la simulation. L'arbre multichronique correspondant à une utilisation de ce type est composé de nœuds issus de la racine représentant une simulation. Il n'est donc pas possible de créer des points de divergence en cours de simulation. Par contre, Multichronia permet de représenter dans l'arbre multichronique la valeur des paramètres pour chaque allocation et de contrôler les instances de simulations individuellement ou en les synchronisant.

Deuxièmement, l'utilisateur peut exécuter l'évolution associée au convoi manuellement (voir la Figure 6.3). Dans ce cas, il effectue l'allocation des paramètres du convoi (SAT et UAT) à chaque pas de temps et un algorithme fait de même pour le paramètre des insurgés. La décision de l'allocation des paramètres du convoi est basée sur la valeur des MoPs observées par l'utilisateur suite à une simulation tactique. L'évolution manuelle des paramètres du convoi lui permet donc de mieux comprendre la manière dont les insurgés évoluent par rapport à l'allocation des paramètres du convoi. Dans ce contexte, Multichronia peut être utilisé afin de prendre des décisions *pendant* l'exécution de la simulation. En effet, l'utilisateur peut à tout moment décider de tester une stratégie plutôt qu'une autre, sans vouloir perdre l'état initial de sa simulation. La fonctionnalité de création de point de divergence de l'arbre multichronique est donc parfaitement désignée pour remplir cette opération.

Troisièmement, l'utilisateur peut contrôler certains paramètres d'évolution du convoi afin de déterminer le résultat à long terme de la coévolution du convoi et des insurgés

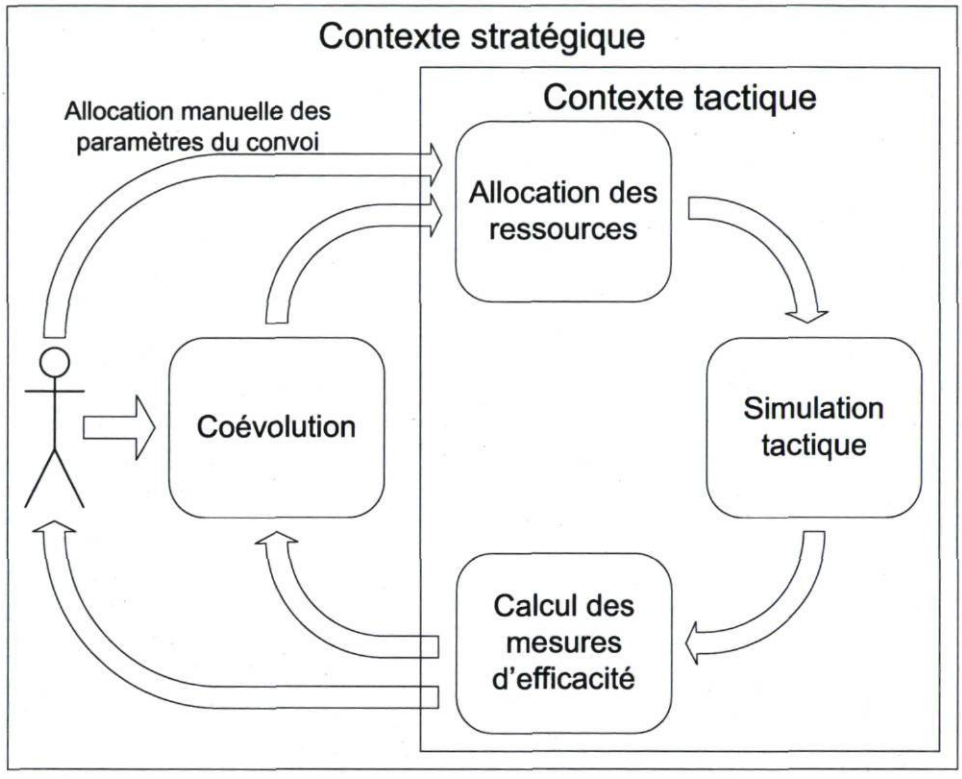


FIGURE 6.3 – Rôle de l'utilisateur dans le contexte de simulation stratégique, inspiré de [Bernier et Rioux, 2008]

(voir la Figure 6.3). Dans ce cas, l'utilisateur démarre une simulation qui tient compte d'une allocation initiale et il laisse aller la coévolution en y modifiant certains paramètres prédéterminés dans la règle d'évolution des agents formant le convoi. Son but est alors de comprendre l'effet du/des paramètres des règles d'évolution sur les MoEs/MoPs et de maintenir ces dernières à des niveaux optimaux. Autrement dit, il s'agit de prendre des décisions à des moments stratégiques afin de maximiser une fonction d'objectif. Les moments auxquels ces décisions sont prises sont totalement arbitraires. En effet, l'utilisateur, grâce à son intuition et son analyse de la situation, décide des moments propices auxquels une décision sur les paramètres d'évolution doit être prise. Il est important de noter que les méthodes formelles d'analyse telles que les designs d'expériences classiques ne peuvent être utilisées dans de telles situations en raison des décisions qui peuvent être prises à chaque itération. Le nombre de paramètres à considérer est ainsi multiplié par le nombre d'itérations d'une simulation, ce qui implique un trop grand nombre de combinaisons à simuler pour obtenir des résultats statistiquement valides. C'est pourquoi l'humain, qui possède une grande capacité d'analyse face à une situation d'une telle complexité, occupe une place très importante dans Multichronia. D'un autre côté, certains mécanismes expérimentaux seront mis en place afin de forcer un utilisateur à prendre des décisions, mais aussi afin de limiter la quantité de décisions prises par un utilisateur à un nombre raisonnable, pour imiter la prise de décision au niveau politique. Multichronia peut donc être utilisé afin de tester différentes stratégies et afin de manipuler les simulations dans le but d'en extraire des connaissances lorsque les données correspondantes sont représentées dans un logiciel de visualisation approprié.

Finalement, dans les deux contextes d'évolution, un utilisateur pourrait être intéressé par l'exécution de la simulation tactique correspondant à une itération donnée. En effet, à la fin de chaque itération de la simulation stratégique, des mesures d'efficacité sont disponibles dans les contextes d'évolution. Or, lorsque ces résultats ne correspondent pas aux attentes de l'utilisateur, il lui est possible de démarrer une simulation tactique qui tient compte de l'allocation de ressources de l'itération considérée afin de pouvoir étudier la dynamique des entités sur le terrain qui a mené au résultat. Multichronia permet à l'utilisateur d'accomplir une telle opération et ainsi favorise l'utilisateur à se questionner sur les résultats qu'il obtient au niveau stratégique.

Chapitre 7

Résultats et discussion

Ce chapitre présente la mise en œuvre de Multichronia et son utilisation dans un contexte de simulation d'attaques de convois militaires par des insurgés. La mise en œuvre d'un tel cadre d'application nécessite premièrement le développement d'une architecture appropriée, adaptable à différents contextes d'application. Les principaux paquetages logiciels seront tout d'abord présentés ainsi que les besoins auxquels ils répondent et les différentes fonctionnalités supplémentaires qu'ils offrent. Par la suite, une implantation du cadre d'application Multichronia sera proposée, suivie de deux applications directes dont les contextes sont respectivement la simulation d'attaque de convois militaires aux niveaux tactique et stratégique, impliquant une utilisation de deux simulateurs différents. Finalement, des mesures qualitatives, qui viennent justifier des choix de design, seront présentées.

7.1 Architecture de base du cadre d'application Multichronia

La Figure 7.1 montre les principaux paquetages logiciels (*software packages*) formant le cadre d'application Multichronia ainsi que leurs liens de dépendance. Quelques-uns de ces paquetages contiennent un certain nombre de classes abstraites et/ou interfaces qui définissent les abstractions de base desquelles les classes formant une application concrète doivent hériter. Les autres paquetages contiennent des classes qui fournissent des méthodes associées à des fonctionnalités de l'arbre multichronique ou au pipeline de données. L'architecture de Multichronia correspond au patron de conception « modèle-vue-contrôleur » puisque l'interface graphique est séparée du modèle de données et du

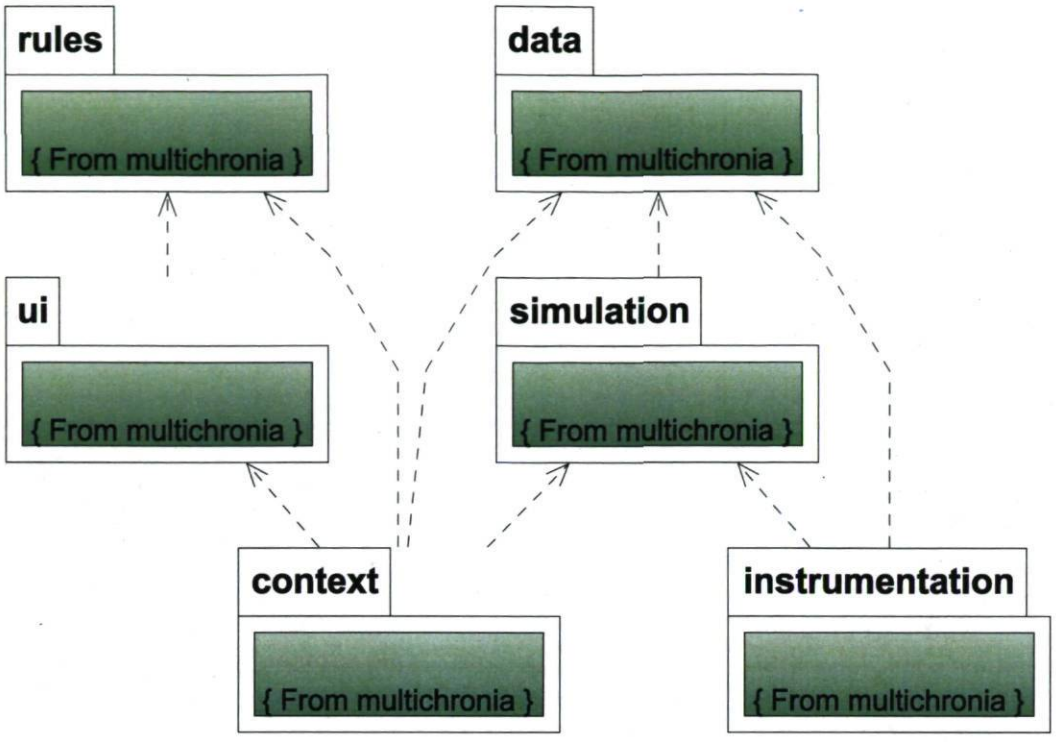


FIGURE 7.1 – Principaux paquetages logiciels du cadre d'application Multichronia et leurs dépendances

manipulateur de données [Reenskaug, 1979]. Toutes les fonctionnalités de Multichronia sont donc regroupées de façon logique par paquetage, et seront décrites dans les paragraphes suivants.

7.1.1 Paquetage « context »

Le paquetage « context » contient des classes de base dont le but est de fournir une interface appropriée afin d'exploiter les classes disponibles dans les autres paquetages. Dans Multichronia, un *contexte* est défini, comme étant « un usage spécifique de l'arbre multichronique et des autres composantes de Multichronia afin d'étudier un problème donné ». La mise en œuvre d'un contexte implique une organisation de l'interface utilisateur de Multichronia (e.g. barres de menus, barres d'outils, actions suite à un clic de souris), une gestion spécifique des données provenant des simulations et un contrôle du simulateur exploité. Le paquetage « context » offre donc un support pour le développement de ces fonctionnalités. De plus, il offre des mécanismes permettant la liaison de plusieurs contextes afin de favoriser l'échange de données et la communication inter-contextes.

7.1.2 Paquetage « ui »

Le paquetage « ui » offre des fonctionnalités relatives à l'affichage de l'interface utilisateur, particulièrement en ce qui a trait à l'affichage de l'arbre multichronique. En effet, ce paquetage rend disponible des classes permettant la réorganisation des nœuds d'un arbre multichronique de différentes manières. Notamment, on retrouve une classe qui dispose les points de divergence sur l'axe de référence en fonction de leur temps de création (*wall clock time*) et une autre qui les dispose en fonction d'une variable arbitraire provenant de la simulation. Il est également possible d'afficher plus d'un axe de référence à la fois afin de visualiser des données de simulation à la manière des coordonnées parallèles (*parallel coordinates*) [Inselberg et Dimsdale, 1990]. Cette technique permet de visualiser des données à plusieurs dimensions et d'observer, entre autres, des tendances de regroupement ou de dispersion. Les classes du paquetage « ui » fournissent également des méthodes facilitant la gestion de la construction d'un arbre multichronique ainsi que la définition du contenu visuel des nœuds qui le composent. Ces classes sont construites de manière à ce qu'il soit facile de modifier l'apparence des nœuds ou d'associer une représentation particulière à un nœud créé suite à une action donnée de l'utilisateur.

En plus de gérer l'affichage de l'interface utilisateur, les classes du paquetage « ui » s'occupent de gérer la réponse du logiciel aux actions posées par l'utilisateur. Typiquement, un utilisateur interagit via l'interface à l'aide de la souris et du clavier d'un ordinateur. Les déplacements de la souris doivent donc occasionner une réponse de l'interface qui correspond à ce à quoi l'utilisateur s'attend. Par exemple, lorsque le curseur de la souris se déplace sur un nœud de l'arbre, ce dernier est affiché en surbrillance, signifiant à l'utilisateur que des actions affectant ce nœud peuvent être exécutées. Les déplacements de la souris au-dessus des menus et des barres d'outils produisent le même effet. D'autre part, les actions exécutées suite à un clic de souris sont entièrement configurables en fonction des besoins de l'utilisateur et affectent toujours l'item situé en-dessous du curseur. Des touches de clavier peuvent également être associées au déclenchement d'actions. Les contextes concrets sont responsables de la définition de ces actions et de leur méthode de déclenchement. Finalement, le modèle flexible de gestion des actions de l'utilisateur permet également aux développeurs d'intégrer à l'interface usager des périphériques additionnels.

7.1.3 Paquetage « rules »

Le paquetage « rules » contient des classes dont le but est de gérer des actions ayant une influence sur l'état de Multichronia. En effet, l'exécution d'actions, que ce soit via l'interaction de l'utilisateur (règles manuelles) ou via des algorithmes de surveillance qui analysent les données provenant des simulations (règles automatiques), dépend de règles qui valident une condition avant d'activer une conclusion qui mène à l'exécution d'une action (voir le diagramme de séquence à la Figure 7.2). Un gestionnaire de règles sert à coordonner l'ensemble des mécanismes impliqués. Les actions sont diverses, configurables pour chaque contexte et peuvent tout autant influencer la lecture d'un cours d'action que modifier le contenu de l'arbre multichronique. Des données permettant de paramétrer une action lui sont passées sous la forme d'arguments nommés qui, par souci de flexibilité, peuvent prendre un type de données arbitraire. Ainsi, une action s'attend à recevoir un certain nombre d'arguments, chacun d'un type particulier. Certains sont obligatoires alors que d'autres sont optionnels. Le développeur d'une action est responsable du rassemblement des arguments nécessaires à l'exécution d'une action et de leur interprétation. D'autre part, les conclusions peuvent être concaténées, rendant possible l'exécution de plusieurs actions à partir d'une seule condition (voir le diagramme de séquence à la Figure 7.2). Finalement, le paquetage « rules » de Multichronia offre un service d'enregistrement dans un historique des actions exécutées afin de rendre possible l'analyse *a posteriori* du travail d'un utilisateur.

7.1.4 Paquetage « simulation »

Le paquetage « simulation » contient des classes dont les fonctionnalités principales sont les suivantes : démarrage et gestion des simulations, synchronisation des cours d'action, gestion du temps processeur pendant l'exécution des simulations et enregistrement des données de simulation reliées à un cours d'action. Avant toute chose, le paquetage « simulation » contient une classe abstraite qui propose des fonctionnalités qu'un simulateur exploité par Multichronia doit implanter ou émuler. Ces dernières sont présentées dans le Tableau 7.1 et accompagnées d'une description de leur objectif. Il est important de noter que les fonctionnalités énumérées ne sont pas toutes obligatoires ; par contre, un simulateur qui les implante toutes sera facilement exploité par Multichronia et un utilisateur pourra profiter de la gamme complète des options offertes par l'arbre multichronique.

Le démarrage et la gestion des simulations s'effectuent différemment pour chaque type de simulateur. En effet, certains possèdent des interfaces de programmation et

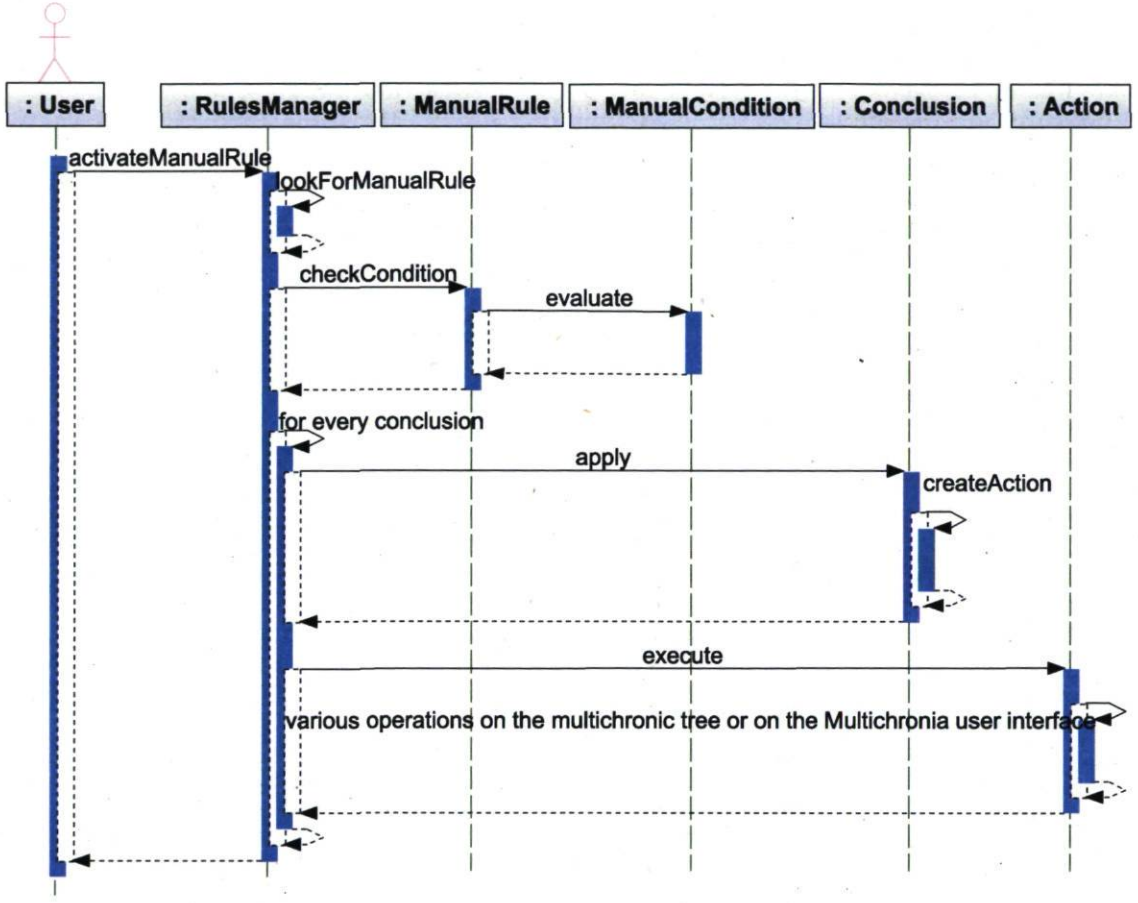


FIGURE 7.2 – Diagramme de séquence d'une interaction de l'utilisateur menant à l'exécution d'une action dans Multichronia

TABLEAU 7.1 – Description des fonctionnalités qu'un simulateur exploité par Multichronia doit implanter ou émuler

Nom de la fonctionnalité	**Paramètre**	**Description**
« start »	-	Démarre une instance de simulateur
« kill »	-	Met fin à une instance de simulateur
« stop »	-	Termine l'exécution d'un noyau de simulation
« pause »	-	Interrompt l'exécution d'une simulation
« resume »	-	Reprend l'exécution d'une simulation ayant été préalablement interrompue
« step »	-	Avance d'un pas de temps et par la suite interrompt l'exécution d'une simulation
« setTime »	# du pas de temps	Charge le pas de temps de simulation dont le numéro correspond au paramètre fourni en argument
« load »	Nom du fichier	Charge le fichier fourni en argument dans le noyau de simulation
« save »	Nom du fichier	Sauvegarde l'état d'une simulation dans le fichier fourni en argument
« sendCommand »	Variable	Exécute des commandes particulières à chaque type de simulateur

peuvent donc être utilisés comme des librairies intégrées dans Multichronia. Par contre, d'autres sont moins « ouverts » et nécessitent le démarrage d'un nouveau processus pour chaque instance de simulation. Multichronia implante donc un service générique qui gère les nouvelles instances de simulations en les enregistrant dans une liste de simulations actives, disponible pour chaque contexte. La communication entre les instances de simulateurs et Multichronia s'effectue à l'aide d'un protocole réseau TCP/IP.

Le service de synchronisation offert par le paquetage « simulation » offre la possibilité aux contextes faisant partie de Multichronia de synchroniser deux ou plusieurs nœuds de simulation, formant ainsi un *groupe de synchronisation.* La *synchronisation* de plusieurs nœuds de simulation implique que les actions portées sur l'un seront automatiquement appliquées sur l'autre. Par exemple, un groupe de synchronisation peut contenir trois nœuds de simulation. Le démarrage de la lecture du cours d'action de l'un de ces nœuds démarrera automatiquement la lecture du cours d'action des deux autres. Il en est de même pour les opérations de pause de la lecture d'un cours d'action et de déplacement d'un nœud de simulation par rapport à l'axe de référence.

