Université de Montréal

Développement d'un outil générique de simulation distribuée de marchés électroniques basés sur les enchères

Par

Mohamed Ali Khemila

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté à la faculté des études supérieures en vue de l'obtention du grade de Maître ès sciences (M.Sc.) en informatique

Avril, 2004

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

Développement d'un outil générique de simulation distribuée de marchés basés sur les enchères

Présenté par :

Mohamed Ali Khemila

a été évalué par un jury composé des personnes suivantes :

Jean Vaucher	président rapporteur
Peter Kropf	directeur de recherche
Teodor Gabriel Crainic	co-directeur de recherche
Michel Gendreau	membre du jury

Mémoire accepté le : 23 septembre 2004

Résumé

Traditionnellement, une simulation est implantée par un programme s'exécutant sur une machine mono-processeur. Dans le cas de simulations manipulant un grand nombre d'événements, ce type d'implantations engendre des temps d'exécution inacceptables. Pour résoudre ce problème, les techniques de simulation distribuées ont été développées pour permettre aux programmes de simulation de profiter du potentiel d'exécution offert par les environnements distribués. La simulation distribuée est utilisée dans beaucoup de domaines, tels que le domaine militaire, académique et industriel. La simulation de marchés électroniques forme un domaine particulier où les techniques de simulation distribuée sont de plus en plus utilisées.

Dans ce mémoire, nous avons développé un outil générique permettant le développement de simulations de prototypes de marchés électroniques basés sur les enchères, développés dans le cadre du projet TEM (Towards Electronic Marketplaces)[CIRANO00]. L'outil offre une librairie de classes C++ permettant de développer de telles simulations. En premier lieu, nous avons étudié l'environnement de simulation sur lequel se base notre outil, à savoir l'environnement *High Level Architecture* (HLA). En deuxième lieu, nous avons fait une étude des marchés ciblés par la simulation. Enfin, nous avons développé notre outil et nous l'avons testé sur le cas particulier du marché financier.

Mots clés : simulation distribuée, High Level Architecture, marchés électroniques, enchères.

Abstract

A computer simulation is traditionally implemented as a single program running on a mono-processor machine. In the case of a simulation manipulating a large number of events, this kind of implementation may cause an unacceptable execution time. To resolve this problem, distributed simulation was introduced to allow simulation programs to use the execution potential of distributed environments. Distributed simulation is used in many fields such as military, research and industry. Electronic marketplaces simulation is a particular field in which distributed simulation techniques are being increasingly used.

In this thesis, we developed a generic tool for the development of auction-based electronic marketplaces simulations, developed in the context of TEM (Towards Electronic Marketplaces) project [CIRANO00]. The tool consists of a library of C++ classes allowing the development of such simulations. As a first step, we studied the *High Level Architecture (HLA)* simulation environment on which our simulation tool is based. As a second step, we studied the marketplaces targeted by our simulation tool. Finally, we developed our tool and tested it in the particular case of the financial marketplace.

Key words: Distributed simulation, High Level Architecture, electronic marketplaces, auctions.

Table des matières

Liste des figures

Liste des abréviations

DDM	Data Distribution Management.
DMSO	Defense Modeling and Simulation Office.
DOD	Department of Defense.
FED	Federation Execution Data.
FOM	Federation Object Model.
HLA	High Level Architecture.
MOM	Management Object Model.
OMT	Object Model Template.
PL	Processus Logique.
RO	Receive Order.
RTI	Run Time Infrastructure.
SOM	Simulation Object Model.
TSO	Timestamp Order.

Remerciements

Je tiens à remercier toute personne qui a participé, de près ou de loin, à l'élaboration de ce travail. J'adresse, en particulier, mes vifs remerciements à mon directeur de recherche le professeur Peter Kropf et à mon co-directeur de recherche le professeur Teodor Gabriel Crainic pour leur effort, leur support et leur patience, qui m'ont permis de mener à bien ce travail.

Jawad Abrache, ami, ex-étudiant en Ph.D au DIRO, et membre du projet TEM, a été très patient et très serviable, et m'a apporté son aide sur tous les plans, tout au long de ce mémoire. Je tiens donc à le remercier vivement pour tout ce qu'il a fait.

J'adresse également mes vifs remerciements aux deux stagiaires au Centre de Recherche sur les Transports, Thomas Stire et Christophe Prevost, qui ont participé à l'élaboration de ce travail pendant la durée de leur stage de six mois. Ils ont, tous deux, fait preuve d'intelligence, d'assiduité et de compétence.

Chapitre 1
Introduction

Les techniques de simulation sont connues et utilisées depuis longtemps pour l'étude de phénomènes trop complexes pour être abordés de façon analytique. Dans son sens général, une simulation est un système qui représente ou émule le comportement à travers le temps d'un autre système, appelé *système physique* [Fujimoto00]. La plupart des systèmes physiques sont trop complexes pour être simulés directement. C'est généralement un *modèle* du système qu'on simule. Un modèle est une abstraction du système physique qui ne représente que les aspects importants de ce système. Les différents objets du système sont représentés dans le modèle par des *entités*. Une entité est caractérisée par des *attributs* dont les valeurs représentent l'état de l'entité à un instant donné. Ces attributs forment un sous-ensemble de l'ensemble des variables, dites *variables d'état*, utilisées par le modèle pour représenter l'*état du système* à un instant donné.

Une grande partie des systèmes physiques sont simulés par la technique de simulation à événements discrets, qui modélise un système dont l'état ne change qu'à un certain nombre de points discrets dans le temps. A cause de la nature dynamique des modèles de simulation à événements discrets, on doit garder trace du temps courant de la simulation le long du déroulement de la simulation [Law00]. Le temps est représenté par une variable appelée *horloge de simulation*. Historiquement, deux mécanismes d'avancement du temps de la simulation ont eu le plus grand succès : l'avancement du temps par en mode *event-driven* et l'avancement du temps en mode *time-stepped*. En mode event-driven, une action qui a lieu dans le système réel et qui peut causer le changement de l'état du système est modélisée par un *événement*. Les événements ont lieu instantanément et à chaque type d'événement est associée une routine d'événement qui implante les actions qui doivent avoir lieu à l'occurrence de l'événement. Le temps de la simulation avance toujours vers le temps où doit avoir lieu le prochain événement. En

mode time-stepped, le temps de la simulation est avancé avec des incréments fixes et à chaque incrément, des changements d'état du système peuvent avoir lieu.

L'approche *orientée processus* est une autre approche de modélisation de simulation couverte par la plupart des outils de simulation contemporains. Dans cette approche, la simulation est vue en terme des entités individuelles impliquées, et le code développé décrit les mouvements d'une entité particulière pendant son évolution à travers le système. Un *processus* est une séquence d'événements ordonnés par le temps et logiquement reliés, qui décrit l'évolution d'une entité à travers le temps [law00]. A chaque type de processus est associée une *routine de processus* qui implante la séquence d'actions décrivant le mouvement de l'entité à travers le processus correspondant. Une simulation adoptant l'approche orientée processus s'exécute, en arrière plan, d'une manière très similaire à celle d'une simulation en mode event-driven.

Quelle que soit l'approche de modélisation utilisée, une simulation à événements discrets est traditionnellement implantée à travers un seul programme, appelé *simulation séquentielle*, qui traite les événements séquentiellement selon l'ordre de leur occurrence. Ce mode d'implantation reste une bonne alternative tant que le nombre d'événements traités est raisonnablement petit. Cependant, dans beaucoup de cas, les simulations traitent un nombre considérable d'événements. C'est le cas, par exemple, d'une simulation du trafic dans un réseau informatique connectant des milliers de nœuds. Dans un tel cas, la simulation séquentielle peut prendre des heures, voire des jours, pour s'exécuter. Lorsque la simulation doit être exécutée plusieurs fois dans le but de tester différents scénarios, le temps d'exécution devient encore plus contraignant. Dans ce genre de situations, l'utilisation d'une simulation distribuée peut réduire considérablement le temps d'exécution. Une simulation distribuée est composée de plusieurs simulateurs séquentiels, distribués sur plusieurs machines, où chaque simulateur simule une partie du système réel (une ou plusieurs entités), et qui interagissent en échangeant des événements. Dans l'exemple d'une simulation d'un réseau informatique, chaque équipement dans le réseau peut être simulé par un simulateur séquentiel qui

s'exécute sur une machine différente. Lorsqu'un simulateur génère un événement qui doit être traité par un autre simulateur, l'événement est envoyé au simulateur en question.

La simulation distribuée est utilisée dans beaucoup de domaines, tels que le domaine militaire, commercial et de recherche scientifique. La simulation de marchés électroniques forme un domaine particulier où les techniques de simulation distribuée sont de plus en plus utilisées. Dans ce mémoire, nous nous intéressons à la simulation distribuée de marchés électroniques développés dans le cadre du projet TEM (Towards Electronic Marketplace) [CIRANO00]. Dans le cadre du projet TEM, on s'intéresse uniquement aux marchés électroniques basés sur les enchères, c'est-à-dire des marchés où des participants interagissent pour transiger des biens et des services, à travers des enchères électroniques organisées par le marché. Le terme marché désigne un ensemble de *places de marché électroniques* -aires où les enchères ont effectivement lieu- où on transige les objets de même nature. Chaque place de marché est un site électronique qui organise ses enchères de manière indépendante des autres places de marché. L'intéressé à une enchère peut y participer en utilisant un programme propre qui interagit avec la place de marché, et qui utilise un outil d'aide à la décision, appelé *aviseur*, qui assiste le participant dans son comportement dans le marché. On peut imaginer, par exemple, un marché électronique où on transige des titres bousiers à travers des enchères, au niveau du Canada. Ce marché peut être composé de trois places de marché électroniques, la première à Montréal, la deuxième à Toronto et la troisième à Vancouver, où chaque place de marché organise ses enchères, à travers son propre site électronique, indépendamment des autres.

Dans sa forme la plus simple, une enchère électronique ressemble à l'enchère utilisée traditionnellement pour vendre des biens, appelée enchère anglaise, dans le sens où un bien est déclaré pour vente et des participants surenchèrent pour le gagner, et le bien est alloué au participant qui offre le meilleur prix. Cependant, cette forme d'enchère reste très limitée pour répondre aux besoins précis des participants en terme d'achats et de ventes. En effet, à un instant donné, plusieurs objets peuvent être annoncés pour enchère dans le même marché, et les intérêts des participants à ces objets peuvent être

interdépendants. Ainsi, un participant peut s'intéresser à acquérir deux objets A et B à la fois, ce qui ne peut lui être garanti lorsque ces objets sont transigés dans deux enchères indépendantes puisque dans ce cas, il doit gagner les deux enchères à la fois. Pour résoudre ce problème, la notion d'*enchère combinatoire* a été introduite dans les marchés dans le but de permettre aux participants de miser sur des « paquets » d'objets avec un prix global. De façon plus générale, dans une enchère combinatoire, un participant peut miser sur une combinaison d'objets où il déclare sa volonté d'acheter, de vendre ou d'acheter et de vendre simultanément plusieurs objets différents. Par exemple, si un participant s'intéresse à procurer à la fois deux objets A et B, alors l'enchère doit lui permettre de miser sur la combinaison (A ET B) avec un prix global. En utilisant différents opérateurs, la combinaison peut prendre différentes formes telles que (A OU B), ((A ET B) OU (A ET C)), etc..

Un marché basé sur les enchères se caractérise par différents aspects qui font l'objet de recherche dans le domaine, dont les plus importants sont les suivants :

- *L'utilisation d'un mécanisme d'enchère* Le développement de mécanismes qui répondent aux besoins des différents cas de marchés est un axe de recherche principal dans le domaine. Pour certains cas de marchés, une enchère aussi simple que l'enchère anglaise peut répondre aux besoins des participants dans le sens où elle leur permet de déclarer leurs intérêts aux objets avec précision, mais pour d'autres cas de marchés, des mécanismes d'enchères plus complexes doivent être développés. Dans le cas d'enchères combinatoires, un défi principal est l'optimisation du problème dit d'allocation et de calcul de prix qui consiste à déterminer les gagnants de l'enchère et de calculer les prix à payer contre les objets alloués, vu le grand nombre de combinaisons possibles pour les mises.

- *L'utilisation d'un vocabulaire de mises* Ce dernier définit un ensemble d'opérateurs logiques permis pour la formulation des mises, ainsi que la manière dont ces opérateurs peuvent être combinés pour construire ces mises.

- *L'utilisation de règles de marchés* Ces dernières déterminent l'ensemble des critères utilisés par le marché pour prendre des décisions reliées au déroulement de l'enchère et à l'acceptation des requêtes des participants.

- *L'utilisation d'aviseurs par les participants* La conception et le développement d'aviseurs performants et optimisés est un axe de recherche principal dans le domaine.

L'un des objectifs finaux du projet TEM est de proposer des prototypes de marchés bâtis autour de ces aspects. Pour évaluer ces prototypes en terme d'utilité économique et de performance, le projet de simulation de marchés, qui fait l'objet de notre mémoire, a été mis en place. Plus précisément, la simulation vise à étudier, entre autres, les aspects suivants :

- la performance des algorithmes d'allocations,

- l'influence du vocabulaire des mises sur les marchés,

- l'influence des règles de marchés sur le déroulement de ce dernier,

- l'évolution du marché sur des périodes de différentes durées,

- la performance des aviseurs.

L'objectif de notre projet n'est toutefois pas de développer des simulations d'instances particulières de marchés, mais c'est de créer un outil (une librairie) qui permet de développer facilement des simulations de marchés basés sur les enchères. En effet, bien que chaque instance de marché ait ses propres caractéristiques, tous les marchés ont des caractéristiques communes, que nous identifierons dans le chapitre 5, tant au niveau de leur structure qu'au niveau de leur exécution. L'implantation de ces aspects communs une fois pour toutes permet donc d'éviter de reprendre à zéro le développement de chaque simulation de marché, ce qui est un processus coûteux. L'outil doit permettre à

l'utilisateur de développer une simulation de marché en implantant uniquement les aspects spécifiques au marché en question.

Un autre objectif important de notre travail est de dégager, au maximum, l'utilisateur qui se sert de notre outil pour développer une simulation de marché, des efforts reliés à la gestion de la simulation qui incluent, entre autres, la gestion du temps, la gestion des événements, la génération des statistiques de la simulation et la gestion des communications entre les simulateurs. Tous ces aspects doivent être gérés par notre outil de manière transparente à l'utilisateur, ce dernier doit plutôt se soucier des aspects reliés aux particularités du marché simulé.

De plus, notre outil est destiné pour développer des simulations distribuées et non séquentielles. En effet, deux raisons principales sont derrière ce choix :

- Une simulation de marché électronique basé sur les enchères traite souvent un très grand nombre d'évènements, et donc un programme de simulation séquentielle risque de donner des résultats en un temps inacceptable.

- Un algorithme d'allocations et de calcul de prix peut prendre une période de temps relativement longue pour donner des résultats, vu le grand nombre de combinaisons dans le cas d'une enchère combinatoire. Dans le cas d'un marché où des enchères s'exécutent simultanément dans des places de marché différentes, l'application séquentielle des algorithmes d'allocation peut prendre un temps inacceptable.

Un autre aspect important de notre travail est de permettre la ré-utilisabilité de simulateurs dans différentes simulations. En effet, puisqu'une simulation distribuée est composée de plusieurs simulateurs séquentiels en interaction, il serait bénéfique qu'un simulateur développé dans le cadre d'une simulation distribuée puisse être réutilisé dans le cadre d'une autre simulation distribuée avec un minimum d'effort d'intégration, ce qui évite le re-développement à zéro du simulateur. Par exemple, un simulateur d'un participant utilisé dans le cadre d'une simulation d'un marché particulier, doit être réutilisable dans le cadre d'une simulation d'un autre marché avec un minimum de

modifications. Cette idée a été un critère principal derrière le choix de l'environnement *High Level Architecture* en tant que plate-forme de simulation sur laquelle se base notre outil.

1-1 Contributions

La simulation des prototypes de marchés développés dans le cadre du projet TEM est indispensable pour l'évaluation de ces prototypes. Or, d'après nos recherches, il n'existe pas d'outils de simulation connus dédiés aux simulations de marchés électroniques basés sur les enchères. Une solution possible consiste en l'utilisation d'outils de simulation distribuée génériques. Néanmoins, cette solution peut s'avérer très coûteuse. Notre outil vient donc, à ce niveau, offrir un moyen plus facile et plus direct pour le développement de simulations distribuées de marchés, en offrant une librairie dédiée à cet effet.

La librairie offre les moyens de développer des algorithmes d'allocations, des vocabulaires de mises, des règles de marché et des aviseurs, ainsi que des simulateurs de places de marché et des simulateurs de participants bâtis autour de ces éléments. La librairie permet également de définir une simulation d'un marché à partir des places de marchés et des participants développés. La librairie encapsule, en grande partie, l'implantation des aspects reliés aux techniques de simulation. L'utilisateur de la librairie n'a donc pas à se soucier de ces aspects mais doit plutôt concentrer ses efforts de développement sur les aspects reliés aux particularités du marché simulé.

En plus de la librairie, l'outil offre un programme générique qui joue le rôle de moniteur qui collecte dynamiquement les informations sur les activités de la simulation, en particulier, sur les interactions entre les simulateurs individuels. Les informations collectées servent à l'utilisateur comme données brutes pour le calcul de statistiques de la simulation.

Enfin, un simulateur individuel d'une place de marché ou d'un participant développé avec notre outil dans le cadre d'une simulation d'un marché particulier est re-utilisable

dans le cadre de simulations d'autres marchés, avec un minimum d'efforts d'intégration. Cette ré-utilisabilité est assurée, en grande partie, par l'environnement de simulation distribuée *High Level Architecture,* sur lequel nous nous sommes basés pour développer notre outil.

1-2 Structure du mémoire

Ce mémoire est organisé de la manière suivante : dans le chapitre 2, nous présentons une revue de littérature sur le domaine de la simulation distribuée. En deuxième lieu, dans le chapitre 3, nous présentons l'environnement de simulation distribuée High Level Architecture que nous avons choisi comme plate-forme de simulation pour les marchés électroniques. En troisième lieu, dans le chapitre 4, nous faisons une classification des marchés ciblés par la simulation. Dans le chapitre 5, nous présentons la solution que nous avons proposée, qui consiste en l'outil que nous avons développé. Ensuite, dans le chapitre 6, nous présentons notre preuve de concept, sur le cas particulier du marché financier. Et enfin, dans un dernier chapitre, nous concluons et nous donnons les perspectives de notre travail.

Chapitre 2
La simulation distribuée

2-1 La notion de simulation et de modélisation

Dans son sens général, une simulation est un système qui représente ou émule le comportement à travers le temps d'un autre système, appelé *système physique* [Fujimoto00]. Ce dernier peut être défini en tant que « une collection d'objets qui interagissent à travers le temps conformément à des règles spécifiques » [Fujimoto00]. Généralement, vue la complexité du système physique, on simule uniquement un modèle de ce dernier. Un *modèle* est une abstraction du système physique en le représentant sous forme de relations mathématiques qui décrivent les relations entre les objets du système [Law00]. Un modèle ne décrit pas tout le système physique mais uniquement les aspects de ce système concernés par la simulation.

Les différents objets du système sont représentés dans le modèle par des *entités*. Une entité est caractérisée par des *attributs* dont les valeurs représentent l'état de l'entité à un instant donné. Ces attributs forment un sous-ensemble de l'ensemble des variables, dites *variables d'état*, utilisées par le modèle pour représenter l'*état du système* à un instant donné. Dépendamment de la façon dont l'état du système évolue avec le temps, on distingue deux types de modèles : les *modèles continus* et les *modèles discrets*. Un modèle continu décrit un système dont l'état évolue de façon continue avec le temps. Cette évolution est souvent exprimée dans le modèle sous forme d'équations différentielles. L'évolution du système solaire est un exemple de système qui peut être modélisé par un modèle continu. Par contraste, un modèle discret décrit un système dont l'état ne change qu'à un certain nombre de points dans le temps où chaque point correspond à l'occurrence d'un événement. Prenons l'exemple d'une queue devant un comptoir bancaire. Un tel système peut être modélisé par un modèle discret dont l'état ne

change que lors de l'arrivée d'un nouveau client, du passage d'un client de la queue vers le comptoir ou du départ d'un client déjà servi.

Étant donné qu'un modèle discret est généralement simulé par la technique de simulation à événements discrets, et que l'étude qui fait l'objet de ce mémoire porte sur un contexte discret, nous nous concentrons sur les *simulations à événements discrets*. En ce qui concerne les simulations continues, le livre [Bennett95] constitue une bonne référence.

2-2 La simulation à événements discrets

Une simulation représente l'évolution de l'état d'un système à travers le temps. Le temps, dont la représentation et la gestion constitue un aspect fondamental de toute simulation, est représenté à travers une abstraction appelée *temps logique* [Fujimoto00]. Durant son exécution, une simulation à événements discrets est appelée à avancer son temps logique en appliquant un *mécanisme d'avancement du temps*. À chaque avancement du temps, la simulation traite des *événements*, met à jour les variables d'état et recalcule ses *compteurs de statistiques*. Dans la plupart des cas, le modèle simulé présente des aspects stochastiques qui dépendent de données aléatoires dont la génération nécessite l'utilisation d'un *générateur de nombres aléatoires*. Dans la suite de cette section, nous nous fixons pour objectif de détailler l'ensemble de ces notions.

2 - 2 - 1 Le temps dans la simulation

La signification du temps est souvent une source de confusion dans le domaine des simulations [Fujimoto98]. On distingue trois notions de temps:

- Le temps physique se réfère au temps du modèle physique simulé.
- Le temps logique est une modélisation du temps physique. Il se réfère à une représentation interne du temps physique, à travers une variable qui prend des valeurs dans un intervalle prédéterminé.

- Le temps d'horloge se réfère au temps réel pendant lequel s'exécute la simulation.

Reprenons l'exemple d'une simulation d'une queue devant un comptoir bancaire. Supposons qu'on veut simuler le guichet sur une période d'une seule journée qui s'étale entre 10h00 et 13h00. Dans ce cas, les événements « ouverture du comptoir » et « fermeture du comptoir » sont repérés dans le monde réel par leurs temps physique 10h00 et 13h00, respectivement. La représentation du temps physique dans la simulation peut être faite, par exemple, en faisant correspondre une minute de temps physique à une unité de temps logique. Ainsi, si on initialise le temps logique à t=0, et on suppose que cet instant correspond à l'instant 10h00 du monde physique, alors les événements «ouverture du comptoir » et « fermeture du comptoir » doivent avoir lieu, dans la simulation, aux temps logiques t=0 et t=180, respectivement.

Dans les simulations dites *as-fast-as-possible*, on ne fait pas de synchronisation du temps logique au temps d'horloge [Fujimoto00]. Pour mieux comprendre ce concept, reprenons l'exemple du comptoir bancaire. Le comptoir ouvre à 10h00 en temps physique, ce qui correspond à t=0 en temps logique. Supposons que le premier client arrive à 10h10, c'est-à-dire à t=10 en temps logique. Dans le monde physique, l'intervalle de temps entre l'ouverture du comptoir et l'arrivée du premier client correspond à 10 minutes de temps d'inactivité. Dans une simulation as-fast-as-possible, les temps d'inactivités sont sautés. Ainsi, le simulation saute de t=0 à t=10 et traite immédiatement l'inscription, simulant ainsi 10 minutes de temps physique en quelques fractions de secondes.

Supposons maintenant qu'on veut simuler le même comptoir bancaire mais en faisant correspondre une minute de temps physique à exactement une minute du temps d'horloge. Ceci veut dire que pour un temps logique t_0 donné, le passage de $t=t_0$ à $t= t_0+1$ dure exactement une minute en temps d'horloge (ou temps d'exécution). Si la tâche exécutée lors de ce passage dure moins d'une minute, le programme de simulation doit attendre la fin de la minute pour continuer. Dans ce cas de figures, des mécanismes de synchronisation doivent être prévus pour synchroniser le temps logique avec le temps d'horloge. On parle, dans ce cas, d'une *simulation en temps réel*. Il est évident que le choix d'implanter une simulation en temps réel doit être motivé par des arguments très

précis. C'est le cas, par exemple, d'une simulation d'entraînement militaire, où des humains interagissent avec un environnement virtuel, et ont donc besoin de percevoir l'environnement de simulation comme réel.

Dans un programme de simulation, le temps (logique) est représenté par une variable qui est initialisée au temps initial, au lancement de la simulation, et qui progresse toujours vers l'avant. Dans la suite de ce document, le mot « temps » désignera par défaut le temps logique d'une simulation. Si une autre signification est voulue elle sera explicitement indiquée.

2 - 2 - 2 Les événements

Dans le système physique, des actions ayant lieu à différents moments dans le temps entraînent le changement de l'état du système. Dans une simulation, une action est modélisée par un *événement*. Ce dernier est une structure qui porte des informations sur la nature de l'action, ainsi qu'une *étiquette de temps* qui indique le temps logique, dans le futur de la simulation, où l'action doit avoir lieu. Pour qu'un événement ait lieu, il faut que la simulation le prévoie à l'avance dans son futur : on dit que la simulation *programme* l'événement. L'événement est alors traité lorsque le temps logique de la simulation atteint celui de l'événement. Pour chaque type d'événement, la simulation implante une *routine d'événement* qui est appelée à chaque fois qu'un événement de ce type est à traiter. Traiter un événement revient donc à appeler la routine d'événement correspondante.

2 - 2 - 3 Mécanismes d'avancement du temps

Une simulation avance progressivement son temps, et à chaque fois que le temps d'un événement est atteint, cet événement est traité. Le résultat de ce traitement est le changement de l'état du système, et éventuellement la programmation de nouveaux événements. Par exemple, si on suppose qu'un client est servi par le comptoir en 3 unités de temps, alors le traitement d'un événement « passage d'un client au service » à un

instant logique t entraîne la programmation d'un événement « départ du client » à l'instant t+3 qui sera traité lorsque la simulation atteint ce temps logique.

La manière dont la simulation avance son temps est appelée *mécanisme d'avancement du temps*. Historiquement, deux mécanismes d'avancement de temps de la simulation ont eu le plus grand succès : l'avancement du temps en mode *time-stepped* et l'avancement du temps en mode *event-driven* [Fujimoto00] qui feront l'objet des deux sous-paragraphes suivants.

2-2-3-1 La simulation en mode time-stepped

Une simulation en mode time-stepped avance son temps à des pas de temps de longueur Δt fixée à l'avance, et au bout de chaque pas, le nouvel état du système est calculé. Tous les événements programmés dans l'intervalle de temps $[t,t+\Delta t)$ sont traités à $t+\Delta t$ et sont donc considérés comme simultanés même s'ils n'ont pas la même étiquette de temps. Lorsqu'il est important que ces événements soient traités dans l'ordre, il faut choisir un pas suffisamment petit pour les discrétiser. Comme mentionné dans [Law00], le mode time-stepped présente deux inconvénients majeurs : l'erreur introduite par le traitement des événements à l'intérieur d'un même pas au même moment, et la nécessité de décider quel événement traiter en premier lieu dans ce cas. En réalité, on n'a pas toujours besoin de mettre à jour l'état du système à chaque pas, mais la simulation time-stepped ne peut avancer son temps que de cette manière. Dans le cas où la simulation fait beaucoup de pas de temps sans mettre à jour l'état du système, il y a manifestement perte de ressources.

2-2-3-2 La simulation en mode event-driven

Dans une simulation event-driven, l'état du système reste inchangé jusqu'à l'occurrence d'un événement. Idéalement, le simulateur utilise une *queue d'événements* qui contient les événements qui ne sont pas encore traités. Programmer un événement revient à l'insérer dans la queue, et chaque événement traité est enlevé de la queue. Au début de la simulation, le simulateur initialise son temps à un temps initial et détermine ses

événements initiaux. Ensuite, il enlève de la queue l'événement le plus proche, avance le temps au temps de cet événement et le traite en appelant la routine d'événement correspondante. Le résultat du traitement de l'événement est la mise à jour de l'état du système, et éventuellement la programmation d'autres événements. Ensuite, le simulateur enlève de la queue l'événement le plus proche, et le même cycle a lieu, jusqu'à ce que la condition de fin de la simulation soit vérifiée. Chaque occurrence d'un événement représente donc un point dans le temps où le simulateur doit mettre à jour son état. Les avancements de temps se font donc à des pas variables. La simulation s'arrête lorsqu'une certaine condition d'arrêt est satisfaite. Plusieurs techniques peuvent être utilisées pour implanter la condition d'arrêt, dont nous nous contentons de présenter les trois techniques classiques suivantes :

- Atteinte d'un certain temps logique défini à l'avance. Par exemple, si on veut simuler un système sur l'intervalle de temps logique [0,60], alors la simulation s'arrête lorsque le prochain événement dans la queue a une étiquette de temps supérieure à 60.
- Utilisation d'un événement particulier « fin de simulation » qui est programmé dès que le temps de fin de la simulation est connu. Lorsque le prochain événement à traiter est l'événement « fin de simulation », la simulation s'arrête.
- Fin des événements dans la queue. Lorsque le dernier événement dans la queue est traité, la simulation s'arrête.

2 - 2 - 4 Aspects stochastiques dans les simulations

La plupart des modèles de systèmes présentent des aspects qui dépendent de données aléatoires. Ainsi, dans l'exemple de la queue devant un comptoir bancaire, l'arrivée d'un nouveau client et le temps qu'il passe devant le guichet sont des phénomènes aléatoires. De tels modèles font des suppositions sur la distribution des événements aléatoires. On peut supposer, dans le modèle exemple, que les clients arrivent selon une distribution de poisson. Pour faire l'échantillonnage de telles distributions, le programme de simulation utilise un *générateur de nombres aléatoires.*

2 - 2 - 5 Collecte des statistiques

La collecte et l'analyse des statistiques est un objectif majeur de la simulation. Les statistiques portent sur l'état de certaines variables, de certaines ressources ou sur les performances du système. Dans notre exemple, la mesure du temps moyen d'attente d'un client dans la queue peut être une statistique importante. La collecte de statistiques se fait à fur et à mesure de l'évolution de la simulation. Au début de la simulation, les compteurs de statistiques sont initialisés, ensuite, au traitement de chaque événement, l'état du système est mis à jour ainsi que les compteurs de statistiques. L'analyse des statistiques collectées par la simulation utilise des techniques qui dépassent le cadre de notre mémoire, le lecteur intéressé peut se référer à [Law00].

2 - 2 - 6 L'exécutif de la simulation

Un programme de simulation discrète utilise typiquement trois structures de données [Law00] :

- les variables d'état qui décrivent l'état du système à un instant donné,
- une queue d'événements programmés dans le futur de la simulation,
- une horloge interne qui représente le temps logique actuel de la simulation.

La gestion de l'horloge et de la queue des événements sont généralement indépendants de la nature du modèle simulé. L'interface qui permet de gérer l'horloge et la liste des événements est généralement commercialisée par des vendeurs sous forme de logiciel de simulation appelé *exécutif de simulation*. La figure 2-1 montre les deux parties composant un programme de simulation. La partie inférieure de la figure représente l'exécutif de simulation. La partie développée par l'utilisateur, représentée dans la partie supérieure du schéma, est appelée *applicatif de simulation*. Cette dernière implante les entités du système, les variables d'état, les entrées/sorties et l'interface avec l'utilisateur.

Idéalement, l'applicatif de simulation initialise l'exécutif en lui fournissant une liste contenant un pointeur vers la routine d'événement pour chaque type d'événements. Par la suite, lorsque l'exécutif est appelé par l'applicatif pour avancer le temps, l'exécutif détermine les événements qui doivent être exécutés et appelle les routines d'événements correspondantes.

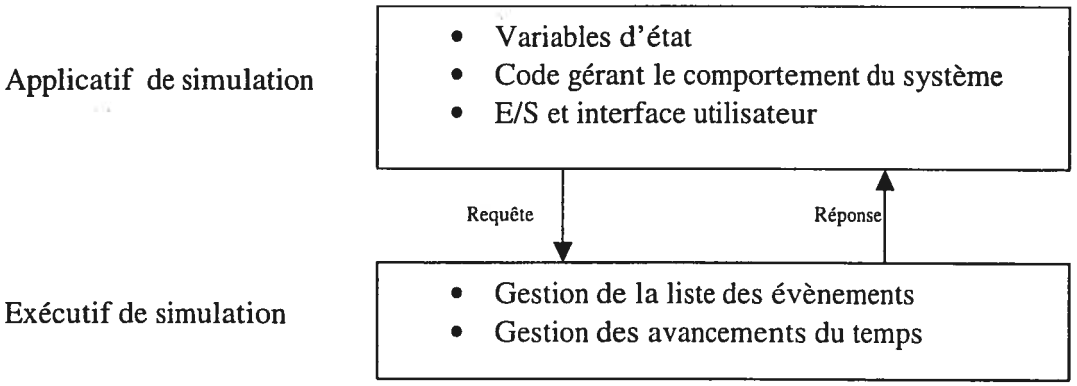

Figure 2-1 Composantes d'un programme de simulation.

2-3 La simulation distribuée à événements discrets

2 - 3 - 1 La notion de simulation distribuée

La simulation séquentielle a prouvé son inefficacité à résoudre les problèmes de grande taille. Pour illustrer cet fait, [Misra86] donne l'exemple d'un grand commutateur téléphonique capable de supporter 100 communications simultanées. Pour l'accomplissement d'un seul appel, le commutateur génère 100 messages internes. La simulation d'un tel commutateur pour une durée de 15 minutes de temps réel nécessite la simulation d'approximativement 10 millions d'événements, ce qui demanderait plusieurs heures d'exécution sur une machine uni-processeur très rapide. Les techniques utilisées traditionnellement dans le calcul parallèle et distribué ne sont pas appropriées aux programmes de simulation. En effet, plusieurs travaux ont été faits dans le but d'exploiter ces techniques dans le cadre de paralléllisation de programmes de simulation, mais elles

ont prouvé leurs limitations ([Fujimoto99], [Misra86]). Une technique spécifique à la paralléllisation et à la distribution des programmes de simulations a donc été développée pour donner naissance aux notions de *simulation parallèle* et *simulation distribuée*. Dans le reste de ce document, nous parlerons uniquement de la simulation distribuée, puisque c'est le sujet de notre mémoire.

Dans une simulation distribuée, le système physique est vu en tant qu'un ensemble de *processus physiques* qui interagissent en des points dans le temps. La simulation distribuée du système est constituée d'un ensemble de simulateurs séquentiels, modélisant chacun un processus physique, qui interagissent en échangeant des événements étiquetés par le temps. Ces simulateurs sont habituellement appelés des *processus logiques (PL)*, une appellation empruntée aux simulations parallèles. Chaque PL maintient une partie des variables d'état du système entier, celles qui représentent l'état du processus physique qu'il simule. Chaque PL maintient également sa propre queue des événements et son propre temps logique. Lorsqu'un PL crée un événement qui peut être approprié à un autre PL, il lui envoie un message contenant cet événement. Le PL récepteur du message programme alors l'événement correspondant dans sa queue d'événements.

N.B : pour rester conforme aux termes utilisés dans la plupart des articles sur la simulation distribuée, nous utiliserons, dans le reste de ce document, le terme « événement » lorsqu'il s'agit d'un événement généré localement par un simulateur, et le terme « message » lorsqu'il s'agit d'un événement généré par un simulateur et envoyé à un autre simulateur.

Pour illustrer la notion de simulation distribuée, prenons l'exemple d'une simulation distribuée du trafic aérien dans le réseau d'aéroports du monde. Chaque aéroport peut être vu comme un processus physique simulé par une simulation séquentielle (PL). Les différents PLs traitent deux types d'événements : « Départ d'un vol vers un aéroport X» et « Arrivée d'un vol provenant d'un aéroport Y ». Lorsqu'un simulateur d'un aéroport A traite l'événement « Départ d'un vol vers l'aéroport B » au temps logique t=T, alors il

envoie en conséquence au simulateur de B un message « Arrivée d'un vol de A » à t=T+ Δt où Δt est la durée du vol en temps logique. Lorsque B reçoit le message, il programme l'événement correspondant (« Arrivée d'un vol provenant de A ») à t=T+ Δt dans sa queue d'événements. Chaque PL traite donc à la fois les *événements internes* programmés par le PL lui-même et les *événements externes* qui lui proviennent des autres PLs.

Dans une simulation distribuée, il n'y a pas de notion de temps global de la simulation, dans la mesure où chaque PL maintient son propre temps logique. Ainsi, à un moment donné de la simulation, deux PLs peuvent être à des temps logiques différents. Cependant, puisque les simulateurs s'échangent des messages étiquetés par temps, il est clair qu'une uniformisation de la notion du temps entre les simulateurs est de mise. Dans une simulation distribuée, le temps utilisé par les PLs est toujours défini par rapport à une même référence appelée *axe de temps de la simulation.* En particulier, un PL qui reçoit un événement d'autres PLs doit interpréter la valeur de l'étiquette de temps de cet événement de la même manière dont il interprète ses événements internes. Ainsi, si un PL A reçoit un événement d'un PL B avec une étiquette de temps égale à 10, alors A doit traiter cet événement après un événement interne à t=9 et avant un événement interne à t=11.

L'exécutif de simulation est commun à tous les PLs. Généralement, c'est un programme indépendant qui communique avec tous les PLs à travers le réseau. Tous les échanges des événements doivent être faits à travers l'exécutif de simulation qui, en plus des services qu'il offre à chaque simulateur séquentiel, doit résoudre un problème relié à la gestion du temps, connu dans le domaine de simulations distribuées sous le nom de *problème de synchronisation.*

2 - 3 - 2 Le problème de synchronisation

Une simulation distribuée d'un système physique doit donner exactement le même résultat qu'une simulation séquentielle du même système. Or, si certaines précautions ne

sont pas prises, cet objectif ne pourra être atteint à cause du problème de synchronisation. Afin de mieux comprendre ce problème, il convient de reprendre l'exemple de la simulation distribuée du trafic aérien dans le réseau d'aéroports du monde. La figure 2-2 représente un scénario d'une simulation distribuée avec trois PLs : PL_A, PL_B et PL_C, s'exécutent sur des machines différentes et simulant respectivement les trois aéroports A,B et C. Dans cet exemple, PL_A envoie à PL_C un message *Arrivee(A)* avec une étiquette de temps t=100, pour signaler l'arrivée à l'aéroport C d'un avion provenant de A à t=100. De même, PL_B envoie à PL_C un message *Arrivee(B)* avec une étiquette de temps t=110, pour signaler l'arrivée à l'aéroport C d'un avion provenant de B à t=110. Toutefois, des problèmes de congestion du réseau font que PL_C reçoit le message *Arrivee(B)* avant le message *Arrivee(A)* et traite l'événement *Arrivee(B)* avant de recevoir l'événement *Arrivee(A).* Le fait que PL_C traite l'événement *Arrivee(B)* avant l'événement *Arrivee(A)* peut causer plusieurs problèmes. D'une part, le traitement de l'événement *Arrivee(B)* peut dépendre de certaines données générées par le traitement de *Arrivee(A),* et donc en traitant les deux événements dans le mauvais ordre, cette dépendance n'est pas prise en compte. De plus, dans ce cas, la simulation donne un résultat différent de celui donné par une simulation séquentielle du même système. Une simulation distribuée doit donc assurer qu'un PL ne traite un événement *E* que lorsqu'il a une garantie qu'il ne recevra pas, plus tard, un message avec une étiquette de temps inférieure à celle de *E.* D'une manière plus générale, une simulation distribuée doit s'assurer que chaque PL traite les événements internes et externes dans l'ordre de leurs étiquettes de temps. À cet égard, une question intéressante peut être soulevée : dans le cas du scénario de la figure 2-2, lorsque PL_C reçoit le message avec l'étiquette t=110, comment pourrait-il savoir s'il y aurait ou non d'autres messages avec une étiquette inférieure à t=110? Le problème d'assurer que les PLs d'une simulation distribuée traitent les événements dans l'ordre de leur étiquette de temps s'appelle *problème de synchronisation.* Un algorithme visant à résoudre ce problème est appelé *algorithme de synchronisation.*

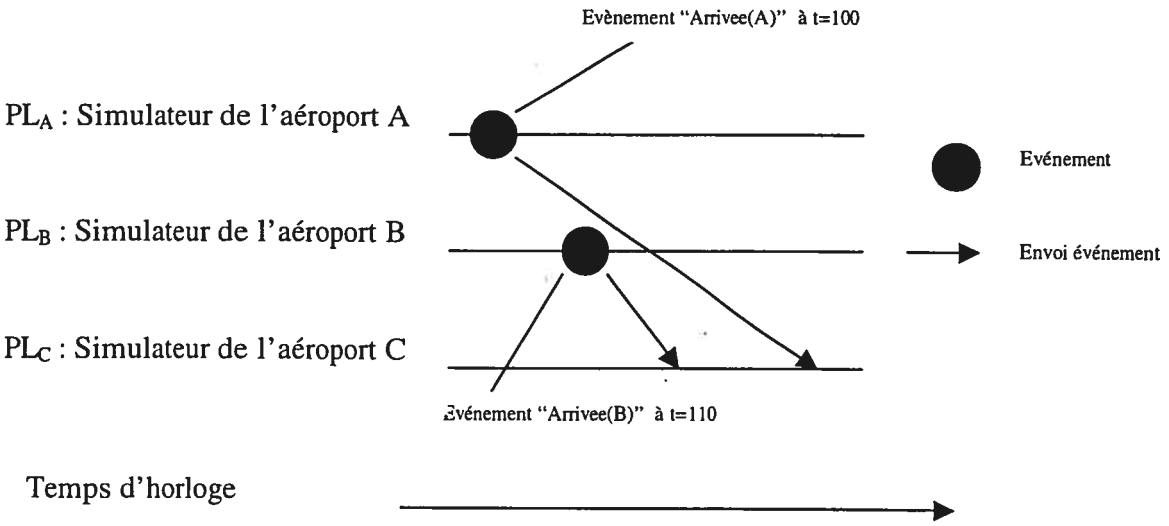

Figure 2-2 Le problème de synchronisation.

Le problème de synchronisation est au cœur même des simulations distribuées. Pour résoudre ce problème, deux approches, que nous décrivons dans les deux sous-paragraphes suivantes, ont historiquement eu le plus grand succès [Fujimoto00] : la *synchronisation conservatrice* et la *synchronisation optimiste*. Plusieurs algorithmes de synchronisation conservatrice et de synchronisation optimiste ont été développés et sont décrits dans [Fujimoto00] et [Ferscha94]. Leur description détaillée dépasse toutefois le cadre de ce mémoire. Nous allons nous contenter, à ce niveau, d'une description générale sommaire de ces deux approches, et décrire en détail, dans le chapitre 3, comment l'environnement High Level Architecture implante ces algorithmes.

2 - 3 - 3 La synchronisation conservatrice

La synchronisation conservatrice se base sur la *contrainte de causalité locale* définie comme suit :

> « Une simulation distribuée à événements discrets obéit à la contrainte de causalité locale si et seulement si chaque PL traite les événements dans l'ordre de leurs étiquettes de temps. »

Il est alors possible de démontrer que si chaque PL, dans une simulation distribuée d'un système, adhère à la contrainte de causalité locale, alors la simulation distribuée donne exactement les mêmes résultats qu'une simulation séquentielle du système, pourvu que les événements ayant des étiquettes de temps égales soient traités dans le même ordre [Fujimoto00].

Un algorithme basé sur l'approche conservatrice tend à empêcher les PLs de violer la contrainte de causalité locale. C'est-à-dire que les PLs traitent les événements dans l'ordre de leur étiquette de temps. En d'autres termes, l'algorithme détermine quand est-ce qu'un événement peut être traité par un PL avec une assurance qu'aucun événement avec une étiquette de temps inférieure ne sera reçu plus tard par le PL. Le PL ne doit pas traiter l'événement avant d'avoir cette assurance.

2 - 3 - 4 La synchronisation optimiste

La deuxième approche de synchronisation d'une simulation distribuée est l'approche basée sur les algorithmes optimistes. Contrairement aux algorithmes conservateurs qui évitent la violation de la contrainte de causalité locale, un algorithme optimiste autorise la violation de cette contrainte, mais il est capable de la détecter et de faire une reprise (rollback). Ce qui signifie que le PL doit faire une sauvegarde de son état juste avant le traitement du mauvais événement, ce qui implique l'utilisation d'une grande quantité de mémoire. L'environnement de simulation le plus connu qui utilise la synchronisation optimiste est l'environnement TimeWarp [TWOS].

2 - 3 - 5 Les outils de simulation distribuée

Actuellement, une multitude d'outils de simulation distribuée sont disponibles sur le marché et offrent des possibilités différentes. Ces environnements sont de deux types : les

langages de simulation et les librairies [Low99]. Un langage de simulation est comparable aux langages de programmation conventionnels, sauf qu'il offre des structures et des routines plus appropriées au développement des simulations distribuées. Une librairie n'offre qu'un ensemble de routines accessibles à travers un langage de programmation conventionnel comme C et C++. Dans [Low99], une étude détaillée a été faite dans le but de classifier et d'évaluer les différents outils de simulation distribuée. Pour notre part, nous nous contentons de donner une brève description de quelques outils populaires :

- Distributed SimJava [Mitre] : c'est un outil basé sur le langage Java qui permet le développement de simulations distribuées à événements discrets, et qui offre un service d'animation permettant de visualiser l'évolution des objets au cours de la simulation.
- ModSim [ModSim] : c'est un langage de simulation orienté objet basé sur la synchronisation optimiste. Il intègre également un service de graphisme et d'animation. Il est idéal pour le développement de simulations visuelles et interactives.
- WARPED [WARPED] : c'est une librairie orientée objet qui permet de développer des simulations séquentielles ou des simulations distribuées basées sur un mécanisme de synchronisation optimiste.
- TWOS [TWOS] : c'est une librairie qui permet de développer des simulations séquentielles ou des simulations distribuées basées sur un mécanisme de synchronisation optimiste. Elle offre également des fonctionnalités de visualisation graphique et d'animation.

Dans tous les environnements de simulation existants, deux aspects ont très souvent fait défaut : la ré-utilisabilité et l'interopérabilité des simulateurs. La ré-utilisabilité est la caractéristique de pouvoir utiliser un simulateur dans le cadre de projets de simulations distribuées différents. L'interopérabilité est la caractéristique de pouvoir intégrer un simulateur à une simulation distribuée sans recours à la re-écriture du code. Il y a quelques années, le Département de la Défense américain (DoD) a proposé une solution à

ces deux problèmes, qui se base sur l'idée suivante : si un standard existait qui définit la manière dont interagissent les simulateurs dans une simulation distribuée, alors il suffirait de développer les simulateurs conformément à ce standard pour garantir leur interopérabilité et leur ré-utilisabilité. Cette solution a mené au développement du standard High Level Architecture (HLA) qui sera le sujet de notre prochain chapitre.

Chapitre 3
High Level Architecture

3-1 Présentation générale

La simulation est un outil d'une importance capitale dans le domaine militaire. La défense américaine utilise particulièrement les simulations par ordinateur d'une façon très fréquente et dans des objectifs très variés : entraînements, tests, évaluations, etc.. L'Office de Simulation et de Modélisation (DMSO) du Département de la Défense américain (DoD) a lancé, en octobre 1993, le projet de développement d'un standard de simulation, qu'elle a nommé High Level Architecture (HLA), capable de répondre à tous les besoins du DoD en terme de simulations [DMSO00]. HLA définit une architecture standard pour les simulations distribuées, composée de trois éléments :

- *HLA rules* : c'est un ensemble de règles aux quelles une simulation distribuée doit obéir pour être conforme au standard HLA.
- *La spécification d'interface* : elle décrit l'architecture de l'exécutif de simulation, appelé *Run Time Infrastructure* ou *RTI*), ainsi que son interface avec les simulateurs.
- *Object Model Template (OMT)* : c'est un format standard de description des données échangées dans la simulation.

La spécification d'interface définit l'architecture de la RTI, ainsi que son interface avec les simulateurs, de manière abstraite, indépendamment de tout langage de programmation et de tout environnement d'exécution. De plus, les services offerts par la RTI sont indépendants du domaine simulé. C'est donc un exécutif à utilité générale. Puisque l'architecture de la RTI est définie d'une manière abstraite, différentes implantations de cette dernière sont possibles. Le DoD a même encouragé les entreprises privées à développer leur propre implantation de la RTI et à la commercialiser. Ce choix garantit que HLA résistera aux évolutions dans les technologies informatiques : bien que les

langages de programmation et les environnements d'exécution évoluent constamment, HLA garde ses caractéristiques puisqu'il suffit de l'implanter pour ces nouvelles technologies.

Bien que développée à l'origine pour être utilisée dans le domaine militaire, l'architecture HLA a gagné une grande popularité dans le domaine civil. Plusieurs organismes et particulièrement les centres de recherche ont commencé à développer des simulations HLA dans plusieurs domaines tels que la circulation routière et les chaînes de production. Cette popularité est due à la politique suivie par le DoD qui veut que HLA devienne un standard utilisé à une grande échelle et dans des domaines variés. Cette politique a été renforcée par les actions suivantes :

- Passation de la standardisation de HLA à l'organisme IEEE.
- Distribution gratuite des logiciels de support et des documentations.
- Création d'un centre de support technique et d'information sur HLA qui offre ses services gratuitement à tous les utilisateurs de HLA.

3-2 Aperçu sur les simulations HLA

Dans la terminologie de HLA, une simulation distribuée est appelée *fédération.* Elle est composée d'un ensemble de simulateurs (PLs), appelés *fédérés*, qui interagissent à travers l'exécutif RTI. Chaque fédéré peut simuler une ou plusieurs entités. Par exemple, dans une simulation du réseau d'aéroports du monde, un fédéré peut simuler tous les aéroports d'un pays particulier. Une fédération doit être créée dynamiquement par un de ses fédérés, généralement le premier lancé. Pour qu'un fédéré devienne membre de la fédération, il doit demander dynamiquement et explicitement, à la RTI, de *rejoindre* la fédération.

Dans une simulation HLA, une entité simulée par un fédéré et visible aux autres fédérés est représentée par un *objet HLA*. L'état d'un objet est défini par la valeur de ses *attributs* appelés *attributs de l'objet.* Un objet est instancié (par le fédéré qui le simule) d'une

classe d'objets HLA complètement définie par l'ensemble de ses attributs appelés *attributs de classe.* La RTI maintient tous les objets de la fédération, cependant les états des objets (les valeurs des attributs) sont maintenus par les fédérés ce qui garantit que la RTI reste indépendante du domaine simulé.

Pour un fédéré, simuler un objet revient à (1) informer le reste de la fédération sur l'existence de l'objet (2) diffuser l'information sur tout changement ultérieur de l'état de l'objet. L'action (1) est réalisée automatiquement par la RTI : dès l'instanciation de l'objet, la RTI informe les autres fédérés de son existence. On dit que la RTI *publie* l'objet et que les fédérés informés de l'existence de l'objet *découvrent* cet objet. L'action (2) est réalisée par le fédéré à travers l'opération de *mise à jour (Update)* qui consiste à informer la RTI, à travers un appel spécifique, du changement de la valeur d'un ou plusieurs attributs et de fournir les nouvelles valeurs. A son tour, la RTI *reflète* la mise à jour aux autres fédérés. Tout reflet d'une mise à jour d'un ou plusieurs attributs est considéré comme un message envoyé par le fédéré initiateur de la mise à jour à tout fédéré ayant reçu le reflet de cette mise à jour. En effet, la mise à jour sur les attributs d'objets est un des deux moyens utilisés par les fédérés HLA pour échanger les messages. Le deuxième moyen est l'envoi d'*interactions HLA*, que nous développerons plus loin dans ce paragraphe.

La simulation d'un objet peut être partagée entre plusieurs fédérés. Par exemple, dans une simulation de la circulation routière, un fédéré F1 peut simuler un véhicule en mettant à jour uniquement sa vitesse alors qu'il dépend d'un autre fédéré F2 (qui s'occupe de l'affichage sur écran) pour mettre à jour la position du véhicule. Dans ce cas, F1 *simule l'attribut Vitesse* de l'objet alors que F2 *simule l'attribut Position* du même objet. Un attribut doit être simulé par au plus un fédéré à la fois. On dit que le fédéré *possède* l'attribut. La possession d'un attribut peut être échangée dynamiquement au cours de la simulation.

Dans une fédération, les fédérés ne sont pas tous intéressés à recevoir tous les messages reliés aux mises à jour d'attributs générées par le reste des fédérés. Par exemple, dans une

fédération de simulation de la circulation routière, le fédéré simulant une automobile s'intéresse aux messages signalant le changement d'état du feu de circulation, par contre le simulateur du feu de circulation ne s'intéresse pas au changement d'état de l'automobile. HLA utilise le mécanisme de *publication* et de *souscription* pour permettre aux fédérés de déclarer les types de messages qu'ils sont capables d'envoyer et les types de messages qu'ils s'intéressent à recevoir. A travers ce mécanisme, chaque fédéré déclare à la RTI les classes des objets qu'il a l'intention d'instancier et les attributs de ces objets qu'il a l'intention de simuler (mettre à jour au cours de la simulation). On dit alors que le fédéré *publie* la classe en question avec les attributs en questions. Chaque fédéré déclare également les attributs de classes dont il s'intéresse à recevoir les mises à jour. On dit que le fédéré se *souscrit* aux attributs de classe en question. Lorsqu'un fédéré F enregistre (instancie) un objet, c'est uniquement les fédérés souscrits à au moins un attribut de la classe de l'objet qui découvrent l'objet. De plus, lorsque F effectue une mise à jour sur des attributs de l'objet, la RTI ne reflète à un fédéré que la mise à jour des attributs auxquels il est souscrit.

HLA offre un deuxième moyen d'échanger les messages : l'envoi d'*interactions HLA*, le premier moyen étant la mise à jour sur les attributs d'objets. Une interaction est utilisée pour envoyer un message qui n'est pas directement lié à un objet. Elle est composée de *paramètres* qui sont des informations atomiques comparables aux attributs dans le cas des objets. De manière similaire aux objets, une interaction est instanciée d'une *classe d'interactions* qui définit sa structure. L'instanciation se fait à travers une requête RTI et entraîne l'envoi immédiat de l'interaction aux autres fédérés. L'envoi et la réception des interactions est gouverné par une procédure de publication et de souscription similaire à celle utilisée par les objets. Toutefois, la publication d'une interaction et la souscription à une interaction doit toujours se faire avec tous les paramètres. Chaque interaction échangée est considérée comme un message. Tout comme pour les objets, un fédéré ne reçoit une interaction que lorsqu'il déclare explicitement à la RTI son intérêt à ce type d'interactions. La différence fondamentale entre un objet et une interaction est qu'un objet persiste alors qu'une interaction ne persiste pas dans la simulation.

Un fédéré peut optionnellement affecter une étiquette de temps à un événement. L'événement ainsi construit est appelé événement *TSO (Time Stamp Order)*, par opposition aux événements *RO (Receive Order)* qui ne portent pas d'étiquettes de temps. Pour les événements TSO, la RTI offre un service de livraison des événements par ordre de leur étiquette de temps.