Pour sa part, le service de gestion du temps processeur pendant l'exécution des simulations permet à toutes celles n'ayant pas encore atteint un état final de partager le temps processeur disponible sur un ordinateur entre elles le plus également possible. En effet, le but de ce service est d'assurer que, par souci de maintien de la réactivité de l'interface utilisateur, tous les simulateurs rendent disponible, dans une mémoire tampon, une quantité minimale de données en prévision de l'exécution par l'utilisateur d'une action d'avance rapide dans le temps de simulation. Les simulations faisant partie de l'arbre multichronique et n'ayant pas encore atteint un état final sont donc classées selon l'ordre de priorité présenté dans le Tableau 7.2. Dans le cas où deux instances de simulation possèdent la même priorité, celle dont le curseur de la souris est la plus proche de sa représentation visuelle, ou celle possédant le plus faible nombre de pas de temps disponibles en mémoire tampon est placée devant dans la liste de priorité. Le nombre de processeurs disponibles pour l'exécution des simulations est égal au nombre de processeurs total présents sur un ordinateur moins un (ou un (1) si un seul processeur). Ainsi, Multichronia alloue en tout temps un processeur à l'interface graphique, afin qu'une bonne réactivité soit maintenue, tout en maximisant l'utilisation des autres processeurs pour l'exécution des modèles de simulation.

Le service d'enregistrement des données de simulation rend ces dernières disponibles lors de la lecture d'un cours d'action. En effet, après avoir été filtrées et transformées par les classes du paquetage « data », les données de simulation sont envoyées à une classe du paquetage « simulation » afin qu'elles soient stockées pour consultation future. Un nœud de données XML contient l'essentiel de l'information pertinente associée à chaque

TABLEAU 7.2 – Caractéristiques déterminant la priorité des exécutions de simulations du service de gestion du temps processeur

Priorité	Description
1	Simulation en cours de lecture et dont le nombre de pas de simulation disponibles est inférieur à un seuil critique
2	Simulation dont le nombre de pas de simulation disponibles est inférieur à un seuil critique
3	Le curseur de la souris est près de la représentation visuelle de la simulation et le nombre de pas de simulation disponibles est inférieur à un seuil non-critique
4	Le nombre de pas de simulation disponibles est inférieur à un seuil non critique
5	Autres cas

pas de temps de simulation. De plus, la lecture et l'écriture de ces données peuvent se dérouler de manière asynchrone ; pendant qu'une simulation génère des données qui sont envoyées vers Multichronia puis enregistrées, le module de lecture peut récupérer et en traiter d'autres qui proviennent de pas de temps antérieurs. Le patron de conception « Observateur » (*Observer*) est employé afin qu'une classe de type « Simulation » puisse en tout temps connaître l'état de la classe de lecture associée. Par exemple, les calculs de la valeur des variables de référence se font en synchronisme avec la lecture des données afin que la représentation visuelle d'une simulation se positionne correctement par rapport à l'axe de référence.

Finalement, le paquetage « Simulation » comprend un gestionnaire de l'historique des paramètres de simulation qui ont été variés. En effet, il est essentiel de garder une trace de ces paramètres et de leur valeur afin de connaître la *provenance* des résultats finaux et intermédiaires d'une simulation [Anderson *et al.*, 2008]. Les paramètres qui sont variés par un utilisateur et leur emplacement dans le modèle des données sont donc encodés sous la forme suivante : leur emplacement est une chaîne de caractères « XPath » qui définit un chemin dans un document XML [Wikipedia, 2009e] et leur valeur est également une chaîne de caractères dans laquelle les données correspondantes sont encodées. À chaque variation de paramètre, une telle entrée est ajoutée à une liste chronologique des valeurs de paramètres. Donc, il est possible de connaître la valeur de tous les paramètres variés à chaque pas de temps de simulation. Cette information est d'ailleurs affichée dans l'arbre multichronique, à la demande de l'utilisateur, dans une infobulle.

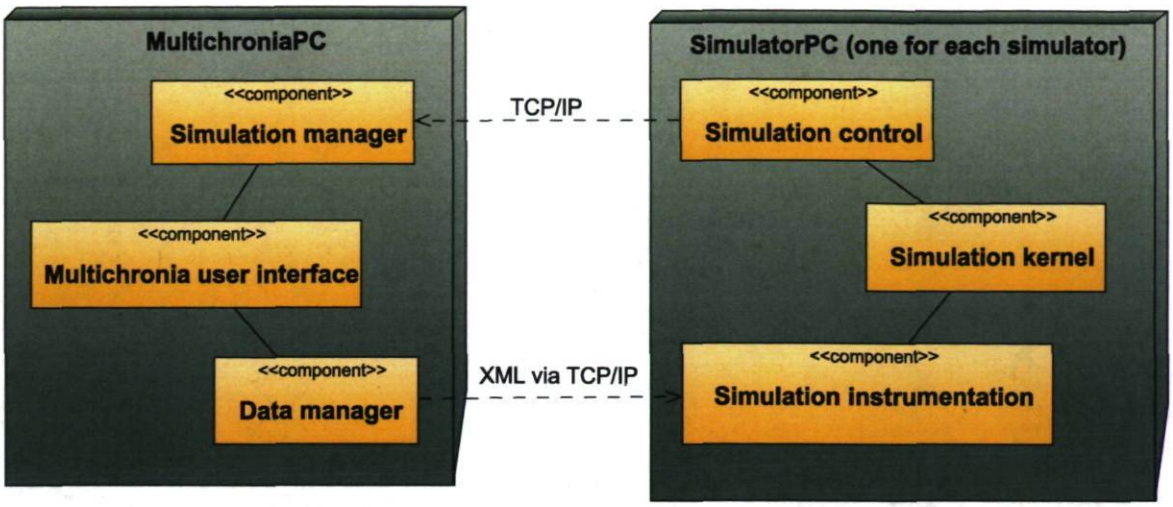


FIGURE 7.3 – Diagramme de déploiement de Multichronia et interconnexions. Le même ordinateur peut héberger Multichronia et les simulateurs

7.1.5 Paquetage « instrumentation »

Le paquetage « instrumentation » fournit des classes de base qui doivent être spécialisées pour tout simulateur exploité par Multichronia. Afin de clarifier la répartition des éléments exposés jusqu'à maintenant, la Figure 7.3 montre le diagramme de déploiement des composantes de Multichronia. Il est clair que les instances de classes des paquetages présentés plus haut sont toutes situées sur l'ordinateur qui affiche l'interface utilisateur de Multichronia. Par contre, les composantes associées au paquetage « instrumentation » sont situées sur le même ordinateur qui exécute l'engin de simulation. Il est important de noter que Multichronia et les instances de simulateurs peuvent être exécutés sur le même ordinateur si une ferme de calcul n'est pas disponible. Dans ce cas, un ordinateur possédant un processeur à plusieurs cœurs offrira une performance supérieure. Finalement, les liens TCP/IP existent pour chacune des instances de simulateurs.

La première classe du paquetage « instrumentation » qui mérite une attention particulière est celle qui amorce l'exécution du simulateur, que ce soit par l'intermédiaire d'une interface de programmation ou en démarrant un nouveau processus. La deuxième classe cruciale est celle qui établit le lien (e.g. lien réseau TCP/IP) entre l'instance du simulateur et Multichronia. Cette classe fournit des fonctionnalités telles que l'interprétation des commandes envoyées par Multichronia afin de contrôler l'exécution de la simulation. Ces dernières correspondent généralement à celles énumérées au Tableau 7.1 en plus des commandes spécifiques à chaque simulateur. De plus, la classe établissant

le lien entre Multichronia et un simulateur implante le patron de conception « Observateur » sur une instance de simulation afin de recueillir des informations sur son état courant (e.g. en pause ou en marche, pas de temps courant) et les envoyer à Multichronia.

L'instrumentation d'un simulateur comprend également un mécanisme de capture des données de simulation. Pour ce faire, tel que présenté conceptuellement au Chapitre 5, le modèle de données spécialisé pour le XML a été mis en œuvre. Dans ce modèle, les étapes du développement du schéma XML, de sa compilation et de la copie des données de l'état de la simulation vers les objets sérialisables sont spécifiques pour chaque simulateur. Par contre, la capture des objets sérialisables et leur envoi vers les classes du paquetage « data » de Multichronia sont des fonctionnalités génériques.

Une fonctionnalité du paquetage « instrumentation » qui ne fait toutefois pas partie du flux de données XML original est le filtrage des données de simulation pendant leur sérialisation en XML. En effet, certains simulateurs produisent une trop grande quantité de données par rapport à la capacité de sérialisation en XML des ordinateurs actuels. Il est donc justifié de permettre un filtrage de ces données afin de ne garder que celles qui sont d'intérêt pour l'analyse. La configuration de ce filtrage s'effectue dans un module de Multichronia dont l'interface est présentée à la Figure 7.4. L'arbre de gauche propose un schéma XML dans lequel l'utilisateur définit, en cochant les cases désirées, les éléments disponibles pour la sérialisation dans un fichier (*logged data*) alors que l'arbre de droite, un sous-arbre de celui de gauche, définit les éléments qui seront envoyés à Multichronia par le réseau (*output data*). La configuration du module de filtrage s'effectue peu après le démarrage du simulateur lors de la réception d'un flux XML en provenance du gestionnaire des simulations de Multichronia qui spécifie le sous-ensemble des données devant être enregistrées par le simulateur. À l'interne, le mécanisme de filtrage implique des modifications au processus de sérialisation des données en XML ; les éléments spécifiés par l'utilisateur sont considérés alors que les autres sont tout simplement ignorés lors de l'écriture d'un flot de données. La sérialisation des données en XML résultante est donc plus rapide et moins de bande passante est gaspillée par l'envoi de données inutiles.

Finalement, le paquetage « instrumentation » offre un mécanisme de *points de sauvegarde périodiques* afin de diminuer la quantité de données à simuler à nouveau, et par conséquent le temps nécessaire pour l'obtention de nouvelles données, lorsqu'un utilisateur crée un point de divergence. La Figure 7.5 montre le phénomène en question ; à un instant précis, un utilisateur décide de créer un nouveau point de divergence (moment « Interaction » = 15 secondes de temps d'exécution de simulation). Puisque la simulation est rendue à un temps ultérieur, il est impossible de tout simplement

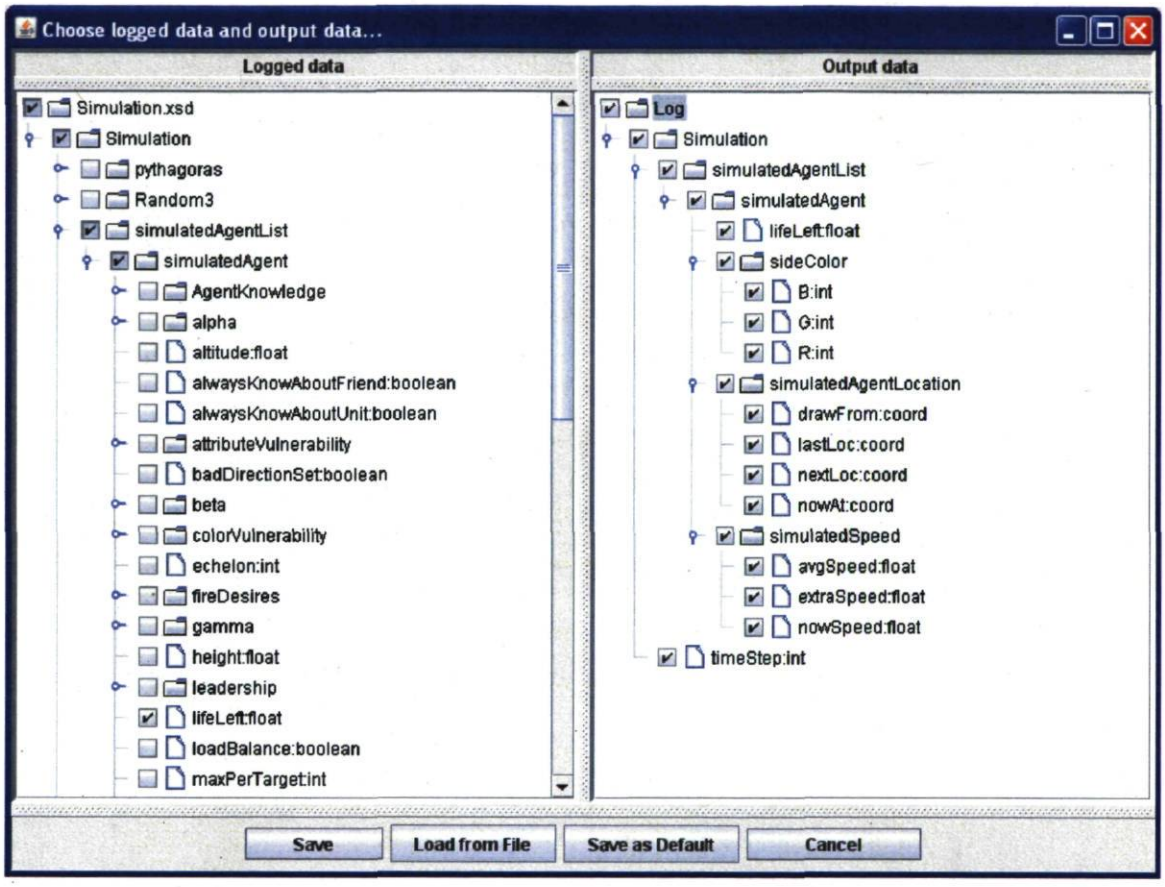


FIGURE 7.4 – Interface de configuration du filtre de données imposé au mécanisme de sérialisation d'un simulateur

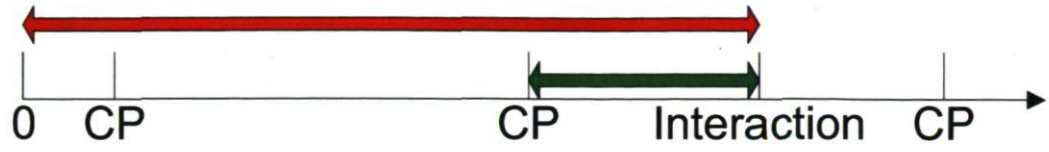


FIGURE 7.5 – Axe du temps montrant l'attente nécessaire pour l'obtention de nouvelles données suite à la création d'un point de divergence (moment « Interaction ») sans (en rouge) et avec (en vert) le mécanisme de points de sauvegarde périodiques (« CP »). La simulation est rendue à un point ultérieur

cloner la simulation actuelle. Il faut donc soit repartir du début et se rendre au point d'interaction (flèche rouge) ou encore repartir d'un point de sauvegarde et se rendre au point d'interaction (flèche verte). Dans le cas présent, l'utilisateur aurait à attendre 15 secondes, soit le temps nécessaire pour exécuter à nouveau les modèles à partir de zéro (0) jusqu'au point « Interaction » (flèche rouge). Par contre, si on suppose qu'un point de sauvegarde est généré à toutes les 1.5 secondes, l'utilisateur n'aurait qu'à attendre au maximum 1.5 secondes (flèche verte), plus le temps nécessaire au clonage, pour que des données à jour soient disponibles, ce qui correspond davantage aux spécifications d'un système interactif [Kalawsky et Nee, 2004].

7.1.6 Paquetage « data »

Les classes du paquetage « data » établissent le lien entre un simulateur et un système de visualisation tout en transformant les données qui y transitent. Tel que proposé au Chapitre 5, le XML est utilisé comme format de données standard. Un *flot de données XML* est un flux de données continu qui débute par un élément racine de type « SimulationLog » suivi par une série d'éléments XML correspondant chacun aux données ayant été retenues par les classes du paquetage « instrumentation » pour un pas de temps de simulation.

La Figure 7.6 montre les classes en interaction lors de la connexion du module « data » d'une instance de simulateur au gestionnaire des données entrantes, un serveur TCP/IP qui fait partie de Multichronia. Il est important de noter que, comme le montre le diagramme de déploiement de la Figure 7.3, l'instance de la classe « Simulation » n'est pas située dans le même processus que les deux autres. Par ailleurs, l'appel « Connect » établit une connexion TCP/IP qui transportera le flot de données XML du simulateur vers le consommateur de données. La création de ce dernier démarre un fil d'exécution (*thread*) qui accepte en entrée un flot de données XML. Le premier nœud de ce flot

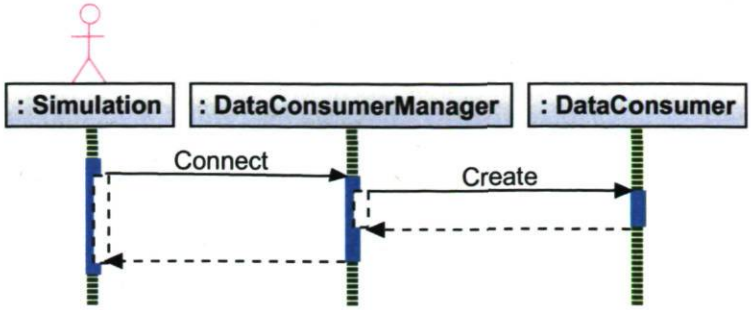


FIGURE 7.6 – Diagramme de séquence montrant une instance de simulation qui se connecte au gestionnaire de données entrantes

de données contient des informations permettant la configuration du consommateur de données et l'association des structures de données de gestion de Multichronia à l'instance de simulation correspondante ainsi qu'à son historique.

La Figure 7.7 montre la séquence des appels effectués par le consommateur de données après la réception de données XML provenant d'une simulation. Avant d'être ajoutées à l'historique de la simulation, les données sont tout d'abord filtrées, si nécessaire, par des objets de type « DataFilter » afin de n'en garder qu'un sous-ensemble ou d'ajouter des informations supplémentaires. Par exemple, une classe de type « DataFilter » pourrait calculer des mesures de performance à partir des données de simulation et les ajouter au flot de données XML, pour utilisation future par le module de visualisation. Ces calculs sont typiquement effectués par des requêtes XQuery, par des transformations XSLT ou par des classes correspondant aux besoins d'un utilisateur. De plus, plusieurs filtres peuvent être exécutés successivement sur les données de simulation entrantes. L'étape suivante consiste à exécuter la méthode « GetOutput » sur les instances de la classe « DataOutput ». Cette méthode retourne des métadonnées qui, bien qu'associées à une simulation, ne dépendent pas du flot de données qui en provient. Par exemple, une classe dérivée de « DataOutput » pourrait ajouter au flot de données un nœud énumérant les paramètres de simulation qui ont été variés par l'utilisateur lors de la création d'un point de divergence. Finalement, la dernière étape suivant la réception des données de simulation consiste à ajouter les nœuds XML filtrés et augmentés de métadonnées à l'historique de simulation. Les données comprises dans cet historique seront récupérées dans le futur suite à une intervention de l'utilisateur et envoyées aux modules qui suivent dans le pipeline de données pour analyse.

Une étape préalable à l'envoi des données de simulation aux modules d'analyse est un traitement dont les résultats dépendent à la fois des données de simulation récupérées de l'historique et des actions de l'utilisateur. La Figure 7.8 montre la séquence des appels et les classes qui interagissent lors de cette dernière étape. Les données

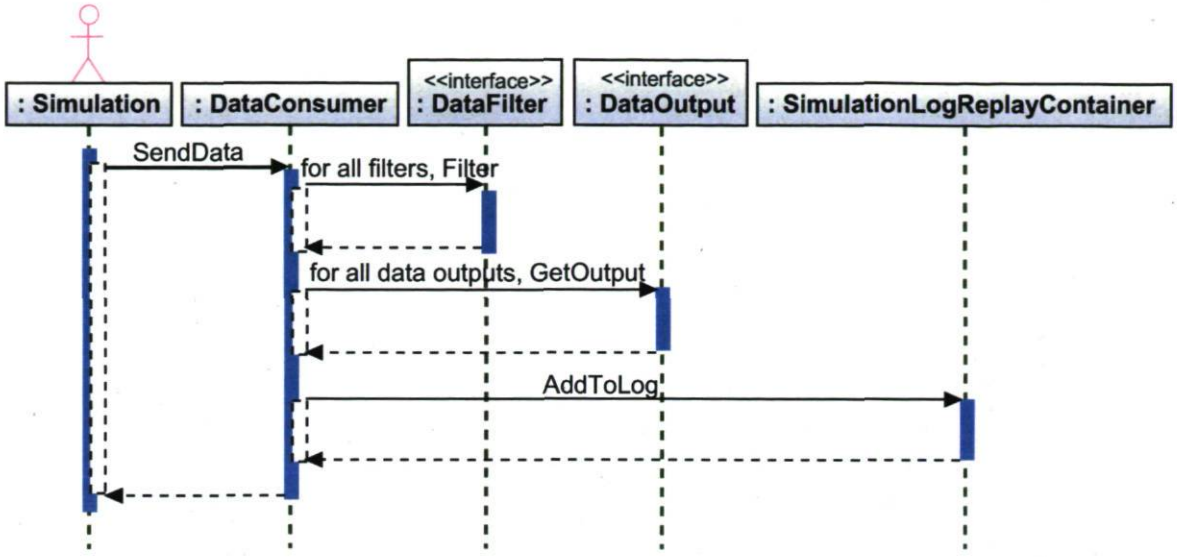


FIGURE 7.7 – Diagramme de séquence montrant les traitements possibles lors de l'envoi de données à partir d'une instance de simulation

sont à nouveau envoyées au consommateur de données parce que ce dernier possède une liste de répartiteurs de résultats (« ResultsDispatcher ») et de gestionnaires de données (« DataHandler ») pour lesquels les méthodes « Dispatch » et « HandleData » sont appelées respectivement. Les objets de type « ResultsDispatcher » permettent l'exécution d'une XQuery sur les données XML récupérées de l'historique de simulation et l'appel d'une méthode qui tient compte du résultat obtenu. Cette méthode, exécutée pour chaque pas de temps extrait de l'historique de simulation, tient compte à la fois des données de simulation et d'autres facteurs faisant partie de l'interface utilisateur de Multichronia. Par exemple, une condition associée à une règle automatique peut être évaluée à « vrai » seulement pour certaines circonstances déterminées par l'état de lecture d'un cours d'action associé à une simulation. Ce dernier exemple justifie donc que cette condition soit évaluée pour chaque pas de temps de simulation relu. Par ailleurs, les objets de type « DataHandler » opèrent de la même manière que les objets de type « ResultsDispatcher », à l'exception de la méthode de traitement des données. En effet, les requêtes ne sont pas limitées aux XQuery ; l'implantation d'un « DataHandler » est par conséquent responsable de la récupération de données en provenance du flot XML et de leur traitement. L'étape finale, suite à la lecture, consiste à envoyer les données correspondant au pas de temps de simulation en cours au module de visualisation et aux autres modules d'analyse.