HLA exige que chaque fédération doit obligatoirement avoir un document appelé *modèle d'objets de fédération ou FOM (Federation Object Model)* qui décrit les classes d'objets et d'interactions utilisées dans la fédération ainsi que d'autres informations sur la manière d'échanger les données entre les fédérés. Le FOM est une forme de contrat entre les fédérés sur les données échangées dans la simulation et les conditions d'échange de ces données. HLA impose également que chaque fédéré doit obligatoirement avoir un document appelé *Modèle d'objets de simulation ou SOM (Simulation Object Model)* qui décrit les classes d'objets et d'interactions utilisés par le fédéré. Le SOM décrit sommairement la nature des données qu'un fédéré peut échanger lorsqu'il est intégré dans une fédération. Les modèles de données SOM et FOM doivent obligatoirement être rédigés dans un format standard appelé *OMT (Object Model Template)* qui fait partie du standard HLA.

Les modèles d'objets sont des composantes qui ne sont pas temps réel. Cependant, un sous-ensemble du FOM appelé *Federation Execution Data (FED)* est utilisé par la simulation sous forme de donnée externe passée à la fois aux fédérés et à la RTI. Les données FED contiennent, entre autres, la définition des classes d'objets et d'interactions qui servent à l'exécution. Les attributs et les paramètres des classes ne sont pas typés dans les données FED. En particulier, la RTI ne connaît pas les sémantiques des données échangées entre les fédérés. Ces derniers sont échangés sous forme de blocs d'octets dans un format conventionné par les fédérés et méconnu à la RTI. L'intérêt de ce choix est de garder la RTI indépendant du domaine simulé.

3-3 Les trois éléments composant HLA

L'architecture HLA est composée de trois éléments : les règles HLA, la Spécification d'Interface (IFSpec) et l'Object Model Template (OMT).

3 - 3 - 1 Les règles HLA

C'est un ensemble de dix règles qui doivent être satisfaites pour qu'un fédéré ou une fédération soit conforme au standard HLA. Les cinq premières règles portent sur la fédération et les cinq autres sur les fédérés. Ces règles, décrites dans le document « *High Level Architecture – Rules* » [IEEE99a], sont les suivantes :

Règle 1 : *Une fédération doit avoir un FOM décrit conformément au format HLA OMT.*

Règle 2 : *Dans une fédération, la représentation de toute instance d'objet associé à la simulation doit être dans les fédérés et non pas dans la RTI.*

Règle 3 : *Durant l'exécution de la fédération, l'échange de toutes les données décrites dans le FOM entre les fédérés doit se faire via la RTI.*

Règle 4 : *Durant l'exécution d'une fédération, les fédérés doivent interagir avec la RTI conformément à la spécification d'interface (HLA Interface Specification).*

Règle 5 : *Durant l'exécution d'une fédération, l'attribut d'un objet doit être possédé par au plus un fédéré à un instant donné.*

Règle 6 : *Un fédéré doit avoir un SOM documenté conformément au format HLA OMT.*

Règle 7 : *Un fédéré doit être capable de mettre à jour et de refléter tout attribut et d'envoyer et de recevoir toute interaction telle que spécifié dans son SOM.*

Règle 8 : *Un fédéré doit être capable de transférer et d'accepter dynamiquement la possession d'attributs tels que spécifiés dans son SOM.*

Règle 9 : *Un fédéré doit être capable de changer les conditions sous lesquelles il permet la mise à jour des attributs tels que spécifié dans son SOM.*

Règle 10 : *Un fédéré doit être capable de gérer son temps local d'une manière qui lui permet de coordonner l'échange de données avec les autres membres de le fédération.*

3 - 3 - 2 La Spécification d'Interface

Le deuxième élément de HLA est la spécification de l'interface décrite dans le document « *High Level Architecture – Interface Specification* » [IEEE99b]. Il définit la spécification de l'interface entre les fédérés et la RTI. La RTI est une implantation conforme à la spécification mais il ne fait pas partie de HLA, c'est uniquement sa spécification qui en fait partie. Ainsi, différentes implantations de la RTI, conformes à la spécification, sont possibles. Un exemple d'implantations est celle du DMSO, la version utilisée dans ce projet et décrite dans [IEEE99d]. La spécification d'interface est définie de manière abstraite et indépendante de tout langage de programmation. Elle distingue les appels faits par les fédérés à la RTI de ceux faits par la RTI aux fédérés. Un appel fait par la RTI au fédéré est appelé *callback* et est noté avec un signe « ^{t} » devant le nom de l'appel. Chaque appel est défini en tant qu'une procédure qui reçoit des arguments et retourne un résultat, avec un ensemble de pré-conditions et de post-conditions sur son exécution, ainsi que des exceptions qui peuvent avoir lieu lors de son exécution.

La spécification d'interface regroupe les services offerts par la RTI en six groupes. Pour chaque groupe, elle décrit l'objectif des services qu'il fournit et la manière d'utiliser ces services dans les différents scénarios. Ces groupes sont les suivants :

1. Gestion de fédération : offre les services de gestion de la fédération incluant, en particulier, les services de démarrage et d'arrêt de la fédération et les services permettant aux fédérés de rejoindre ou de quitter une fédération.
2. Gestion du temps : offre les services permettant l'application de mécanismes d'avancement du temps et de livraison des messages.
3. Gestion de déclaration : offre les services reliés à la publication et à la souscription des classes d'objets et d'interactions.
4. Gestion d'objets : offre les services reliés à l'enregistrement et à la découverte des objets, aux mises à jours des attributs et aux échanges d'interactions.

5. Gestion de possession : offre les services reliés à l'échange de la possession des attributs.
6. Gestion de distribution de données : peut être considérée comme une extension aux services de gestion de déclaration en fournissant un mécanisme d'échange des informations basé sur les valeurs réelles des attributs et des paramètres.

3 - 3 - 3 L'Object Model Template (OMT)

Le troisième élément de HLA est le standard de documentation des modèles d'objets (OMT), décrit dans le document officiel « *HLA Object Model Template Specification* » [IEEE99c]. Il a été introduit par souci de ré-utilisabilité et d'interopérabilité qui veut que chaque information gérée par un fédéré et visible à l'extérieur de celui-ci doit être spécifiée en détail dans un format commun.

L'OMT décrit trois modèles d'objets : le *modèle d'objet de simulateur (SOM)*, le *modèle d'objet de fédération (FOM)* et le *modèle d'objet de gestion (MOM)*. Ces modèles sont de nature non dynamique et n'interviennent donc pas dans l'exécution de la simulation.

- *FOM* : chaque fédération doit avoir un FOM qui décrit les classes d'objets et les classes d'interactions utilisées dans la fédération. En formalisant l'échange des données dans une fédération en utilisant un format standard, le FOM fournit les éléments clés d'une fédération et constitue ainsi un assistant principal dans la ré-utilisabilité de celle-ci.
- *SOM* : chaque fédéré doit avoir un SOM qui décrit les classes d'objets et d'interactions qu'un fédéré est capable d'utiliser dans une fédération. Intuitivement, le SOM décrit avec quoi un fédéré peut participer à une fédération. Il peut être utilisé dans le futur en tant que moyen pour décider si le fédéré est approprié à être utilisé dans une nouvelle fédération. Dans le cas des simulations les plus larges, bien que les informations nécessaires sur les fédérés peuvent aller beaucoup plus loin que celles décrites par le SOM, ce dernier fournit les informations minimales pour décider s'il

serait bénéfique d'investir dans le fédéré. Généralement, le FOM d'une fédération est développé en fusionnant les SOMs des différents fédérés qui la composent.

- *MOM* : la RTI elle-même peut échanger des messages avec les fédérés à travers des interactions et des objets spécifiques, dans l'objectif de fournir les informations générales sur la fédération. Les classes d'objets et d'interactions correspondantes sont prédéfinies et font partie du standard OMT et sont donc communes à toutes les simulations. Le MOM est le modèle d'objet qui décrit ces classes.

OMT supporte la notion d'héritage entre les classes d'objets et les classes d'interactions. Lorsqu'une classe A hérite d'une classe B, on dit que B est la super-classe et A est la sous-classe. La classe A est considérée comme une spécialisation de la classe B, c'est-à-dire que A contient automatiquement toutes les caractéristiques de B (attributs de classe d'objet ou paramètres de classe d'interaction) et définit d'autres caractéristiques qui la spécialisent par rapport à B.

Avec l'OMT, les modèles d'objets sont décrits sous forme tabulaire en utilisant sept tables:

- La table d'identification du modèle d'objets : elle contient les informations sur le modèle d'objets. Celles-ci incluent, le cadre dans lequel il a été développé, l'organisation qui l'a produit, les personnes à contacter pour avoir des informations sur le modèle, etc..
- La table de structure de classes d'objets : elle identifie les classes d'objets, définies d'une manière hiérarchique.
- La table de structure de classes d'interactions : elle identifie les classes d'interactions, définies d'une manière hiérarchique.
- La table d'attributs d'objets : elle décrit les attributs de chaque classe d'objet. Elle décrit, entre autres, le type de données de chaque attribut, sa résolution, sa cardinalité, etc..

- La table de paramètres d'interactions : elle décrit les paramètre de chaque classe d'interactions. Elle décrit, entre autres, le type de données de chaque paramètre, sa résolution, sa cardinalité, etc..
- La table du lexique du FOM/SOM : elle définit tous les termes utilisés dans les autres tables.
- La table d'espaces de routage : elle contient les informations nécessaires à la distribution de données dans la fédération, qui est le moyen utilisé par HLA pour distribuer les données de manière plus efficace. La distribution de données dans HLA sera expliquée en détail dans le paragraphe 3-6.

3-4 La version de la RTI utilisée

Dans ce paragraphe, nous présentons une implantation particulière de la RTI : celle fournie gratuitement par le DMSO et téléchargeable à partir de leur site Web [DMSO00]. Dans notre projet, nous avons travaillé avec la version RTI1.3-NG. Dans le reste du document, nous nous référons par le terme RTI à cette implantation particulière.

3 - 4 - 1 Composition d'une fédération

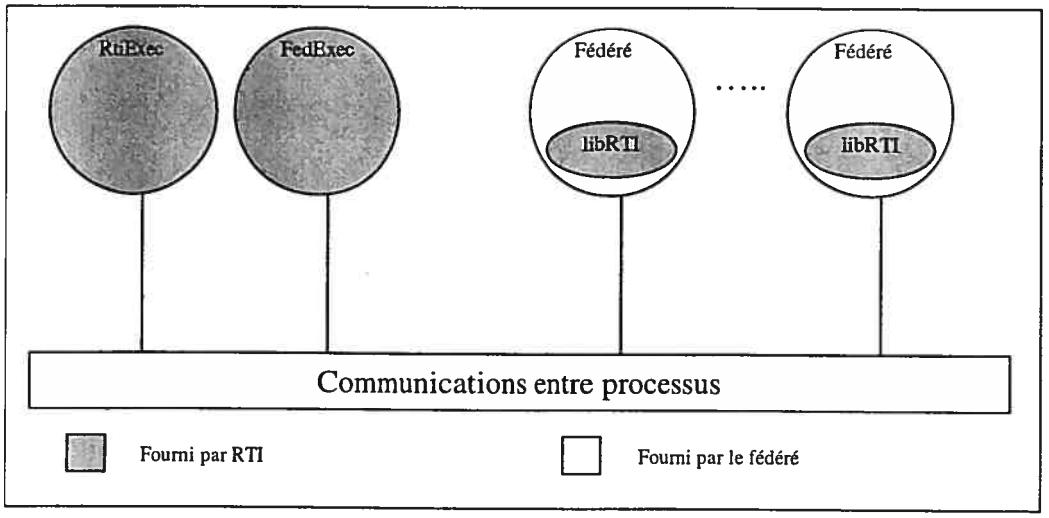

Figure 3-1 Composition d'une fédération sous DMSO RTI.

La figure 3-1 montre les composantes d'une fédération s'exécutant sous la RTI. Tous les éléments de la figure appartiennent à la fédération sauf le programme RtiExec. Voici ce que représente chaque élément :

- Fédéré : c'est un fédéré particulier participant à la fédération. Il est composé d'un simulateur et d'un SOM.
- Fédération : elle est composée de l'ensemble des fédérés en interaction et d'un FOM.
- FedExec (Federation Executive) : c'est un programme fourni par la RTI qui gère la fédération. Il permet aux fédérés de rejoindre et de quitter la fédération. Il permet également l'échange des messages entre les fédérés participant à une même fédération. Il y a un programme FedExec par fédération.
- RtiExec (RTI Executive) : c'est un programme fourni par la RTI qui se charge de la création et de la destruction des fédérations. RtiExec crée un programme FedExec unique pour chaque fédération nouvellement créée et lui assigne un nom unique. Le programme RtiExec redirige chaque fédéré vers le programme FedExec de la fédération qu'il a rejointe.
- LibRTI : c'est une librairie qui contient l'ensemble des méthodes utilisées par les fédérés pour appeler les services de la RTI. Elle est écrite en C++ et offre des interfaces en C++, Java, Ada et CORBA IDL.

3 - 4 - 2 Composition d'un fédéré

La figure 3-2 montre les différents éléments composant un fédéré. Pour obtenir le fédéré, l'édition des liens est établie entre le code de l'utilisateur et la librairie *libRTI*. A l'intérieur d'un fédéré, la partie de *libRTI* fournissant la fonctionnalité externe telle que spécifiée dans le document officiel « HLA – Interface Specification » s'appelle *Local RTI Component* (*LRC*). Elle implante la classe *RTIambassador*. Tous les appels faits par le fédéré pour demander les services de la RTI sont des méthodes de la classe *RTIambassador*. La partie de *libRTI* fournissant l'interface utilisée par la RTI pour appeler les callbacks au niveau du fédéré est implantée dans la classe *FederateAmbassador*. Cette dernière ne contient que des méthodes purement virtuelles

(qui sont les callbacks) et n'est donc pas directement instanciable. L'utilisateur doit par contre passer par une classe dérivée dont il doit implanter les méthodes.

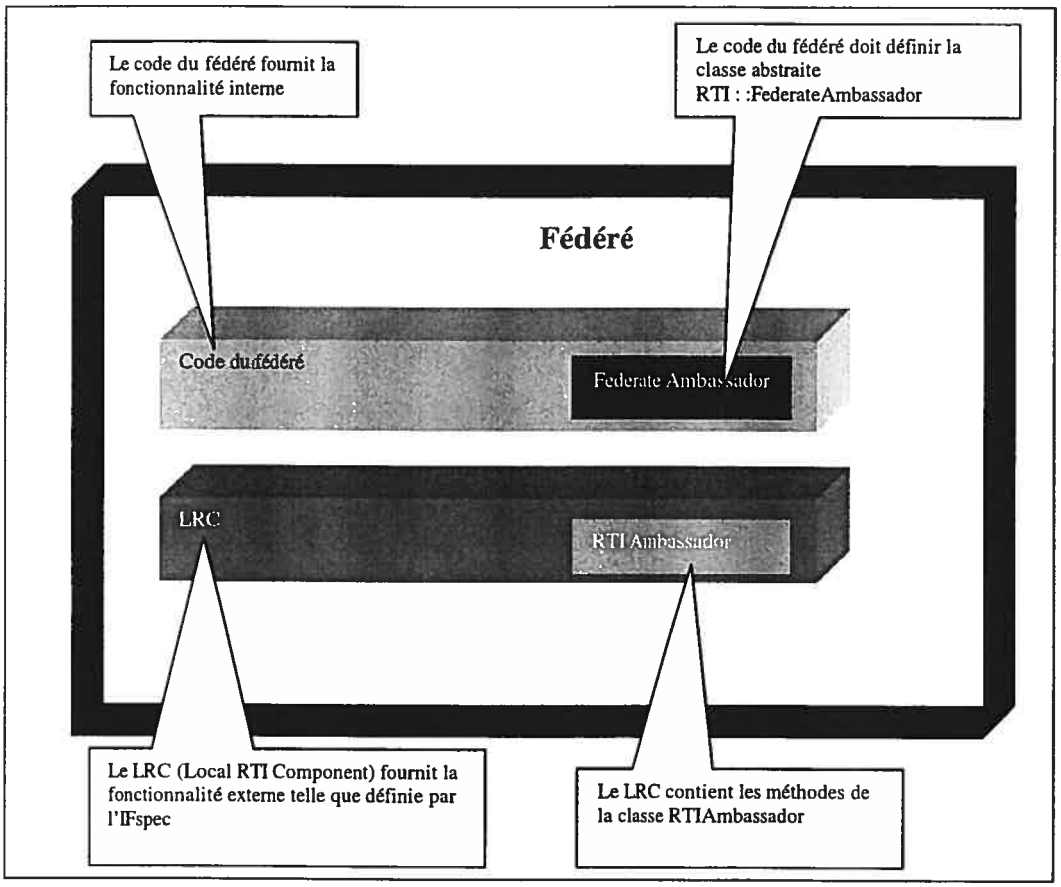

Figure 3-2 Composition d'un fédéré sous DMSO RTI.

3-5 La gestion du temps

La RTI offre un service de gestion de temps logique qui permet aux fédérés d'avancer leur temps et d'échanger des messages étiquetés par le temps. La gestion du temps réel est une responsabilité des fédérés qui est totalement transparente à la RTI. A cet égard, HLA offre un service de gestion du temps assez flexible pour répondre aux besoins variés du Département de la Défense américain en terme de simulation. Ce service inclut :

- Un service d'avancement du temps en mode time-stepped et en mode event-driven au niveau des fédérés.
- Un mécanisme de synchronisation du temps entre les fédérés. Deux modes de synchronisation sont permis : la synchronisation conservatrice et la synchronisation optimiste.

Nous rappelons que l'un des objectifs principaux de HLA est de permettre la re-utilisabilité et l'interopérabilité des simulateurs. Un même fédéré doit donc être capable d'interagir indifféremment avec des fédérés en mode time-stepped ou en mode event-driven. HLA offre cette possibilité à travers la *transparence de gestion du temps* qui signifie que les fédérés n'indiquent pas explicitement à la RTI le mode de gestion du temps qu'ils utilisent [Fujimoto98]. Un fédéré interagit donc avec les autres fédérés sans avoir d'idée sur le mode d'avancement du temps qu'ils utilisent.

En ce qui concerne les événements, HLA en supporte deux types : les événements ordonnés par le temps ou TSO (*Time Stamp Order*) et les événements asynchrones ou RO (*Receive Order*). Les événements TSO ont la propriété d'être délivrés aux fédérés dans l'ordre de leurs étiquettes de temps. En effet, la RTI maintient ces événements dans une queue interne et ne les délivre aux fédérés qu'après être assurée qu'ils sont délivrés dans l'ordre. Les événements RO, par contre, sont délivrés de manière asynchrone.

3 - 5 - 1 La synchronisation conservatrice

HLA offre deux modes de synchronisation du temps : le mode conservateur (ou mode de synchronisation conservatrice) et le mode optimiste. En mode conservateur, un fédéré ne doit jamais recevoir d'événements dans son passé. Les avancements de temps d'un fédéré doivent donc passer par la RTI. De son coté, la RTI ne doit accorder l'avancement à un fédéré qu'après être assurée que le fédéré ne recevra plus d'événements dans son passé. Pour obtenir cette assurance, la RTI se base sur une technique qui consiste à imposer à chaque fédéré de déclarer une durée de temps logique, appelée *lookahead*, pendant laquelle il garantit de ne pas envoyer de messages. En d'autres termes, un fédéré au temps t=T avec un lookahead égal à L ne peut pas envoyer des messages avec une étiquette de temps inférieure à T+L. Un fédéré doit déclarer explicitement son lookahead

à la RTI. Ce dernier détient la valeur du lookahead ainsi que la valeur du temps logique de chaque fédéré, c'est ce qui lui permet de contrôler l'avancement du temps des fédérés. La détermination du lookahead n'est pas toujours facile et dépend largement de la nature du problème simulé.

Pour mieux comprendre l'utilité du lookahead dans le mécanisme d'avancement du temps en mode de synchronisation conservatrice, prenons l'exemple de la figure 3-3 (repris sur [IEEE99d]. La figure représente trois fédérés F1, F2 et F3. F1 est au temps logique 18 et il a déclaré un lookahead de 9. Donc le message le plus proche qu'il pourra envoyer aura une étiquette de temps de 27. De même, F2 qui est au temps 17 avec un lookahead de 3 ne pourra pas envoyer de messages avec une étiquettes inférieure à 20. F3 à son tour est au temps 17 avec un lookahead de 4 et ne pourra donc pas envoyer de messages avec des étiquettes inférieures à 21. Supposons qu'à cet instant, F2 demande à la RTI d'avancer son temps à t=25. Si on suppose que la RTI accorde à F2 le temps demandé, alors F3 pourrait envoyer à ce dernier des messages avec t=24 et donc dans son passé. Plutôt, la RTI met F2 en attente jusqu'à ce que F3 avance suffisamment dans le temps pour garantir qu'il n'envoie pas de messages à F2 dans son passé. Entre temps, F2 ne pourra plus envoyer de messages avec des étiquettes de temps inférieures au temps demandé (c'est-à-dire 25) plus son lookahead, ce qui correspond à 29. Supposons maintenant que F3 finit ses traitements et demande d'avancer son temps à t=26. La RTI accorde alors à F3 le temps demandé, car F1 et F2 ne peuvent pas envoyer des messages avec des étiquettes inférieures à 27 et 29, respectivement. Par suite, la RTI peut accorder à F2 (encore en attente) le temps t=25 qu'il avait demandé, puisque le risque qu'il reçoive des messages dans son passé de la part de F3 est éliminé.

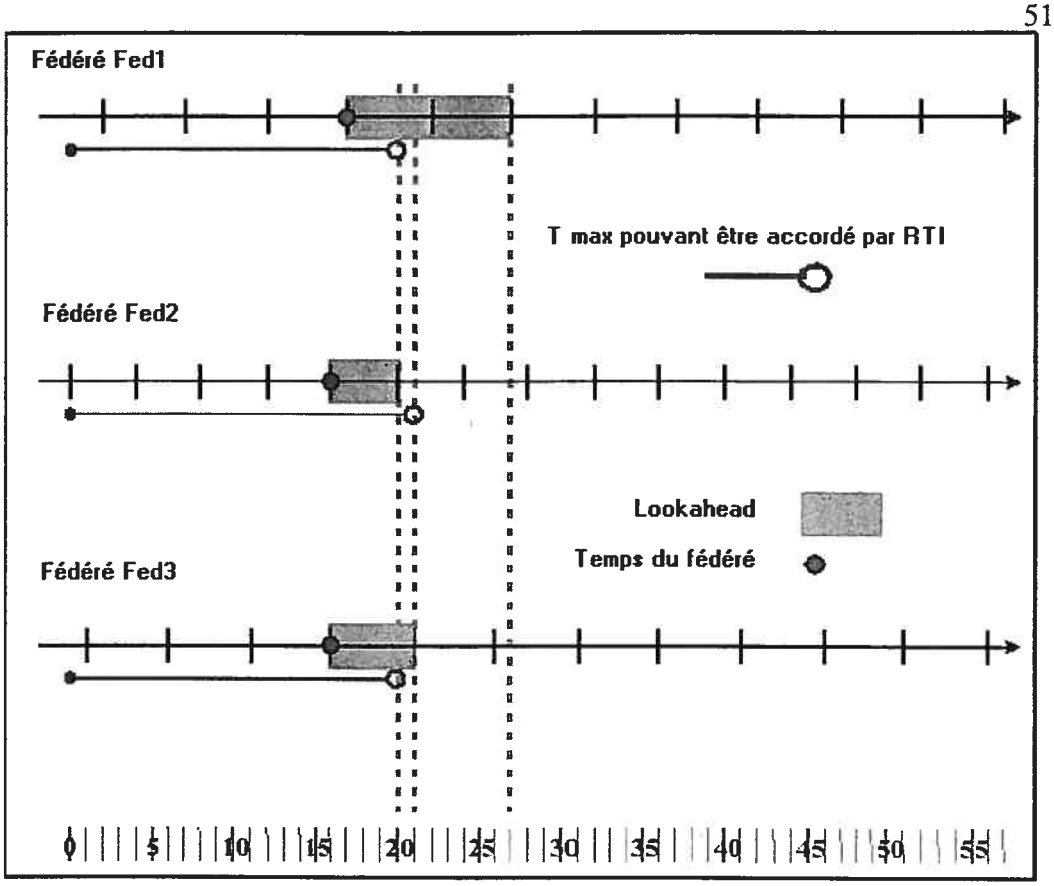

Figure 3-3 Avancement du temps en mode de synchronisation conservatrice.

La RTI offre deux interfaces pour l'avancement du temps en mode conservateur :

- l'appel *TimeAdvanceRequest(T)* : Cette interface est appropriée à un fédéré qui désire avancer son temps à des pas fixes, c'est à dire en mode time-stepped. Le paramètre T désigne la valeur du temps désirée. HLA n'impose pas au fédéré que les avancements soient forcément à des pas fixes mais c'est à l'utilisateur de respecter cette contrainte s'il le désire. Suite à un appel *TimeAdvanceRequest(T)*, la RTI délivre au fédéré tous les messages dont l'étiquette de temps est inférieure à T. Ce n'est que lorsqu'il s'assure que le fédéré ne recevra plus d'autres messages avec une étiquette de temps inférieure à T que la RTI accorde au fédéré le temps T demandé, à travers le callback *TimeAdvanceGrant*.

- L'appel *NextEventRequest(T)* : cette interface est appropriée à un fédéré qui désire avancer son temps en mode event-driven. Suite à cet appel, la RTI délivre au fédéré le message *E* le plus proche avec une étiquette de temps t_E inférieure ou égale à T ainsi que tous les autres messages à t_E, puis lui accorde le temps de ce message à travers le callback *TimeAdvanceGrant()*. Si aucun message n'existe avec un temps inférieur à T, alors le temps T est accordé. Idéalement, T représente le temps du prochain événement dans la queue du fédéré.

3 - 5 - 2 La synchronisation optimiste

Pour un fédéré optimiste, la RTI offre le servie *FlushQueueRequest()* qui force la RTI de délivrer au fédéré tous les messages qui lui sont destinés, même s'il sait qu'il est possible que d'autres messages avec des étiquettes plus petites puissent parvenir plus tard et seront donc dans le passé du fédéré. Si le fédéré reçoit un message dans son passé, il doit reculer dans le temps et restaurer son dernier état avant le temps de ce message. De plus, il doit annuler tous les mauvais messages qu'il a dû envoyer suite au traitement des événements incorrects. L'annulation d'un message envoyé se fait à travers le service RTI *Retract()*. En réaction à l'appel de ce service, la RTI propage la demande d'annulation à chaque fédéré ayant reçu le message à annuler. Ces fédérés, à leur tour, doivent effectuer une reprise et générer éventuellement d'autres demandes d'annulation.

Un fédéré optimiste doit calculer et déclarer (à la RTI) le temps le plus loin dans le passé pour lequel il peut faire une reprise. Ce temps appelé, *Global Virtual Time (GVT),* permet au fédéré de calculer la quantité de ressource mémoire nécessaire pour la sauvegarde de son état et de faire les opérations d'entrée / sortie qui ne peuvent plus être annulées. Un fédéré optimiste garantit qu'il n'envoie pas de messages avec une étiquette inférieur à son GVT plus son lookahead (GVT+lookahead). Ceci permet à la RTI de savoir quels messages délivrer aux fédérés conservateurs avec une assurance que ces messages ne seront pas annulés plus tard. Ainsi, un fédéré conservateur ne reçoit jamais une demande d'annulation de messages.

Une propriété intéressante de la gestion du temps optimiste dans HLA est qu'un fédéré conservateur ne se rend pas compte de l'existence de fédérés optimistes, puisqu'il ne

reçoit jamais de demandes d'annulations de messages. Un fédéré, conservateur ou optimiste, interagit avec les autres fédérés sans savoir quels modes de synchronisation ils utilisent.

Enfin, HLA ne résout pas le problème classique relié à la gestion du temps optimiste et qui consiste à empêcher les fédérations d'avancer trop loin dans le futur, mais il délègue cette responsabilité aux fédérés.