Les paragraphes précédents justifient le choix du XML pour la transmission des données provenant des simulateurs vers le module de visualisation. En effet, des fonctionnalités essentielles telles que le filtrage de données et des requêtes sur les données

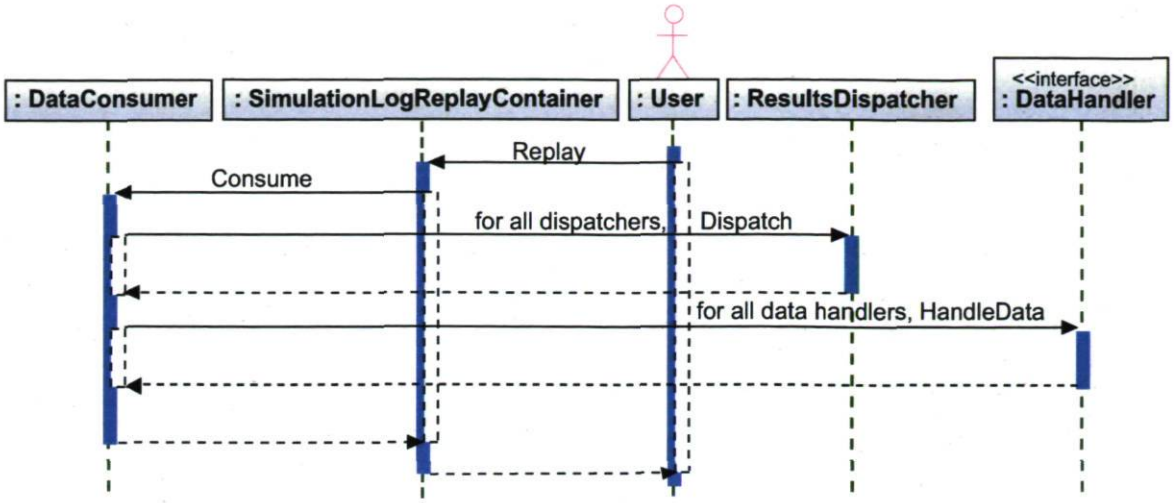


FIGURE 7.8 – Diagramme de séquence montrant les traitements possibles lors de la lecture d'un cours d'action, opération qui est activée par l'utilisateur

entrantes sont possibles avec les outils standards du XML comme le XQuery et le XSLT. De plus, il est trivial d'ajouter des métadonnées ou de modifier les flots de données entrants. Finalement, le XML se stocke facilement soit en mémoire, soit dans un fichier.

7.2 Implantation de Multichronia

Un prototype de Multichronia a été implanté en Java™, un langage de programmation orienté objet très facile d'utilisation de par la disponibilité des outils de développement et des librairies nécessaires pour la mise en œuvre de logiciels sophistiqués. L'environnement de développement Eclipse [Eclipse Foundation, 2009] a été utilisé puisqu'il offre plusieurs fonctionnalités facilitant le développement logiciel telles que la refonte des classes (*refactoring*), une interface de modélisation UML et des outils aidant à la mise en œuvre d'interfaces graphiques. Les librairies standard de Java incluent les fonctionnalités suivantes :

- Communication réseau TCP/IP (serveur et client).
- Affichage d'interfaces graphiques.
- Gestion de structures de données essentielles (e.g. listes, vecteurs, tableaux associatifs (*maps*), itérateurs).
- Gestion de plusieurs fils d'exécution (*multithreading*) et leur synchronisation.
- Disponibilité d'un analyseur de syntaxe XML.
- Récupération automatique de la mémoire (*garbage collection*).

Contrairement à d'autres langages de programmation très populaires comme le C++, le code généré par un compilateur Java n'est pas du code machine directement exécutable par un ordinateur, mais plutôt du code binaire (*bytecode*) exécuté par la machine virtuelle Java (*Java Virtual Machine*, JVM). Le rôle de la JVM est d'interpréter le code binaire et d'exécuter les fonctions correspondantes sur l'ordinateur hôte. Le code binaire Java est donc portable sur toutes les plateformes possédant une JVM. Typiquement, un programme Java nécessite une quantité de mémoire plus importante et un temps d'initialisation plus long que son équivalent C++. Par contre, ces facteurs ne sont pas critiques pour l'exécution de Multichronia. Les avantages au niveau de la facilité de développement et de la grande quantité d'outils/librairies disponibles ont fait en sorte que Java a été choisi comme langage de programmation lors du développement de Multichronia.

L'implantation des classes faisant partie de la plupart des paquetages ne nécessite pas de librairies particulières, mais exploite plutôt celles qui sont disponibles dans un environnement d'exécution standard de Java. Par contre, des fonctionnalités de la librairie Nux [LBL, 2008] ont été utilisées, notamment celles qui permettent l'exécution de requêtes XQuery simples et sur des flots de données, de même que celles qui permettent le stockage de documents XML dans un format en arbre efficace, XOM™. D'autre part, des fonctionnalités d'affichage de graphes de la librairie Prefuse ont été utilisées afin d'offrir aux utilisateurs une représentation visuelle d'un document XML [Heer *et al.*, 2005]. La Figure 7.9 montre un exemple d'une telle représentation visuelle sous la forme d'un arbre. Dans cet exemple, les nœuds « feuille » et les attributs nommés des éléments sont en rouge, signifiant qu'il est possible de modifier la valeur associée. Il est également important de noter que le chemin du nœud sélectionné jusqu'à la racine de l'arbre est affiché en surbrillance.

7.2.1 Affichage et interaction de l'arbre multichronique

L'affichage de l'arbre multichronique nécessite l'utilisation d'une librairie graphique avancée étant donné qu'il s'agit d'une représentation originale non disponible dans les librairies communes. Dans le prototype courant, l'environnement Processing a été utilisé pour sa flexibilité et sa facilité d'utilisation tant au niveau de l'affichage graphique que de la gestion des actions de l'utilisateur via la souris et le clavier de l'ordinateur [Reas et Fry, 2007; Processing, 2009]. En effet, Processing offre une interface de programmation complète permettant l'affichage de formes simples (e.g. lignes, courbes, carrés, chaînes de caractères) à l'aide de Java 2D™, Java 3D™ ou JOGL, une librairie Java de classes enveloppantes aux classes natives de OpenGL® [JOGL, 2009]. L'affichage JOGL a été préféré aux deux autres dans Multichronia parce que l'exécution des fonctions d'affi-

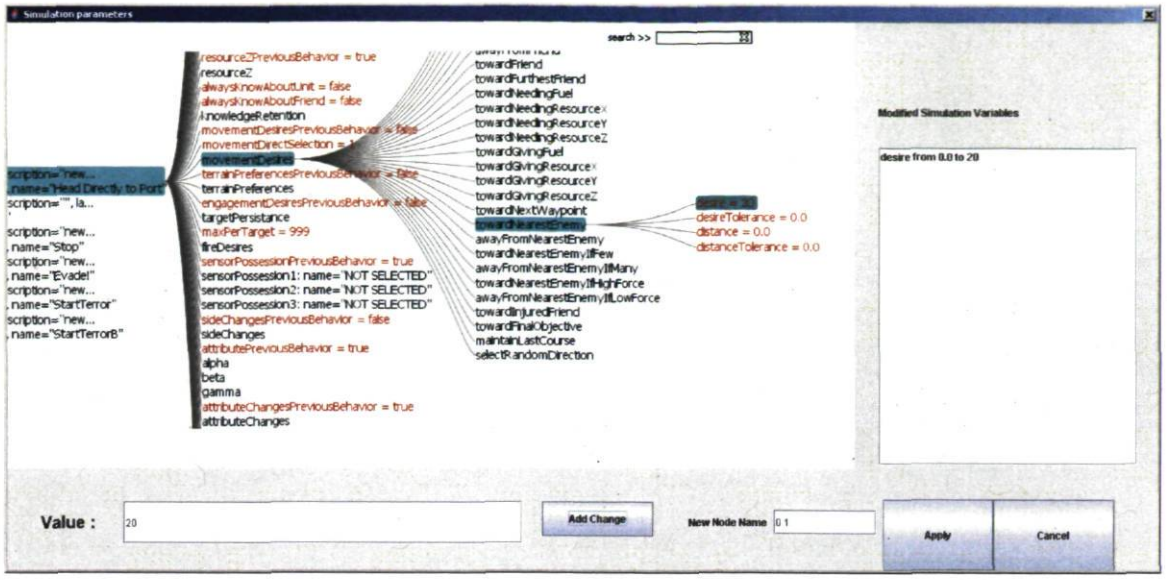


FIGURE 7.9 – Représentation visuelle d'un document XML incluse dans une boîte de dialogue permettant le changement de paramètres de simulation

chage se déroule beaucoup plus rapidement lorsque la quantité d'information est grande. De plus, OpenGL offre beaucoup plus de flexibilité pour les fonctionnalités graphiques 3D, notamment par l'utilisation de ses extensions et la possibilité de programmer des *shaders*.

La librairie Processing possède toutefois un point faible au niveau de l'affichage de ses interfaces graphiques. En effet, elle ne fournit pas de méthodes permettant de facilement afficher des menus et des barres d'outils, qui sont des éléments essentiels pour que l'utilisateur ait accès aux fonctionnalités complètes de l'arbre multichronique. C'est pourquoi les éléments relatifs à l'interface graphique de la librairie PhyloWidget [Jordan et Piel, 2008], basée sur Processing, ont été intégrés à Multichronia. Plus précisément, PhyloWidget fournit 1) des classes permettant l'affichage de barres de menus verticaux, de barres d'outils et de menus contextuels circulaires et 2) des classes remplissant les fonctions de gestion des événements provenant de la souris et du clavier. La Figure 7.10 montre une capture d'écran de Multichronia sur laquelle les fonctionnalités interactives sont mises en évidence. On y retrouve notamment les différents contrôles interactifs issus de la librairie PhyloWidget ainsi qu'un arbre multichronique affiché par Processing via OpenGL.

Les fonctionnalités interactives de Multichronia sont accessibles via des menus, des touches sur le clavier ou par manipulation directe des éléments graphiques de l'arbre multichronique. Par exemple, le menu vertical montré à la Figure 7.10 offre plusieurs

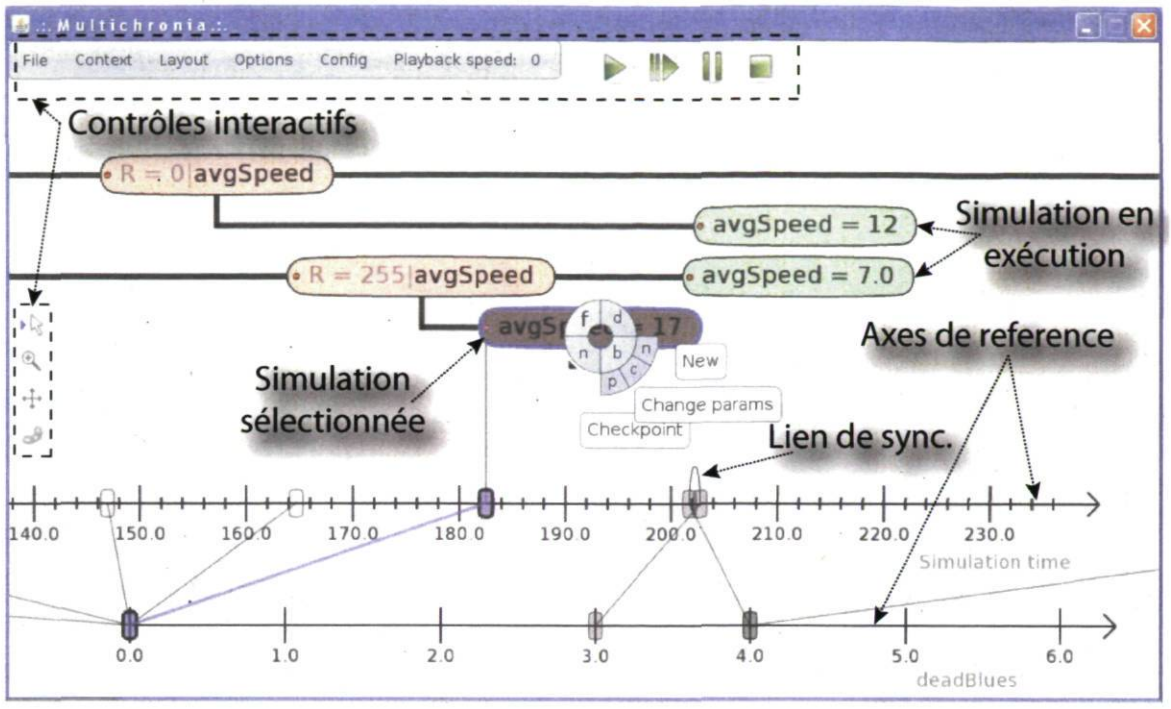


FIGURE 7.10 – Capture d'écran de Multichronia montrant les fonctionnalités interactives offertes à l'utilisateur pour le contexte de simulation tactique

fonctionnalités regroupées par thèmes. Le menu « File » offre les fonctionnalités de sauvegarder un espace de travail dans un fichier, d'en recharger un à partir d'un fichier, de démarrer/arrêter l'enregistrement des interactions de l'utilisateur et de quitter le logiciel. Pour sa part, le menu « Context » permet le changement de contexte. Ces deux derniers menus sont présents dans tous les contextes ; les autres sont spécifiques. Par exemple, pour le contexte montré à la Figure 7.10, le menu « Layout » permet de changer la disposition des nœuds dans l'arbre multichronique. D'autre part, la barre d'outils supérieure permet de contrôler la lecture d'un cours d'action. Un utilisateur devrait reconnaître les icônes et les employer de manière intuitive. Pour ce qui est de la barre d'outils située à gauche de la Figure 7.10, elle offre des fonctionnalités telles que la manipulation directe des nœuds de l'arbre, la possibilité de zoomer vers gros plan/vers plan général, de déplacer l'ensemble de l'arbre à droite ou à gauche et d'établir des liens de synchronisation entre des nœuds de simulation. Finalement, la capture d'écran de la Figure 7.10 montre un arbre multichronique possédant deux axes de référence, soit un indiquant le temps de simulation et l'autre le nombre d'agents bleus tombés au combat (« deadBlues »). Il est important de noter que les nœuds de l'arbre multichronique sont alignés par rapport au point rouge situé à leur extrême gauche. Le positionnement du pointeur de la souris sur un nœud indique clairement cet alignement et met en évidence les valeurs correspondantes sur les axes de référence.

7.2.2 Exploitation des processeurs multi-cœurs

Puisque Multichronia est un logiciel interactif, il est impératif qu'il demeure réactif suite aux actions de l'utilisateur et que ce dernier connaisse en tout temps l'état du logiciel. C'est pourquoi l'affichage de l'interface graphique s'effectue dans un fil d'exécution distinct du traitement des données et de la gestion des instances de simulation. En effet, l'affichage de l'arbre multichronique est effectué de manière asynchrone par rapport à la réception des données de simulation et à leur traitement. Ainsi, l'ensemble des éléments graphiques sont redessinés 30 fois par seconde, indépendamment de ce qui se produit dans les autres modules de Multichronia. De plus, le module de gestion du temps processeur attribue un processeur (partagé dans le cas d'un ordinateur muni d'un seul processeur) spécifiquement au rendu de l'interface. Les fils d'exécution, dont les caractéristiques sont résumées dans le Tableau 7.3, assurent que les services correspondants seront toujours disponibles pour les modules de Multichronia qui en ont besoin.

TABLEAU 7.3 – Principaux fils d'exécution de Multichronia et leur description

Nom du fil d'exécution	**Description**
EventLoop (« main »)	Capture les interactions de l'utilisateur et les envoie au gestionnaire de règles qui exécute les actions correspondantes. S'occupe aussi de la mise à jour de l'arbre multichronique et de son affichage à l'écran
ListeningServer	Fait partie du gestionnaire des simulations. Il agit comme serveur de connexions TCP/IP afin que les simulations nouvellement créées puissent établir le contact avec Multichronia
Reply-Commands	Fait partie du gestionnaire des simulations. Il envoie les paramètres initiaux aux nouvelles instances de simulation (e.g. leur nom, le schéma XML partiel de configuration des filtres de données)
Cluster-Manager	Établit une priorité pour toutes les exécutions de simulations n'ayant pas atteint l'état final et leur alloue du temps processeur en fonction de ces priorités
TooltipTimer	Gère l'affichage d'une infobulle dans l'arbre multichronique. Cette dernière apparait lorsque l'utilisateur maintient le pointeur de la souris fixe sur la représentation visuelle d'un nœud de simulation pendant un certain temps

Suite à la page suivante

TABLEAU 7.3 – suite de la page précédente

Nom du fil d'exécution	**Description**
DataConsumer-Manager	Agit comme serveur de connexions pour les flots de données de simulation. Il initialise le lien de données et démarre des instances de « DataConsumer »
DataConsumer	Reçoit les données d'une simulation, les filtre, les manipule, puis les envoie dans la classe d'historique correspondante
Results-Dispatcher	Répartit des données en provenance du consommateur de données de manière asynchrone. La répartition asynchrone vise à ne pas bloquer le fil d'exécution de réception des données.
Replay	Gère l'historique des données de la simulation correspondante et rend disponible à l'utilisateur des données archivées, à sa demande
Simulation	Gère une instance de simulation, du démarrage jusqu'au moment où l'utilisateur décide d'y mettre fin. Une nouvelle simulation peut soit être démarrée dans un nouveau processus, soit dans le fil d'exécution courant
Link	Fait le lien de contrôle entre Multichronia et une instance de simulation. Toutes les commandes de contrôle passent par ce fil d'exécution
Multichronia-Link	Établit le lien de contrôle avec l'instance de Multichronia. Situé au niveau du simulateur
Checkpoint-Thread	Effectue la sauvegarde ponctuelle de l'état de la simulation périodiquement. Situé au niveau du simulateur
Simulator	Exécute les modèles et envoie les données vers Multichronia. Situé au niveau du simulateur

La Figure 7.11 montre différents modules de Multichronia ainsi que les fils d'exécution qui les composent. De plus, elle montre la séquence des opérations et les différents échanges de données/contrôles entre les fils d'exécution suite à la création d'un point de divergence. Il est important de noter que l'instanciation d'une nouvelle simulation par le gestionnaire d'événements (étape 1) et le contrôle de l'historique d'une simulation sont activés par un utilisateur via l'interface utilisateur ou par une règle automatique. Les étapes 1 à 5, qui correspondent à l'initialisation complète du simulateur et de ses modules satellites, se déroulent successivement dès que la règle de création d'une nouvelle instance de simulateur est activée. D'autre part, les étapes 6 et 7 sont activées automatiquement, mais de manière asynchrone, lorsque le gestionnaire de temps processeur alloue du temps processeur au simulateur nouvellement instancié. L'envoi de la commande de démarrage au simulateur enclenche le reste du processus (étapes 8 et 9),

ce qui fait intervenir les modules de traitement du pipeline de données. Les étapes 8 et 9 sont donc exécutées jusqu'à l'atteinte d'un état final dans le scénario de simulation.

La Figure 7.11 montre aussi à quel point l'arbre multichronique est indépendant des autres modules au niveau de son affichage. En effet, l'instanciation d'une simulation et le contrôle de l'historique d'une simulation sont les deux seuls endroits où l'interface graphique doit attendre l'exécution d'un autre module avant de retourner à la boucle de rendu. Dans l'implantation courante, ces deux appels s'exécutent de manière asynchrone, ce qui règle le problème d'attente. De plus, des mécanismes de synchronisation des données ont été mis en place afin que le fil d'exécution de l'interface graphique n'ait à attendre qu'à de rares occasions, c'est-à-dire lors des mises-à-jour du module « TooltipTimer » ou de l'historique des simulations. Finalement, l'utilisation du langage de programmation Java a grandement facilité la mise en œuvre des mécanismes de synchronisation et des multiples fils d'exécution qui forment le noyau de Multichronia puisque ces fonctionnalités sont natives au langage.

7.2.3 Fonctionnalités additionnelles

Des fonctionnalités additionnelles ont été implantées dans Multichronia en réponse à certains besoins évoqués par des utilisateurs futurs ou afin de tester des alternatives/compléments au design proposé.

Intégration du périphérique de contrôle « SpacePilot™ » à Multichronia

Le périphérique de contrôle « SpacePilot » donne la possibilité à ses utilisateurs de manipuler un objet sur six degrés de liberté, en plus de mettre à leur disposition plus de 21 boutons programmables (Figure 7.12). Ce périphérique est généralement utilisé dans les logiciels de conception assistée par ordinateur afin de manipuler des objets en trois dimensions. Selon la compagnie 3DConnexion, qui distribue le produit, un utilisateur qui intègre le « SpacePilot » à son environnement de travail devrait pouvoir exécuter ses opérations usuelles en cliquant 50% moins souvent et en étant 20% plus productif dans l'accomplissement de sa tâche.