3-6 La distribution de données

Un exemple illustratif permet de comprendre la distribution de données dans HLA. Il s'agit d'une fédération de bataille comprenant des simulateurs d'avions et des simulateurs de radars. Dans cette fédération, un avion est simulé par un objet qui décrit la position de l'avion par trois attributs : longitude, latitude et altitude. Si le simulateur d'un radar réalise une souscription à la classe d'objets « avion », alors il va découvrir tous les avions et recevoir toutes les mises à jour (Update) sur leurs positions même s'ils ne sont pas dans son champ. Dans ce cas de figure, le fédéré doit simplement ignorer ces mises à jour. Dans le cas où la fédération contient des milliers de simulateurs d'avions et de radars, il est clair qu'il y aura un grand nombre de messages non utiles délivrés aux fédérés. La distribution de données dans HLA offre une solution à ce problème en fournissant aux fédérés le moyen de déclarer les événements qu'ils s'intéressent à recevoir en se basant sur les valeurs effectives des attributs des objets. La distribution de données ne remplace pas le mécanisme de publication/souscription mais, plutôt, elle le complète. Son utilisation reste toutefois optionnelle.

La distribution de données fonctionne de la manière suivante : les fédérés doivent être préalablement d'accord sur un espace, appelé *espace de routage,* dans lequel varient les valeurs des attributs. Ce dernier est défini par un ensemble de *dimensions*. On suppose, dans notre exemple, que l'espace est défini par les trois dimensions : l'altitude qui varie entre 1 km et 12 km, la latitude qui varie entre 44°E et 48°E et la longitude qui varie entre 30°N et 40°N (figure 3-4). Un fédéré qui simule un radar doit se souscrire aux objets « avion » en spécifiant une *région de souscription* qui encadre l'ensemble des positions

d'avion qui l'intéressent, c'est-à-dire celles que le radar est capable de détecter (volume à droite dans la figure). De son côté, un fédéré qui simule un avion doit déclarer, pour chaque avion qu'il simule, une *région de mises à jour* qui encadre l'ensemble des positions possibles de son avion durant la simulation (volume à gauche dans la figure). Par la suite, une mise à jour effectuée sur un objet « avion » n'est reflétée à un fédéré « radar » que si la région de souscription du radar et celle des mises à jour de l'objet avion se chevauchent. Dans l'exemple de la figure 3-4, les deux régions (de mise à jour et de souscription) sont des volumes tridimensionnels. Une région peut être définie par l'union de plusieurs volumes tridimensionnels, appelés des « extents ». De façon plus générale, dans HLA, une région est définie par un ensemble d'extents. Chaque extent est un volume dans l'espace de routage, défini par l'ensemble des intervalles de valeurs qui le délimitent selon chacune des dimensions. Ainsi, la région de mises à jour de la figure 3-4 est composée d'un seul extent défini par les trois intervalles [1km,11km], [45° E,47° E] et [34° N,38° N] qui le délimitent selon les axes altitude, latitude et longitude, respectivement.

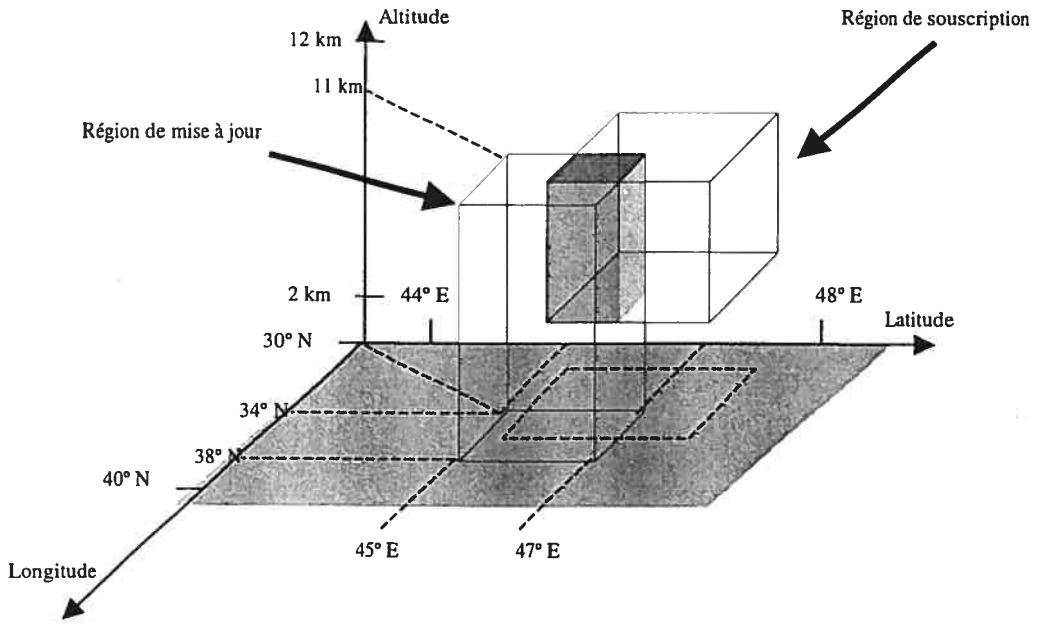

Figure 3-4 Région de souscription et région de mises à jour.

Pour que la RTI reste indépendante du problème simulé, un schéma de représentation générique est utilisé pour exprimer les extents [IEEE99d]. Dans le cadre de ce schéma, une valeur exprimée selon une dimension D doit être projetée dans l'intervalle [*MIN_EXTENT*, *MAX_EXTENT*] où *MIN_ EXTENT* et *MAX_ EXTENT* sont deux valeurs réelles fixes, prédéfinies dans la RTI. La figure 3-5 illustre la formule utilisée pour convertir une valeur *V* exprimée selon la dimension D en une valeur dans l'intervalle [*MIN_EXTENT*, *MAX_EXTENT*]. Dans la figure, la projection de la valeur minimale possible *Dmin*, selon la dimension *D*, donne la valeur *MIN_EXTENT*, et la projection de la valeur maximale possible *Dmax*, selon la dimension *D*, donne la valeur *MAX_EXTENT*. Pour illustrer comment utiliser cette méthode, reprenons l'exemple de la figure 3-5. Dans cet exemple, la région de mises à jour contient un seul extent, délimité par l'intervalle [1km,11km] selon la dimension altitude, l'intervalle [45° E,47° E] selon la dimension latitude et l'intervalle [34° N,38° N] selon la dimension longitude. Avant de déclarer cet extent à la RTI, un fédéré doit projeter les valeurs des bornes de chacun de ces intervalles sur l'intervalle [*MIN_EXTENT*, *MAX_EXTENT*], conformément à la méthode présentée dans la figure 3-5.

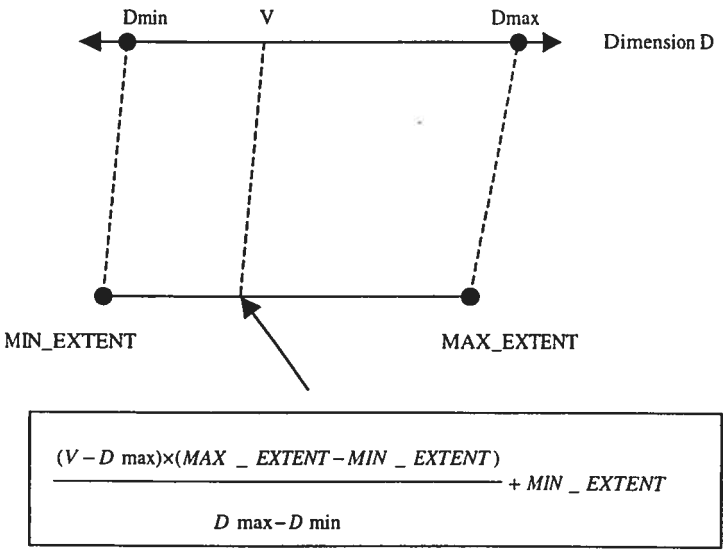

Figure 3-5 Conversion d'une dimension dans le cadre de DDM.

Notons enfin que la distribution de données peut être utilisée avec les interactions d'une façon tout à fait similaire au cas des objets.

3-7 Les raisons du choix de HLA pour la simulation de marchés

Le choix de HLA comme plate-forme de simulation des prototypes de marchés développés dans le cadre du projet TEM a été fait préalablement à notre projet. Cependant, nous partageons ce choix pour les raisons suivantes :

- HLA est un standard qui facilite la ré-utilisabilité et l'interopérabilité des simulateurs, ce qui est très utile dans notre projet. Par exemple, si l'on développe un simulateur d'un « participant dans une place de marché » dans le cadre d'une simulation d'un marché, alors il sera très bénéfique de pouvoir réutiliser ce simulateur dans le cadre d'autres simulations de marché sans recours aux modifications de code ou avec un minimum de modifications.
- Le développement des différents simulateurs peut être réalisé par des équipes différentes. Puisque ces simulateurs sont conformes au standard HLA, leur interopérabilité est garantie.
- La version gratuite de HLA fournie par [DMSO00] est une librairie qui offre des interfaces en C++, Java, ADA et CORBA IDL. Ces derniers, contrairement aux langages de simulation, sont beaucoup plus adaptés à l'implantation des différents aspects des marchés, qui utilisent largement des méthodes de recherche opérationnelle.
- HLA est un standard qui prend de plus en plus de popularité dans le monde de simulation, que ce soit sur le plan académique ou industriel. Cette popularité permettra, en particulier, l'échange d'informations avec d'autres groupes de recherche. Le téléchargement de simulateurs prêts à être utilisés sera même possible dans le futur proche.

Chapitre 4
Caractéristiques des marchés à simuler

Dans ce chapitre, nous donnons une description détaillée des caractéristiques des marchés ciblés par notre outil. Notre objectif n'est pas d'expliquer l'utilité économique et les raisons d'être des différentes notions utilisées dans les marchés. Notre objectif est plutôt de décrire des aspects ayant une relation directe avec notre projet de simulation.

Nous sommes intéressés, dans notre projet, par les marchés électroniques basés sur les enchères. Donc, dans le reste de ce document, lorsque nous utilisons le terme « marché », nous sous-entendons un marché électronique basé sur les enchères.

4-1 Éléments de marchés

La figure 4-1 représente une vue d'ensemble d'un marché électronique. Les éléments représentés sont les suivants :

Marché et place de marché électronique

Un *marché électronique* est composé d'un ensemble de *places de marchés électroniques* et d'un ensemble de participants qui interagissent pour transiger des biens ou des services à travers ces places de marchés.

Une place de marché est l'aire où se font effectivement les enchères. C'est un site auquel les participants peuvent se connecter pour participer aux enchères. Chaque place de marché gère un système d'informations où elle maintient les données sur les enchères qui s'y déroulent ainsi que les informations sur les participants à ces enchères.

Un agent appelé *encanteur* contrôle le déroulement des enchères. En particulier, c'est l'encanteur qui déclare le début et la fin d'une enchère, séquence ses étapes, applique les règles du marché, reçoit les mises des participants et leur communique les résultats.

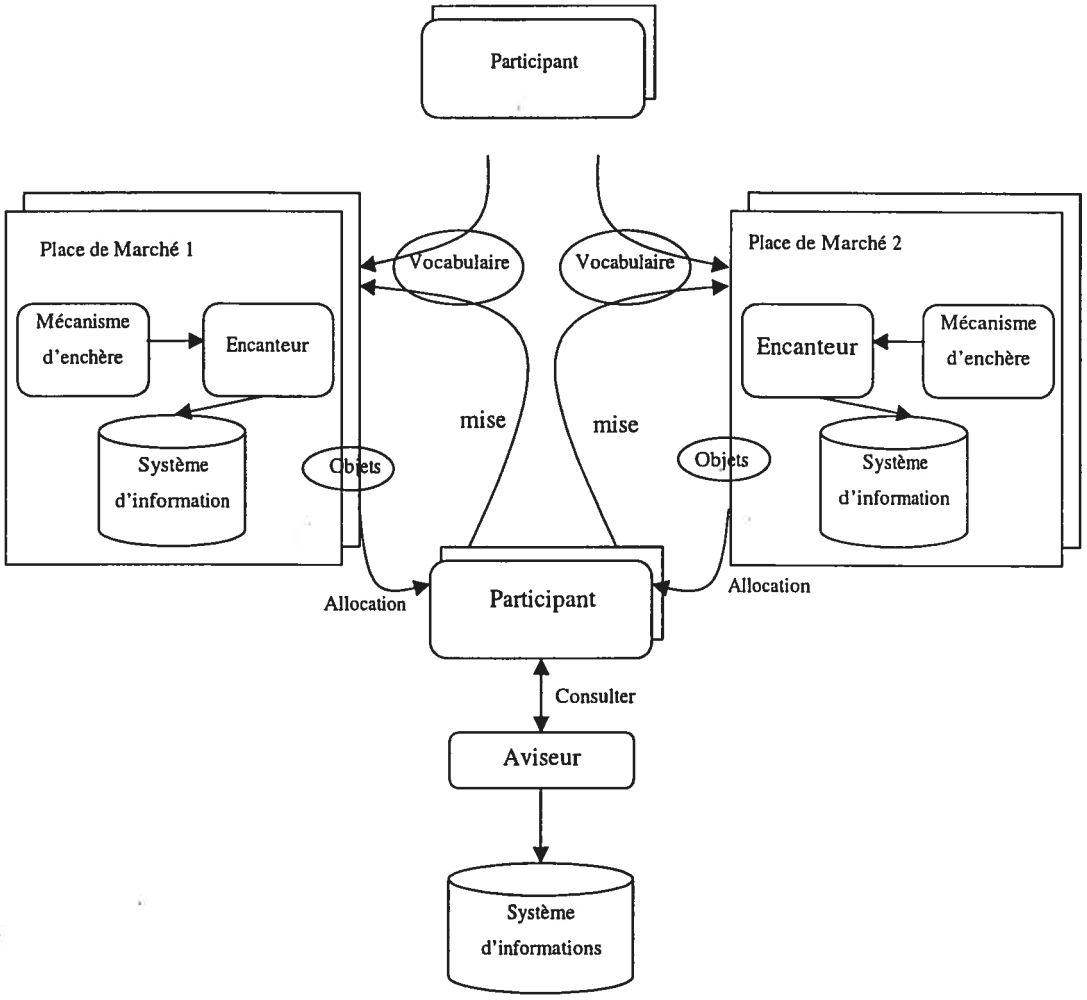

Figure 4-1 Vue d'ensemble d'un marché électronique.

La place de marché ouvre de façon continue (une fois par jour, par exemple), organise une session d'enchères et ferme à la fin de la session. La période entre l'ouverture et la fermeture d'une place de marché est appelée une *occurrence du marché.*

N.B : Le mot marché, bien qu'il désigne l'ensemble de places de marchés et des participants en interaction, est souvent utilisé pour désigner aussi une place de marché. La signification voulue doit donc être comprise du contexte.

Objets

C'est l'ensemble des biens et des services sur lesquels portent les enchères.

Participant et aviseur

Dans les enchères de façon générale, le terme *participant* désigne un intervenant qui participe à une enchère dans le but de transiger des biens ou des services à travers cette enchère. Dans le contexte des enchères électroniques, le participant est représenté par un logiciel tout capable de se connecter à une place de marché dans le but de participer aux enchères. Dans ce contexte, le participant peut utiliser un outil d'aide à la décision, appelé *aviseur,* qui le guide dans son comportement dans le marché. En particulier, il lui indique des *stratégies de mise*. Une stratégie de mises indique au participant les objets sur lesquels il doit miser, le moment où il doit soumettre une mise, le prix à proposer, etc.. Pour pouvoir prendre des décisions, un aviseur a besoin d'accéder au système d'informations du participant. Par exemple, dans un marché de transport, un aviseur d'un transporteur doit pouvoir accéder aux informations sur la disponibilité des camions, des ressources, des coûts de transport, etc..

Mise, annonce et vocabulaire des mises

Dans les marchés dits *non combinatoires*, une mise est une déclaration d'une volonté d'achat ou de vente d'un objet à une quantité donnée et à un prix donné. Dans ce cas, on parle de *mise atomique*. Dans les marchés dits *combinatoires*, une mise est soit une mise atomique soit une *mise combinée*. Cette dernière désigne une combinaison de plusieurs mises atomiques, avec un prix global. Par exemple, si un participant s'intéresse à se procurer deux objets A et B, mais A seul ou B seul ne l'intéressent pas, alors l'enchère doit lui permettre de miser sur la combinaison (A ET B) avec un prix global. La combinaison peut prendre différentes formes en utilisant différents opérateurs, par

exemple (A OU B) ou ((A ET B) OU (A ET C)). L'ensemble des opérateurs permis pour la formulation des mises ainsi que la manière de les combiner définit le *vocabulaire des mises*.

Dans beaucoup de cas de marchés, un objet n'est connu que lorsqu'un participant le déclare pour la première fois dans le marché (pour vente ou pour achat), c'est-à-dire qu'il déclare sa volonté d'acheter ou de vendre un objet dont il donne les caractéristiques. Cette déclaration est appelée *annonce*. Les mises ont pour objectif de gagner les enchères sur les objets annoncés. L'annonce a une durée de vie, c'est-à-dire que l'objet annoncé doit être alloué avant l'expiration de sa durée de vie. La mise soumise par le gagnant d'un objet, à travers laquelle il a gagné l'objet, est appelée *mise gagnante*.

Mécanisme d'enchère, allocation et prix

Les allocations et les prix forment le résultat de l'enchère. L'*allocation* désigne les objets gagnés et les gagnants. Dans certains cas de marchés, les objets sont transigés avec des *quantités*. Dans ce cas, l'allocation est accompagnée de la quantité gagnée pour chaque objet alloué. Par exemple, dans un marché boursier, un participant peut miser sur 50 actions d'une entreprise particulière, mais le marché lui alloue uniquement 30 actions qui correspondent à la quantité allouée.

Le *mécanisme d'enchère* applique le scénario de l'enchère et l'algorithme de résolution de deux problèmes : le problème d'allocation et le problème de détermination des prix. L'enchère peut être *ascendante* ou *descendante*. Dans une enchère ascendante, l'objet est déclaré pour vente et les acheteurs tentent de le gagner en proposant des prix de plus en plus élevés. Dans une enchère descendante, l'objet est déclaré pour achat et les acheteurs tentent de le gagner en proposant des prix de plus en plus bas. Un domaine approprié pour l'application des enchères descendantes est l'achat de services où l'acheteur du service cherche le vendeur qui offre le prix le plus bas.

Dans la suite de ce chapitre, nous prendrons nos exemples du marché de bourse de fret que nous présentons à ce niveau. Nous présentons une bourse de fret très simple mais,

dans la réalité, elle peut être beaucoup plus compliquée. Pour voir les progrès faits dans le projet TEM dans le cas de la bourse de fret, le lecteur peut se référer à [Abrache01a].

Dans une bourse de fret, on transige des *ordres de transport*. Un ordre de transport désigne une volonté de transporter une marchandise d'un point de départ vers un point d'arrivée à une date donnée. D'autres informations peuvent être incluses dans l'ordre, telles que la nature de la marchandise, son volume, etc.. Les participants dans le marché sont, d'une part les particuliers qui désirent transporter leurs marchandises, et d'autre part les transporteurs qui veulent gagner les ordres de transport. Ainsi, au cours de l'enchère, les particuliers annoncent des ordres de transport et les transporteurs misent sur ces ordres pour les gagner. L'objectif de l'enchère est de désigner quel transporteur a gagné quel ordre et à quel prix.

4-2 Les objets transigés

Dans ce paragraphe, nous donnons une classification des objets transigés dans les marchés électroniques.

Description des objets

C'est la description physique des objets. Le fonctionnement de la place de marché et des aviseurs peut en dépendre. Par exemple, dans le cas d'une bourse de fret, les aviseurs doivent connaître les points de départ et les points d'arrivée des ordres de transport pour pouvoir vérifier s'ils ont des camions disponibles pour faire le trajet.

Les objets peuvent être *indivisibles* ou *infiniment divisibles* [Babin01]. Par exemple, les capacités dans un réseau de télécommunications sont divisibles, mais pas les droits de passage sur les chemins de fer [Abrache01b].

Objets dans les marchés multi-unités

Dans un marché multi-unités, les objets se transigent par quantités. Par exemple, les titres dans un marché financier sont des objets multi-unités : un participant peut miser sur le titre IBM avec une quantité de 50 titres.

Objets dans les marchés multi-objets

Beaucoup de marchés annoncent des enchères sur plusieurs objets en même temps. Lorsque, pour les participants, l'évaluation d'un objet dépend des résultats des enchères sur les autres objets, les enchères sont appelées *enchères multi-objets*.

4-3 Rôle des participants dans le marché

Dépendamment du rôle des participants, une enchère peut être unilatérale ou multilatérale. Dans une enchère *unilatérale*, le seul vendeur (acheteur) est l'encanteur lui-même et tous les autres participants sont acheteurs (vendeurs). Par exemple, si le gouvernement désire acheter le service de déneigement de la ville, alors il peut déclarer une enchère sur son propre site Internet. Dans ce cas, le gouvernement est lui-même l'encanteur et est le seul acheteur, tandis que les entreprises de déneigement sont les vendeurs. Dans une enchère *multilatérale*, chaque participant peut être acheteur, vendeur ou les deux à la fois, dans le sens où il peut soumettre à la fois des mises d'achat et des mises de vente. Dans le cas des enchères combinatoires, le participant peut même créer des mises combinant à la fois des achats et des ventes, c'est-à-dire qu'il peut annoncer des objets pour enchère et miser sur les objets annoncés par les autres participants. Par exemple, dans un marché financier, où on vend des titres boursiers, un participant peut soumettre une mise combinée où il déclare sa volonté d'acheter 50 titres IBM et de vendre 100 titres Microsoft avec un prix global (à recevoir ou à payer) pour l'ensemble.

Dans notre outil, nous avons fusionné le traitement du cas d'une enchère unilatérale avec celui d'une enchère multilatérale, en considérant qu'une enchère unilatérale peut être assimilée à une enchère multilatérale où l'un des participants représente l'encanteur.

4-4 Les enchères

4 - 4 - 1 Objectif de l'enchère

L'enchère doit se dérouler de telle sorte qu'elle converge vers un objectif pré-déterminé. L'enchère peut être optimisée ou non-optimisée. L'objectif d'une enchère optimisée peut être, entre autres, la maximisation du revenu de l'enchère ou la maximisation du bien-être social de tous les participants [Abrache01a].

4 - 4 - 2 Enchères combinatoires

Les enchères combinatoires forment un type d'enchères particulièrement intéressant. Pour comprendre la notion d'enchère combinatoire, reprenons l'exemple du marché de bourse de fret, avec un scénario repris sur [Abrache01b]. Supposons que parmi les ordres de transport annoncés dans le marché, il y a les deux ordres suivants :

- Ordre O1 : transport d'une marchandise de Montréal vers Québec qui doit être exécuté au jour j.
- Ordre O2 : transport d'une marchandise de Québec vers Montréal qui doit être exécuté au même jour j.

Supposons maintenant qu'un transporteur T1 dispose d'un camion disponible au jour j et qui s'intéresse aux deux ordres O1 et O2. Supposons que T1 évalue l'ordre O1 à 500$, c'est-à-dire qu'il est prêt, pour ce montant, à transporter la marchandise de Montréal vers Québec et son camion retourne à Montréal vide. Supposons également qu'il évalue l'ordre O2 à 500$, c'est à dire qu'il est prêt, pour ce montant, à envoyer son camion vide à Québec au jour j pour transporter la marchandise vers Montréal. Le fait que les deux ordres soient à exécuter au même jour j donne la raison à notre transporteur de penser à gagner les deux ordres en même temps, de telle sorte que son camion transporte la marchandise de O1 de Montréal vers Québec au jour j et retourne à Montréal le même jour avec la marchandise de O2. Il est évident que l'évaluation par T1 de l'ensemble des deux ordres O1 et O2 est inférieure à la somme des évaluations des deux ordres puisque

les trajets à vide sont éliminés. Supposons qu'il évalue l'ensemble des deux ordres à un montant global de 800$. Si le marché annonce des enchères indépendantes sur O1 et O2 alors T1 n'est pas capable d'annoncer son intérêt aux deux ordres réunis pour le montant de 800$. D'autre part, T1 ne s'intéresse pas à prendre le risque de miser sur chaque ordre à part à un prix inférieur à son évaluation (en misant 400$ pour O1 et 400$ pour O2, par exemple) car s'il gagne un ordre et perd l'autre, alors il sera obligé d'exécuter l'ordre gagné à un prix inférieur à son évaluation de cet ordre. Ce problème appelé *problème d'exposition* ne se pose pas dans le cas des enchères dites combinatoires.

Dans une enchère *combinatoire*, on permet aux participants de combiner plusieurs objets à la fois. Dans l'exemple ci-dessus, une enchère combinatoire devrait permettre au transporteur de faire une mise de la forme :

((O1 pour 500$) **OR** (O2 pour 500$) **OR** (O1 **AND** O2 pour 800$)).

Avec cette stratégie, si le transporteur gagne l'ensemble des deux ordres, alors il les exécutera à 800$, mais s'il ne gagne qu'un seul alors il l'exécutera à 500$. Les enchères combinatoires posent beaucoup de défis, le défi majeur étant d'optimiser le mécanisme d'enchère, vu le grand nombre de combinaisons possibles.

De façon plus générale, l'utilité des enchères combinatoires est de permettre aux participants de prendre en compte les facteurs de complémentarité et de substituabilité entre les objets transigés. Deux objets sont dits *complémentaires* pour un participant, si l'évaluation, par ce dernier, des deux objets réunis est supérieure à la somme des évaluations de chaque objet à part. Deux objets sont dits *substituables* pour un participant, si l'évaluation, par ce dernier, des deux objets réunis est égale à la somme des évaluations de chaque objet à part.

4 - 4 - 3 La chorégraphie de l'enchère

La chorégraphie de l'enchère détermine si les participants ont le droit de modifier leur mise, et de quelle manière [Abrache01b]. De ce point de vue, on distingue deux types d'enchères :

- *Les enchères mono-rondes* : dans ce type d'enchère, les participants n'ont le droit de miser qu'une seule fois. Le mécanisme reçoit les mises et réalise l'allocation une fois pour toute.
- *Les enchères multi-rondes* : dans ce cas, l'enchère se fait en plusieurs rondes, et au cours de chaque nouvelle ronde, les participants peuvent améliorer leur mise. Au bout de chaque ronde, il y a une allocation temporaire, l'allocation définitive est réalisée au bout de la dernière ronde. Le nombre de rondes peut être pré-déterminé ou peut être déterminé dynamiquement suivant la progression du marché. Ce type d'enchère permet aux participants d'ajuster leurs mises suivant ce qui se passe dans le marché et donc de soumettre des mises plus réalistes.

4 - 4 - 4 Les enchères dans les marchés continus et dans les marchés discrets

On distingue deux types de marchés par rapport au moment où les enchères commencent dans le marché :

4-4-4-1 Les marchés continus

Dans un *marché continu* (figure 4-2), les objets sont annoncés de manière asynchrone et à chaque fois qu'un objet est annoncé, une enchère est immédiatement lancée sur ce dernier. On se trouve donc avec des enchères indépendantes qui se déroulent en parallèle. L'encanteur reçoit les mises sur un objet de façon continue jusqu'à ce que la règle d'arrêt de l'enchère sur cet objet soit vérifiée auquel cas une allocation est réalisée : les enchères sont donc mono-rondes. De plus, puisque ces enchères sont indépendantes, on ne peut pas faire d'enchères combinatoires dans les marchés continus.