Le but de l'intégration du « SpacePilot » dans Multichronia est de permettre à l'utilisateur de contrôler l'arbre multichronique en utilisant ses deux mains, et de manière naturelle. La métaphore d'interaction est la suivante : le déplacement de la rondelle dans le plan déplace le curseur de la souris de la même manière. La sélection d'un

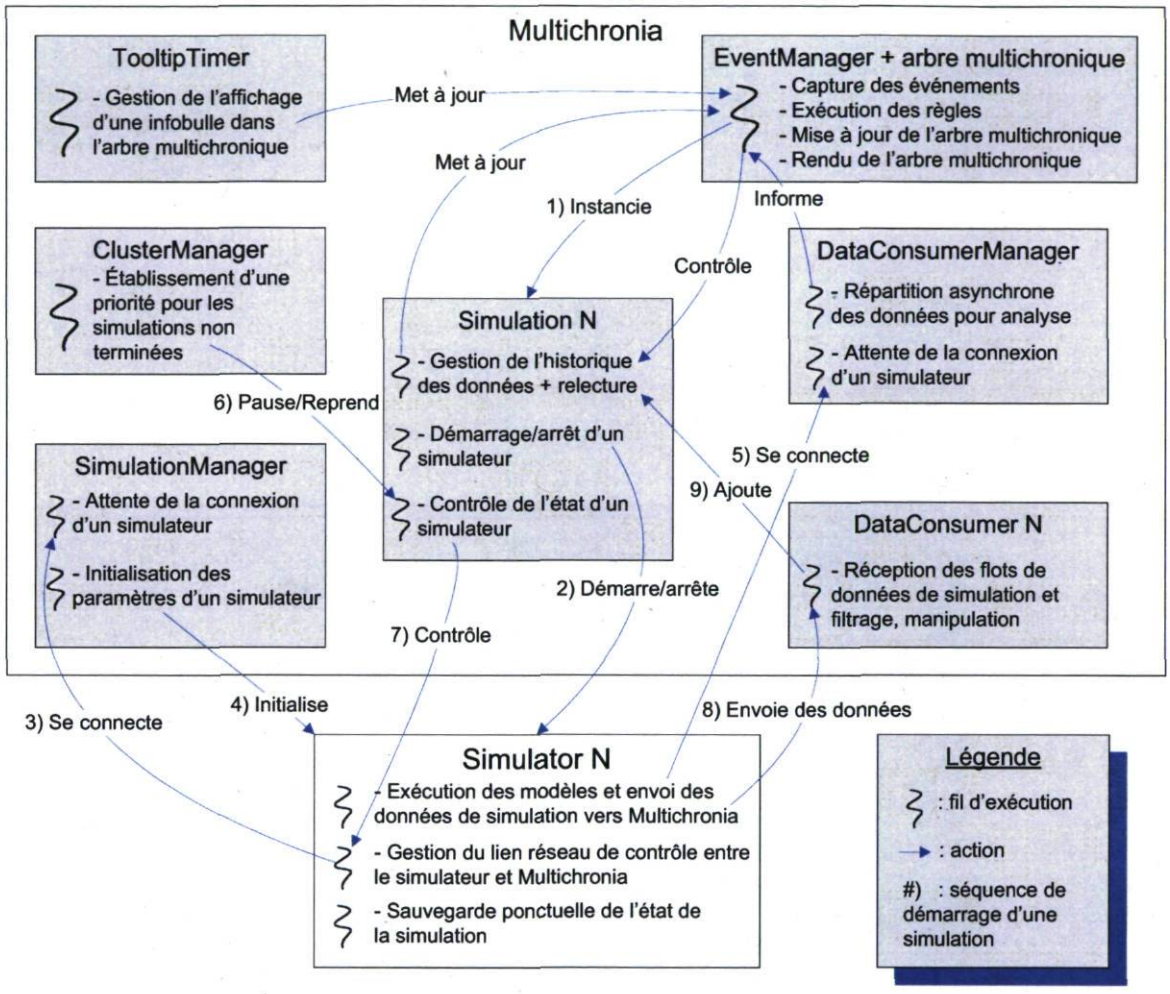


FIGURE 7.11 – Représentation des différents fils d'exécution faisant partie de Multichronia et leurs interactions. Les actions numérotées indiquent la séquence des actions accomplies à l'interne suite à la création d'un point de divergence

FIGURE 7.12 – Photo du « SpacePilot ». La rondelle sur le dessus du périphérique sert à manipuler des objets virtuels avec six degrés de liberté

nœud de simulation s'effectue avec un bouton. Le déplacement dans la hiérarchie de l'arbre s'effectue en basculant la rondelle vers le haut ou vers le bas, ce qui correspond respectivement à l'opération de sélection du nœud précédent ou du nœud suivant. La bascule de la rondelle vers la droite sur un nœud sélectionné démarre la lecture d'une simulation si cette dernière était dans l'état « pause », ou interrompt la lecture inverse d'une simulation. Similairement, le bascule de la rondelle vers la gauche sur un nœud sélectionné interrompt la lecture d'une simulation ou démarre sa lecture inverse. Finalement, la rotation de la rondelle vers la gauche ou vers la droite déplace un nœud de simulation par rapport à l'axe de référence respectivement vers la gauche ou vers la droite. Certains boutons sont dédiés à l'exécution de fonctionnalités interactives de l'arbre multichronique telles que la création d'une nouvelle branche de simulation ou le changement de l'outil courant.

Des résultats qualitatifs préliminaires montrent que certains utilisateurs préfèrent le « SpacePilot » à la souris pour le contrôle de l'arbre multichronique. Le nombre de clics nécessaires afin d'accéder à toutes les fonctionnalités associées est effectivement diminué parce que les opérations de sélection de nœuds de simulation, de démarrage/interruption de lecture et de déplacement d'un nœud par rapport à l'axe de référence s'effectuent strictement à l'aide du nouveau périphérique. Par contre, une étude quantitative sur plusieurs utilisateurs démontrerait si le « SpacePilot » offre réellement un gain d'efficacité lorsqu'utilisé dans Multichronia.

Sauvegarde des données de simulation dans une base de données SQL

La sauvegarde des données de simulation dans une base de données SQL a été implantée afin de permettre à des logiciels qui ne supportent pas le chargement de données XML de pouvoir afficher les données qui transitent par Multichronia. Le logiciel Tableau Software® en est un exemple typique ; il peut charger une base de données SQL dont les données pourront être visualisées [Tableau Software, 2009].

L'envoi de données de Multichronia vers une base de données SQL a facilement été implanté. Dans le cas présent, le serveur MySQL® a été choisi parce qu'il est gratuit et supporté par un grand nombre d'applications [MySQL, 2009]. De plus, un connecteur Java existe pour cette base de données. La traduction des données du XML vers le SQL est triviale (commande SQL « INSERT INTO »), mais il est essentiel de connaître le type des données afin de pouvoir créer une table SQL contenant des données du bon format (commande SQL « CREATE TABLE »). Dans le cas présent, les données ajoutées à la base de données SQL sont figées dans le code. Par contre, il existe des techniques permettant la création d'une table SQL à partir d'un schéma XML [Bourret, 2005]. L'implantation de ces techniques rendrait le processus de sauvegarde en SQL entièrement automatique.

Contrôle de Multichronia à distance

Dans l'optique d'intégrer Multichronia à un environnement de travail complet, il est essentiel de pouvoir exécuter certaines opérations sur Multichronia à distance. Par exemple, un utilisateur observant des données dans un logiciel de visualisation sur mesure devrait avoir la possibilité, par manipulation directe des objets faisant partie de son monde virtuel, de modifier des paramètres de simulation pendant l'exécution. Ce dernier exemple justifie donc la présence d'un lien de contrôle entre le logiciel de visualisation et Multichronia. Le format des commandes envoyées vers Multichronia à partir d'un module externe est le suivant : « `rule NomDeLaRègle ; NomParamètre1=ValeurParamètre1 ; ... ; NomParamètreN=ValeurParamètreN` ». Il est donc possible, par ce lien réseau, d'activer à distance une règle enregistrée dans Multichronia. Le lien inverse est également implanté, mais pas encore exploité par les logiciels utilisés.

7.3 Discussion sur les contextes de simulation implantés dans Multichronia

Le Chapitre 7 serait incomplet sans une discussion sur les deux principaux contextes de simulation qui ont été implantés lors du développement de Multichronia. Même s'ils sont tous les deux intégrés à la première phase d'expérimentation sur des sujets humains du projet IMAGE de RDDC Valcartier, ces deux scénarios exploitent des simulateurs totalement différents et nécessitent un traitement particulier. Par contre, l'intégration de ces deux simulateurs a été facilitée par le fait que Multichronia est un cadre d'application ; les classes de base n'ont qu'à être spécialisées, instanciées puis enregistrées dans les différents gestionnaires pour qu'un contexte soit fonctionnel.

7.3.1 Simulation tactique d'attaque de convois militaires

Le contexte de simulation tactique d'attaque de convois militaires correspond au scénario présenté à la Section 6.2. Le simulateur exploité est Pythagoras, un simulateur multiagent développé par Northrop Grumman pour le Naval Postgraduate School (NPS) [Bitinas, 2002]. Ce logiciel, disponible en sources ouvertes, est distribué gratuitement sur le site web du NPS [NPS, 2009]. Les modèles d'agents de Pythagoras offrent suffisamment de flexibilité pour permettre la modélisation précise du scénario de la Section 6.2. De plus, ils permettent d'enlever toute source de variabilité due aux phénomènes aléatoires, une condition essentielle à l'utilisation de Multichronia dans son état actuel. Pythagoras possède également une fenêtre de visualisation de l'état de la simulation montrant le terrain sur lequel les agents évoluent de même que l'état actuel de ceux-ci (voir Figure 7.13).

Modifications apportées à Pythagoras

Bien que Pythagoras réponde à plusieurs des conditions essentielles à son exploitation par Multichronia, certaines ne sont pas remplies. Notamment, il n'est pas possible d'effectuer un changement de paramètres direct, ni d'effectuer la sauvegarde ponctuelle de l'état d'une simulation. L'une de ces deux conditions doit toutefois être satisfaite pour que la fonctionnalité de création de nouveaux points de divergence à partir d'une simulation originale soit accessible. La méthode jugée la plus facile à implanter a été développée, soit la *sauvegarde ponctuelle de l'état d'une simulation.*

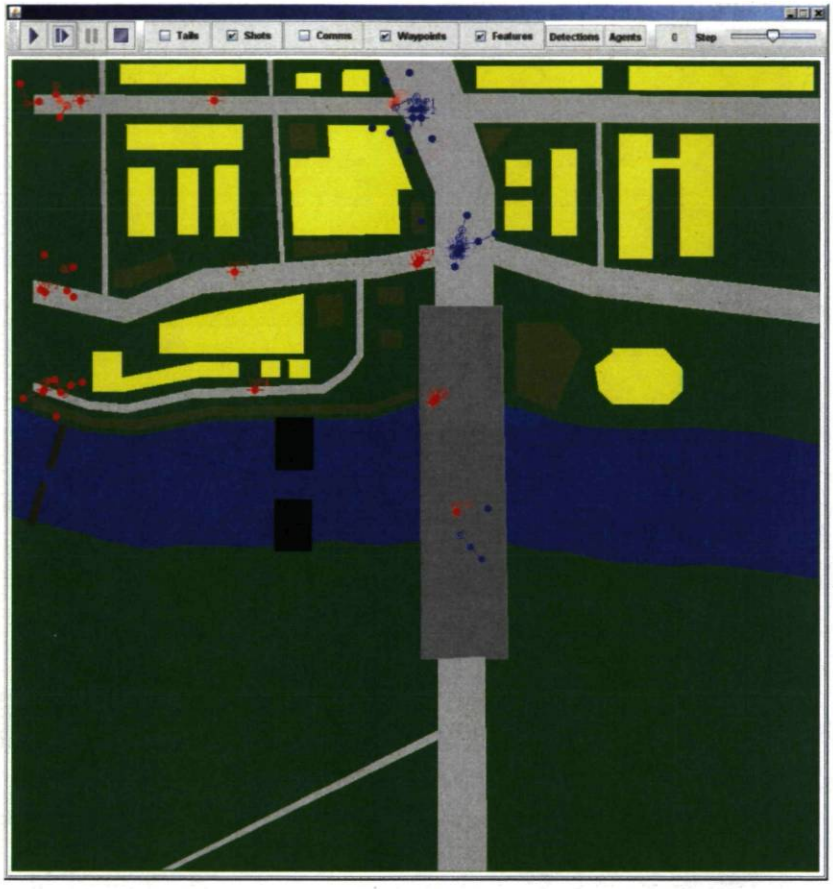


FIGURE 7.13 – Capture d'écran d'un scénario typique implanté dans Pythagoras. Les agents sont représentés par les pastilles bleues et rouges

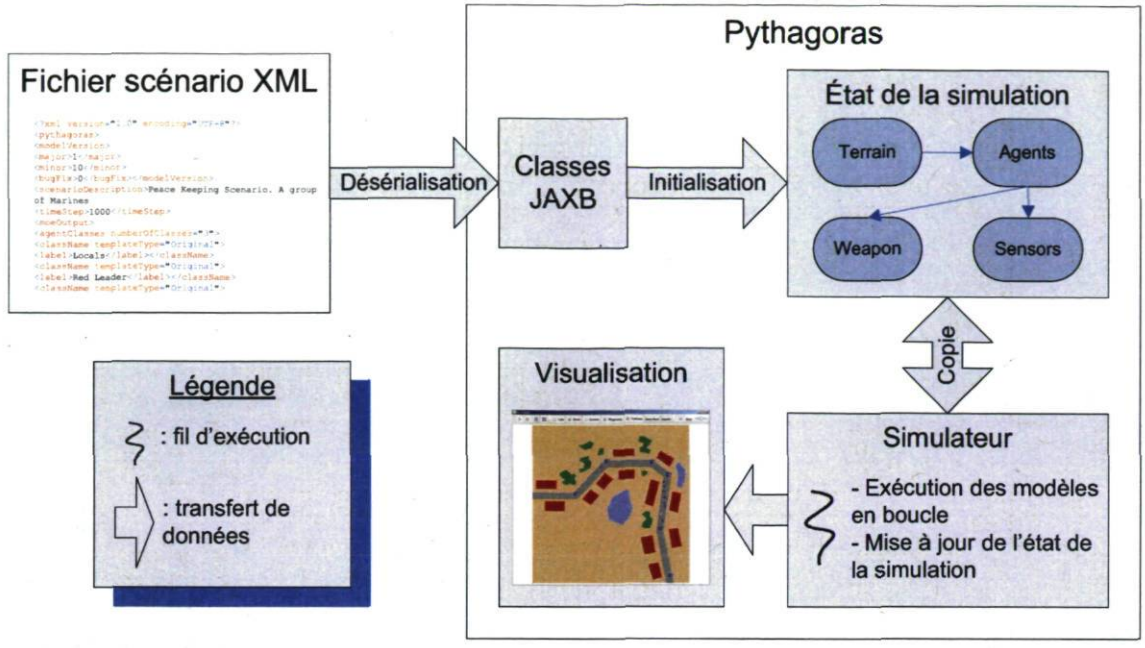


FIGURE 7.14 – Flot de données original dans Pythagoras

Comme le montre la Figure 7.14, les données correspondant à l'état de la simulation ne voyagent que dans un sens dans la version originale de Pythagoras, soit d'un fichier de scénario XML vers l'état de la simulation. Afin d'être compatible avec les fonctionnalités de l'arbre multichronique, ce cheminement des données doit être bidirectionnel. Le fait que Pythagoras utilise la technologie de liaison XML version Java (JAXB) pour charger un scénario de simulation a facilité l'implantation de la sauvegarde ponctuelle de l'état complet d'une simulation. En effet, conformément à la Section 5.3.2 qui porte sur une méthodologie de développement du pipeline de données basé sur le XML, la première étape est de compléter le schéma XML de la version originale de Pythagoras afin d'y inclure des structures de données visant à capturer l'état de la simulation pendant son exécution. Ces structures de données incluent notamment l'état individuel de chaque agent instancié dans le scénario, le pas de temps de simulation courant et l'état du générateur de nombres aléatoires qui, bien qu'inutile dans le scénario courant, peut devenir fort important pour des applications dans lesquelles des variables stochastiques sont impliquées. La deuxième étape de la méthodologie, qui consiste à effectuer la liaison XML à partir d'un schéma, a été accomplie à l'aide de la librairie JAXB [JAXB, 2009]. Finalement, la dernière étape menant à la sauvegarde ponctuelle d'une simulation est le développement des méthodes écrivant les données contenues dans les classes Java formant l'état de la simulation vers les classes Java générées par JAXB (méthodes « toXML »). Similairement, le chargement d'une simulation à partir des classes JAXB implique des méthodes transférant les données contenues dans les classes JAXB vers les classes formant l'état de la simulation (méthodes « fromXML »).

La Figure 7.15 montre le flot de données combiné de Pythagoras et Multichronia suite à l'ajout de la capacité de sauvegarder/charger l'état de la simulation. Les désérialisations d'un fichier de scénario XML et d'un fichier de sauvegarde XML s'effectuent dans des instances des mêmes classes de liaison JAXB, mais la sérialisation n'est seulement possible que pour un fichier de sauvegarde XML puisque l'initialisation de l'état de la simulation instancie des agents, chacun possédant des attributs distincts. Puisque Pythagoras ne permet pas le changement direct de paramètres, la modification des paramètres pendant l'exécution s'effectue en trois étapes : 1) Activé par une commande provenant de Multichronia, Pythagoras sérialise l'état complet de la simulation dans un fichier de sauvegarde XML, 2) Multichronia charge ce fichier dans une interface permettant à l'utilisateur d'avoir accès aux paramètres, de les modifier et d'enregistrer dans un nouveau fichier XML l'état de la simulation, incluant les modifications aux paramètres (voir Figure 7.9) et 3) Multichronia ordonne à Pythagoras de charger le nouveau fichier XML dans l'état de la simulation et de poursuivre l'exécution des modèles. Ce processus, essentiel pour l'utilisation de Pythagoras en mode interactif, a été testé par plusieurs utilisateurs. Il s'avère être très robuste et d'une grande flexibilité. Toutefois, quelques limitations y sont associées. Premièrement, il est impossible d'ajouter de nouvelles instances d'agents pendant l'exécution de la simulation. De plus, l'utilisateur est responsable de la validité des changements qu'il apporte au document XML, ce qui requiert de sa part une connaissance approfondie du schéma XML qui définit la structure d'un fichier de sauvegarde.

L'expérience pratique de la méthodologie de conception du pipeline de données XML a mis en évidence quelques leçons apprises sur la facilité avec laquelle elle peut être mise en œuvre avec un simulateur existant. Par exemple, la *disponibilité du code source* facilite grandement la mise en œuvre de la méthodologie, permettant ainsi au développeur d'application de modifier le simulateur à sa guise. Bien qu'une phase d'ingénierie inverse ait été nécessaire pour la compréhension du fonctionnement de Pythagoras, le développement des fonctionnalités manquantes aurait été beaucoup plus difficile si les sources avaient été fermées. D'autre part, *une ingénierie basée sur les modèles s'utilise plus facilement lorsque des modèles sont disponibles.* En effet, puisqu'aucun modèle UML n'était disponible avec la version originale de Pythagoras, l'avantage de construire un modèle pour ensuite générer un schéma XML complet, par rapport à modifier manuellement le schéma, n'était pas clair. C'est pourquoi cette dernière méthode a été adoptée. Finalement, *l'utilisation d'un type de données standard, le XML*, dans Pythagoras a grandement facilité son intégration dans le pipeline de données. D'autres leçons apprises à saveur plus technique sont décrites dans un article dédié entièrement au pipeline de données [Rioux *et al.*, 2008].

Pythagoras a également été modifié de manière à séparer le module de visualisation

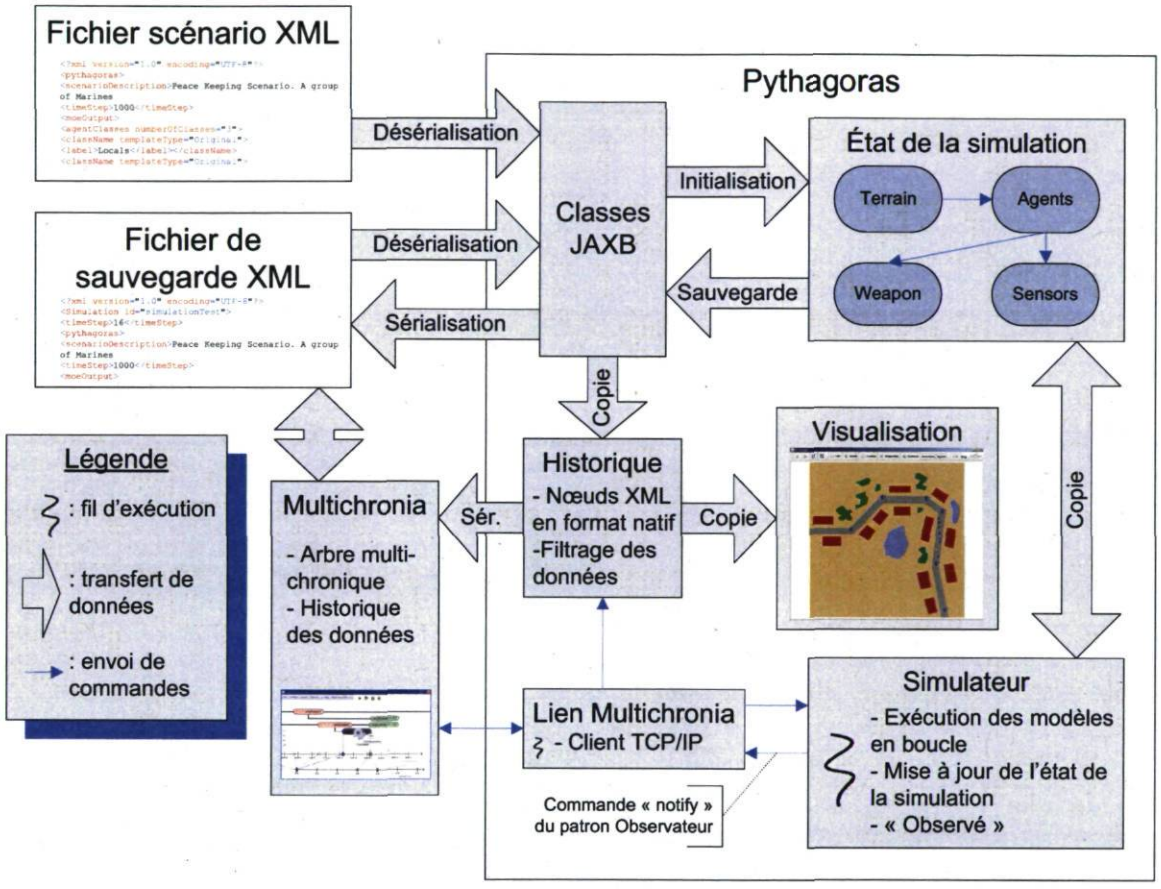


FIGURE 7.15 – Flot de données de Pythagoras qui inclut les modifications qui ont été apportées afin de l'exploiter dans Multichronia

de l'état de la simulation du noyau de simulation qui, lui, exécute les modèles et génère les données visualisées. En effet, ces deux modules étaient, dans la version originale du simulateur tel que montré à la Figure 7.14, indissociables ; les rendre indépendants permet au noyau de simuler en arrière plan des nouvelles données tandis que le module de visualisation affiche les données selon les besoins de l'utilisateur. L'absence d'un lien entre le simulateur et le module de visualisation à la Figure 7.15 montre que les deux modules sont indépendants. Afin de minimiser le transfert de données de Pythagoras vers Multichronia, les données de simulation sont enregistrées au niveau de Pythagoras, pour éventuellement être rechargées dans le module de visualisation, à la demande de l'utilisateur, via des commandes envoyées à l'historique de Pythagoras par le module « lien Multichronia ».

Finalement, Pythagoras a été instrumenté en y ajoutant des appels aux fonctions des classes du paquetage « instrumentation », présentées à la Section 7.1.5. Entre autres, les classes formant le noyau de simulation implantent la classe « observée » du patron de conception « Observateur » (voir Figure 7.15). De plus, des méthodes accessoires visant à démarrer/pauser l'exécution d'une simulation ont été ajoutées. Et enfin, à chaque pas de temps de simulation, le noyau du simulateur capture son état courant et l'envoie vers les classes du paquetage « instrumentation » qui se chargent par la suite de filtrer les données et de les envoyer vers Multichronia. L'instrumentation vise à être minimalement invasive afin que le moins de modifications possible aient à être implantées dans un simulateur existant.

Exploitation de Pythagoras dans l'arbre multichronique

Un certain nombre de classes de base du cadre d'application Multichronia ont dû être spécialisées afin que Pythagoras soit exploitable dans l'arbre multichronique. Premièrement, une classe définissant le contexte de scénario tactique Pythagoras a été développée. Cette classe définit principalement les différents manipulateurs de données, les variables de référence qui entrent en ligne de compte et les règles pouvant être activées par l'utilisateur ou par un mécanisme automatique. De plus, l'arbre multichronique est créé dans la méthode d'initialisation du contexte Pythagoras.

Deuxièmement, l'interface graphique entourant l'arbre multichronique est créée par une classe dont le but est la configuration des éléments graphiques associés à un contexte. Dans le cas présent, cette classe instancie notamment les menus contextuels circulaires qui sont disponibles lorsque l'utilisateur effectue un clic droit dans l'arbre multichronique. De plus, elle instancie des barres d'outils et un menu vertical permettant à l'utilisateur de choisir différentes options sur la disposition des nœuds de l'arbre

multichronique. Finalement, un gestionnaire des clics de souris et des touches de clavier enfoncées est démarré afin de gérer l'activation de règles suite à des événements prédéfinis. Tous ces objets, des instances de classes de Phylowidget, sont activées seulement dans le contexte de simulation tactique Pythagoras. Lorsqu'il y a changement de contexte, ils sont tout simplement désactivés afin de laisser la place aux fonctions graphiques du contexte alternatif.

Enfin, les classes relatives au contrôle de la simulation et de la réception des données ont été spécialisées. Une classe faisant le lien de commande avec Pythagoras a tout d'abord été implantée. Il s'agit d'un lien TCP/IP avec le « lien Multichronia » de la Figure 7.15. Les commandes standard y sont formulées, de même que des commandes spécifiques à Pythagoras (e.g. « focus » qui met le focus sur la simulation concernée, « name » qui ajuste le titre de la fenêtre de visualisation). Une seconde classe qui démarre une instance de Pythagoras a également vu le jour. Il est possible de charger Pythagoras en deux modes, soit dans un nouveau processus qui implique le démarrage d'une nouvelle machine virtuelle Java, soit dans le même processus que Multichronia. La première méthode consomme plus de mémoire, mais assure une indépendance au niveau des processus, ce qui signifie que si, pour une raison quelconque, Pythagoras rencontre des problèmes d'exécution, Multichronia va demeurer stable. Par contre, la deuxième méthode est très utile pour déverminer les interactions entre Multichronia et Pythagoras.

En résumé, bien qu'il ait été nécessaire de modifier quelques classes de Pythagoras pour arriver au but visé, l'exploitation de Pythagoras par Multichronia a facilement été rendue possible grâce à une architecture de classes générique laissant énormément de flexibilité au développeur. Le fait que Pythagoras utilisait déjà la technologie XML pour le chargement de scénarios a également facilité la tâche de développement. D'autre part, le premier prototype de Multichronia a été développé strictement pour être utilisé avec Pythagoras. La version présentée dans cette thèse est donc la deuxième itération du prototype et vise à être générique par rapport au simulateur utilisé. C'est pourquoi un deuxième contexte d'utilisation sera présenté afin de démontrer la flexibilité du logiciel.

7.3.2 Simulation stratégique d'attaque de convois militaires

Le contexte de simulation stratégique d'attaque de convois militaires a été développé sur mesure afin de répondre aux besoins de l'expérimentation V0 du projet IMAGE de RDDC Valcartier.

Simulateur d'attaque de convoi militaires au niveau stratégique

Le simulateur exploité dans ce contexte est le résultat du travail conjoint de Maya Heat Transfer Technology Ltd., une firme privée externe, et de RDDC Valcartier. Ce simulateur, programmé en Java, répond strictement aux besoins de simulation stratégique, ou de coévolution, présentés à la Section 6.3. Son architecture est montrée à la Figure 7.16. Une simulation stratégique consiste à faire coévoluer les membres du convoi et les insurgés de manière à ce que chacun s'adapte au comportement de l'autre. Les mesures de performance nécessaires pour l'adaptation sont évaluées suite à une simulation tactique de type Pythagoras. Par contre, le scénario considéré, qui comporte seulement trois paramètres discrets, a permis d'optimiser la vitesse d'exécution d'une coévolution. Plutôt que de simuler chaque cours d'action tactique, desquels seulement les mesures de performance sont intéressantes, le simulateur de convoi stratégique utilise des résultats pré-simulés et stockés dans une table de conversion (*look-up table*) qui lie l'état de la simulation à des mesures de performance (voir Figure 7.16). La table de conversion comprend une entrée pour chaque combinaison de paramètre, soit 726 au total. Cette optimisation permet donc d'accélérer l'exécution d'une simulation stratégique par un facteur 1000. Toutefois, cette technique est liée au scénario et serait difficilement utilisable pour des paramètres prenant des valeurs continues.

La récupération des données se fait de manière similaire à Pythagoras, c'est-à-dire que l'état de la simulation est sauvegardé dans des classes JAXB, puis sérialisé pour être passé par réseau à Multichronia. La quantité de données, par rapport à Pythagoras, est beaucoup moins grande. En conséquence, aucun mécanisme de filtrage n'est nécessaire pour le transport des données. D'autre part, le simulateur de convoi stratégique permet un changement direct des paramètres de simulation [Brooke *et al.*, 2003]. Autrement dit, des commandes prédéfinies, reçues par le simulateur via un contrôleur client, mènent directement à des changements de paramètres de coévolution, influençant ainsi le cours de la simulation du convoi stratégique. Bien qu'il soit plus simple à utiliser, ce mécanisme est beaucoup moins flexible que celui implanté dans Pythagoras. En effet, l'ajout de paramètres additionnels pouvant être modifiés dans la coévolution implique obligatoirement un supplément de code dans les classes qui font le lien entre Multichronia et le simulateur de convoi stratégique afin de gérer les commandes correspondantes. Par contre, l'intégration d'un tel mécanisme à Multichronia montre que le logiciel peut supporter les deux approches de modification de paramètres.

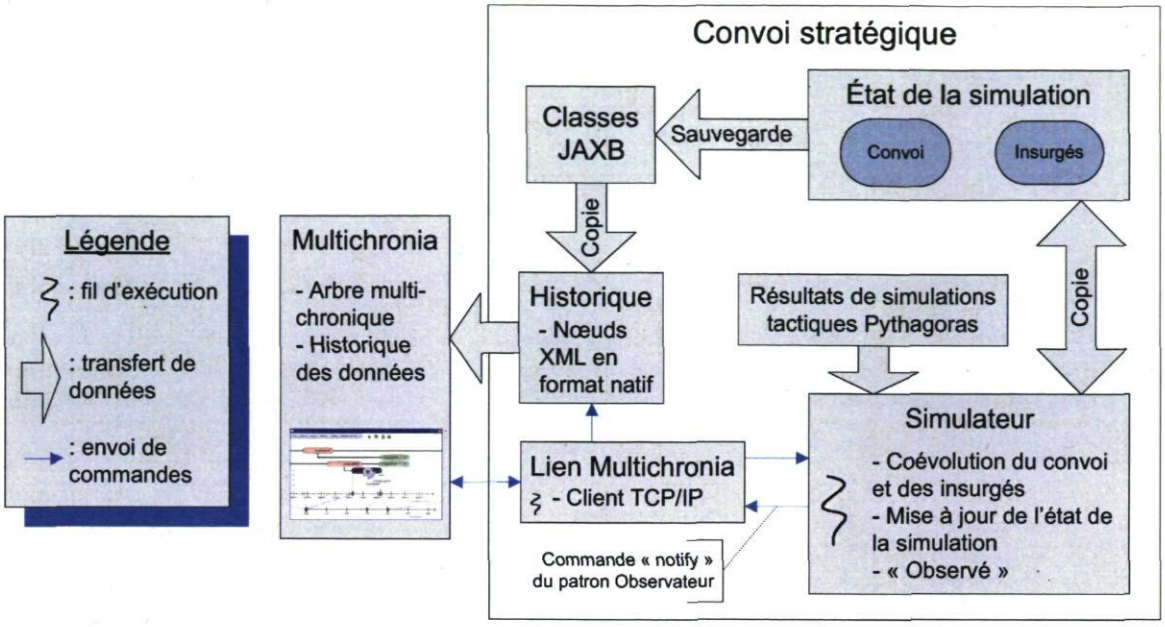


FIGURE 7.16 – Flot de données et architecture du simulateur de convoi stratégique

Exploitation du simulateur de convoi stratégique dans l'arbre multichronique

Le simulateur de convoi stratégique est utilisé pour implanter deux modes d'opération du scénario d'attaque de convois militaires par des insurgés, soit le mode purement stratégique, dans lequel l'utilisateur doit varier des paramètres de coévolution, et le mode opérationnel, dans lequel l'utilisateur doit réaliser l'allocation des ressources pour chaque pas de temps de simulation. Ces deux modes d'opération sont implantés dans le même contexte Multichronia. Comme pour le contexte de simulation tactique de Pythagoras, la classe maîtresse du contexte a d'abord été complétée ainsi que celle rendant disponibles les éléments de l'interface graphique. Les classes qui démarrent une simulation de convoi stratégique et celles qui établissent un lien réseau ont également été réalisées et intégrées à Multichronia en dérivant des mêmes classes de base que pour le scénario tactique de Pythagoras. Encore une fois, le simulateur de convoi stratégique est démarré dans un nouveau processus afin de minimiser le risque d'instabilité dans Multichronia. Au niveau de l'apparence de l'arbre multichronique il n'y a pas de différences majeures avec le contexte tactique, si ce n'est une signification différente des identificateurs présents à l'intérieur de chaque point de divergence et des paramètres supplémentaires dans les infobulles affichées à la demande de l'utilisateur. Comme le montre la Figure 7.17, les identificateurs consistent en un nom unique pour chaque simulation suivi d'un nombre représentant une quelconque variable. Ces identificateurs sont évidemment entièrement configurables et doivent être adaptés selon les commen-

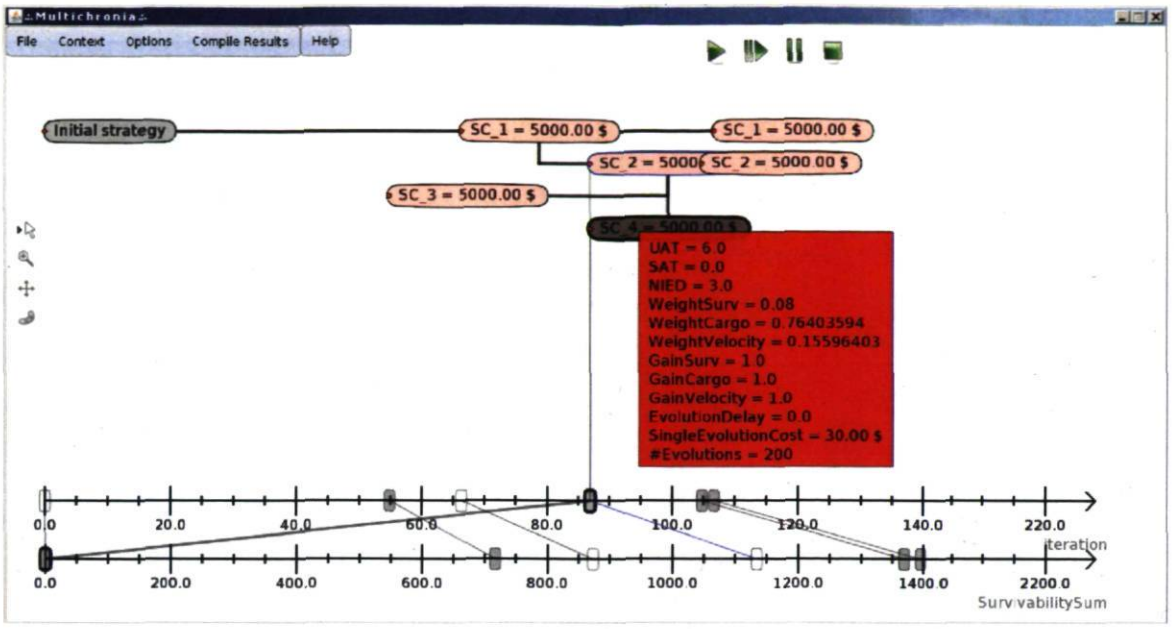


FIGURE 7.17 – Capture d'écran d'un arbre multichronique dans le contexte de simulation stratégique d'attaque de convois militaires

taires donnés par les utilisateurs du logiciel. L'infobulle rouge montre les paramètres d'évolution stratégique de la simulation mise en évidence.

Une différence majeure dans l'implantation du contexte stratégique par rapport au contexte tactique est la manière dont les paramètres sont changés. Premièrement, plutôt que de lire l'état de la simulation directement du fichier de scénario de sauvegarde, le contexte stratégique a accès à l'historique des données provenant de la simulation sous format XML natif. C'est ainsi que l'action de *création d'une nouvelle branche* obtient l'état courant de la coévolution. Une boîte de dialogue sur mesure permet alors de modifier les paramètres de la simulation stratégique (gauche de la Figure 7.18) et, lors de l'allocation initiale, une autre boîte permet d'ajuster les paramètres tactiques (droite de la Figure 7.18). Ces boîtes de dialogue font contraste par rapport à la représentation en arbre d'un fichier XML disponible dans le contexte de simulation tactique Pythagoras, mais permettent tout de même de varier tous les paramètres de coévolution disponibles. Ces paramètres sont d'ailleurs envoyés par réseau TCP/IP au module « contrôleur client » du simulateur stratégique afin d'être pris en compte au démarrage de la simulation.

Puisque le simulateur stratégique n'offre pas de module de visualisation de l'état de la simulation, des logiciels commerciaux tels que Microsoft® Excel, SysStat® ou Tableau Software sont utilisés afin d'analyser les données résultant de la coévolution.

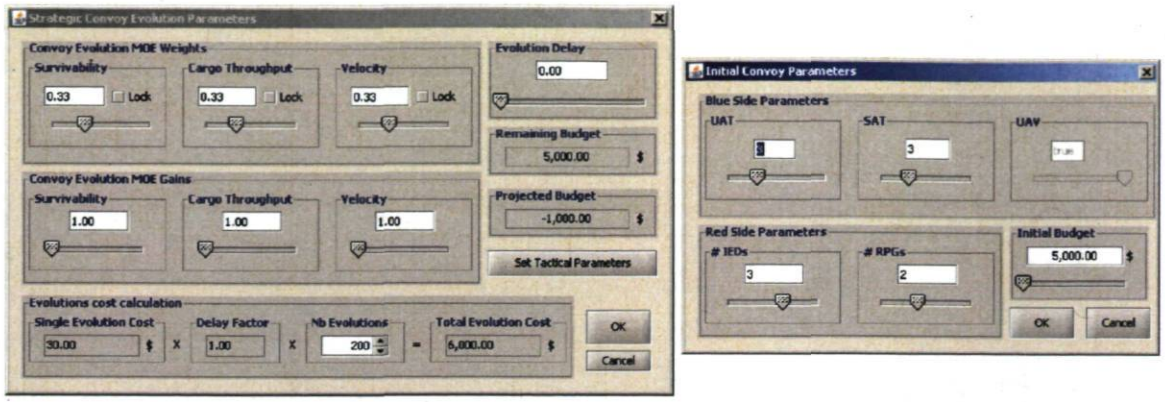


FIGURE 7.18 – Gauche : boîte de dialogue permettant de modifier des paramètres de simulation stratégique (encore au stade de prototype). Droite : boîte de dialogue permettant de modifier l'allocation initiale d'un cours d'action

Par contre, puisque les données XML ne sont pas prises en charge par ces logiciels, les données de simulation doivent être sauvegardées dans un autre format. C'est pourquoi un module de sauvegarde de données texte (*Comma Separated Values*, CSV) a été implanté dans le simulateur stratégique. Ce module sauvegarde donc l'historique des données d'une simulation stratégique dans un fichier. De son côté, le contexte de simulation tactique offre la fonctionnalité de combiner tous les fichiers CSV dans un fichier global pouvant être importé dans les logiciels énumérés plus haut. De plus, la classe implantant la fonctionnalité d'historique de simulation stratégique au niveau de Multichronia peut écrire des données dans une base de données SQL afin que le logiciel Tableau Software puisse charger les données dans son espace de travail.

Afin de compléter la gamme des fonctionnalités disponibles dans le contexte de simulation stratégique, ce dernier offre la possibilité aux utilisateurs de démarrer une simulation tactique correspondant à un pas de temps de la coévolution. Le démarrage d'une simulation tactique permet donc de considérer les données tactiques dans l'analyse d'une simulation stratégique. Cette fonctionnalité implique toutefois la possibilité pour deux contextes d'échanger de l'information. Ainsi, les paramètres d'allocation pour la simulation stratégique sont copiés dans un fichier XML de scénario qui est par la suite chargé dans une instance de simulation tactique de Pythagoras. Un nœud de simulation est également ajouté à la racine de l'arbre multichronique des simulations tactiques. Par conséquent, lorsque des résultats inattendus surviennent dans l'évolution stratégique, l'utilisateur a la possibilité d'en trouver la source en explorant le cours d'action tactique en entier.

7.4 Conclusion

En conclusion, les classes contenues dans les paquetages du cadre conceptuel Multichronia le rendent compatible avec plusieurs applications concrètes. Les fonctionnalités de base qui ont été identifiées satisfont tous les besoins rencontrés jusqu'à présent. Des mécanismes génériques d'extension ont par ailleurs été implantés afin de subvenir à des besoins futurs. L'implantation de Multichronia a été réalisée avec succès et est entièrement fonctionnelle. De plus, les patrons de conception utilisés et l'architecture générique simplifient la tâche des développeurs pour l'ajout de diverses fonctionnalités, dont plusieurs ont déjà été rajoutées au moment d'écrire cette thèse afin de combler les besoins de IMAGEV0. En somme, l'expérience préliminaire des développeurs et des utilisateurs s'est avérée positive.

Parmi les points positifs à retenir, mentionnés par les développeurs, on retrouve la flexibilité du pipeline de données XML. En effet, le choix des données de simulation sauvegardées et envoyées vers Multichronia s'effectue de manière transparente, sans besoin de recompiler le logiciel. De plus, le mécanisme sélectionné pour effectuer des requêtes sur les données entrantes, soit le XQuery, gère avec succès toutes les requêtes rencontrées dans IMAGEV0. Pour le moment, Multichronia en compte environ une dizaine, dont certaines combinent plusieurs nœuds XML et effectuent des opérations mathématiques simples.

D'un autre côté, la réactivité du logiciel est une caractéristique appréciée des utilisateurs. En effet, l'utilisation de plusieurs fils d'exécution pour la gestion de l'interface graphique et des modules relatifs à la simulation et la réception des données rend la couche visible par l'utilisateur indépendante de ce qui se passe sous le capot. L'interface est fluide, réactive, et l'utilisation d'animations la rend davantage attrayante. De plus, l'utilisateur connaît à tout moment l'état des simulations faisant partie de son espace de travail grâce aux rétroactions visuelles offertes pour tout changement d'état.

Finalement, au moment d'écrire cette thèse, une série d'expérimentations sur des sujets humains est en cours de planification. Seules ces expérimentations sauront dire si les utilisateurs de Multichronia en bénéficient par rapport à des techniques classiques. Par ailleurs, le logiciel implantant les techniques classiques sera une version de Multichronia dépourvue de fonctionnalités permettant l'observation simultanée de plusieurs cours d'action. D'ici à ce que des mesures quantitatives soient disponibles, des mesures qualitatives continueront à être recueillies afin de constamment améliorer l'interface graphique de Multichronia et de rendre le logiciel davantage fonctionnel aux yeux de ses utilisateurs et développeurs potentiels.

Chapitre 8

Conclusion

Cette thèse a traité des éléments essentiels à la mise en œuvre de Multichronia, un cadre conceptuel de simulation interactive. Le Chapitre 3 a introduit les principes de base relatifs à Multichronia, notamment quatre boucles d'interaction visant à explorer quatre espaces différents, soit l'espace des paramètres, l'espace des simulations, l'espace des données et l'espace visuel. Le Chapitre 4 a présenté en détails la représentation graphique du cheminement d'un utilisateur dans l'exploration d'un problème, l'arbre multichronique, et les opérations interactives associées. Le Chapitre 5 a discuté du modèle d'un pipeline de données conceptuel visant à simplifier le transport des données provenant d'un simulateur vers un module d'analyse. Ce pipeline de données a été spécialisé pour le XML dans le but d'exploiter toutes les possibilités offertes par les outils offerts par cette technologie. Le Chapitre 6 a présenté deux scénarios qui serviront de bancs d'essai afin d'évaluer les outils interactifs disponibles dans Multichronia lors d'expérimentations sur des sujets humains. Ces scénarios, faisant partie intégrante du projet IMAGE de RDDC Valcartier, concernent les simulations tactique et stratégique de l'attaque de convois militaires par des insurgés. Finalement, le Chapitre 7 a traité de la mise en œuvre de Multichronia. Son architecture générale a été présentée, de même que certains détails d'implantation. Les deux scénarios concrets ont ensuite été détaillés, démontrant ainsi la flexibilité de l'architecture de Multichronia et la possibilité de s'adapter à différents contextes de simulation interactive.

8.1 Retour sur les objectifs

L'objectif principal de cette thèse était de *bâtir une architecture générique de simulation interactive pouvant être utilisée lors d'expérimentations sur des sujets humains.* Le développement du cadre conceptuel Multichronia est le résultat qui correspond à cet objectif. Multichronia définit des boucles d'interaction essentielles à l'implantation de plusieurs fonctionnalités interactives que l'on retrouve dans la littérature, de même que certaines fonctionnalités originales, proposées dans cette thèse. De plus, le cadre conceptuel établit les spécifications d'un pipeline de données générique comprenant plusieurs unités de transformation de données. Chacune est associée à certains traitements affectant les flots de données qui y circulent. Un modèle de données permettant aux développeurs de simulation interactive de facilement adapter un simulateur à Multichronia a également été établi. Sa spécialisation pour la technologie XML a pu démontrer à quel point il est facile de mettre en œuvre la méthodologie de conversion d'un simulateur standard en un simulateur pouvant être exploité par un logiciel de simulation interactive.