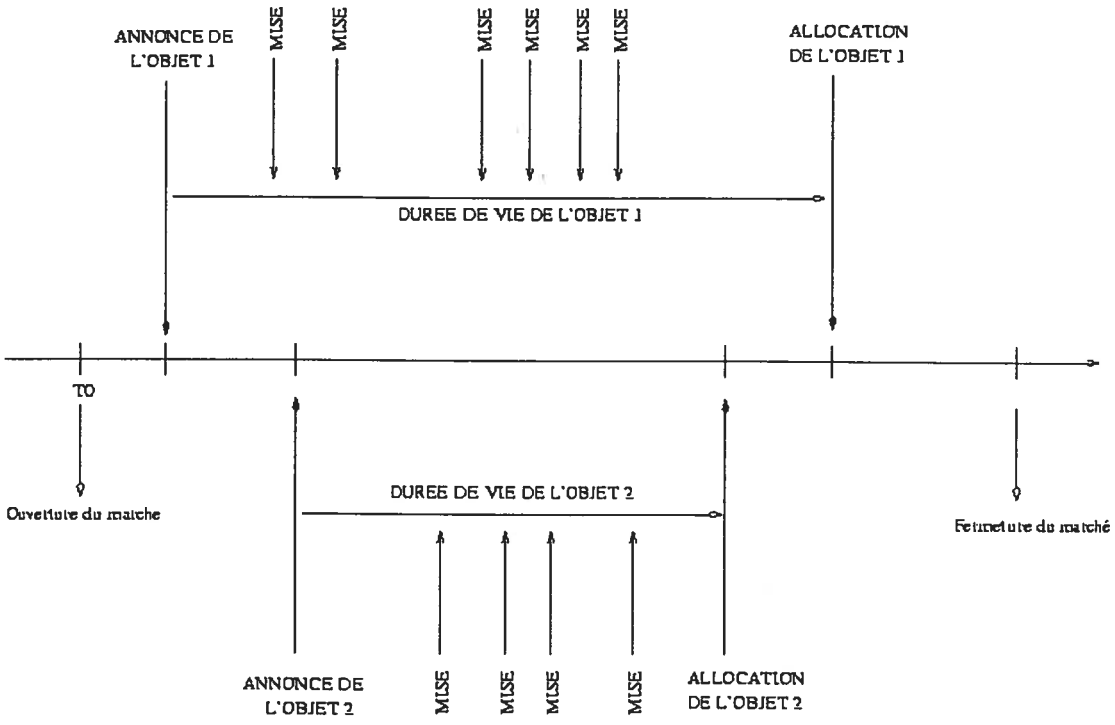

Figure 4-2 Déroulement d'un marché continu.

4-4-4-2 Les marchés discrets

Dans un *marché discret*, les enchères commencent en même temps, à un moment précis spécifié par l'encanteur. On distingue alors deux cas, par rapport au moment où interviennent les acheteurs et les vendeurs :

- Le premier cas est illustré par la figure 4-3. Les enchères sont précédées par une période d'annonces où les participants annoncent leurs objets. Immédiatement, après la fin de la période des annonces, commencent les enchères sur les objets annoncés (ou une seule enchère combinatoire). Les acheteurs et les vendeurs interviennent ainsi en deux périodes différentes du marché. On parle alors d'un *marché discret avec positions d'achat et vente exclusives.*

- Dans le deuxième cas, les enchères commencent immédiatement au lancement du marché. Chaque participant peut soumettre des mises de vente ou des mises d'achat en même temps. Dans le cas combinatoire, un participant peut même placer des mises où il combine des achats et des ventes. Un participant qui place une mise de vente ne sait pas forcément s'il y a des acheteurs intéressés, et un participant qui place une mise d'achat ne sait pas forcément s'il y a des vendeurs. Le mécanisme d'enchère reçoit alors toutes les mises d'achat et de vente et réalise les allocations des objets. Dans ce type de marchés, les acheteurs et les vendeurs interviennent au même moment, pour cette raison on l'appelle *marché discret avec positions d'achat et vente simultanées*.

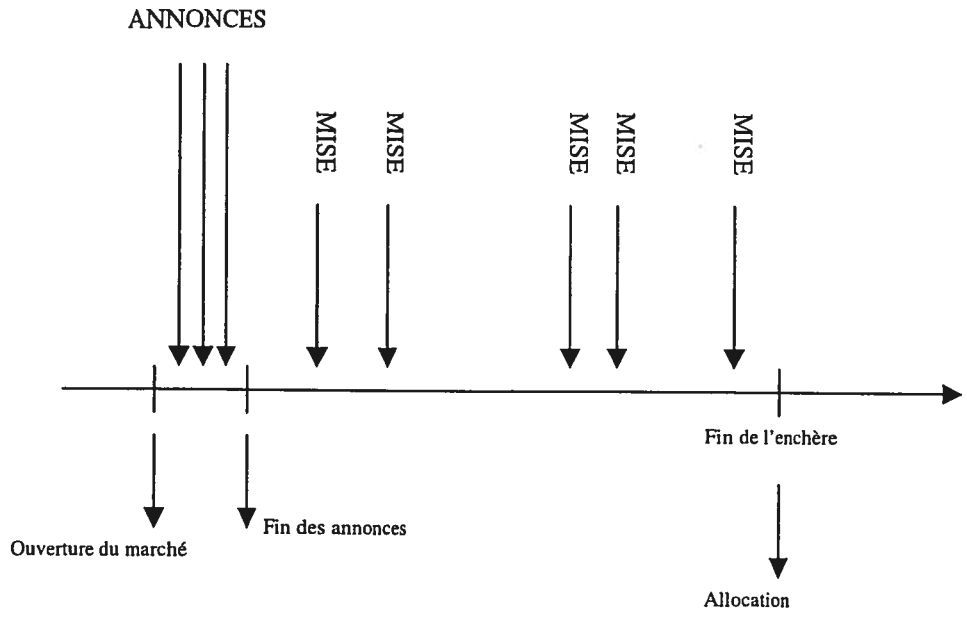

Figure 4-3 Déroulement d'un marché discret avec positions d'achats et ventes exclusives.

4 - 4 - 5 Enchères dans un marché multi-phases

Un marché multi-phases est un marché où les allocations se font en une succession de phases. Dans chaque phase, des enchères intermédiaires sur les objets ont lieu et dont les résultats sont des allocations provisoires, sauf pour la dernière phase où elles sont

définitives. Chaque phase peut ainsi être considérée comme une enchère qui applique son propre mécanisme. Le nombre de phases d'un marché peut être connu à l'avance ou déterminé dynamiquement en fonction de la progression du marché. Une phase peut être continue ou discrète, simple ou combinatoire, mono-ronde ou multi-rondes.

Une utilité économique de l'enchère multi-phases est qu'elle permet aux participants de mieux évaluer les objets négociés. Par exemple, dans certains marchés, on utilise deux phases où la première applique un mécanisme d'enchère anglaise dont l'objectif est de réchauffer le marché, et la deuxième utilise une enchère combinatoire qui aboutit aux allocations définitives.

La figure 4-4 montre la succession des phases dans un marché multi-phases. Les phases peuvent avoir des caractéristiques différentes : on peut, par exemple, avoir une phase continue suivie d'une phase multi-rondes avec positions d'achats et ventes exclusives, etc.. Les allocations de chaque phase sont provisoires sauf pour la dernière phase où elles sont définitives. L'enchère de chaque phase dépend des informations qu'elle reçoit de la phase précédente, à savoir les allocations provisoires et éventuellement d'autres informations (les mises, les participants inscrits, etc..).

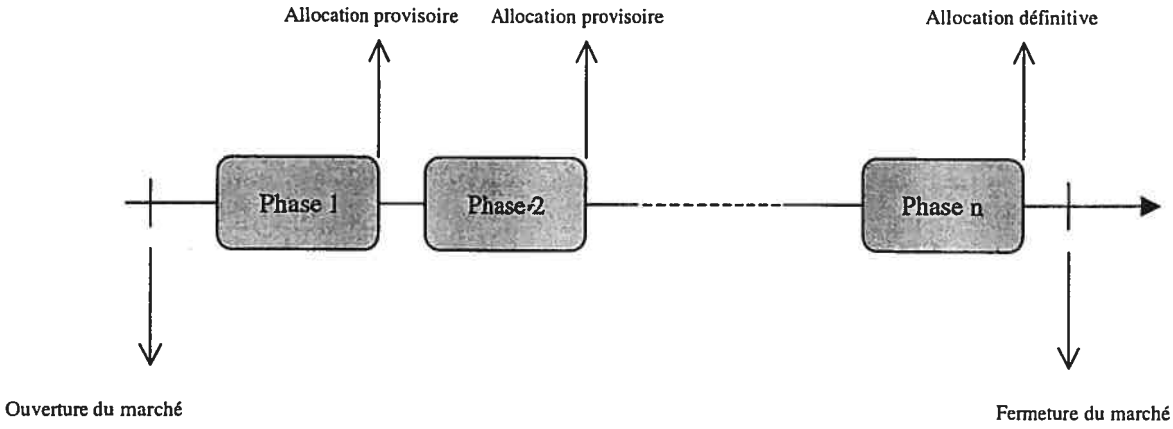

Figure 4-4 Succession des phases dans un marché multi-phases.

4-5 Le vocabulaire de mises

Un autre défi des enchères électroniques est de définir le vocabulaire de mises [Abrache02a]. Ce dernier définit l'ensemble des opérateurs permis, ainsi que la manière d'utiliser ces opérateurs pour formuler des mises combinées. Des exemples d'opérateurs classiques incluent le « OR », le « AND » et le « XOR ».

4-6 Les règles d'un marché

Les règles du marché sont l'ensemble des critères utilisés par l'encanteur pour prendre des décisions quant au déroulement de l'enchère et à l'acceptation des requêtes des participants. Les règles peuvent être exprimées dans différentes dimensions telles que le temps, le nombre des mises, le nombre de participants, etc.. On peut imaginer par exemple un marché où la règle d'ouverture du marché est exprimée dans la dimension temps : « le marché ouvre tous les jours à 8h00 », la règles de début de l'enchère est exprimée dans la dimension « nombre de participants inscrits dans le marché » : « l'enchère commence lorsqu'il y a 25 participants inscrits dans le marché », et la règle d'arrêt de l'enchère est exprimée dans la dimension « nombre de rondes »: « l'enchère finit au bout de cinq rondes ». Chaque marché peut avoir des dimensions qui lui sont propres. Les règles peuvent évidemment être exprimées en une combinaison de dimensions. On distingue trois types de règles :

- *les règles d'admissibilité* : ce sont les règles qui déterminent si une information provenant d'un participant doit être acceptée. On distingue deux types de règles d'admissibilité :
    - *Les règles d'admissibilité des mises* déterminent si une mise d'un participant doit être acceptée. Par exemple, dans une enchère multi-rondes, une règle d'admissibilité peut être : « l'amélioration d'une mise doit être supérieure à 10% dans chaque nouvelle ronde ».

- *Les règles d'admissibilité des annonces* déterminent si une annonce d'un participant doit être acceptée.

- *Les règles d'activité* : ce sont les règles qui déterminent le comportement exigé d'un participant pour rester actif dans le marché. L'objectif de ce genre de règles est d'obliger les participants à donner leurs vraies évaluations des objets au début de l'enchère pour éviter les risques de jeu [Abrache01b].

- *Les règles d'arrêt* : c'est l'ensemble des critères qui déterminent quand est-ce qu'une étape de l'enchère doit finir. On distingue plusieurs types de règles d'arrêt :

  - *Les règles d'arrêt de l'enchère* déterminent la condition de fin de l'enchère.

  - *Les règles d'arrêt du marché* déterminent la condition de fin du marché.

  - *Les règles d'arrêt de phase* déterminent les critères d'arrêt de chaque phase de l'enchère.

  - *Les règles d'arrêt de l'enchère sur un objet annoncé* déterminent, dans le cas d'une phase continue, l'ensemble des critères de fin de l'enchère sur chaque objet annoncé.

  - *Les règles d'arrêt de ronde* déterminent, dans le cas d'une phase discrète, l'ensemble des critères de fin de chaque ronde.

  - *Les règles d'arrêt des annonces* déterminent, dans le cas d'une phase discrète avec positions d'achat vente exclusives, l'ensemble des critères de fin de la période des annonces.

- *Les règles d'ouverture du marché* déterminent la condition de l'ouverture de chaque occurrence du marché. On peut, par exemple, avoir la règle suivante : « Le marché ouvre tous les jours à 8h00 ».

- *Les règles de début de l'enchère* déterminent la condition de début de chaque enchère dans le marché.

4-7 Les participants et les aviseurs

Chaque participant dispose d'un module intelligent, appelé aviseur, qui le guide dans son comportement dans le marché. Le rôle le plus important de l'aviseur est de décider des *stratégies de mises*. Une stratégie de mise indique au participant les objets sur lesquels il doit miser, en quel moment et à quel prix. Un aviseur possède plusieurs caractéristiques :

- *Les marchés supportés* : un aviseur est développé pour participer à un ou plusieurs places de marchés particulières. Il doit connaître les caractéristiques de ces places de marché pour pouvoir s'y comporter correctement. Ainsi, l'aviseur doit connaître la nature et l'enchaînement des phases d'enchère, les mécanismes d'enchères utilisés, les règles du marché, etc..

- *Le système d'informations :* l'espace économique du participants englobe, entre autres, ses activités internes, ses activités externes, ses relations avec ses clients, ses relations avec ses fournisseurs et ses obligations envers ses clients. Par exemple, l'espace économique d'un transporteur englobe les informations sur la disponibilité des camions, les changements de la flotte, les coûts des opérations de transport, etc.. L'aviseur a besoin de ces informations, qui lui sont accessibles à travers un système d'informations, pour déterminer le comportement du participant dans le marché.

- *Profil d'aviseur :* le profil d'un aviseur définit le trait général du comportement de l'aviseur. Par exemple, on peut définir un aviseur ayant le profil « agressif » par celui qui fait des mises très compétitives pour éliminer les autres participants de l'enchère. La notion de profil dépend largement du domaine du marché, et une définition des profils possibles d'un aviseur ne peut être faite avant de connaître la nature du marché.

Chapitre 5
L'outil développé

5-1 Le prototype de marché

Dans le chapitre précédent, nous avons décrit les éléments communs aux marchés ciblés par notre outil de simulation. La description donnée, qui a constitué notre point de départ, ne décrit pas un prototype de marché, mais décrit les éléments servant à la construction de prototypes de marchés. Aussi, la première partie de notre tâche consiste-t-elle à développer un prototype de marché paramétrable basé sur ces éléments de marchés. Les paramètres de ce dernier sont les aspects spécifiques à chaque cas de marché, c'est-à-dire les aspects qui distinguent un cas particulier de marché par rapport aux autres (les règles du marché, les phases du marché, les mécanismes d'enchères, etc..). Une fois le prototype développé, nous avons développé l'outil qui permet à l'utilisateur de développer la simulation du prototype en spécifiant uniquement les aspects spécifiques de son cas particulier de marché. Dans la suite de ce chapitre, nous allons décrire le prototype développé.

Présences des participants dans le marché

Dans notre prototype, nous supposons que tous les participants sont présents en permanence dans le marché. Par « présent », nous voulons dire que le participant existe dans le marché, ce qui ne signifie pas qu'il est actif, mais qu'il est capable d'observer en permanence ce qui se passe dans le marché et de décider, quand il en trouve les raisons, de s'engager dans une enchère ou une autre.

Inscription des participants

Avant de pouvoir participer à une enchère dans une place de marché, un participant doit, au préalable, faire une *inscription* à l'enchère en question auprès de la place de marché.

Une période d'inscriptions doit donc avoir lieu avant le début de chaque enchère, où les participants soumettent une demande d'inscription à la place de marché organisatrice de l'enchère. Ceci permettra à la place de marché de connaître tous les participants dans une enchère, ce qui lui sert, entre autres, de contrôler les activités des participants.

De plus, un participant est développé pour participer dans des places de marchés particulières dont il connaît les caractéristiques, mais il n'est pas supposé être capable de participer à toutes les places de marché. Cette caractéristique est très importante, autrement, les participants doivent être capables de détecter dynamiquement les caractéristiques d'une place de marché et de s'y adapter, ce qui dépasse le cadre de notre projet de simulation.

Le vocabulaire utilisé

Aucun vocabulaire suffisamment général, flexible et succinct n'existant dans le domaine des enchères combinatoires, un groupe de recherche du projet TEM a proposé un cadre méthodologique que nous utilisons pour notre prototype. Cette architecture, décrite dans [Abrache04], est basée sur une représentation à deux niveaux d'une mise combinée. Au niveau interne, des mises atomiques désignent des ordres d'achat ou de vente d'objets, et des opérateurs de mise sont utilisés pour exprimer des conditions reliées aux proportions d'exécutions des mises atomiques, formant ainsi des mises partielles. Au niveau externe, les mises partielles sont combinées par l'intermédiaire d'opérateurs de mises logiques, pour former une mise finale que le participant soumet à l'encanteur. Un vocabulaire de mise particulier est une instance particulière du cadre méthodologique.

Bien qu'il est possible que d'autres vocabulaires de mises soient développés dans le futur, nous limitons notre prototype de marché, par souci de simplification, pour qu'il supporte uniquement les vocabulaires conformes à l'architecture décrite ci-dessus. La mise à jour de l'outil pour supporter d'autres vocabulaires de mises pourra faire l'objet de projets futurs. Deux places de marchés différentes peuvent utiliser des vocabulaires de mises différents, pourvu que le vocabulaire en question soit conforme à l'architecture décrite plus haut.

5 - 1 - 1 Scénarios du marché

Dans ce sous-paragraphe, nous développons les scénarios d'exécution du prototype de marché. Nous commençons par donner le scénario global du marché, ensuite, nous détaillons le scénario global en allant en profondeur.

Pour compléter les scénarios du marché, nous avons introduit la notion de « signaux ». Un signal représente une information qu'envoie une place de marché aux participants du marché pour signaler le changement d'état de la place de marché (début d'une enchère, fin d'une ronde, etc..). Un signal peut également être utilisé par la place de marché pour signaler à un participer l'acceptation ou le refus d'une mise ou d'une annonce émise par ce participant.

Nous représentons les scénarios sous forme graphique. Dans tous les schémas des scénarios, les flèches verticales représentent les différents intervenants dans le marché. Les flèches sont dirigées vers le bas pour indiquer que le scénario évolue dans le temps en allant du haut du schéma vers le bas. Les flèches horizontales représentent les interactions entre les différents intervenants. Lorsque la flèche est en ligne double, elle indique que l'interaction est un signal. Une flèche en pointillés représente une boucle, c'est-à-dire une séquence d'actions qui se répète.

Scénario global d'un marché

Le scénario du déroulement global d'un marché est donné dans la figure 5-1. La figure représente une seule place de marché avec plusieurs participants, mais en réalité, plusieurs places de marché peuvent avoir lieu en même temps. Le déroulement de la place de marché est une succession d'occurrences de marché. Le début de chaque occurrence est déterminé par les règles d'ouverture du marché, ce qui permet de définir des moments précis pour l'ouverture du marché. Ainsi, si une occurrence de marché finit, l'encanteur détermine l'instant de début de l'occurrence suivante en se basant sur les règles d'ouverture du marché. Au bout de chaque occurrence, l'encanteur consulte les

règles d'arrêt de marché. Si ces règles indiquent la fin du marché, alors l'encanteur finit le marché et envoie un signal de fin de marché, sinon il lance une nouvelle occurrence.

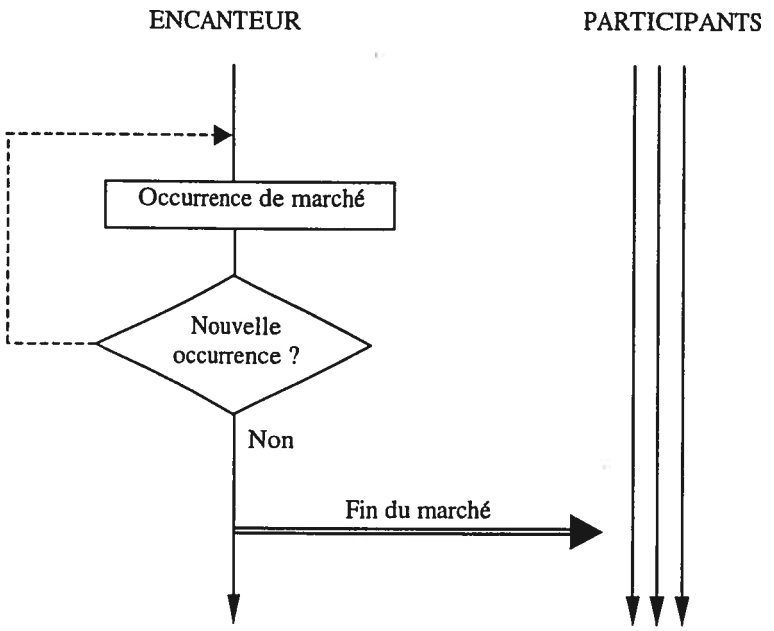

Figure 5-1 Scénario global d'un marché.

Scénario d'une occurrence de marché

Au niveau d'une place de marché, chaque occurrence de marché se déroule conformément au scénario donné dans la figure 5-2. L'occurrence commence par une initialisation du marché. Ensuite, le marché est ouvert et un signal d'ouverture du marché est envoyé aux participants. Ensuite, débute une période d'inscription et un signal correspondant est envoyé aux participants. Lorsque les règles du marché indiquent le début de l'enchère, un signal de début d'enchère est envoyé pour signaler la fin de la période des inscriptions et le début de l'enchère. La succession des phases est ensuite appliquée. Le début de chaque phase est annoncé aux participants, à travers le signal « début phase ». La fin de chaque phase est déterminée par les règles d'arrêt de phases.

Chaque phase se déroule conformément à un scénario qui dépend de la nature de la phase (continue, discrète avec positions d'achats et ventes simultanées ou discrète avec positions d'achats et ventes exclusives). A la fin de l'enchère, un signal de fin d'enchère est envoyé aux participants suivi d'un signal de fermeture de marché.

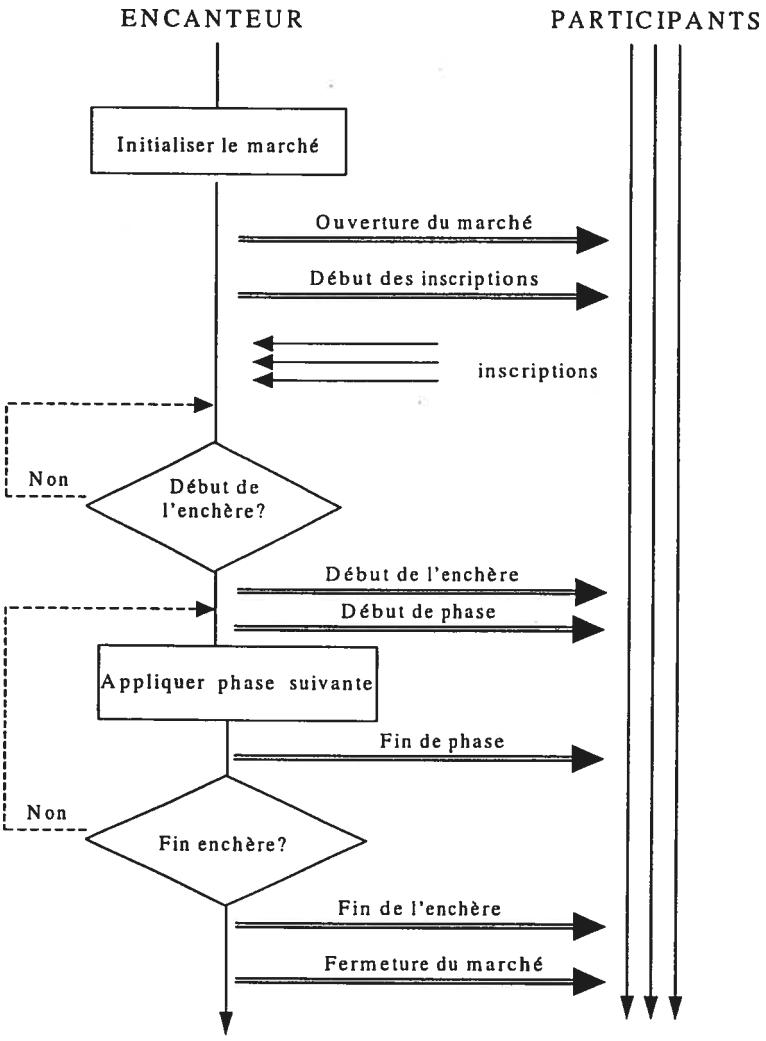

Figure 5-2 Scénario d'une occurrence de marché.

Scénario d'une phase continue

La figure 5-3 montre le déroulement d'une phase continue. Comme nous l'avons expliqué dans le chapitre IV, dans une phase continue, les annonces viennent de façon asynchrone et dès qu'une annonce est réalisée, une enchère est lancée sur l'objet annoncé. Nous rappelons également que les mises combinatoires ne sont pas permises dans le cas d'une phase continue. Dans la figure, on ne représente que le scénario d'une enchère sur un objet A mais, en réalité, le même scénario peut avoir lieu, sur des objets différents, en tout moment de la phase. La figure représente également un seul participant annonceur et un seul participant miseur, mais dans la réalité, il peut y en avoir plusieurs. En premier lieu, l'objet est annoncé par le participant annonceur. L'encanteur vérifie alors les règles d'admissibilité sur l'objet et renvoie, en conséquence, un accord ou un refus de l'objet. Dans le cas de notre schéma, nous supposons que l'objet est accepté, alors l'encanteur envoie un signal d'acceptation au participant annonceur et reflète l'annonce aux autres participants. Chaque participant intéressé peut miser sur l'objet. L'encanteur vérifie chaque mise en utilisant les règles d'admissibilité des mises et envoie une acceptation ou un refus au participant ayant soumis la mise. Lorsque les règles d'arrêt d'une ronde indiquent un arrêt de ronde, l'encanteur déclare une nouvelle ronde, et réalise une allocation[1] provisoire. Lorsque les règles d'arrêt de l'enchère indiquent la fin de l'enchère, l'encanteur réalise une allocation définitive.

---

[1] Bien que le terme « allocation » est usuellement utilisé pour désigner les objets gagnés et les gagnants de l'enchère, et le terme « prix » pour désigner le prix à payer contre les objets gagnés, nous utiliserons, dans le reste de ce document, par abus de langage, le terme « allocation » pour désigner à la fois les objets gagnés, les gagnants et les prix à payer. Donc, quand nous disons que l'encanteur réalise une allocation, nous signifions qu'il détermine les objets gagnés, les gagnants et les prix.

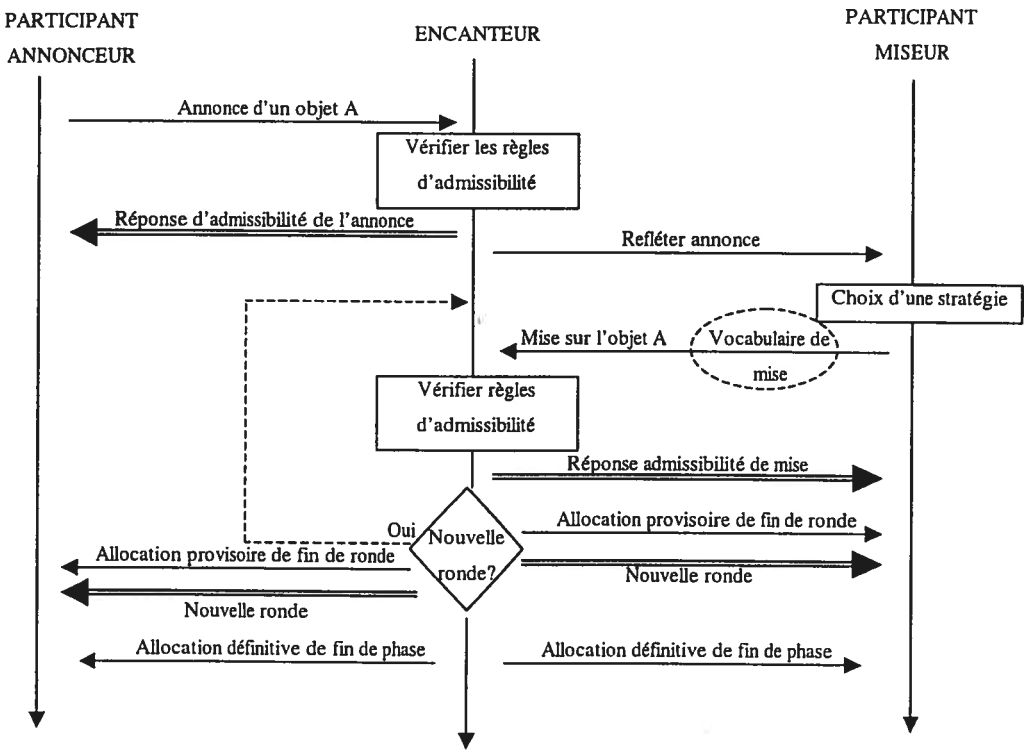

Figure 5-3 Scénario d'une phase continue (avec un seul objet annoncé).