Le deuxième objectif principal était de *concevoir une interface graphique de simulation interactive en utilisant des techniques modernes d'interaction.* Pour ce faire, plusieurs fonctionnalités essentielles ont été répertoriées dans la littérature. Certaines limitations au niveau des interfaces existantes ont été notées, en particulier le manque de fonctionnalités interactives de la représentation visuelle du cheminement d'un utilisateur dans sa recherche d'une meilleure compréhension du problème sous étude. C'est ainsi que, basé sur des travaux existants, l'arbre multichronique a été conçu et implanté. Cet arbre affiche, d'une manière structurée et formelle, toutes les modifications de paramètres effectuées par un utilisateur. La contribution se situe au niveau des fonctionnalités interactives qu'il offre en plus de l'affichage. En effet, il permet de « jouer » avec les simulations en reculant et en avançant par rapport à un axe de référence, de synchroniser plusieurs cours d'action, et une foule d'autres fonctionnalités sur lesquelles plus de détails sont donnés au Chapitre 4. La mise en œuvre de l'arbre multichronique, à l'aide de logiciels évolués et de techniques d'interaction modernes, et son déploiement dans des applications concrètes, prouvent que cet objectif a été atteint avec succès. Des améliorations possibles sont toutefois discutées dans les sections suivantes.

Le troisième et dernier objectif principal était de *développer au moins deux applications concrètes basées sur le cadre conceptuel Multichronia et pouvant être utilisées lors d'expérimentations de psychologie cognitive.* Le fait que le travail réalisé pour cette thèse soit partie intégrante du projet IMAGE a permis de fixer le contexte des scénarios envisagés. En effet, deux scénarios dont l'objet est l'attaque de convois militaires

par des insurgés ont été développés. L'un d'eux implique l'allocation de ressources au niveau tactique, c'est-à-dire que tous les éléments présents sur le champ de bataille sont simulés. L'autre implique des décisions en cours d'exécution de la part de l'utilisateur au niveau stratégique, ou autrement dit, des décisions de haut niveau qui influencent indirectement les agents présents sur le champ de bataille. Les résultats concrets de l'implantation de ces deux applications sont encourageants pour la suite des travaux. En effet, le simulateur utilisé dans le contexte de simulation tactique est Pythagoras, un simulateur multiagent développé essentiellement pour des applications de « data farming ». Ce logiciel a été modifié afin de s'adapter aux applications proposées dans le cadre de cette thèse *pendant* le développement de Multichronia. La charge de travail nécessaire pour réaliser ces modifications est donc difficile à quantifier. Toutefois, il est clair qu'elles sont mineures par rapport à la quantité de code nécessaire à l'élaboration d'un simulateur en entier. Elles consistent essentiellement à l'ajout de fonctions implantant la sauvegarde et le chargement de l'état de la simulation et de fonctions qui contrôlent l'exécution d'une simulation. Le simulateur associé au deuxième scénario a été développé dans l'optique d'être exploité par un logiciel de simulation interactive, mais pas spécifiquement pour Multichronia. Ce simulateur stratégique de coévolution a également nécessité des modifications mineures afin d'être intégré à Multichronia. Puisqu'il est de moins grande envergure que Pythagoras, moins de deux jours de travail ont été nécessaires afin de construire toutes les classes essentielles à son exploitation dans Multichronia. L'objectif d'implanter au moins deux applications concrètes exploitant Multichronia a donc été rempli avec succès.

8.2 Limitations

Bien que la plupart des fonctionnalités nécessaires au bon fonctionnement d'un logiciel de simulation interactive aient été implantées dans Multichronia, il n'en demeure pas moins que le cadre d'application proposé possède plusieurs limitations non-critiques, surtout au niveau de la configuration du logiciel. En effet, plusieurs paramètres de Multichronia sont fixés dans le code, ce qui signifie que toute modification nécessite une recompilation de l'application. La configuration du logiciel, soit dans un fichier texte, soit dans une interface de configuration, permettrait à un développeur d'être plus efficace lors de l'utilisation du logiciel. Bien que cette limitation affecte l'étape de développement des contextes d'applications, elle n'affectera en rien la phase d'expérimentation à laquelle seront soumis plusieurs sujets humains qui n'auront qu'à utiliser le logiciel et non à le configurer.

Bien qu'il ait été énuméré comme un problème non-abordé, le problème de la distri-

bution des calculs sur une ferme d'ordinateurs constitue une limitation dans Multichronia. Les applications actuelles ne causent pas de problème lorsqu'elles sont exécutées sur des ordinateurs qui possèdent quatre processeurs. De même, étant donné que, dans les applications courantes, tous les points de divergence sont créés par l'utilisateur, les surcharges de calcul causant une baisse de réactivité de l'interface sont quasi-inexistantes. Par contre, des mécanismes pouvant démarrer plusieurs simulations d'un coup sont en place et leur utilisation rendrait l'application beaucoup moins interactive qu'elle ne l'est présentement dû à une charge de calcul trop grande. La distribution des calculs sur une ferme d'ordinateurs est une solution envisageable pour régler ce problème. Des travaux sur l'extensibilité de Multichronia, visant à intégrer des simulateurs capables de générer davantage de données, pourraient également être entrepris afin d'éliminer les possibles goulots d'étranglement au niveau du pipeline de données.

Par ailleurs, l'approche proposée n'a pas été vérifiée quant au gain en performance offert à des utilisateurs dont le but est de comprendre un problème complexe. Dans son état actuel, l'interface utilisateur liée à Multichronia n'est probablement pas optimale au niveau de son utilisabilité. Cette limitation est cruciale afin qu'un utilisateur emploie Multichronia de manière naturelle et intuitive. Comme tests préliminaires, le logiciel a été distribué à plusieurs personnes membres du projet IMAGE et utilisé afin de mettre les applications développées à l'épreuve. Certains commentaires pertinents concernant l'interface graphique ont été recueillis et des modifications ont été apportées afin de corriger les imperfections. De plus, le logiciel a été testé par des sujets pilotes appelés à solutionner un problème donné à l'aide d'une suite d'outils logiciels. Multichronia en faisait partie, de même qu'un logiciel de visualisation commercial et un autre servant à représenter les connaissances acquises sous la forme d'un graphe conceptuel. Les résultats sont encourageants puisque les utilisateurs ont su utiliser Multichronia adéquatement. Par contre, seules des expérimentations menées en bonne et due forme pourront révéler le gain d'un utilisateur à utiliser Multichronia par rapport aux techniques classiques. Ces expérimentations seront entreprises par une équipe de psychologie de la cognition au moment de déposer cette thèse et les résultats seront connus à la mi 2009.

8.3 Perspectives

En plus des expérimentations de psychologie cognitive visant à étudier l'effet d'outils de simulation interactive sur la compréhension des systèmes complexes par des sujets humains, Multichronia ouvre plusieurs axes de recherche qui n'ont par encore été visités jusqu'à ce jour.

Premièrement, puisque la majorité des modèles de simulation utilisés afin de résoudre des problèmes réels sont stochastiques, il serait intéressant que Multichronia puisse gérer ces derniers. Les travaux de Fischer *et al.* sont un bon pas en avant, mais le système qu'ils ont développé n'est pas très interactif en comparaison avec Multichronia [Fischer *et al.*, 2007]. Par ailleurs, les simulations stochastiques impliquent la réplication de plusieurs simulations possédant le même ensemble de paramètres de simulation initiaux mais différents paramètres de générateurs de nombres aléatoires. Les résultats de travaux futurs sur l'intégration de simulations stochastiques dans un processus de simulation interactive devront fournir une représentation originale pour les simulations stochastiques générées par des tests d'hypothèse. De plus, il faudra revoir la manière d'interagir avec les résultats, que ce soit via un arbre ou toute autre métaphore. La présence d'algorithmes de surveillance qui proposent à l'utilisateur un sous-ensemble des résultats est une solution envisageable afin de diminuer l'effort cognitif qu'aurait à fournir un utilisateur pour gérer cette énorme quantité de données.

Similairement, il serait intéressant d'intégrer des outils formels d'analyse à Multichronia. En effet, la simulation interactive ne vise pas à remplacer les études formelles, mais à les complémenter. Les méthodes formelles sont très utiles afin d'échantillonner grossièrement l'espace des paramètres. Accomplir cette tâche manuellement deviendrait répétitif pour l'utilisateur et ne serait pas très efficace. D'un autre côté, seul l'utilisateur peut comprendre les résultats qu'il obtient et donc se questionner lorsque des résultats inattendus, tels que des observations aberrantes, surviennent. C'est pourquoi une parfaite symbiose entre les méthodes d'exploration formelles et les méthodes interactives dans un même logiciel est un axe de recherche prometteur. Le design d'expériences interactif est également une solution à envisager.

L'arbre multichronique, tel qu'il a été présenté dans cette thèse, n'est que la première itération d'une interface graphique visant à représenter le cheminement d'un utilisateur dans une étude de simulation interactive. En effet, l'aspect visuel de l'arbre pourrait être grandement amélioré par l'ajout de données supplémentaires, sans pour autant surcharger l'utilisateur cognitivement. Il serait possible, par exemple, de représenter l'arbre multichronique en trois dimensions et d'associer un axe de référence supplémentaire à cette troisième dimension. De plus, l'interaction avec l'arbre pourrait être davantage intuitive et se dérouler dans un environnement de réalité virtuelle. Les possibilités graphiques seraient alors grandement améliorées, de même que l'interaction de l'utilisateur.

Finalement, l'intégration de Multichronia à un espace de travail complet ne ferait que bénéficier à l'utilisateur. En effet, l'intégration d'un environnement de visualisation générique rendrait Multichronia beaucoup plus facile d'utilisation et complèterait le cadre

conceptuel présenté dans cette thèse. Un des buts du projet IMAGE est d'ailleurs la réalisation d'une telle intégration. Le défi est grand, mais Multichronia a été conçu afin que l'intégration de modules externes se réalise le plus simplement possible. La résolution de problèmes autres que la simulation de systèmes complexes à l'aide de Multichronia est également envisageable et souhaitable. Multichronia peut en effet répondre au besoin de garder une trace visuelle des essais effectués pour tout type d'application, allant du développement de modèles aux applications en ligne en passant par la conception assistée par ordinateur.

Bibliographie

[Abadi *et al.*, 2005] D. ABADI, Y. AHMAD, M. BALAZINSKA, U. ÇETINTEMEL, M. CHERNIACK, J. H. HWANG, W. LINDNER, A. S. MASKEY, A. RASIN, et E. RYVKINA. The design of the Borealis stream processing engine. *CIDR Conference*, 2005.

[Abadi *et al.*, 2003] D. J. ABADI, D. CARNEY, U. ÇETINTEMEL, M. CHERNIACK, C. CONVEY, S. LEE, M. STONEBRAKER, N. TATBUL, et S. ZDONIK. Aurora : a new model and architecture for data stream management. *The International Journal on Very Large Data Bases*, 12(2) :120–139, 2003.

[Aigner *et al.*, 2008] W AIGNER, S MIKSCH, W MÜLLER, H SCHUMANN, et C TOMINSKI. Visual methods for analyzing time-oriented data. *IEEE Transactions on Visualization and Computer Graphics*, pages 47–60, 2008.

[Allen *et al.*, 2000] G. ALLEN, W. BENGER, T. GOODALE, H. C. HEGE, G. LANFERMANN, A. MERZKY, T. RADKE, E. SEIDEL, et J. SHALF. The Cactus Code : a problem solving environment for the grid. *The Ninth International Symposium on High-Performance Distributed Computing*, pages 253–260, 2000.

[Altova, 2009] ALTOVA. Xmlspy. [En ligne]. `http://www.altova.com/products/xmlspy/xml_editor.html` (Page consultée le 30 aout 2009).

[Anderson *et al.*, 2008] EWA ANDERSON, JP HEITMANN, K HABIB, et S SILVA. Provenance in comparative analysis : A study in cosmology. *Computing in Science & Engineering*, 10(3) :30–37, 2008.

[Arnott, 1998] D. ARNOTT. A taxonomy of decision biases. Rapport Technique, Monash University, 1998.

[Atkinson et Kuhne, 2003] C. ATKINSON et T. KUHNE. Model-driven development : A metamodeling foundation. *Software, IEEE*, 20(5) :36–41, 2003.

[Bell, 1989] P. C. BELL. Stochastic visual interactive simulation models. *Journal of the Operational Research Society*, 40(7) :615–624, 1989.

[Bell et O'Keefe, 1987] P. C. BELL et R. M. O'KEEFE. Visual interactive simulation–history, recent developments, and major issues. *SIMULATION*, 49(3) :109, 1987.

[Bell et O'Keefe, 1994] P. C. BELL et R. M. O'KEEFE. Visual interactive simulation : A methodological perspective. *Annals of Operations Research*, 53(1) :321–342, 1994.

[Bell et O'Keefe, 1995] P. C. BELL et R. M. O'KEEFE. An experimental investigation into the efficacy of visual interactive simulation. *Management Science*, 41(6) :1018–1038, 1995.

[Bernier et Rioux, 2008] François BERNIER et François RIOUX. Convoy scenario for complexity study – Co-evolution for strategic red-teaming. Rapport Technique, RDDC Valcartier, 2008.

[Bitinas, 2002] E. BITINAS. Pythagoras : The newest member of the Project Albert family. *Computing Advances in Military OR-WG 31. 70 thMilitary Operations Research Society Symposium*, pages 18–20, 2002.

[Borland, 2009] BORLAND. Together. [En ligne]. `http://www.borland.com/us/products/together/index.html` (Page consultée le 30 aout 2009).

[Bourret, 2005] R. BOURRET. XML and databases. [En ligne]. `http://www.rpbourret.com/xml/XMLAndDatabases.htm` (Page consultée le 30 aout 2009).

[Boyno, 2006] E. A. BOYNO. XML : What, what, who and where. Dans *ISECON*, 2006.

[Brandstein et Horne, 1998] A. BRANDSTEIN et G. HORNE. Data farming : A meta-technique for research in the 21st century. *Maneuver Warfare Science*, 1998.

[Brodlie *et al.*, 1993] K. BRODLIE, A. POON, H. WRIGHT, L. BRANKIN, G. BANECKI, et A. GAY. GRASPARC–A problem solving environment integrating computation and visualization. *IEEE Visualization*, pages 102–109, 1993.

[Brodlie *et al.*, 2004] K. BRODLIE, J. WOOD, D. DUCE, et M. SAGAR. gViz : Visualization and computational steering on the grid. *UK e-Science All Hands Meeting*, 2004.

[Brodlie et Wood, 2007] K. W. BRODLIE et J. D. WOOD. Computational steering in visualization dataflow environments. Dans *MODSIM International Congress on Modelling and Simulation*, 2007.

[Brooke *et al.*, 2003] J. M. BROOKE, P. V. COVENEY, J. HARTING, S. JHA, S. M. PICKLES, R. L. PINNING, et A. R. PORTER. Computational steering in RealityGrid. Dans *UK e-Science All Hands Meeting*, pages 2–4, 2003.

[Brooks, 1988] F. P. BROOKS. Grasping reality through illusion-interactive graphics serving science. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1–11, 1988.

[Cannataro *et al.*, 2004] M. CANNATARO, C. COMITO, F. L. SCHIAVO, et P. VELTRI. Proteus, a grid based problem solving environment for bioinformatics : Architecture and experiments. *IEEE Computational Intelligence Bulletin*, 3(1) :7–18, 2004.

[Carlson, 2006] D. CARLSON. Semantic models for XML Schema with UML tooling. Dans *2nd International Workshop on Semantic Web Enabled Software Engineering*, 2006.

[Chau et Bell, 1995] P. Y. K. CHAU et P. C. BELL. Designing effective simulation-based decision support systems : An empirical assessment of three types of decision support systems. *Journal of the Operational Research Society*, 46(3) :315–331, 1995.

[Choo *et al.*, 2007] CS CHOO, CL CHUA, et SHV TAY. Automated red teaming : A proposed framework for military application. *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1936–1942, 2007.

[Cioppa, 2002] T. M. CIOPPA. *Efficient Nearly Orthogonal and Space-Filling Experimental Designs for High-Dimensional Complex Models.* Thèse de doctorat, Naval Postgraduate School, 2002.

[Cioppa et Lucas, 2007] T. M. CIOPPA et T. W. LUCAS. Efficient nearly orthogonal and space-filling latin hypercubes. *Technometrics*, 49(1) :45–55, 2007.

[Codella *et al.*, 1992] C. CODELLA, R. JALILI, L. KOVED, J. B. LEWIS, D. T. LING, J. S. LIPSCOMB, D. A. RABENHORST, C. P. WANG, A. NORTON, et P. SWEENEY. Interactive simulation in a multi-person virtual world. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 329–334, 1992.

[Cohen et Harel, 2007] I. R. COHEN et D. HAREL. Explaining a complex living system : Dynamics, multi-scaling and emergence. *Interface*, 4 :175–182, 2007.

[Concannon et Becker, 1990] K. CONCANNON et P. BECKER. A tutorial on GENETIK simulation and scheduling (tutorial session). *Proceedings of the 22nd conference on Winter simulation*, pages 140–145, 1990.

[Dane et Pratt, 2007] E. DANE et M. G. PRATT. Exploring intuition and its role in managerial decision making. *The Academy of Management Review (AMR)*, 32(1) :33–54, 2007.

[Davis et Kottemann, 1994] F. D. DAVIS et J. E. KOTTEMANN. User perceptions of decision support effectiveness : Two production planning experiments. *Decision Sciences*, 25(1) :57–76, 1994.

[Davis *et al.*, 1991] F. D. DAVIS, J. E. KOTTEMANN, et W. E. REMUS. What-if analysis and the illusion of control. *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, 3, 1991.

[De Rainville *et al.*, 2009] François-Michel DE RAINVILLE, Christian GAGNÉ, Olivier TEYTAUD, et Denis LAURENDEAU. Optimizing low-discrepancy sequences with an evolutionary algorithm. Dans *GECCO*, pages 1491–1498, 2009.

[Decker *et al.*, 2000] S. DECKER, S. MELNIK, F. van HARMELEN, D. FENSEL, M. KLEIN, J. BROEKSTRA, M. ERDMANN, et I. HORROCKS. The semantic web : the roles of XML and RDF. *IEEE Internet Computing*, 4(5) :63–73, 2000.

[Defense Update, 2004a] DEFENSE UPDATE. Countering the rpg threat. [En ligne]. `http://www.defense-update.com/features/du-1-04/rpg-threat.htm` (Page consultée le 30 aout 2009).

[Defense Update, 2004b] DEFENSE UPDATE. Ied – a weapons' profile. [En ligne]. `http://www.defense-update.com/features/du-3-04/IED.htm` (Page consultée le 30 aout 2009).

[Defense Update, 2005] DEFENSE UPDATE. Convoy protection & escort. [En ligne]. `http://www.defense-update.com/features/du-2-05/protection-9.htm` (Page consultée le 30 aout 2009).

[Denk, 2007] G. DENK. Modelling and simulation of transient noise in circuit simulation. *Mathematical and Computer Modelling of Dynamical Systems*, 13(4) :383–394, 2007.

[Dörner, 1980] D. DÖRNER. On the difficulties people have in dealing with complexity. *Simulation & Gaming*, 11(1) :87, 1980.

[Dörner, 1996] D. DÖRNER. *The Logic of Failure : Recognizing and Avoiding Problems in Complex Situations.* Perseus Press, 1996.

[Dos Santos, 2007] M. I. R. DOS SANTOS. Estimating and validating nonlinear regression metamodels in simulation. *Communications in Statistics-Simulation and Computation*, 36(1) :123–137, 2007.

[Drogoul *et al.*, 2002] A. DROGOUL, D. VANBERGUE, et T. MEURISSE. Multi-agent based simulation : Where are the agents. *Third International Workshop Multi-Agent-Based Simulation*, pages 15–16, 2002.

[Eclipse Foundation, 2009] ECLIPSE FOUNDATION. Eclipse. [En ligne]. `http://www.eclipse.org` (Page consultée le 30 aout 2009).

[Edmonds, 1999] B. EDMONDS. *Syntactic Measures of Complexity.* Thèse de doctorat, University of Manchester, 1999.

[Esnard, 2005] A. ESNARD. *Analyse, conception et réalisation d'un environnement pour le pilotage et la visualisation en ligne de simulations numériques parallèles.* Thèse de doctorat, Université de Bordeaux, 2005.

[Esnard *et al.*, 2004] A. ESNARD, M. DUSSERE, et O. COULAUD. A time-coherent model for the steering of parallel simulations. *Europar*, 2004.

[Fischer *et al.*, 2007] Matthias FISCHER, Christoph LAROQUE, Daniel HUBER, Jens KROKOWSKI, Bengt MUECK, Michael KORTENJAN, Mark AUFENANGER, et Wilhelm DANGELMAIER. Interactive refinement of a material flow simulation model by comparing multiple simulation runs in one 3D environment. Dans *European Simulation and Modelling Conference*, pages 499–505. EUROSIS, 2007.

[Fishwick, 1997] P. A. FISHWICK. Computer simulation : Growth through extension. *TRANSACTIONS of The SCS*, 14 :13–23, 1997.

[Flexsim, 2009] FLEXSIM. Flexsim simulation software. [En ligne]. `http://www.flexsim.com` (Page consultée le 30 aout 2009).

[Geer, 2005] D GEER. Will binary XML speed network traffic ? *Computer*, 38(4) :16–18, 2005.

[Geist *et al.*, 1996] G. A. GEIST, J. A. KOHL, et P. M. PAPADOPOULOS. CUMULVS : Providing fault-tolerance, visualization and steering of parallel applications. *Environment*, 1996.

[Germans *et al.*, 2001] D. GERMANS, H. J. W. SPOELDER, L. RENAMBOT, et H. E. BAL. High-level steering : Measuring in virtual reality environments. *Proceedings of the ASCI*, 2001.

[Gill et Kalantzis, 2007] Andy GILL et Eugenia KALANTZIS. Limited objective experiment 0502 convoy force protection. Rapport Technique AEC-R 0502, Department of National Defence, March 2007 2007.

[Gilman et Billingham, 1989] A. R. GILMAN et C. BILLINGHAM. A tutorial on SEE WHY and WITNESS. *Proceedings of the 21st conference on Winter simulation*, pages 192–200, 1989.

[Grinstein *et al.*, 2001] G. GRINSTEIN, M. TRUTSCHL, et U. CVEK. High-dimensional visualizations. *Proceedings of the 7th Data Mining Conference-KDD*, 2001.

[Gu et Eisenhauer, 1994] W. GU et G. EISENHAUER. Falcon : On-line monitoring and steering of large-scale parallel programs. Dans *Symposium on the Frontiers of Massively Parallel Computation*, 1994.

[Haber et McNabb, 1990] R. B. HABER et D. A. MCNABB. Visualization idioms : A conceptual model for scientific visualization systems. *Visualization in Scientific Computing*, pages 74–93, 1990.

[Hakola, 2004] MB HAKOLA. *An Exploratory Analysis of Convoy Protection Using Agent-Based Simulation.* Thèse de doctorat, Naval Postgraduate School, 2004.

[Harrison *et al.*, 2005] N. HARRISON, B. GILBERT, A. JEFFREY, R. LESTAGE, M. LAUZON, et A. MORIN. KARMA : Materializing the soul of technologies into models. Dans *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*, 2005.

[Hauge et Paige, 2001] J. W. HAUGE et K. N. PAIGE. Learning SIMUL8 : The complete guide. *Bellingham, WA : PlainVu Publishers*, 2001.

[Heer *et al.*, 2005] J. HEER, S. K. CARD, et J. A. LANDAY. Prefuse : A toolkit for interactive information visualization. *Human Factors in Computing Systems*, pages 421–430, 2005.

[Henriksen et Crain, 1989] J. O. HENRIKSEN et R. C. CRAIN. *GPSS/H Reference Manual.* Wolverine Software, 1989.

[Hobbs, 2003] R. L. HOBBS. Using XML to support military decision-making. *Proceedings of the XML Conference and Exposition*, 2003.

[Horne, 2001] G. HORNE. Beyond point estimates : Operational synthesis and data farming. *Maneuver Warfare Science*, pages 1–8, 2001.

[Horne et Meyer, 2004] G. E. HORNE et T. E. MEYER. Data farming : discovering surprise. Dans *Winter Simulation Conference*, volume 1, 2004.

[Houstis et Rice, 2000] E. N. HOUSTIS et J. R. RICE. Future problem solving environments for computational science. *Mathematics and Computers in Simulation*, 54(4-5) :243–257, 2000.

[Hübler, 2007] A. W. HÜBLER. Understanding complex systems : Defining an abstract concept : Simply complex. *Complexity*, 12(5) :9–11, 2007.

[Hurrion, 1976] R. D. HURRION. *The design, use and required facilities of an interactive visual computer simulation language to explore production planning problems.* Thèse de doctorat, University of London, England, 1976.

[Huyet et Pierreval, 2008] A. L. HUYET et H. PIERREVAL. Modeling human real time decisions : an approach based on automatic learning and visual interactive simulation. *Expert systems*, 18(19) :28, 2008.

[Hybinette et Fujimoto, 2001] M. HYBINETTE et R. M. FUJIMOTO. Cloning parallel simulations. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 11(4) :378–407, 2001.

[Iba, 2004] T. IBA. A framework and tools for modeling and simulating societies as evolutionary complex systems. *2nd. International Conference of the European Social Simulation Association*, 2004.

[IBM, 2009] IBM. Rational rose. [En ligne]. `http://www.rational.com` (Page consultée le 30 aout 2009).

[Ilachinski, 1997] A. ILACHINSKI. Irreducible semi-autonomous adaptive combat (ISAAC) : An artificial-life approach to land warfare. *Center for Naval Analyses Research Memorandum CRM*, 9 :7–61.10, 1997.

[Inselberg et Dimsdale, 1990] A. INSELBERG et B. DIMSDALE. Parallel coordinates : a tool for visualizing multi-dimensional geometry. *Proceedings of the First IEEE Conference on Visualization*, pages 361–378, 1990.

[Jablonowski *et al.*, 1993] D. J. JABLONOWSKI, J. D. BRUNER, B. BLISS, et R. B. HABER. VASE : The visualization and application steering environment. *Supercomputing*, pages 560–569, 1993.

[Jankun-Kelly *et al.*, 2007] T. J. JANKUN-KELLY, K. L. MA, et M. GERTZ. A model and framework for visualization exploration. *IEEE Transactions on Visualization and Computer Graphics*, pages 357–369, 2007.

[JAXB, 2009] JAXB. Java architecture for xml binding (jaxb). [En ligne]. `https://jaxb.dev.java.net/` (Page consultée le 30 aout 2009).

[Jennings et Wooldridge, 2000] N. R. JENNINGS et M. WOOLDRIDGE. Agent-oriented software engineering. *Handbook of Agent Technology*, 2000.

[JOGL, 2009] JOGL. Java bindings for opengl. [En ligne]. `https://jogl.dev.java.net/` (Page consultée le 30 aout 2009).

[Johnson *et al.*, 1999] C. JOHNSON, S. G. PARKER, C. HANSEN, G. L. KINDLMANN, et Y. LIVNAT. Interactive simulation and visualization. *Computer*, 32(12) :59–65, 1999.

[Johnson, 2007] J. JOHNSON. Science and policy in designing complex futures. *Futures*, 2007.

[Jordan et Piel, 2008] G. E. JORDAN et W. H. PIEL. Phylowidget : Web-based visualizations for the tree of life. *Bioinformatics*, 2008.

[Kalawsky et Nee, 2004] R. S. KALAWSKY et S. P. NEE. Important issues concerning interactive user interfaces in grid based computational steering systems. *Proceedings of the UK e-Science All Hands Meeting*, 2004.

[Kalawsky *et al.*, 2005] R. S. KALAWSKY, J. O'BRIEN, et P. V. COVENEY. Improving scientists' interaction with complex computational - visualization environments based on a distributed grid infrastructure. *Philosophical Transactions : Mathematical, Physical and Engineering Sciences*, 363(1833) :1867–1884, 2005.

[Kelton et Barton, 2003] W. D. KELTON et R. R. BARTON. Experimental design for simulation. *Proceedings of the Winter Simulation Conference*, 1, 2003.

[Kleijnen, 1992] J. P. C. KLEIJNEN. Regression metamodels for simulation with common random numbers : comparison of validation tests and confidence intervals. *Management Science*, 38(8) :1164–1185, 1992.

[Kleijnen, 2007] J. P. C. KLEIJNEN. Kriging metamodeling in simulation : A review. *European Journal of Operational Research*, 2007.

[Kleijnen *et al.*, 2005] J. P. C. KLEIJNEN, S. M. SANCHEZ, T. W. LUCAS, et T. M. CIOPPA. A user's guide to the brave new world of designing simulation experiments. *INFORMS Journal on Computing*, 17(3) :263–289, 2005.

[Kleijnen et Sargent, 2000] J. P. C. KLEIJNEN et R. G. SARGENT. A methodology for fitting and validating metamodels in simulation. *European Journal of Operational Research*, 120(1) :14–29, 2000.

[Kohl et Papadopoulas, 1998] J. A. KOHL et P. M. PAPADOPOULAS. Efficient and flexible fault tolerance and migration of scientific simulations using CUMULVS. *SIGMETRICS symposium on Parallel and distributed tools*, pages 60–71, 1998.

[Kottemann *et al.*, 1994] J. E. KOTTEMANN, F. D. DAVIS, et W. E. REMUS. Computer-assisted decision making : performance, beliefs, and the illusion of control. *Organizational behavior and human decision processes*, 57(1) :26–37, 1994.

[Kreylos *et al.*, 2002] O. KREYLOS, A. M. TESDALL, B. HAMANN, J. K. HUNTER, et K. I. JOY. Interactive visualization and steering of CFD simulations. *Proceedings of the symposium on Data Visualisation*, pages 25–34, 2002.

[Kurtev et van den Berg, 2005] I. KURTEV et K. van den BERG. Building adaptable and reusable XML applications with model transformations. *Proceedings of the 14th international conference on World Wide Web*, pages 160–169, 2005.

[Lanner Group, 2009] LANNER GROUP. WITNESS. [En ligne]. `http://www.lanner.com` (Page consultée le 30 aout 2009).

[Lauren et Stephen, 2002] M. K. LAUREN et R. T. STEPHEN. Map-aware non-uniform automata (MANA) - a new zealand approach to scenario modelling. *Journal of Battlefield Technology*, 5(1) :27–31, 2002.

[Law, 2006] A. LAW. *Simulation Modeling and Analysis with Expertfit Software.* McGraw-Hill Science/Engineering/Math, 4th edition édition, 2006.

[LBL, 2008] LBL. Nux – overview. [En ligne]. `http://acs.lbl.gov/nux` (Page consultée le 30 aout 2009).

[Le Bars et Le Grusse, 2008] M. LE BARS et P. LE GRUSSE. Use of a decision support system and a simulation game to help collective decision-making in water management. *Computers and Electronics in Agriculture*, 62(2) :182–189, 2008.

[Lizotte *et al.*, 2008] M. LIZOTTE, F. BERNIER, M. MOKHTARI, E. BOIVIN, M. DUCHARME, et D. POUSSART. IMAGE : Simulation for understanding complex situations and increasing future force agility. Dans *Proceedings of the 26th Army Science Conference*, page 7, 2008.

[Lizotte *et al.*, 2007] Michel LIZOTTE, Marielle MOKHTARI, Michel DUCHARME, Éric BOIVIN, François BERNIER, et Denis POUSSART. IMAGE - an interactive computer aided cognition capability for C4ISR complexity discovery - a human assistant to understand complex C4ISR situations, executive overview. Rapport Technique SL 20078-497, DRDC Valcartier, 2007.

[Lloyd *et al.*, 2004] C. M. LLOYD, M. D. HALSTEAD, et P. F. NIELSEN. CellML : its future, present and past. *Prog Biophys Mol Biol*, 85(2-3) :433–50, 2004.

[Lloyd *et al.*, 2007] S. LLOYD, D. GAVAGHAN, A. SIMPSON, M. MASCORD, C. SENEURINE, G. WILLIAMS, J. PITT-FRANCIS, D. BOYD, D. MAC RANDAL, et L. SASTRY. Integrative biology – the challenges of developing a collaborative research environment for heart and cancer modelling. *Future Generation Computer Systems*, 23(3) :457–465, 2007.

[Marshall *et al.*, 1990] R. MARSHALL, J. KEMPF, S. DYER, et C. C. YEN. Visualization methods and simulation steering for a 3D turbulence model of Lake Erie. *Symposium on Interactive 3D graphics*, pages 89–97, 1990.

[McCormick *et al.*, 1987] B. H. McCormick, T. A. DeFanti, et M. D. Brown. Visualization in scientific computing - a synopsis. *IEEE Computer Graphics & Applications*, 7(7) :61–70, 1987.

[McDonald et Upton, 2005] ML McDonald et SC Upton. Investigating the dynamics of competition : coevolving red and blue simulation parameters. *Proceedings of the 37th conference on Winter simulation*, pages 1008–1012, 2005.

[McGrath, 2003] R. E. McGrath. XML and scientific file formats. *Seattle Annual Meeting*, 2003.

[Montgomery, 2001] D. C. Montgomery. *Design and Analysis of Experiments.* John Wiley & Sons, New York, 5 edition, 2001.

[Mulder, 1998] J. D. Mulder. *Computational Steering with Parametrized Geometric Objects.* Thèse de doctorat, Universiteit van Amsterdam, 1998.

[MySQL, 2009] MySQL. MySQL : : The world's most popular open source database. [En ligne]. `http://www.mysql.com` (Page consultée le 30 aout 2009).

[Norman et Draper, 1986] D. A. Norman et S. W. Draper. *User centered system design.* Lawrence Erlbaum Associates Hillsdale, NJ, 1986.

[Novotný, 2004] R. Novotný. CASE tool for developing XML schemas and XSLT. Mémoire de maîtrise, Comenius University, 2004.

[NPS, 2009] NPS. Pythagoras. [En ligne]. `http://harvest.nps.edu` (Page consultée le 30 aout 2009).

[Osman, 2008] M. Osman. Observation can be as effective as action in problem solving. *Cognitive Science : A Multidisciplinary Journal*, 32(1) :162–183, 2008.

[Parker, 1999] S. G. Parker. *The SCIRun Problem-Solving Environment and Computational Steering Software System.* Thèse de doctorat, University of Utah, 1999.

[Parker *et al.*, 1997] S. G. Parker, D. W. Weinstein, et C. R. Johnson. The SCIRun computational steering software system. *Modern software tools for scientific computing table of contents*, pages 5–44, 1997.

[Patton, 2003] J. R. Patton. Intuition in decisions. *Management Decision*, 41(10) :989–996, 2003.

[Philippakis, 1988] A. S. Philippakis. Structured what if analysis in DSS models. *Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences*, 3, 1988.

[Pickles *et al.*, 2004a] S. M. Pickles, R. J. Blake, B. M. Boghosian, J. M. Brooke, J. Chin, P. E. L. Clarke, P. V. Coveney, N. GonzÃ¡lez-Segredo, R. Haines, et J. Harting. The TeraGyroid experiment. Dans *Proceedings of the Workshop on Case Studies on Grid Applications at GGF*, volume 10, page 2004, 2004.

[Pickles *et al.*, 2004b] S. M. PICKLES, R. HAINES, R. L. PINNING, et A. R. PORTER. Practical tools for computational steering. *UK e-Science All Hands Meeting*, 31, 2004.

[Pickles *et al.*, 2005] S. M. PICKLES, R. HAINES, R. L. PINNING, et A. R. PORTER. A practical toolkit for computational steering. *Mathematical, physical and engineering sciences Philosophical transactions*, 363(1833) :1843–1853, 2005.

[Poussart, 2006] Denis POUSSART. Complexity : An overview of its nature and manifestations, and of S&T convergence, with comments on their relevance to Canadian Defense. Rapport Technique, RDDC Valcartier, 2006.

[Power et Sharda, 2007] D. J. POWER et R. SHARDA. Model-driven decision support systems : Concepts and research directions. *Decision Support Systems*, 43(3) :1044–1061, 2007.

[Processing, 2009] PROCESSING. Processing 1.0 (BETA). [En ligne]. `http://www.processing.org` (Page consultée le 30 aout 2009).

[Reas et Fry, 2007] C. REAS et B. FRY. *Processing : A Programming Handbook for Visual Designers and Artists.* MIT Press, 2007.

[Reenskaug, 1979] T. M. H. REENSKAUG. Thing-Model-View-Editor, an example from a planning-system. Rapport Technique, Xerox PARC, 1979.

[Renambot *et al.*, 2000] Luc RENAMBOT, Henri E. BAL, Desmond GERMANS, et Hans J.W. SPOELDER. CAVEStudy : An infrastructure for computational steering in virtual reality environments. Dans *IEEE International Symposium on High Performance Distributed Computing*, pages 57–61, 2000.

[Rice et Boisvert, 1996] J. R. RICE et R. F. BOISVERT. From scientific software libraries to problem-solving environments. *IEEE Computational Science and Engineering*, 3(3) :44–53, 1996.

[Riehle, 2000] D. RIEHLE. *Framework Design - A Role Modeling Approach.* Thèse de doctorat, Swiss Federal Institute of Technology Zurich, 2000.

[Rioux *et al.*, 2008] François RIOUX, François BERNIER, et Denis LAURENDEAU. Design and implementation of an XML-based, technology-unified data pipeline for interactive simulation. Dans *Winter Simulation Conference*, Miami, FL, 2008.

[Robinson, 2004] S. ROBINSON. *Simulation : The Practice of Model Development and Use.* Wiley, 2004.

[Robinson *et al.*, 2004] S. ROBINSON, R. E. NANCE, R. J. PAUL, M. PIDD, et S. J. E. TAYLOR. Simulation model reuse : definitions, benefits and obstacles. *Simulation Modelling Practice and Theory*, 12(7-8) :479–494, 2004.

[Rockwell Automation Inc., 2009] ROCKWELL AUTOMATION INC.. Arena simulation. [En ligne]. `http://www.arenasimulation.com` (Page consultée le 30 aout 2009).

[Röhl et Uhrmacher, 2005] M. RÖHL et A. M. UHRMACHER. Flexible integration of XML into modeling and simulation systems. *Proceedings of the 37th conference on Winter simulation*, pages 1813–1820, 2005.

[Sanchez, 2005] S. M. SANCHEZ. Work smarter, not harder : Guidelines for designing simulation experiments. *Proceedings of the Winter Simulation Conference*, pages 69–82, 2005.

[Sanchez, 2009] S. M. SANCHEZ. NOLHdesigns spreadsheet. [En ligne]. `http://harvest.nps.edu` (Page consultée le 30 aout 2009).

[Sanchez *et al.*, 2006] S. M. SANCHEZ, F. MOEENI, et P. J. SANCHEZ. So many factors, so little time – simulation experiments in the frequency domain. *International Journal of Production Economics*, 103(1) :149–165, 2006.

[Santner *et al.*, 2003] T. J. SANTNER, B. J. WILLIAMS, et W. NOTZ. *The Design and Analysis of Computer Experiments*. Springer, 2003.

[Schruben et Cogliano, 1981] L. W. SCHRUBEN et V. J. COGLIANO. Simulation sensitivity analysis : A frequency domain approach. *Proceedings of the 13th conference on Winter simulation*, pages 455–459, 1981.

[Schruben et Cogliano, 1987] L. W. SCHRUBEN et V. J. COGLIANO. An experimental procedure for simulation response surface model identification. *Communications of the ACM*, 30(8) :716–730, 1987.

[Sips *et al.*, 2007] M. SIPS, J. SCHNEIDEWIND, et D. A. KEIM. Highlighting space-time patterns : Effective visual encodings for interactive decision making. *International Journal of Geographical Information Science*, 21(8) :879, 2007.

[Spector, 2008] J. M. SPECTOR. Cognition and learning in the digital age : Promising research and practice. *Computers in Human Behavior*, 24(2) :249–262, 2008.

[Standridge *et al.*, 1990] C.R. STANDRIDGE, K.M. MATWICZAK, D.A. DAVIS, K.J. MUSSELMAN, et D.T. BRUNNER. Interactive simulation. Dans *Winter Simulation Conference*, pages 453–458, 1990.

[Sterman, 1994] J. D. STERMAN. Learning in and about complex systems. *System Dynamics Review*, 10(2-3) :291–330, 1994.

[Sterman, 2004] J. D. STERMAN. *Business dynamics : systems thinking and modeling for a complex world*. McGraw-Hill, Boston, 2004.

[Tableau Software, 2009] TABLEAU SOFTWARE. Data analysis | tableau software | visual analysis. [En ligne]. `http://www.tableausoftware.com` (Page consultée le 30 aout 2009).

[The Canadian Press, 2008] THE CANADIAN PRESS. Canadian army 'regrets' killing 2 Afghan children when car approached convoy. [En ligne]. `http://ca.news.yahoo.com/s/capress/080728/world/afghan_cda_children` (Page consultée le 1er octobre 2008).

[The Mathworks, 2009] THE MATHWORKS. Matlab. [En ligne]. `http://www.mathworks.com/` (Page consultée le 30 aout 2009).

[van Liere *et al.*, 1996] R. van LIERE, J. D. MULDER, et J. J. van WIJK. *Computational Steering.* Computer Science, Department of Interactive Systems, CWI, 1996.

[van Wijk et van Liere, 1994] J. J. van WIJK et R. van LIERE. *An Environment for Computational Steering.* Centrum voor Wiskunde en Informatica, 1994.

[W3C, 2009a] W3C. Mathematical markup language (MathML). [En ligne]. `http://www.w3.org/TR/MathML/` (Page consultée le 30 aout 2009).

[W3C, 2009b] W3C. W3C XML Query (XQuery). [En ligne]. `http://www.w3.org/XML/Query/` (Page consultée le 30 aout 2009).

[W3C, 2009c] W3C. XSL Transformation (XSLT). [En ligne]. `http://www.w3.org/TR/xslt` (Page consultée le 30 aout 2009).

[Wang et Shan, 2007] G. G. WANG et S. SHAN. Review of metamodeling techniques in support of engineering design optimization. *Journal of Mechanical Design*, 129 :370, 2007.

[Wikipedia, 2009a] WIKIPEDIA. Computer simulation. [En ligne]. `http://en.wikipedia.org/wiki/Computer_simulation` (Page consultée le 30 aout 2009).

[Wikipedia, 2009b] WIKIPEDIA. Extensible markup language (XML). [En ligne]. `http://en.wikipedia.org/wiki/XML` (Page consultée le 30 aout 2009).

[Wikipedia, 2009c] WIKIPEDIA. XML data binding. [En ligne]. `http://en.wikipedia.org/wiki/XML_data_binding` (Page consultée le 30 aout 2009).

[Wikipedia, 2009d] WIKIPEDIA. XML database. [En ligne]. `http://en.wikipedia.org/wiki/XML_database` (Page consultée le 30 aout 2009).

[Wikipedia, 2009e] WIKIPEDIA. XPath. [En ligne]. `http://en.wikipedia.org/wiki/XPath` (Page consultée le 30 aout 2009).

[Winfield, 1998] A. J. WINFIELD. A virtual laboratory notebook for simulation models. Dans *Pacific Symposium on Biocomputing*, volume 177, page 88, 1998.

[Wood *et al.*, 2003] J. WOOD, K. BRODLIE, et J. WALTON. gViz – visualization and steering for the Grid. *e-Science All Hands Meeting*, 2003.

[Wright, 2004] H. WRIGHT. Putting visualization first in computational steering. Dans *UK e-Science All Hands Meeting*, pages 326–331, 2004.

[Wright et Walton, 1996] H. WRIGHT et J. WALTON. HyperScribe : A data management facility for the dataflow visualization. Rapport Technique, IRIS Explorer Technical Report IETR/4, NAG Ltd., 1996.

[Wu, 2002] H. F. WU. *Spectral Analysis and Sonification of Simulation Data Generated in a Frequency Domain Experiment.* Thèse de doctorat, Naval Postgraduate School, 2002.

[X. J. Technologies, 2009] X. J. TECHNOLOGIES. Anylogic 6. [En ligne]. `http://www.xjtek.com/anylogic/` (Page consultée le 30 aout 2009).