Scénario d'une phase discrète avec positions d'achats et ventes exclusives

La figure 5-4 montre le déroulement d'une phase discrète avec positions d'achats et ventes exclusives. L'encanteur commence la phase par déclarer une période d'annonces. Pendant cette période, les participants envoient leurs annonces à l'encanteur, qui vérifie chaque annonce en utilisant les règles d'admissibilité. Selon que l'annonce est acceptée ou non, un accord ou un refus est envoyé au participant qui l'a générée. Les annonces acceptées sont reflétées aux autres participants. Au bout de la période des annonces, déterminée par les règles d'arrêt de période d'annonces, les participants sont informés, la période des enchères sur les objets annoncés commence immédiatement, et les participants peuvent commencer à miser. Lorsque les règles d'arrêt de ronde indiquent une fin de ronde, l'encanteur réalise une allocation. Si la ronde est la dernière alors l'allocation est définitive pour la phase en cours, sinon elle est provisoire.

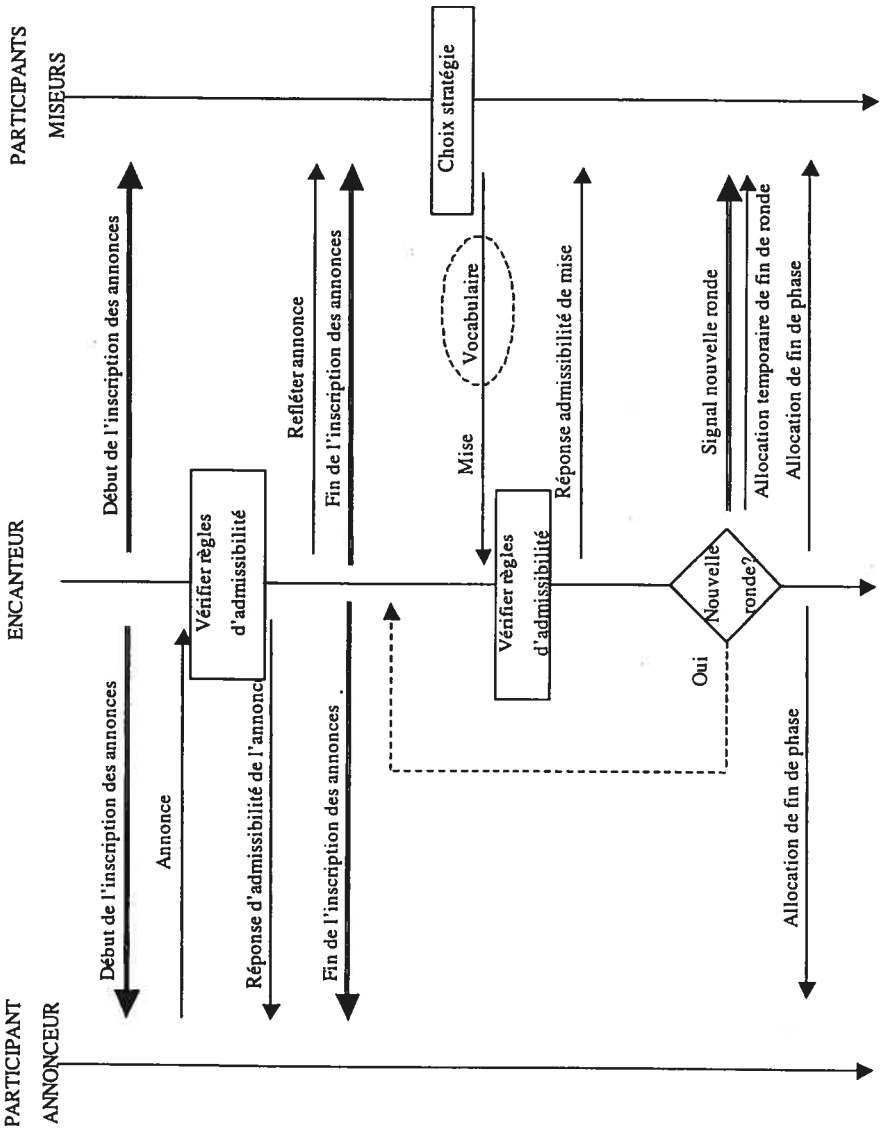

Figure 5-4 Scénario d'une phase discrète avec positions d'achats et ventes exclusives.

Scénario d'une phase discrète avec positions d'achats et ventes simultanées

La figure 5-5 montre le déroulement d'une phase discrète avec positions d'achats et ventes simultanées. Dans ce cas, il n'y a pas de période d'annonces. La première ronde

est entamée au lancement de la phase. A l'intérieur de chaque ronde, les participants soumettent leurs mises. Lorsque les règles de d'arrêt de ronde indique une fin de ronde, l'encanteur envoie une allocation temporaire et un signal de fin de ronde, si la ronde n'est pas la dernière, sinon il envoie une allocation définitive.

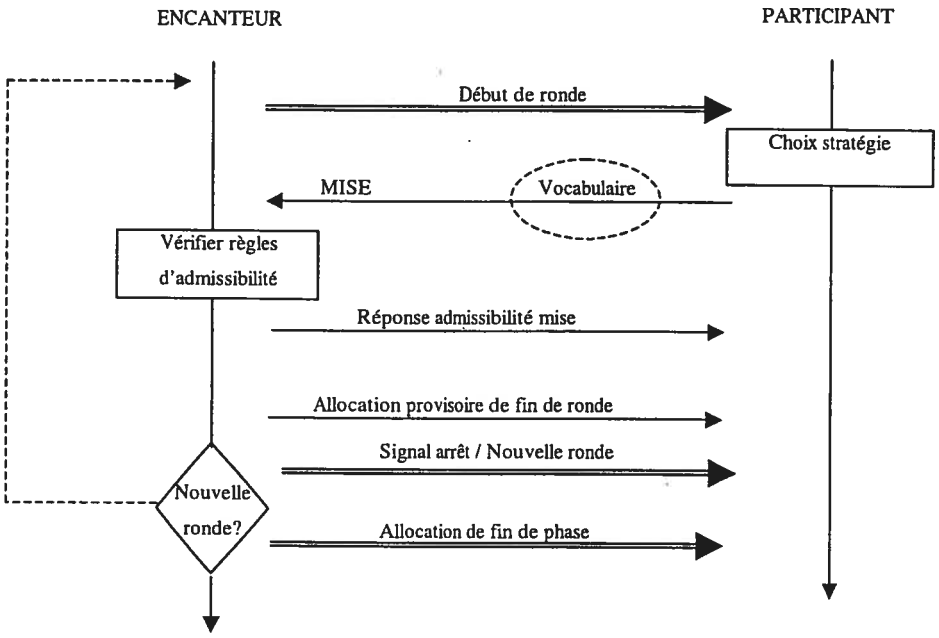

Figure 5-5 Scénario d'une phase discrète avec positions d'achats et ventes simultanées.

5 - 1 - 2 Identification des aspects spécifiques des marchés

Notre outil doit encapsuler les aspects génériques de la simulation du prototype de marché et donner à l'utilisateur le moyen (une interface) pour définir les aspects spécifiques à son cas particulier. Ces aspects spécifiques, que nous identifions dans ce paragraphe, forment l'entrée pour l'instanciation de la simulation d'un marché particulier.

Chaque cas de marché se distingue par :

- Le nombre et les caractéristiques des places de marchés qui y sont présentes.

- Le nombre et les caractéristiques des participants qui y participent.

De son côté, une place de marché se caractérise par les spécificités suivantes :

- La liste ordonnée des phases de l'enchère et la nature de chaque phase (continue, discrète avec positions d'achats et ventes simultanées ou discrète avec positions d'achats et ventes exclusives).

- Le mécanisme d'enchère appliqué dans chaque phase.

- Les règles de la place de marché.

- Le vocabulaire des mises utilisé par la place de marché.

- La nature physique des objets transigés dans la place de marché.

Tous les autres aspects spécifiques d'une place de marché (durée de la place de marché, nombre d'occurrences, nombre de rondes, durée de chaque ronde, etc.) peuvent être exprimés sous forme de règles de marché. Prenons l'exemple d'une place de marché qui ouvre tous les jours de 08h00 à 16h00, qu'on veut simuler sur une période de 5 jours : il y a donc cinq occurrences de marché. Supposons que dans notre simulation, une unité de temps logique correspond à une minute du temps physique et que le temps logique t=0 correspond à 08h00 de la première journée du marché. Pour implanter cette situation, nous avons besoin de trois règles :

- Une règle d'ouverture du marché qui détermine le moment d'ouverture de chaque occurrence du marché, pour chaque journée. Elle peut être exprimée de la manière suivante : « ouverture du marché lorsque $t = 0 + (N_{occ} - 1)* (24*60)$ », où N_{occ} est le numéro de l'occurrence du marché qui commence à 1. Le nombre 0 représente le temps logique correspondant à 8h00 du premier jour. Le nombre 24*60 représente le nombre de minutes dans une période de 24 heures.

- Une règle d'arrêt d'occurrence de marché qui détermine le moment d'arrêt de chaque occurrence de marché. Elle peut être exprimée de la manière suivante : « fin d'occurrence du marché lorsque $t = (16 - 8) * 60 + (N_{occ} - 1)* (24*60)$ », où N_{occ} est le numéro de l'occurrence du marché qui commence à 1. Le nombre (16-8)*60 représente le temps logique correspondant à 16h00 du premier jour. Le nombre 24*60 représente le nombre de minutes dans une période de 24 heures.

- Une règle d'arrêt du marché qui détermine la fin du marché. Elle peut être exprimée de la manière suivante : « fin du marché lorsque $N_{occ} = 5$ », où N_{occ} est le numéro d'occurrence du marché.

De son côté, un participant se caractérise par les spécificités suivantes :

- L'aviseur : nous considérons que l'aviseur est une boite noire et nous laissons à l'utilisateur le soin de l'implanter. Nous ne fournissons que la définition de l'interface entre l'aviseur et son participant. En effet, la structure d'un aviseur dépend largement de l'objectif du participant, et donc nous croyons que c'est un tout, indivisible en modules, qui doit être entièrement re-développé pour chaque cas de marché.

- La nature des objets que le participant est capable de transiger.

5-2 Aspects stochastiques d'une simulation de marché

Dans la simulation de marchés, nous considérons que le seul aspect qui peut dépendre de données aléatoires est le comportement de l'aviseur, tout le reste de la simulation est déterministe. En effet, les simulations de marchés sont censées utiliser des modèles réels d'éléments de marché, sauf pour les aviseurs où des modèles simulés seront utilisés. Ainsi, les phases, les règles, les mécanismes d'enchères et les encanteurs ne sont pas simulés, mais des modèles réels de ces derniers seront utilisés. Un aviseur sera simulé

puisqu'il doit simuler le comportement interne du participant. Par exemple, la simulation de l'aviseur d'un transporteur englobe la simulation de la gestion de sa flotte, la gestion de ses relations avec les clients, le calcul des coups des opérations, etc..

Puisque le comportement interne du participant dépend largement de la nature de ce dernier, les aspects stochastiques de la simulation de l'aviseur dépendent à leur tour de la nature du participant et doivent donc être re-déterminés et re-implantés pour chaque cas de marché.

5-3 Vue d'ensemble de l'outil développé

L'outil développé est composé de quatre éléments :

1. Une librairie de classes C++ : elle contient les différentes classes C++ qui servent à l'utilisateur pour développer ses fédérés, qui sont de deux types :

   - Le fédéré « place de marché » : il simule une place de marché.
   - Le fédéré « participant » : il simule un participant et son aviseur.

   Par souci de simplification, nous utiliserons, dans la suite de ce document, les termes « fédéré PM », et « fédéré PA » pour désigner, respectivement, un fédéré « place de marché » et un fédéré « participant ».

2. Un format pour la description d'un vocabulaire de mises sous forme de fichier XML : c'est un format qui permet de décrire un vocabulaire de mises utilisé par une place de marché. Le fichier est fourni en entrée au fédéré PM.

3. Trois modèles d'objets HLA génériques : le FOM de la fédération, le SOM du fédéré PA et le SOM du fédéré PM. Ces trois modèles sont valables, quel que soit le marché simulé. L'utilisateur n'a donc pas à re-développer les modèles de données pour chaque instance de marché.

4. Un fédéré « moniteur » : c'est un fédéré générique, valable quelque soit l'instance de marché simulé, qui sert pour le lancement de la fédération, ainsi que pour la collecte et l'enregistrement des données échangées entre les fédérés pendant la simulation. Ces données servent comme partie de la sortie de la simulation.

Les places de marchés et les participants sont les entités de la simulation. Un fédéré PM implante la simulation de l'entité « place de marché », et un fédéré PA l'entité « participant ». Une fédération est constituée d'un ensemble de fédérés PA et de fédérés PM, où chaque fédéré PA simule une instance d'un participant et chaque fédéré PM simule une instance de la place de marché.

La figure 5-6 représente un fédéré PM et un fédéré PA, ainsi que les différents éléments, fournis par notre outil, qui permettent de les développer. Dans la figure, les rectangles à l'intérieur des fédérés représentent des objets C++. La librairie offre les classes *Encanteur*, *Phase*, *Regle*, *Mecanisme*, *Evenement* et *interfaceBD* utilisées au niveau du fédéré PM, ainsi que les classes *Participant*, *Aviseur* et *Evenement* utilisées au niveau du fédéré PA. La classe *interfaceRTI* représentée dans la figure est utilisée par les fédérés en interne et n'est pas visible à l'utilisateur, elle implante l'interface du fédéré avec la RTI. D'autres classes de moindre importance sont utilisées par la librairie en interne, et ne sont pas représentées dans la figure.

Tous les éléments représentés sur la figure (à l'exception de la RTI) sont génériques et donc implantés par l'outil, sauf les éléments suivants:

- les règles du marché et les mécanismes d'enchères au niveau du fédéré PM,
- l'aviseur au niveau du fédéré PA,
- le fichier de vocabulaire de mises utilisé par le fédéré PM.

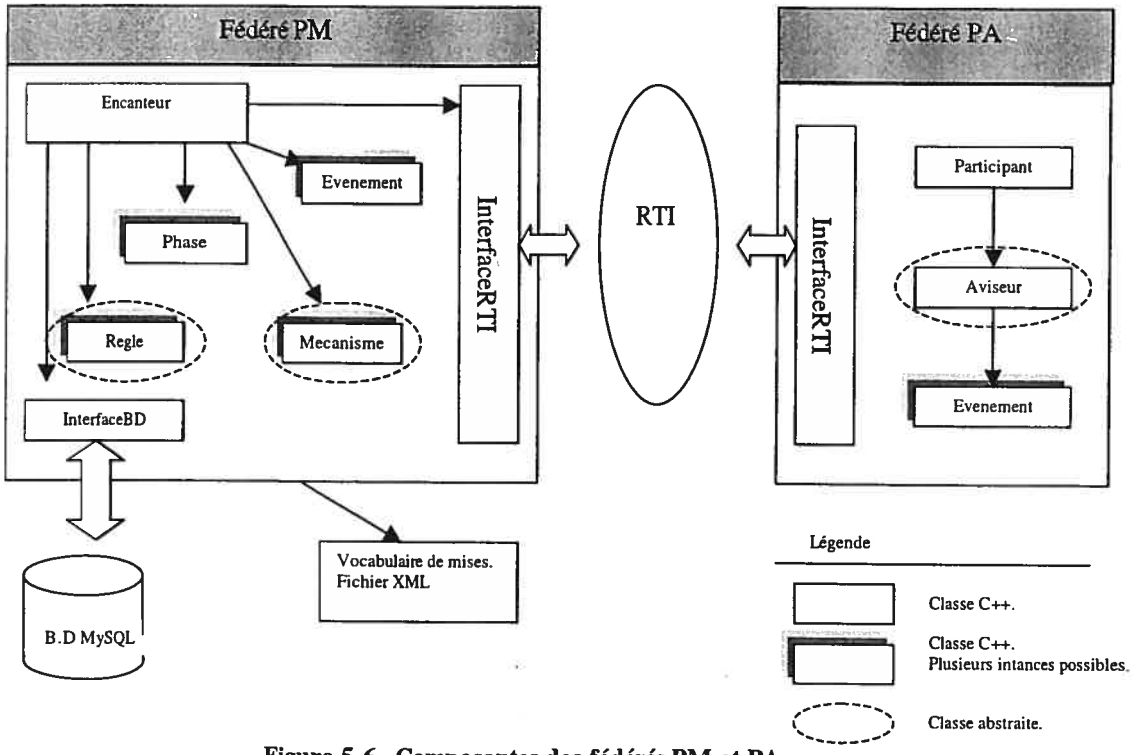

Figure 5-6 Composantes des fédérés PM et PA.

Les règles du marché, les mécanismes d'enchères et les aviseurs sont des boîtes noires dont l'outil ne fournit que l'interface, puisque leur comportement dépend de chaque fédéré et doit donc être ré-implanté pour chaque nouveau fédéré. Ces boîtes noires sont fournies par la librairie à travers les classes abstraites *Regle*, *Mecanisme* et *Aviseur*, entourées dans la figure par des ellipses en pointillés. En C++, une classe abstraite est une classe dont l'interface est définie, mais une ou plusieurs méthodes, appelées à leur tout des méthodes abstraites, ne sont pas implantées. Par exemple, la classe *Mecanisme* définit une méthode abstraite *allouer()* dont l'objectif est de calculer l'allocation. Cette méthode n'est pas implantée, puisqu'il y a un algorithme différent de calcul des allocations pour chaque mécanisme d'enchère. Le soin est laissé à l'utilisateur pour implanter cette méthode pour chaque mécanisme d'enchère. Le fichier « Vocabulaire de mise » représenté dans la figure 5-6 contient, au format XML, conformément à une structure que nous imposons, la spécification du vocabulaire de mise utilisé par la place de marché, qui

sert à la place de marché pour vérifier si les mises qu'il reçoit sont conformes au vocabulaire.

La classe *Evenement* est la racine de la hiérarchie des classes qui implantent les événements de la simulation. Ces classes sont utilisées tant au niveau du fédéré PM qu'au niveau du fédéré PA. La classe *Phase* est la racine de la hiérarchie des classes qui implantent les phases d'enchère. La classe *interfaceBD* est utilisée au niveau du fédéré PM pour accéder à son système d'information en lecture et en écriture. Le fédéré PA simule à la fois le participant et son aviseur, modélisés respectivement par les classes *Participant* et *Aviseur*.

La base de données *MySQL*, représentée dans la figure, est utilisée par le fédéré PM pour sauvegarder les informations sur toutes les enchères effectuées par la place de marché depuis le début de la simulation. Ces informations concernent les participants inscrits, les mises, les annonces et les allocations. Toute l'information sur un objet annoncé est représentée sous forme d'une chaîne de caractères, dont la structure est conventionnée par les fédérés, stockée dans un attribut de la table « Annonce ». Ainsi, la structure de la base de données devient indépendante de la nature physique des objets. C'est donc une base de données générique, valable quelque soit la nature du marché simulé. La base de données est accessible par l'encanteur à travers un objet instancié de la classe *InterfaceBD*.

Pour être conformes au standard HLA, les fédérés et la fédération doivent avoir des modèles de données rédigés dans le format OMT de HLA. Au niveau des modèles de données, nous avons choisi de modéliser une mise sous forme d'une interaction HLA ayant un seul paramètre qui décrit toute la mise sous forme d'une chaîne de caractères dont la structure est conventionnée par les fédérés, ce qui rend la structure des modèles de données indépendante du vocabulaire de mises. De même, nous avons modélisé une annonce par une interaction HLA ayant un seul paramètre qui décrit toute l'information sur l'objet annoncé, sous forme d'une chaîne de caractères, ce qui rend la structure des modèles indépendants de la nature des objets transigés. Avec ce choix, nous avons rendu les modèles de données indépendants de la nature du marché, ils sont donc génériques.

Ces modèles de données, qui consistent en un SOM pour le fédéré PA, un SOM pour le fédéré PM et un FOM pour toute la fédération, sont fournis par notre outil.

Notre outil offre, en outre, un fédéré spécial que nous avons appelé *Moniteur* qui doit faire partie de toute simulation de marché. Son objectif est de lancer la simulation, puis de collecter les événements échangés entre les fédérés et de les enregistrer dans un fichier qui constitue la sortie de la simulation. Le contenu du fichier sert de données bruts pour le calcul de statistiques. Le moniteur affecte également à chaque fédéré un code unique au niveau de la fédération, passé au fédéré en ligne de commande.

5 - 3 - 1 Représentation des objets transigés

En dehors des boîtes noires (classes abstraites) représentées dans la figure 5-6, les objets transigés sont représentés sous forme d'une chaîne de caractères, ce qui rend les différents éléments, autres que les classes abstraites, indépendants de la nature physique des objets transigés, et donc génériques. Ceci nous a permis de développer ces éléments une fois pour toutes et de les fournir comme partie de notre outil. Ainsi, l'encanteur, les phases, la base de données du fédéré PM, et le participant au niveau du fédéré PA sont génériques et fournis pas notre outil. Nous n'imposons cependant pas une structure spécifique pour la chaîne de caractères, mais nous laissons à l'utilisateur le soin de définir la structure qui convient à son cas de marché. A l'intérieur des classes abstraites, l'utilisateur est libre de choisir une représentation pour ses objets physiques. Il peut, par exemple, les représenter par des objets C++. Cependant, l'utilisateur prend la responsabilité de codifier les objets sous forme de chaîne de caractères avant de les échanger avec les autres classes utilisées par les fédérés.

Supposons, par exemple, que dans un marché de transport, un objet physique (ordre de transport) est défini par les informations suivantes : la ville de départ, la ville d'arrivée, la date de départ, le volume de la marchandise et la nature de la marchandise. Dans ce cas, un objet physique peut être représenté, par exemple, par une chaîne de caractères qui représente les différentes informations séparées par des points-virgules, avec l'ensemble entre accolades :

{Ville de départ ; ville d'arrivée ; date de départ ; volume ; nature de la marchandise}.

Un exemple d'un ordre de transport peut, par exemple, être :

"{MONTREAL ;QUEBEC ;05/02/2004 ;3000 ;CARBURANT}".

5 - 3 - 2 Aperçu de l'exécution d'une fédération

Dans une fédération d'un marché, le moniteur est lancé en premier lieu. Ce dernier reçoit un fichier en entrée contenant l'information sur les différents fédérés qui doivent participer à la fédération, l'exécutable de chaque fédéré et l'adresse de la machine où il doit être exécuté. Avant de lancer la simulation, le moniteur doit s'assurer, à travers les données du MOM, que tous les fédérés sont prêts, c'est-à-dire qu'ils ont tous joint la simulation. Au lancement du marché, chaque place de marché et chaque participant a un code qui l'identifie d'une manière unique au niveau du marché.

Au niveau du fédéré PM, c'est l'objet *Encanteur* qui contrôle l'exécution de la simulation de la place de marché. Au lancement, le programme principal (développé par l'utilisateur) du fédéré PM doit instancier un objet *Encanteur* et lui donner immédiatement la main, en appelant la méthode *Encanteur.executer()*. L'encanteur commence par initialiser la simulation. L'initialisation comprend, entre autres, la publication des classes HLA simulées par le fédéré, la souscription aux classes auxquelles il s'intéresse, l'initialisation du temps. Ensuite, il applique la boucle principale de la simulation de la place de marché. Chaque place de marché doit publier un objet de la classe HLA *CaracteristiquesPM,* qui modélise les caractéristiques de la place de marché (ses règles, ses phases, les mécanismes d'enchères qu'elle utilise, etc.). Tous les fédérés PA doivent se souscrire à cet objet et découvrir donc l'objet, ce qui permet à un fédéré PA d'identifier les fédérés PM auxquels il est capable de participer. Au niveau du fédéré PA, c'est l'objet *Participant* qui contrôle l'exécution de la simulation.

Les deux fédérés PM et PA fonctionnent en mode event-driven. Tous les événements externes, échangés par les fédérés, sont modélisés par des interactions HLA. Ces

événements sont : les mises, les annonces, les allocations, les inscriptions des participants et les signaux. Enfin, les fédérés utilisent la distribution de données offerte par le service DDM de HLA, pour éliminer les échanges d'événements inutiles.

5-4 Les modèles de données

Les modèles de données que nous avons développés, et qui sont fournis avec notre outil, sont indépendants du marché simulé, et sont donc valables pour tous les cas de marchés développés par notre outil. Dans le contexte de HLA, le FOM d'une fédération est généralement construit en fusionnant les SOMs de ses fédérés. Nous nous suffisons donc, dans ce paragraphe, de parler du FOM.

Le FOM définit une seule classe d'objets : la classe *CaracteristiquesPM*. Chaque fédéré PM publie un objet de cette classe et met à jour ses attributs une seule fois pendant toute la simulation. Chaque fédéré PA se souscrit à cette classe pour recevoir la mise à jour des attributs. Toutes les autres données échangées entre les fédérés ont été modélisées par des interactions. En effet, toutes ces données ne sont pas persistantes au cours de la simulation, et doivent donc naturellement être modélisées par des interactions. Notre modèle de données définit six classes d'interactions :

- La classe d'interaction *Mise* modélise une mise. Un fédéré PA envoie cette interaction pour soumettre une mise dans une place de marché.

- La classe d'interaction *Annonce* modélise une annonce. Un fédéré PA envoie cette interaction pour annoncer un objet dans une place de marché. L'interaction contient les informations concernant la description de l'objet annoncé, sa durée de vie, le prix minimum désiré, etc..

- La classe d'interaction *Inscription* modélise une demande d'inscription. Un participant envoie cette interaction pour demander une inscription à une enchère dans

une place de marché. L'interaction contient les informations concernant le participant en question.

- la classe d'interaction *Allocation* modélise une allocation. Un fédéré PM envoie cette interaction pour annoncer le résultat d'une allocation. L'allocation contient l'information sur le gagnant, la mise gagnante, la quantité gagnée et le prix.

- La classe d'interaction *Accord* modélise une acceptation, par une place de marché, d'une mise ou d'une annonce, envoyée au participant ayant soumis la mise ou l'annonce.

- La classe d'interaction *Refus* modélise un refus, par une place de marché, d'une mise ou d'une annonce, envoyée au participant ayant soumis la mise ou l'annonce. L'attribut *Motif* contient un code qui indique le motif du refus.

- La classe d'interaction *Signal* modélise un signal. L'attribut TypeSignal contient l'information sur la nature du signal (signal de début de phase, de fin de ronde, d'ouverture de marché, etc..).

Dans la table d'espaces de routage du FOM, nous définissons l'espace de routage *Destinataire* et la seule dimension *CodeFedere* qui servent aux fédérés pour router les événements à travers le service DDM. Nous parlerons de cet aspect plus loin dans ce chapitre, lorsque nous expliquerons comment notre outil utilise le service DDM.

5-5 Le fédéré PM

Au sein du fédéré PM, c'est l'encanteur qui contrôle l'exécution globale de la simulation. En particulier, l'encanteur initialise le fédéré, avance le temps, traite les événements externes, envoie les événements aux autres fédérés et met à jour l'état de la place de marché. Pour développer ce fédéré, l'utilisateur a besoin de quatre classes C++ fournies

par la librairie, qui sont les classes *Encanteur*, *Phase*, *Regle* et *Mecanisme*. L'encanteur utilise également une base de données et un fichier de vocabulaire de mises.

5 - 5 - 1 La base de données

La base de données relationnelle stocke les informations sur les participants, les annonces, les mises et les allocations de toutes les enchères en cours ainsi que toutes les enchères ayant eu lieu dans la place de marché depuis le lancement de la simulation. Les informations sur les enchères précédentes peuvent être utiles, puisque le comportement de la place de marché peut en dépendre. Par exemple, on peut imaginer que la durée de l'instance actuelle du marché dépend des résultats de l'instance précédente.

5 - 5 - 2 Aperçu sur les classes de la librairie

La classe Regle

La librairie offre un ensemble de classes qui permettent à l'utilisateur d'implanter les règles d'arrêt, les règles d'admissibilité et les règles d'activité. Les classes des règles sont abstraites puisqu'elles offrent une méthode abstraite *verifier()*, qui doit être implantée par l'utilisateur. Selon que la règle est d'admissibilité, d'activité ou d'arrêt, la méthode *verifier()* est utilisée d'une manière différente :

- Pour les règles d'admissibilité : l'encanteur appelle la méthode *verifier()* avec, en paramètre, une mise ou annonce dont la validité est à vérifier. La méthode retourne alors un boolean qui indique si la mise ou l'annonce est admissible.

- Pour les règles d'activité : lorsqu'à un instant donné de la simulation, l'encanteur désire vérifier l'activité d'un participant, il appelle la méthode *verifier()* de la règle d'activité en question avec, en paramètre, le code du participant. La méthode retourne un boolean indiquant si le participant doit rester actif.

- Pour les règles d'arrêt : la méthode *verifier()* teste une condition d'arrêt et retourne le résultat du test (un boolean). Lorsqu'à un instant donné, au milieu d'une ronde, par exemple, l'encanteur désire vérifier si la ronde doit être arrêtée, il appelle la méthode *verifier()* de la règle d'arrêt de ronde qui retourne le résultat du test de la condition d'arrêt de ronde.

Une règle peut être composée d'un ensemble de règles atomiques. La règle globale est vérifiée lorsqu'au moins une des règles atomiques est vérifiée. Chaque règle atomique est représentée par un objet de la classe Regle, et la règle globale est représentée par l'ensemble de ces objets. La vérification de la règle revient donc à vérifier toutes les règles atomiques, en appelant la méthode *verifier()* pour chacun des objets implantant ces règles.

La classe Mecanisme

La classe *Mecanisme* sert à l'utilisateur pour implanter un mécanisme d'enchère d'une phase. La classe est abstraite et offre la méthode abstraite *allouer()* appelée pour calculer les allocations à la fin de chaque ronde.

Dans le cas d'une phase continue, la méthode *allouer()* alloue un seul objet à la fois puisque les enchères sont indépendantes, elle prend donc en paramètre le code de l'objet transigé sujet de l'allocation et retourne un objet de la classe *Allocation* qui contient une référence sur la mise gagnante, un prix et une quantité allouée s'il y a lieu. Dans le cas d'une phase discrète, la méthode *allouer()* ne prend pas de paramètres puisqu'elle doit allouer l'ensemble de tous les objets transigés. Elle retourne un ensemble d'objets de la classe *Allocation*.

La classe Phase

La classe *Phase* est la racine de la hiérarchie des classes qui implantent les phases. Une phase continue, une phase discrète avec positions d'achats et ventes exclusives, et une

phase discrète avec positions d'achats et ventes simultanées sont modélisées par trois classes différentes. Un objet phase se caractérise principalement par :

- Le mécanisme d'enchère utilisé par la phase.

- Les règles de la phase.

Le cas de phases multi-rondes a été implanté en utilisant les règles d'arrêt de rondes et les règles d'arrêt de phases de la manière suivante : lorsque les règles d'arrêt de ronde indiquent la fin d'une ronde, l'encanteur examine les règles d'arrêt de phase. Si ces dernières indiquent une fin de phase, l'encanteur déclare la fin de la phase, sinon il déclare une nouvelle ronde. Par exemple, dans le cas d'une phase qui applique deux rondes de cinq unités de temps chacune, on peut attribuer à la phase la règle d'arrêt de ronde : « fin de ronde lorsque : t = temps de début de ronde + 5 », et la règle d'arrêt de phase : « fin de phase lorsque : nombre de rondes appliquées = 2 ».

La classe Encanteur

La classe *Encanteur* implante un encanteur. Elle utilise une liste ordonnée des phases de l'enchère qu'elle exécute dans l'ordre. La classe offre une méthode *executer*() dont l'objectif est d'appliquer la boucle de la simulation du côté de la place de marché.

5 - 5 - 3 La gestion du temps du fédéré PM

5-5-3-1 Le mode de synchronisation

Notre outil de simulation utilise le mode de synchronisation conservatrice pour le fédéré PM. Le mode de synchronisation optimiste n'est pas approprié à ce fédéré. En effet, un fédéré qui avance son temps en mode optimiste doit pouvoir faire une reprise à chaque fois qu'il reçoit un événement dans son passé. Le mode optimiste est approprié à un fédéré lorsque les reprises que doit faire ce dernier ne sont pas fréquents. Or, le fédéré PM reçoit fréquemment des événements de la part des fédérés PAs (mises, annonces,

inscriptions). Donc, si le fédéré PM avance son temps en mode optimiste, la probabilité qu'il reçoive des événements dans son passé est grande et, par conséquent, les reprises qu'il est appelé à faire sont fréquents. L'utilisation du mode de synchronisation conservatrice, par contre, ne présente aucune contrainte particulière.

5-5-3-2 La représentation du temps

Il est essentiel, dans notre cas, de faire une correspondance entre le temps logique et le temps physique, puisque nous simulons le temps. En effet, les règles du marché peuvent être exprimées dans la dimension temps, et donc une projection du temps physique en temps logique est nécessaire. Cependant, nous avons laissé à l'utilisateur le soin de choisir le rapport de projection, lors de l'implantation de sa simulation de marché. En effet, puisque l'utilisateur est amené à traduire les valeurs exprimées en temps physique dans le système réel, en des valeurs exprimées en temps logique avant de les utiliser dans le programme de simulation, nous lui donnons la possibilité de choisir un système de projection pour lequel cette traduction est la plus directe.

5-5-3-3 Le mode d'avancement du temps et le traitement des événements

Le fédéré PM ne met à jour son état qu'à l'occurrence d'un événement. Il doit donc naturellement être simulé en mode event-driven. En effet, le fédéré met à jour son état à l'occurrence d'un événement externe (mise, annonce ou inscription) ou du seul événement interne que nous avons introduit dans la simulation de la place de marché et dont l'occurrence aboutit à la vérification d'une règle d'arrêt. En effet, une règle d'arrêt peut être exprimée dans la dimension temps, et il est donc nécessaire qu'un événement ait lieu pour indiquer qu'une règle est vérifiée (c'est-à-dire que la condition d'arrêt implantée par la règle est vérifiée). Par exemple, si on suppose qu'une règle d'arrêt de ronde est « arrêt de ronde lorsque : t= 150 », alors il faut qu'au temps t = 150, le fédéré PM traite la fin de ronde. Ceci peut être réalisé en utilisant un événement spécial « arrêt de ronde », programmé, au début de la ronde, à t = 150, dont l'occurrence aboutit au traitement d'une fin de ronde. Cependant, cette méthode n'est pas possible lorsque la

règle utilise des valeurs de temps relatives, comme dans l'exemple de la règle d'arrêt de ronde suivante :

Arrêt de ronde lorsque : (t = temps de la dernière mise + 5).

Cette règle a pour objectif d'arrêter la ronde si aucune mise n'a été faite depuis cinq unités de temps. Dans le cas de cette règle, le temps de fin de ronde ne peut être connu à l'avance par l'encanteur, et l'utilisation d'un événement « arrêt de ronde » n'est donc pas possible. La solution que nous avons trouvée introduit un événement interne *VerifierRegle* dont l'occurrence aboutit à la re-vérification d'une règle particulière. En d'autres termes, à l'occurrence de l'événement *VerifierRegle*, la routine de l'événement appelle la méthode *verifier()* de cette règle afin de savoir si la condition d'arrêt est vérifiée. Nous allons expliquer comment fonctionne la méthode dans le cas des règles d'arrêt de ronde. Pour les autres types de règles d'arrêt, le fonctionnement de la méthode est similaire. Avec les règles d'arrêt de ronde, l'événement *VerifierRegle* est utilisé de la manière suivante : dès qu'une ronde débute, l'encanteur appelle, pour chaque règle d'arrêt de ronde, la méthode *InitialiserRegle()*, implantée par l'utilisateur. Cette dernière détermine le premier instant logique t_v où la règle d'arrêt de ronde doit être re-vérifiée, et programme un événement *VerifierRegle* à l'instant t_v : il y a donc un événement *VerifierRegle* programmé pour chaque règle. L'événement porte le code de la règle en question dans l'un de ses attributs. Lors du traitement d'un événement *VerifierRegle*, l'encanteur détermine la règle en question, puis appelle la méthode *verifier()* de cette règle. Si la condition d'arrêt de la règle n'est pas vérifiée, la méthode *verifier()* détermine le prochain instant logique $t_{v'}$ où la règle doit être re-vérifiée, reprogramme un nouvel événement *VerifierRegle* à l'instant $t_{v'}$ et retourne la valeur FALSE. Par contre, si la condition d'arrêt de la règle est vérifiée, la méthode *verifier()* retourne la valeur TRUE. Dans ce dernier cas, tous les événements *VerifierRegle* des règles d'arrêt de ronde sont retirés de la queue des événements, et donc annulés, puisque la ronde est terminée.

Pour mieux comprendre ce choix, nous proposons l'exemple d'un marché qui utilise les deux règles d'arrêt de ronde suivantes :

- Règle *R1*, instanciée de la classe *ClasseR1*: « fin de ronde lorsque : (t = TDR+10 et NMR =15) ou (t = TDR+20 et NMR = 15)) », où TDR est le temps de début de ronde et NMR est le nombre de mises reçus dans la ronde.

- Règle *R2*, instanciée de la classe *ClasseR2*: « fin de ronde lorsque (t = TDR+30) ».

L'objectif de la règle *R1* est d'accorder aux participants 10 unités de temps pour miser, à partir du début de la ronde. Si, au bout de cette période, au moins 15 mises sont reçues, alors la ronde doit s'arrêter, sinon, la ronde continue et 10 unités de temps supplémentaires sont accordées. Si, au bout de cette nouvelle période, le nombre total des mises reçues depuis le début de ronde est au moins égal à 15, alors la ronde s'arrête, sinon la ronde continue. La règle *R2* vient alors pour arrêter la ronde lorsque 30 unités de temps se sont écoulées après le début de la ronde, quelque soit le nombre de mises reçues.

La figure 5-7 représente l'implantation des classes *ClasseR1* et *ClasseR2*. Dès la déclaration d'une nouvelle ronde, l'encanteur appelle la méthode *InitialiserRegle()* des règles *R1* et *R2*. L'initialisation de *R1* entraîne la programmation d'un événement *E1* du type *VerififerRegle* à t=TDR+10 et celle de *R2* d'un événement *E3* du type *VerifierRegle* à t=TDR+30. Lorsque le fédéré atteint le temps logique t=TDR+10, il applique l'événement *E1*, ce qui entraîne la vérification de la règle *R1* (en appelant la méthode *verifier()*). Si, à cet instant, le nombre de mises reçues est supérieur à 15, alors la règle est vérifiée, la méthode *verifier()* retourne la valeur TRUE et l'encanteur déclare la fin de ronde et retire de la queue les événements *VerifierRegle* des règles d'arrêt de ronde (uniquement *E3* dans ce cas). Si, par contre, le nombre des mises et inférieur à 15, alors la méthode *verifier()* de *R1* programme l'événement *E2* avec t=TDR+20 et retourner la valeur FLASE. Dans ce dernier cas, le fédéré continue la simulation jusqu'à ce qu'il atteigne le temps logique t=TDR+20 et applique l'événement *E2*, ce qui entraîne la re-vérification de la règle *R1*. Si, à cet instant, le nombre de mises est supérieur à 15, alors la condition d'arrêt de la règle est vérifiée, la méthode *verifier()* retourne la valeur TRUE, et l'encanteur déclare la fin de ronde et retire de la queue les événements *VerifierRegle* des règles d'arrêt de ronde (uniquement *E3* dans ce cas). Si, par contre, le nombre des

mises est inférieur à 15, alors la méthode *verifier()* de *R1* retourne la valeur FLASE. Dans ce dernier cas, le fédéré continue la simulation jusqu'à ce qu'il atteint le temps logique t=TDR+30 et applique l'événement *E3*, ce qui entraîne la re-vérification de la règle *R2*. Dans ce cas, la condition d'arrêt de *R2* est vérifiée, puisque le temps t est égal à TDR+30, la fin de ronde est déclarée et tous les événement du type *VerifierRegle* relatifs aux règles d'arrêt de ronde sont supprimés (il n'y en a pas dans notre cas).

Le fédéré PM traite évidemment les événements externes, qu'il reçoit des autres fédérés, parallèlement aux événements internes *VerifierRegle*. Chaque événement externe reçu est automatiquement inséré dans la queue des événements. Un événement externe est toujours traité avant un événement interne *(VerifierRegle)* ayant la même étiquette de temps. En effet, le traitement de l'événement interne aboutit à la vérification d'une règle d'arrêt. Or, la condition d'arrêt de la règle porte sur l'état du fédéré. Il est donc nécessaire de mettre à jour l'état du fédéré, en appliquant les événements externes, avant de vérifier la règle.

```
// ----- Implantation de la règle ClasseR2 -----

ClasseR1::InitialiserRegle() {
    TDR = temps de début de la ronde en cours;
    CR = Code de cette règle;
    E1 = new évènement VerifierRegle(CR);
    Programmer E1 à t=TDR+10; // événrmemt qui aboutit à la première vérification de la règle
}

ClasseR1::verifier() {
    CR = Code de cette règle;
    TDR = temps de début de la ronde en cours;
    NMR = nombre de mises reçues depuis le début de la ronde en cours.

// Si on est à t=TDR+10 et nombre de mises reçus est au moins de 15.
// alors il y a fin de ronde
    Si (Now==TDR+10) alors
            Si (NMR>=15) alors
                    Retourner TRUE;
            Sinon
                    E2 = new évènement VerifierRegle(CR);
                    Programmer E2 à t=TDR+20;
                    Retourner FALSE;

// Si on est à t=TDR+20 et nombre de mises reçus est au moins de 15.
// alors il y a fin de ronde
    Si ( (Now==TDR+20) alors
            Si (NMR>=15) alors
                    Retourner TRUE;
            Sinon
                    Retourner FLASE;
    Sinon
            Retourner FALSE;
}

// ----- Implantation de la règle ClasseR2 -----

ClasseR2::InitialiserRegle() {
    TDR = temps de début de la ronde en cours;
    CR = Code de cette règle.
    E3 = new évènement VerifierRegle(CR);
    Programmer E3 à t=TDR+30; // événrmemt qui aboutit à la première vérification de la règle
}

ClasseR2::verifier() {

// Si on est à t=TDR+30 alors il y a fin de ronde
    TDR = temps de début de la ronde en cours;
    Si (Now==TDR+30) alors
            Retourner TRUE;
    Sinon
            Retourner FALSE;
}
```

Figure 5-7 Implantation d'un exemple de règles d'arrêt.

5-5-3-4 La valeur du lookahead

Dans le cas des simulations distribuées utilisant un algorithme conservateur, le choix du lookahead n'est pas toujours évident. En effet, il n'existe pas de méthode directe qui permet de déterminer le lookahead, mais le choix d'une bonne valeur pour le lookahead dépend largement du système simulé. Dans notre cas, nous pouvons dériver la valeur du lookahead à partir des imprécisions temporelles tolérées dans le marché. En effet, l'utilisation du lookahead introduit une imprécision temporelle dans les instants logiques d'envois des événements, puisqu'il engendre un décalage, dans le futur, de l'instant logique d'envoi d'un événement. Le lookahead peut donc être choisi de telle sorte que cette imprécision soit tolérée. Par exemple, si la simulation d'une place de marché génère un événement à un instant T, mais le fait qu'elle lui assigne l'étiquette de temps T+1 seconde est tolérable, alors la simulation peut utiliser un lookahead de 1.

Cependant, puisque l'imprécision temporelle tolérable peut différer d'une place de marché à l'autre, alors nous avons laissé à l'utilisateur le soin de choisir la valeur du lookahead, qu'il fournit en paramètre à la classe Encanteur.

5-5-3-5 La boucle principale de l'encanteur

L'encanteur exécute la boucle représentée dans la figure 5-8. Dans la figure, la variable *Now* représente le temps logique du fédéré, initialisé à la valeur 0 au début de la simulation. Dans la boucle, l'encanteur commence par l'appel *NextEventRequest(T)*, où T est le temps du prochain événement dans la queue. A travers cet appel, le fédéré demande la livraison du prochain événement externe, s'il y a lieu, ou avancer le temps de la simulation au temps du prochain événement dans la queue. Lorsque la RTI n'a aucun événement avec une étiquette inférieure ou égale à T à délivrer au fédéré, la RTI accorde le temps T au fédéré, à travers le callback *TimeAdvanceGrant()* dont le code remet le temps logique *Now* au temps accordé, à savoir T. Par contre, lorsque la RTI détient des événements avec des étiquettes de temps inférieures ou égales à T destinés au fédéré, la RTI délivre au fédéré l'événements ayant l'étiquette de temps la plus petite inférieure à T, soit T', ainsi que tous les événements à T', en appelant le callback *ReceiveInteraction()*

pour chaque événement, puis accorde au fédéré le temps T', à travers le callback *TimeAdvanceGrant()*. Le code du callback *ReceiveInteraction()* insère chaque événement reçu dans la queue des événements. Dans le callback *TimeAdvanceGrant()*, le fédéré remet la variable *Now* au temps accordé par la RTI. Au retour de l'appel *NextEventRequest()*, le fédéré traite les événements ayant une étiquette égale au temps logique *Now*.

```
Now := 0
Tant que (Simulation pas finie)
    T := temps du prochain événement dans la queue ;
    NextEventRequest(T) ;
    Traiter les événement dans la queue au temps égal à Now;
Fin Tant que ;

   // Le callbak ReceiveInteraction() est appelé au niveau du fédéré pour
   // chaque événement reçu.
   // Le code du callback insère tous les événements externes (interaction)
   // reçus dans la queue des événements.
ReceiveInteraction()
    Insérer l'événement reçu dans la queue;

   // Le callbak TimeAdvanceGrant() est appelé au niveau du fédéré
   // pour accorder un avancement du temps.
   // Le code du callback met la variable Now au temps accordé par RTI.
TimeAdvanceGrant(TimeGranted)
    Now := TimeGranted;
```

Figure 5-8 Boucle principale de l'encanteur

5-6 Le fédéré PA

La structure du fédéré PA est beaucoup plus simple que celle du fédéré PM, et l'interface qu'offre notre outil pour le développement du fédéré PA est, par conséquent, plus simple. Ceci est dû au fait que l'aviseur, qui implante la plus grande partie du comportement du participant, doit être implanté par l'utilisateur pour chaque participant. En effet, un aviseur est censé simuler le comportement interne du participant, puisque la prise de décisions faites par l'aviseur dépend de ce comportement. Par exemple, un aviseur d'un transporteur, participant dans un marché de transport, doit simuler la gestion de son espace économique qui englobe, entre autres, la gestion de sa flotte, la gestion de ses

relations avec les clients et le calcul des coups des opérations. Or, le comportement interne du participant est largement dépendant de la nature de ce dernier, et donc les aspects communs aux aviseurs ne sont pas nombreux. Nous considérons donc que l'aviseur est une boite noire dont nous ne définissons que l'interface avec le participant. Cependant, notre outil offre des services pour le développement du fédéré PA, à plusieurs niveaux :

- Il implante la partie du participant qui joue le rôle d'intermédiaire entre l'aviseur et le reste de la simulation. Cette implantation est réalisée à travers la classe *Participant*.

- Il définit une interface commune d'interaction entre un participant et un aviseur.

- Il implante l'avancement du temps, la gestion des événements et l'interaction avec la RTI, puisque ces aspects sont indépendants de la nature du participant.

5 - 6 - 1 Les classes Participant et Aviseur

Notre librairie offre deux classes C++ principales permettant de développer un fédéré PA : la classe *Participant* et la classe *Aviseur*, qui modélisent respectivement un participant et son aviseur. La classe Aviseur est naturellement abstraite, et son implantation doit donc être complétée par l'utilisateur selon les besoins.

L'objet *Participant* contrôle le déroulement de la simulation du participant. Il implante également la gestion du temps, la gestion de la queue des événements et l'interaction du fédéré avec la RTI. Le programme principal, développé par l'utilisateur, instancie un objet *Participant* et appelle la méthode *executer()* de cet objet, dont le but est d'exécuter la boucle principale de la simulation du participant.

5 - 6 - 2 La gestion du temps du fédéré PA

Notre outil de simulation utilise le mode de synchronisation conservatrice pour le fédéré PA. Le mode de synchronisation optimiste n'est pas approprié à ce fédéré. En effet,

puisque le fédéré PA reçoit fréquemment des événements de la part des fédérés PM, l'utilisation de la synchronisation optimiste l'obligerait à faire fréquemment des reprises.

Puisque le fonctionnement de l'aviseur dépend de sa nature, il n'est pas possible de décider à l'avance lequel des mécanismes d'avancement du temps, time-stepped et event-driven, est le plus approprié pour le fédéré participant. Toutefois, nous avons implanté uniquement le mode event-driven, puisque ce mode est plus approprié aux aviseurs existants qui seront simulés par notre outil. L'implantation du mode time-stepped pourra faire l'objet d'une extension future de l'outil. Puisque l'aviseur doit être re-implanté pour chaque cas de marché, nous ne pouvons pas, non plus, connaître à l'avance les types d'événements internes gérés par l'aviseur. L'outil doit donc offrir à l'utilisateur le moyen d'introduire ses événements internes et de les simuler.

Dans la solution que nous proposons, le travail de simulation est partagé entre le participant et l'aviseur de la manière suivante (figure 5-9) : le participant reçoit les

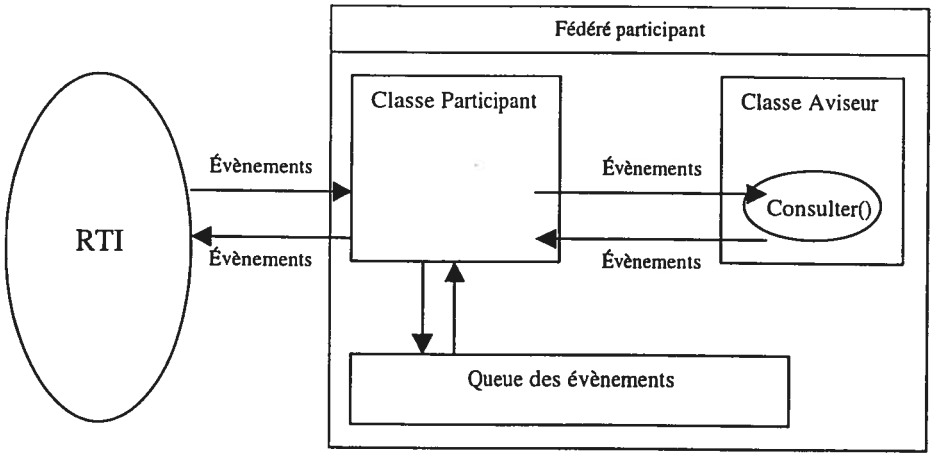

Figure 5-9 Rôle des classes *Participant* et *Aviseur* dans la simulation.

événements externes de la part des autres fédérés et les insère dans la queue, qui contient également les événements internes. Le participant avance son temps à travers l'appel *NextEventRequest()*, retire le prochain événements de la queue et le délivre à l'aviseur en paramètre de la méthode *consulter()*. Cette dernière traite l'événement et retourne, en

sortie, des événements externes (annonces, mises, inscriptions) à envoyer aux autres fédérés et des événements internes, reliés à la simulation du comportement interne du participant, à programmer dans la queue. Le participant insère les événements internes dans la queue et envoie les événements externes aux autres fédérés.

Le choix du lookahead pour le fédéré PA est tout à fait similaire au cas du fédéré PM. En effet, nous nous basons sur la valeur des imprécisions temporelles permises par le fédéré pour déterminer le lookahead. Cependant, puisque la valeur des imprécisions permises dépend du participant, nous laissons à l'utilisateur le soin de choisir la valeur du lookahead.

La fin de la simulation d'un participant a lieu lorsque toutes les places de marché quittent la fédération. En effet, nous considérons que le participant est présent en permanence dans le marché et ne le quitte que lorsque le dernier fédéré PM quitte le marché.

5-7 Utilisation du service DDM

Dans une simulation de marché, un participant peut envoyer trois types d'événements : les mises, les annonces et les demandes d'inscription. Ces événements sont toujours destinés à une place de marché particulière. Or, par défaut, la RTI diffuse toujours les événements à tous les fédérés souscrits à la classe de l'événement. Par défaut, la RTI diffuserait donc, à tous les fédérés PM, une mise, une annonce ou une demande d'inscription envoyée par un participant, alors que l'événement est destiné à un seul fédéré PM. De même, un fédéré PM envoie des événements qui seraient, par défaut, reçus par tous les fédérés PA, même ceux qui ne sont pas concernés par l'événement.

Les fédérés se basent sur le service Data Distribution Management (DDM), offert par HLA, pour éliminer l'échange d'événements inutiles. Les fédérés utilisent le service DDM en jouant sur les codes des fédérés. L'espace de routage est défini par une seule dimension *CodeFedere*, définie par l'intervalle [1,1000] de points discrets représentant l'ensemble des codes fédérés qui varient de 1 à 10000 (nous rappelons que chaque fédéré

se voit attribuer, à son lancement, un code qui l'identifie de manière unique dans la fédération).

La gestion de distribution de données (DDM) a été utilisée pour l'envoi d'interactions des fédérés PA aux fédérés PM et vice-versa. Pour illustrer comment nous utilisons le service DDM dans notre outil, prenons l'exemple de l'envoi des allocations. Un fédéré PM envoie une allocation à tous les fédérés PA inscrits à l'enchère. L'allocation est modélisée par la classe d'interactions *Allocation*. Chaque fédéré PA se souscrit à la classe *Allocation* avec une région de souscription formée d'un seul « extent » contenant un seul point de la dimension *CodeFedere,* représentant le code du fédéré PA en question. De même, un fédéré PM envoie ses allocations avec une région d'envoi formée par un ensemble de points discrets de la dimension *CodeFedere*, représentant les codes des fédérés PA inscrits dans la place de marché. Chaque point forme un « extent », et la région est l'union de tous les « extents ». La RTI ne livre alors une allocation à un fédéré PA que lorsque la région d'envoi du fédéré PM contient le code du fédéré PA, c'est-à-dire qu'il y a intersection entre la région d'envoi et la région de souscription. Prenons l'exemple d'une fédération de trois fédérés PA : PA1, PA2 et PA3, ayant les codes respectifs 1, 2 et 3, et un fédéré PM1 ayant le code 4. Supposons que les deux fédérés PA1 et PA2 (mais pas PA3) sont inscrits à une enchère organisée par le fédéré PM1. Le fédéré PM1 doit envoyer ses allocations avec une région d'envoi formée uniquement par les codes des deux fédérés PA1 et PA2, puisqu'ils sont inscrits à l'enchère. Donc la région d'envoi est R_envoi = {1,2}. Chaque fédéré PA doit se souscrire à la classe d'interactions *Allocation* avec une région de souscription contenant uniquement le code du fédéré PA en question. Donc les régions de souscription des fédérés PA1, PA2 et PA3 sont respectivement:

R_souscriptionA1 = {1},
R_souscriptionA2 = {2},
R_souscriptionA3 = {3}.