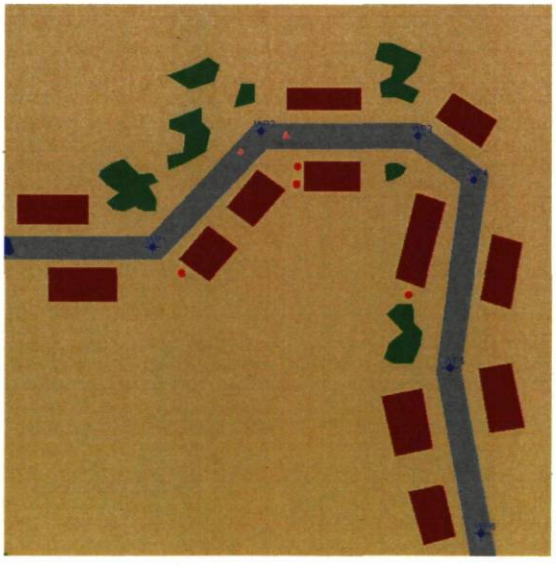
[Yau *et al.*, 2006] S. M. YAU, E. GRINSPUN, V. KARAMCHETI, et D. ZORIN. SimX : Parallel system software for interactive multi-experiment computational studies. Dans *International Parallel and Distributed Processing Symposium*, page 10, 2006.

[Zeigler *et al.*, 2000] B. P. ZEIGLER, H. PRAEHOFER, et T. G. KIM. *Theory of Modeling and Simulation.* Academic Press, 2000.

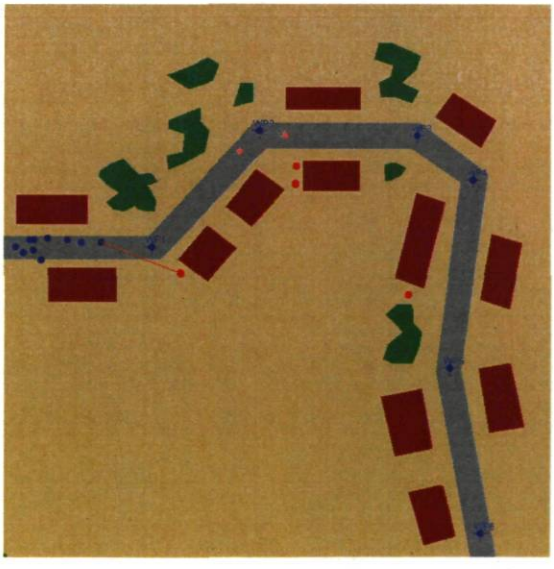
Annexe A

Dynamique du scénario d'attaque de convoi militaire

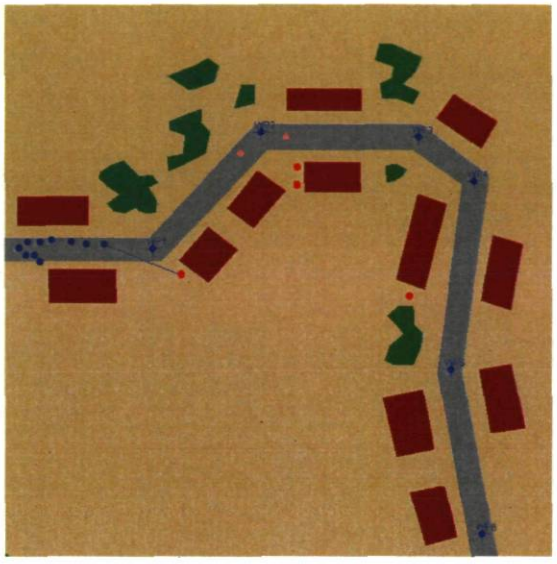
1) Emplacement initial du convoi qui est initialement groupé (points bleus). Sur la route, 4 insurgés de type RPG (points rouges), 1 agent IED faux positif et 1 agent IED non détecté (points roses).



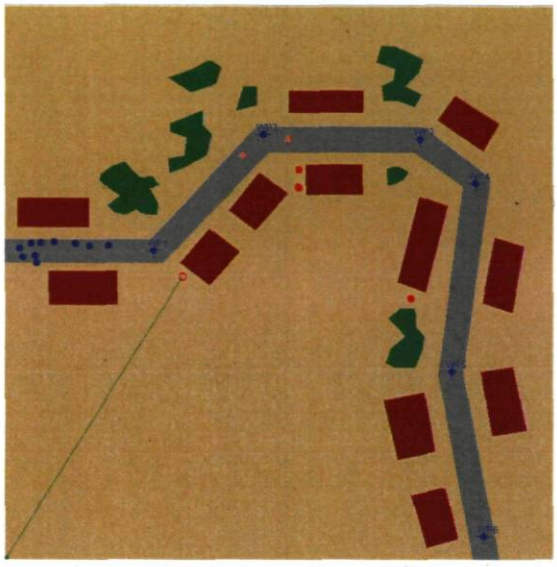
2) Le senseur du premier insurgé RPG détecte un véhicule du convoi puisqu'il entre dans sa zone de couverture (120 unités). L'insurgé attaque le convoi avec le RPG immédiatement. Des dommages sont causés au premier véhicule qui est représenté par un symbole bleu plus pâle.



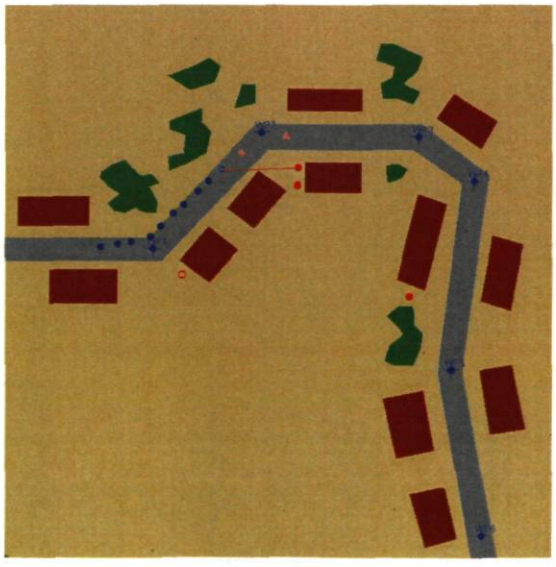
3) Le convoi continue de se déplacer à une vitesse plus lente, étant en mode « défense ». Le premier véhicule réplique à l'insurgé.



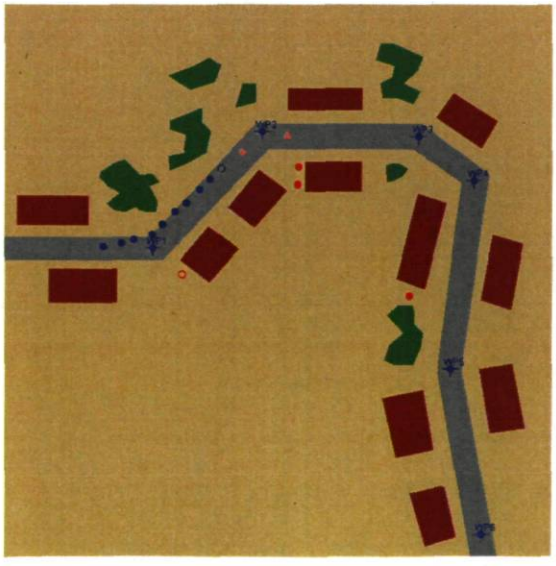
4) L'insurgé est éliminé, d'où le cercle vide qui le représente. Le convoi continue sa route vers le prochain point de rencontre.



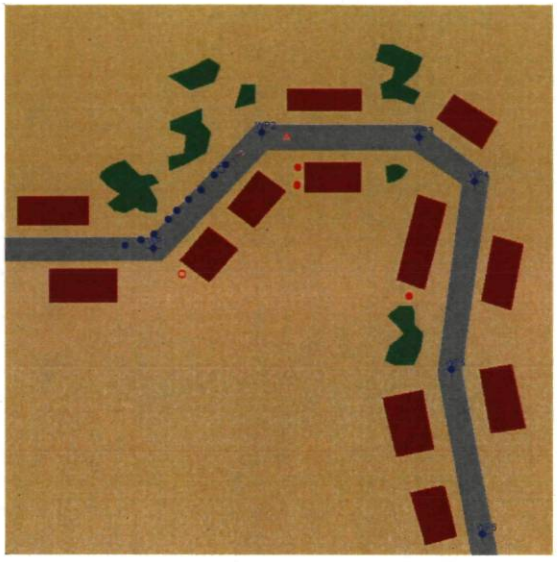
5) Le premier véhicule du convoi est attaqué à nouveau lorsqu'il entre dans la zone de couverture d'un insurgé RPG.



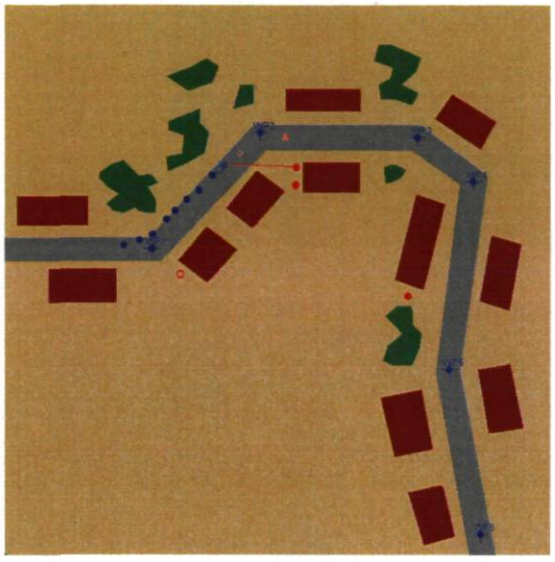
6) Le premier véhicule du convoi est éliminé. Les autres véhicules se mettent dans l'état « défense » pour 5 pas de temps en se déplaçant plus lentement, mais ne parviennent par à repérer le RPG qui a attaqué le premier véhicule du convoi. Le convoi retourne à l'état de mouvement initial.



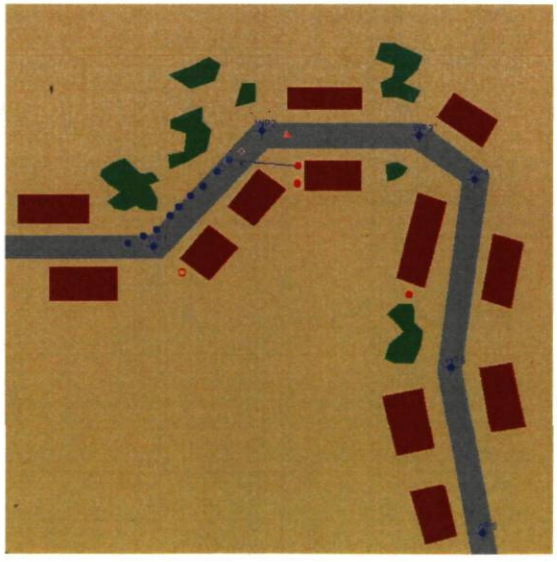
7) Le premier IED est désarmé lorsque repéré par le nouveau véhicule de tête du convoi. Le convoi est alors stoppé pour 10 pas de temps.



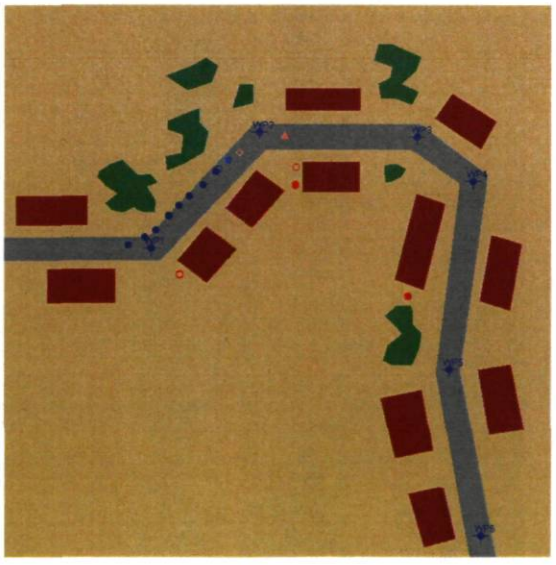
8) Pendant que les véhicules du convoi désarment le IED, l'insurgé RPG a eu le temps de se réarmer et il tire donc sur le véhicule de tête du convoi.



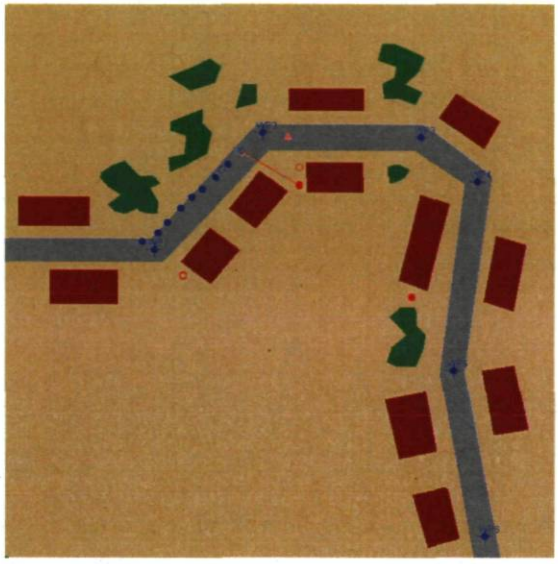
9) Les véhicules du convoi entrent en mode « défense » et répliquent à l'insurgé RPG.



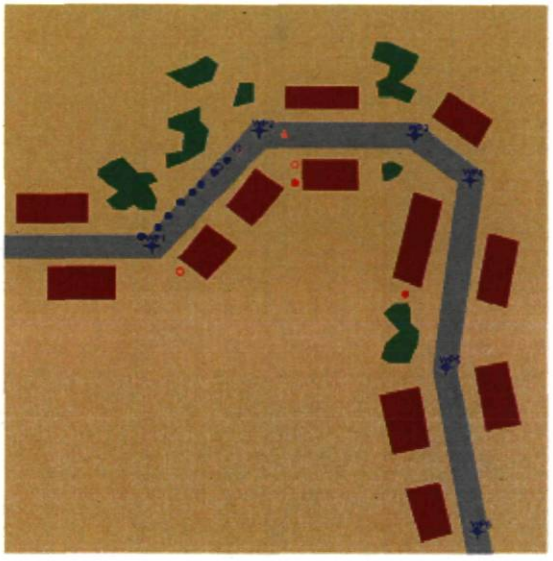
10) L'insurgé RPG est éliminé. Le convoi poursuit sa route vers le prochain point de rencontre.



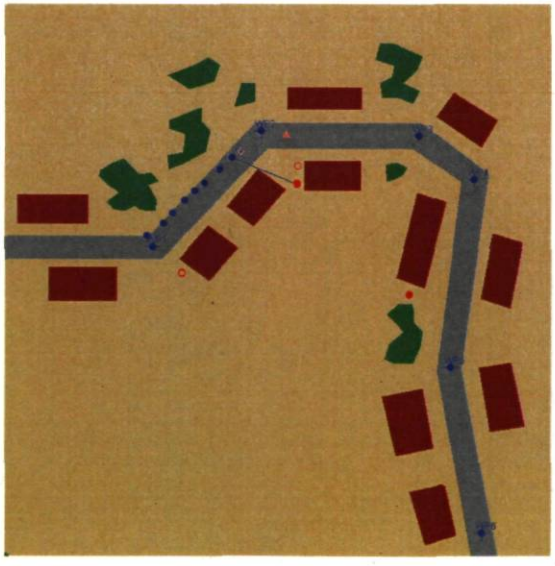
11) Le premier véhicule du convoi est attaqué par un autre insurgé RPG.



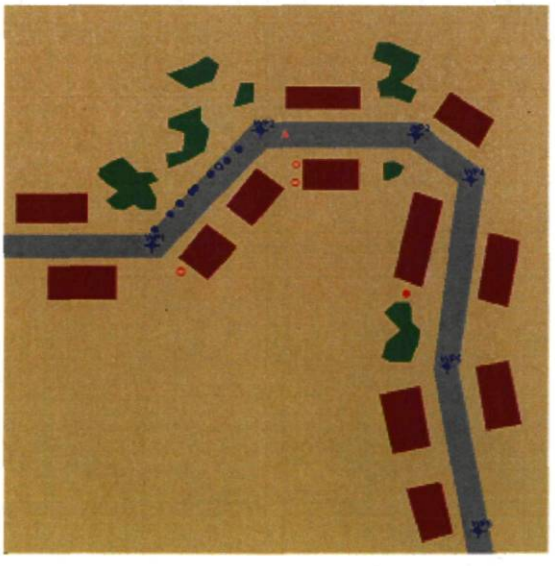
12) Le véhicule de tête est éliminé et les autres membres du convoi entrent dans le mode « défense ».



13) Le véhicule de tête réplique au tir du RPG lorsqu'il entre dans sa ligne de vue.

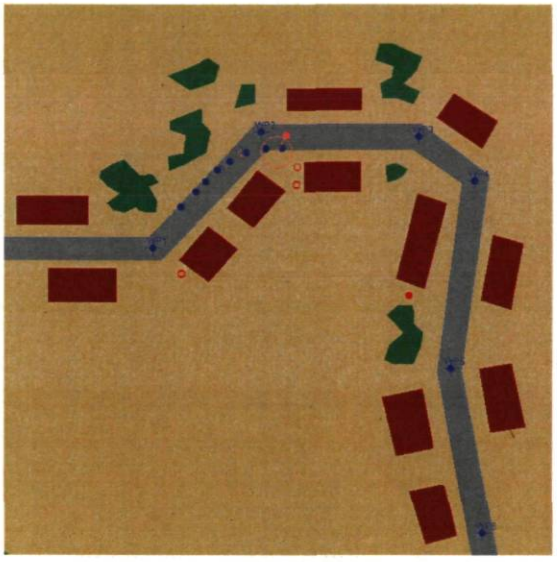


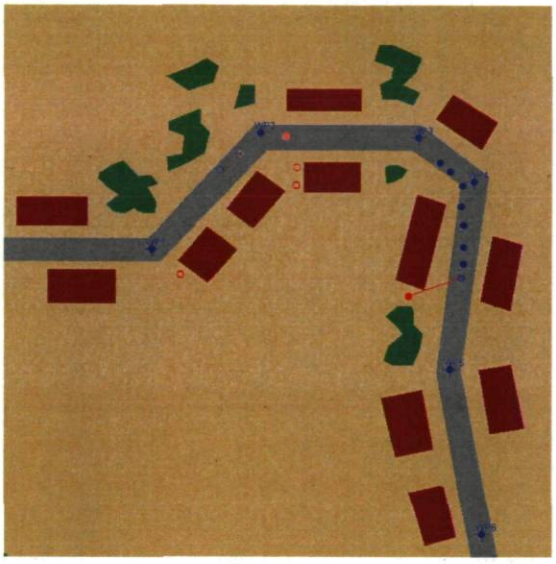
14) L'insurgé est éliminé et le convoi continue son chemin vers le prochain point de rencontre.



15) Un IED explose lorsque le premier véhicule s'en approche à moins de 20 unités. Le véhicule de tête est endommagé et le convoi tombe dans l'état « défense » pour 5 pas de temps, mais ne perçoit pas l'ennemi dans sa ligne de vue. Le convoi continue vers le prochain point de rencontre.

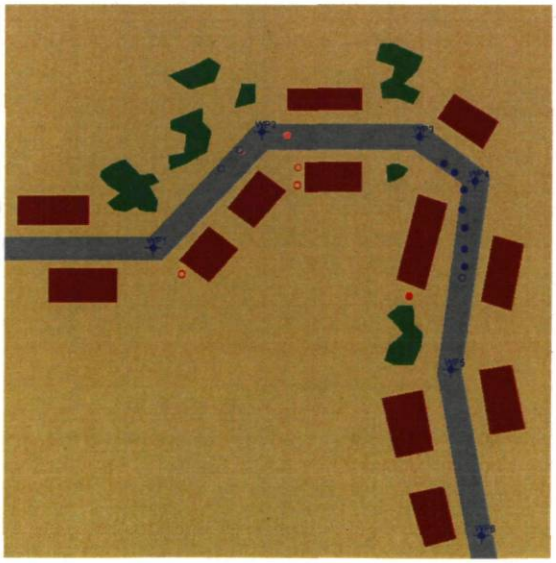
16) Un insurgé RPG attaque le premier véhicule du convoi, l'éliminant.

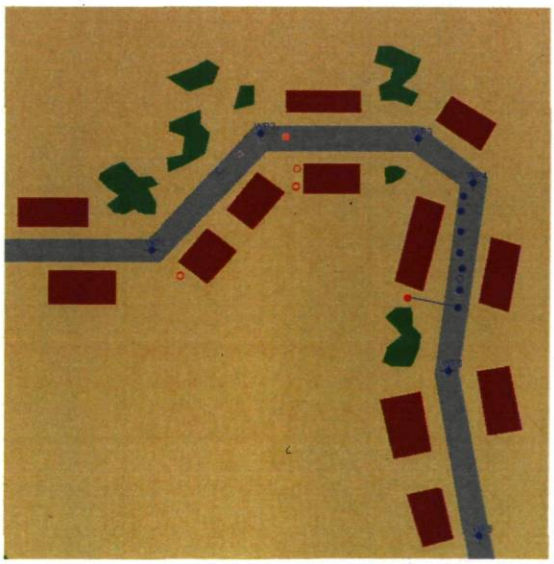




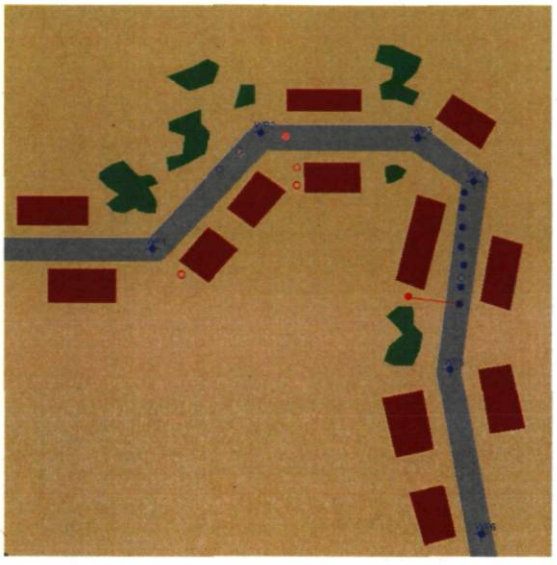
17) Les autres membres du convoi tombent dans l'état « défense » pour 5 pas de temps, mais ne réussissent pas à détecter l'ennemi.

18) L'insurgé attaque de nouveau après 10 pas de temps de réarmement.





19) Le véhicule de tête réplique à l'insurgé qui est éliminé. Le convoi poursuit sa route vers la destination finale.



20) Le convoi arrive à destination, puis s'arrête.