Et puisque uniquement les régions R_souscriptionA1 et R_souscriptionA2 ont une intersection avec la région d'envoi du fédéré PM1, alors uniquement ces fédérés reçoivent les allocations envoyées par le fédéré PM1.

De façon plus générale, un fédéré se souscrit toujours à une classe d'interactions avec une région de souscription contenant uniquement le code du fédéré. Un fédéré envoie toujours ses interactions avec une région d'envoi contenant l'ensemble des codes des fédérés auxquels sont destinées les interactions. La RTI ne livre alors l'interaction au fédéré récepteur que lorsque le code du fédéré expéditeur appartient à la région du fédéré récepteur, c'est-à-dire lorsqu'il y a intersection entre la région d'envoi et la région de souscription.

5-8 Le fédéré moniteur et la génération des statistiques

La génération des statistiques dépend de chaque cas de marché. En effet, les statistiques désirées dans le cas d'un marché financier sont naturellement différentes de celles désirées dans le cas d'un marché de transport. Pour cette raison, notre objectif n'est pas de définir des variables statistiques particulières et de les calculer, mais c'est de donner à l'utilisateur le moyen de définir ses propres variables et de les calculer. La méthode utilisée, pour collecter les statistiques de la fédération, consiste à faire une collecte de tous les événements échangés entre tous les fédérés et de les enregistrer dans un fichier, ce qui servira de données brutes pour le calcul de statistiques après la fin de la simulation. La collecte est assurée par le fédéré *Moniteur* qui se souscrit à tous les types de données du FOM, c'est-à-dire à tous les événements échangés au niveau de la fédération, et reçoit ainsi une « copie » de tous les événements échangés et les enregistre dans un fichier dans un format prédéfini. Notons que le fédéré *Moniteur* n'utilise pas le service DDM, c'est-à-dire qu'il ne définit pas une région de souscription, et donc il reçoit automatiquement tous les événements auxquels il est souscrit.

Le fichier généré contient une entrée par ligne, contenant une description d'un événement capturé par le moniteur. Chaque ligne contient le type de l'événement (mise, annonce, signal, etc..), le code et l'étiquette de temps logique de l'événement, le code du fédéré générateur de l'événement, ainsi que d'autres informations spécifiques au type de l'événement. Une description détaillée de la structure de ce fichier est donnée en annexe A.

Avec cette structure, le fichier contient un historique de l'exécution de la fédération. Le contenu du fichier donne le moyen de calculer une multitude de statistiques possibles. A partir de ce fichier, il est possible, par exemple, de calculer le nombre moyen des mises par ronde et la durée moyenne, la fréquence moyenne des mises et la durée moyenne d'une ronde. Le calcul de telles variables nécessite le développement d'un programme capable de parcourir le fichier et de l'analyser. Nous laissons à l'utilisateur le soin de développer ce programme, puisque les statistiques désirées dépendent du cas de marché simulé.

Chapitre 6

Application : une simulation de marché financier combinatoire

Dans les marchés financiers, les traders ont souvent besoin de rééquilibrer leur portefeuille d'actifs en fin de journée, afin de respecter au mieux les directives de leur entreprise ou de leurs clients, concernant la composition sectorielle de leur portefeuille, ou afin de suivre un indice de performance donné. Pour réaliser ce rééquilibrage dans les marchés actuels qui transigent les actifs sur une base individuelle, c'est-à-dire actif par actif, un trader doit négocier en parallèle de nombreux ordres parfois sur des places de marchés différentes.

Cette pratique engendre des frais de transactions importants, une analyse stratégique complexe, et surtout un risque significatif de se retrouver avec un portefeuille non équilibré. En effet, le risque d'exposition prend ici tout son sens: lorsqu'un trader transige plusieurs actifs séparément, il prend le risque qu'une des transactions soit partiellement exécutée ou pas du tout. Dans ce contexte, des marchés financiers combinatoires dédiés au rééquilibrage de porte-feuille offrent une alternative intéressante.

Dans de tels marchés, les traders soumettent, à l'encanteur, des ordres combinés afin de vendre et acheter simultanément différents actifs. Ils indiquent pour chaque paquet le prix qu'ils sont prêts à payer ou recevoir si celui-ci est exécuté. Après avoir reçu tous les ordres, l'encanteur détermine pour chaque paquet la proportion exécutée et les prix (reçus ou payés) associés.

Dans le cadre de notre projet, l'intérêt premier de cette application est de montrer que notre plate-forme générique de simulation fonctionne correctement et qu'elle répond aux objectifs que nous nous étions fixé. En d'autres termes, nous cherchons à montrer qu'avec notre outil, nous pouvons facilement implanter la simulation d'un marché

financier et qu'à partir de cette simulation, nous pouvons bien aboutir aux statistiques sur les aspects des marchés que nous désirons évaluer. De plus, nous voulons prouver qu'en utilisant l'outil développé, l'effort de développement de la simulation du marché financier est concentré sur les aspects reliés aux particularités du marché en question et que l'intervention de l'utilisateur dans l'implantation des aspects reliés aux techniques de simulation est minimale.

6-1 Description du marché

Le marché proposé est très simple. Il se compose d'une place de marché et d'un ensemble L de participants (traders). Le mécanisme d'enchère comporte une phase discrète avec positions d'achats et ventes simultanées. Il n'y a donc pas de périodes d'annonces, mais l'enchère commence directement par une période de mises. Le mécanisme d'allocation est mono-phase mono-ronde. Par souci de simplification, la ronde est déclarée lorsque chaque participant a réalisé une et une seule mise. Le marché sera simulé sur trois journées successives. Chaque journée commence à 08h00 et finit à 9h00 en temps physique. La période des inscriptions dure 15 minutes après l'ouverture du marché, et l'enchère commence donc chaque journée à 8h15.

6-2 Les actifs transigés

Notons G l'ensemble des actifs transigés dans le marché. Nous avons choisi de travailler sur 100 actions cotées à la bourse de Paris, pour lesquelles nous avons relevé les cours d'une journée particulière. Nous avons classé ces 100 titres en secteurs selon le système de classification internationale FTSE utilisé par les bourses de Paris, Amsterdam et Bruxelles.

6-3 Le vocabulaire des mises

Une mise se décompose en 4 niveaux (figure 6-1) :

- Au niveau atomique, on indique un titre r et un entier q_{ljr} qui est la quantité maximale de l'actif r que le trader l désire transiger dans l'ordre j ; $q_{ljr} > 0$ correspond à un achat, $q_{ljr} < 0$ à une vente.

- Au niveau interne qui correspond à la formation des paquets, l'opérateur accepté est l'opérateur EQUAL, qui signifie que tous les titres du paquet doivent être transigés dans une proportion égale. Il faut également spécifier un réel p_{lj} qui est le prix que le trader l veut payer (prix positif) ou recevoir (prix négatif) si la mise sur son paquet j est totalement exécutée, ainsi qu'une borne inférieure $b_{lj} \in [0,1]$ qui est la proportion en dessous de laquelle le trader l préférerait que sa mise sur le paquet j ne soit pas exécutée du tout.

- Aux niveaux externes 1 (respectivement 2), seul l'opérateur logique OR (respectivement XOR) est autorisé.

Notations complémentaires

Notons J_l l'ensemble des paquets formés par le trader l, $l \in L$.

Une relation XOR que l'on notera χ, formulée par le trader l, est un sous-ensemble de paquets de J_l, qui indique qu'au plus un paquet dans χ doit être exécuté. Soit χ_l l'ensemble des relations XOR définies par le trader l.

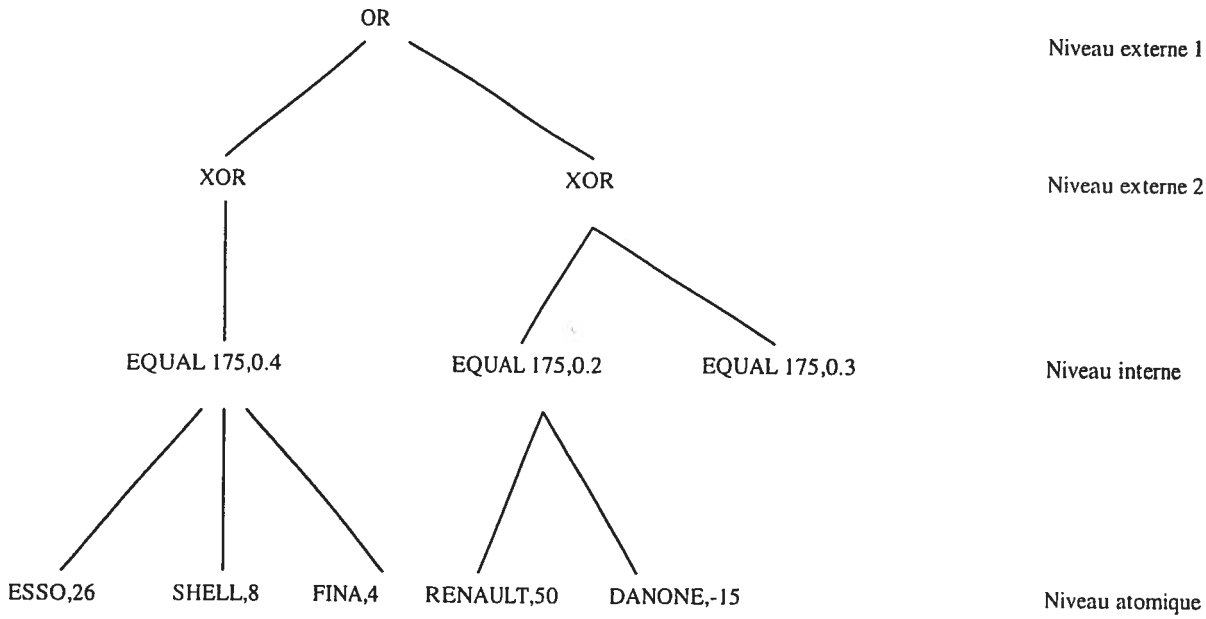

Figure 6-1 Exemple de mise combinée.

Le fichier XML décrivant le vocabulaire que nous venons de décrire pour le marché financier est représenté dans la figure 6-2.

```
<ConfigVocabulaire>
        <NomVocabulaire> Marche Financier Combinatoire </NomVocabulaire>
        <PermissionPrixAtomiqueSignificatif> false </PermissionPrixAtomiqueSignificatif>
        <OuterLayer>
                <PermissionPrixSignificatif> false </PermissionPrixSignificatif>
                <NombreOperateurs> 1 </NombreOperateurs>
                <OperateurOut> OR </OperateurOut>
        </OuterLayer>
        <OuterLayer>
                <PermissionPrixSignificatif> false </PermissionPrixSignificatif>
                <NombreOperateurs> 100 </NombreOperateurs>
                <OperateurOut> XOR </OperateurOut>
        </OuterLayer>
        <InnerLevel>
                <PermissionPrixSignificatif> true </PermissionPrixSignificatif>
                <NombreOperateurs> 10000 </NombreOperateurs>
                <OperateurIn> EQUAL </OperateurIn>
        </InnerLevel>
</ConfigVocabulaire>
```

Figure 6-2 Fichier XML décrivant le vocabulaire des mises.

6-4 Le mécanisme d'allocation

Les variables de décision du problème d'allocation sont :

- x_{lj}, la proportion transigée de la mise portant sur le paquet j, soumise par le trader l.

- y_{lj}, une variable binaire caractérisant l'exécution de la mise portant sur le paquet j, soumise par le trader l; i.e. $y_{lj} = 1$ si elle est exécutée, $y_{lj} = 0$ sinon.

Avec les notations précédentes, le problème d'allocation peut être formulé comme le problème linéaire d'optimisation suivant :

$$\max \sum_{l \in L} \sum_{j \in J_l} p_{lj} x_{lj} \qquad (1)$$

$$s.t. \sum_{l \in L} \sum_{j \in J_l} q_{ljr} x_{lj} = 0, \quad r \in G \qquad (2)$$

$$y_{lj} b_{lj} \le x_{lj} \le y_{lj}, \quad l \in L, j \in J_l \qquad (3)$$

$$\sum_{j \in \chi} y_{lj} \le 1, \quad \chi \in \chi_l, l \in L \qquad (4)$$

La fonction objectif (1) correspond à la maximisation du surplus économique du marché. Les contraintes (2) expriment l'équilibre du marché: pour chaque actif r, la quantité achetée est égale à la quantité vendue. Les contraintes (3) assurent que, pour chaque paquet, la proportion accordée sera supérieure à la borne inférieure correspondante. Enfin, les contraintes (4) résultent de l'application des opérateurs XOR.

Le mécanisme d'allocation et de calcul de prix a été implanté par la classe *MFCalculAllocationPrix*, qui hérite de la classe *Mecanisme* offerte par notre librairie, et implante la méthode abstraite *allouer()*. Lorsque la fin de ronde est atteinte, l'encanteur appelle la méthode *MFCalculAllocationPrix .allouer()*. La méthode lance une requête auprès de la base de données pour retrouver toutes les mises émises par les participants,

ensuite calcule les allocations et les prix et retourne à l'encanteur une liste d'objets de la classe *Allocation*. Une allocation porte sur une seule mise gagnante et contient une liste d'allocations atomiques correspondant chacune à une mise atomique gagnante. La figure 6-3 montre l'exemple d'une allocation pour la mise de la figure 6-1. Dans l'exemple, uniquement la mise partielle de la branche gauche de la mise représentée dans la figure 6-1 a gagné, et a été exécutée avec une proportion de 50%. Chaque allocation atomique contient le nom du titre gagné, la quantité allouée et le prix à payer.

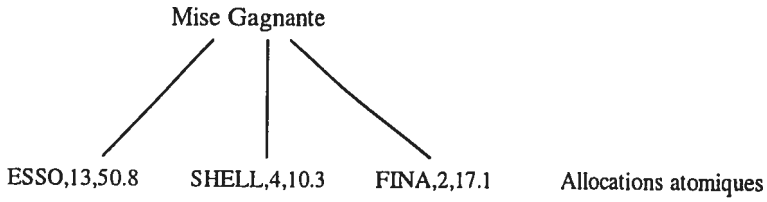

Figure 6-3 Un exemple d'allocation.

6-5 Temps au niveau du fédéré PM

Dans l'implantation du marché financier, nous avons choisi de faire correspondre une unité de temps logique à une minute de temps physique. Le temps initial t=0 correspond à 8h00 du temps physique, heure du début du marché.

6-6 Les règles du marché

Le marché utilise un nombre de règles dont nous présentons les plus importantes. La première règle que nous identifions est la règle d'ouverture du marché. La règle est implantée à travers la classe *ROuvertureMarche*, représentée dans la figure 6-4. La première étape réalisée par le fédéré PM, après l'initialisation de la simulation, est l'appel de la méthode *InitialiserRegle()* de la règle d'ouverture du marché. La méthode programme l'événement *VerifierRegle* à t=0 avec, comme code règle, le code de la règle

en question. Cet événement est le premier événement programmé au niveau du fédéré PM. Ensuite, le fédéré lance la boucle d'événement. Puisque le premier (et le seul) événement dans la queue est l'événement *VerifierRegle* à t=0, le fédéré le retire de la queue et appelle la routine d'événement correspondante. Le routine appelle la méthode *RouvertureMarche ::verifier()*. Cette méthode retourne alors la valeur TRUE, signifiant que l'ouverture du marché, et donc la période des inscriptions, doit avoir lieu. Ensuite la routine envoie le signal d'ouverture du marché aux participants et appelle les méthodes :

```
RouvertureMarche ::InitialiserRegles()
        Programmer un événement VerifierRegle à t=15;

RouvertureMarche ::Verifier()
        if (t=0) then
                return TRUE;
        else
                return FALSE;
```

Figure 6-4 Implantation de la règle d'ouverture du marché.

- *RArretMarche ::InitialiserRegle()* où *RArretMarche* est la classe qui implante la règle de fin de l'occurrence du marché. En effet, dès que le marché ouvre, l'encanteur programme son arrêt. La méthode programme un événement *VerifierRegle* à t=60 (qui correspond à 9h00, heure de fermeture du marché). Lorsque le fédéré atteint le temps t=60, plus tard, le traitement de cet événement aboutit à la fin du marché. En effet, la routine de l'événement appelle la méthode *RArretMarche ::verifier()* qui retourne la valeur TRUE lorsque le temps logique est égal à 60.

- *RDebutEnchère ::InitialiserRegle()* où *RDebutEnchere* est la classe qui implante la règle de début de l'enchère. Cette méthode programme un événement *VerifierRegle* à t=15 (qui correspond à 8h15). Lorsque le fédéré atteint le temps t=15, plus tard, le traitement de cet événement aboutit à la fin de la période des inscriptions et au début de l'enchère. En effet, à l'ouverture du marché, l'encanteur programme le début de l'enchère. La période d'inscriptions a lieu entre l'ouverture du marché et le début de l'enchère.

Lorsque l'événement *VerifierRegle* programmé par la méthode *RDebutEnchère ::InitialiserRegle()* est traité à t=15, son traitement aboutit au lancement de l'enchère, et donc au lancement de la première phase. A cet instant, le fédéré appelle les méthodes :

- *RArretEnchere ::InitialiserRegle()*. En effet, dès que commence l'enchère, l'encanteur appelle cette méthode pour programmer la fin de l'enchère. Puisque l'arrêt de l'enchère a lieu au bout de la première phase (car l'enchère est mono-phase), la règle d'arrêt de l'enchère n'est pas exprimée dans la dimension temps, et donc cette méthode ne programme pas d'événement *VerifierRegle*. Le code de cette méthode ne fait donc aucune action. Par contre, la méthode *RarretEnchere ::verifier()* exécute le test suivant :

```
if (nombre de phases appliquées == 1) then
        return TRUE;
else
        return FALSE;
```

  L'encanteur appelle cette méthode au bout de chaque phase. Au bout de la première phase, la condition d'arrêt est vérifiée, et l'enchère est arrêtée.

- *RArretPhase ::InitialiserRegle()*. Puisque l'arrêt de la phase a lieu lorsque chaque participant a soumis une mise, alors cette méthode n'est pas exprimée dans la dimension temps, et la méthode ne programme pas d'événement *VerifierRegle*. Par contre, la méthode *RarretPhase ::verifier()* exécute le test suivant :

```
Retrouver les mises à partir de la base de données.
if (chaque participant a fait une mise) then
        return TRUE;
else
        return FALSE;
```

  Cette méthode est appelée au traitement de chaque mise. Lorsque chaque participant a soumis une mise, la méthode retourne la valeur TRUE, et la fin de phase est atteinte.

6-7 Aviseur et génération des mises

Le modèle d'aviseur qui nous a été proposé pour tester l'application est très simple et déterministe, et ne présente donc aucun aspect stochastique. Les participants débutent la simulation avec une somme d'argent et un portefeuille initial. Chaque participant est associé à son propre aviseur, qui lui dicte la stratégie à adopter. L'objectif de l'aviseur est de faire converger le portefeuille du participant vers un portefeuille cible. Le comportement du participant suit le scénario imposé par la place de marché: il reçoit des événements et y répond. Par exemple, il attend l'ouverture d'une place de marché puis s'y inscrit. Si son inscription est acceptée, il attend le début de l'enchère et génère une mise. Ensuite, il attend la réponse à sa mise, puis l'allocation selon laquelle il doit mettre à jour son portefeuille. Cette séquence se répète à chaque ouverture de marché.

Pour pouvoir simuler un panel intéressant de comportements, il a fallu définir le profil d'un aviseur : un ensemble de paramètres que l'utilisateur peut faire varier pour simuler les différents comportements des traders. Chaque participant reçoit donc, en entrée, un ensemble de valeurs pour ces paramètres qui déterminent le profil de son aviseur. Les dimensions du profil sont les suivantes :

1. Portefeuille initial

    a. *Liquidité initiale* : la somme d'argent du départ.

    b. *Investissement initial* : le pourcentage de l'argent initial du participant qui est utilisé pour créer le portefeuille.

2. Portefeuille cible

    a. *L'aversion aux actifs risqués* : un entier indiquant le niveau de volatilité des actions que le participant souhaite acquérir.

b. *La diversification des actifs* : la liste des secteurs dans lesquels le participant souhaite investir.

3. Structure de la mise

a. *La fréquence d'utilisation du XOR* : un nombre compris entre 0 et 1 désignant la proportion des mises pour lesquelles le participant va chercher à définir des paquets équivalents et ainsi utiliser le XOR.

b. *L'amplitude du XOR* : deux entiers indiquant les nombres minimum et maximum de paquets équivalents que va chercher à soumettre le participant lorsqu'il utilise un XOR.

c. *La fréquence d'achat vente combinés dans un même paquet* : un nombre compris entre 0 et 1 désignant la proportion de paquets EQUAL dans lesquels des achats et des ventes seront combinés, si possible, c'est-à-dire s'il reste des titres à vendre dans le portefeuille du participant.

d. *La taille désirée d'un paquet* : deux entiers désignant les nombres minimum et maximum d'items regroupés dans un paquet.

4. Prix et quantités demandés

a. *L'évaluation personnelle d'un titre (Δ)* : les participants, selon leur contexte, ne doivent pas avoir la même vision d'un titre donné. La valeur d'un actif sera donc une perturbation aléatoire de sa cotation. L'amplitude de cette perturbation est un nombre Δ compris entre 0 et 1.

b. *Le pourcentage des liquidités investi dans un achat (α)* : lorsqu'un participant effectue une mise contenant un achat, il faut déterminer l'argent investi et calculer les quantités en conséquence.

c. *Le facteur de spéculation (β)* : un nombre compris entre 0 et 1 indiquant si le participant est prêt à spéculer pour obtenir un meilleur prix que sa propre évaluation du paquet, ce qui diminue ses chances d'obtenir le paquet. Si β est faible, le prix soumis lors d'une mise par le participant sera proche de son prix de réserve. Au contraire, une valeur forte définira un participant qui essaie d'obtenir mieux que son prix de réserve.

d. *La fréquence d'utilisation de la borne inférieure* : un nombre compris entre 0 et 1 désignant la proportion de mises pour lesquelles le participant va utiliser une borne inférieure dans ses mises. Comme décrit dans [Abrache02a], la borne inférieure est la quantité minimale, exprimée en proportion, exigée par le participant lors de l'allocation de sa mise. Par exemple, avec une valeur de 0.25, il interdit que ses mises soient partiellement exécutées en dessous d'un quart, i.e. que des quantités inférieures au quart de celles demandées lui soient attribuées.

e. *L'amplitude de la borne inférieure* : deux nombres compris entre 0 et 1 indiquant les valeurs minimales et maximales de la borne inférieure des mises.

Nous avons conçu un exécutable permettant de générer des *populations* de profils, sous forme de fichiers, à partir de la saisie au clavier d'un utilisateur. Le principe est de demander, pour chaque population, un intervalle de valeurs pour chacune des dimensions du profil, puis de choisir parmi celles-ci des valeurs aléatoires pour créer chaque élément de la population. Cet exécutable crée aussi une base de données communes à tous les participants, qui contient la liste des actifs, classés par secteur, avec leurs niveaux de risque et les cotations des jours d'enchères.

Ensuite, pour chaque profil généré, on crée un aviseur. En fonction des actifs et des deux premières dimensions du profil, on peut déterminer les titres qu'il veut acheter (qui correspondent à sa diversification et à son niveau de risque). Le portefeuille initial est créé à partir des liquidités initialement investies, en achetant uniquement parmi les actifs

que le participant ne veut pas, de manière aléatoire. En effet, ce qui nous intéresse en premier lieu est sa façon de converger vers le portefeuille cible.

6 - 7 - 1 Génération des mises

6-7-1-1 Présentation de l'algorithme

L'aviseur possède une liste de titres à vendre et une liste de titres à acheter, classés par secteur. Le trader effectue une mise par secteur voulu, et les relie par un OR. Pour chaque secteur, l'aviseur décide de générer un ou plusieurs paquets équivalents, reliés par un XOR, selon les critères 3.a et 3.b du profil.

Pour chaque paquet équivalent, l'aviseur choisit entre trois stratégies : achat pur (paquet ne se composant que d'achats), vente pure ou achat et vente combinés. Le choix entre ces trois options se fait de la manière suivante. Si le participant n'a plus rien à vendre, il fait un ordre d'achat pur. Sinon, le choix est fait de manière aléatoire, avec une probabilité d'utiliser des achats et des ventes combinés dans un même EQUAL égale à la dimension 3.c.

Le nombre de titres dans un EQUAL, c'est-à-dire la taille T du paquet, est choisie aléatoirement entre les valeurs du critère 3.d. Il reste alors à déterminer quels titres choisir. Pour les achats et ventes dans un même paquet, le nombre de ventes *NBVente* est choisi de manière aléatoire entre 0 et $2/3 * T$. Ensuite, pour les achats comme pour les ventes, le choix se fait de manière aléatoire parmi les titres voulus, jusqu'à ce que la taille du paquet soit atteinte ou qu'il n'y ait plus de titres à transiger.

Remarque : nous verrons que, dans le cas d'une vente, le participant cherche à transiger la quantité totale possédée de l'actif à vendre. Il est donc possible, si l'encanteur accorde la totalité de cette transaction, que le participant ne possède plus ce titre. C'est pourquoi, dans une même mise, il faut garantir qu'un titre à vendre n'apparaisse pas dans plusieurs paquets qui peuvent être exécutés en même temps, pour empêcher que le trader vende plus de titres qu'il ne le peut. Pour résoudre ce problème, le choix des titres à vendre est

fait aléatoirement parmi les titres non encore vendus dans une autre branche du OR. Remarquons enfin qu'un titre à vendre peut cependant apparaître dans plusieurs paquets composant un même XOR.

6-7-1-2 Prix et quantités dans un paquet

L'évaluation des actifs est certainement le point le plus complexe de l'aviseur d'un participant à un marché financier. La méthode que nous proposons ici est forcément éloignée de la manière de raisonner des traders, qui se servent de méthodes quantitatives complexes et surtout de leur expérience personnelle. Elle a l'avantage d'être simple et d'aboutir à des évaluations cohérentes. Nous générons les prix à partir des cours de référence des actifs sur le marché public, connues par tous les participants, et de simples perturbations aléatoires.

Evaluation personnelle d'un titre

L'évaluation personnelle d'un titre i est une valeur positive $p_i = Cotation_i \times \delta_i$, où $Cotation_i$ est sa cotation dans le marché public, au jour de l'enchère. δ_i est une perturbation aléatoire comprise entre $1-\Delta/2$ et $1+\Delta/2$ (dimension 4.a du profil d'aviseur).

Prix de réserve et quantités dans un paquet de ventes

Pour un paquet Γ se composant uniquement de ventes, le participant attribue la totalité de la quantité qu'il possède à chaque titre. L'évaluation du paquet, que nous appelons prix de réserve, est alors :

$$P_{reserve} = \sum_{i \in \Gamma} q_i \times p_i \qquad (5)$$

où q_i est la quantité transigée. Notons que puisqu'il s'agit d'une vente, q_i est négatif et donc le prix de réserve aussi.

Prix de réserve et quantités dans un paquet comprenant un achat

Pour un paquet comprenant l'achat d'au moins un titre, le participant commence par calculer l'argent qu'il va investir dans ce paquet. Cette somme sera le prix de réserve, à partir duquel on déterminera les quantités correspondant à son évaluation personnelle du paquet. Il y a au maximum un paquet par secteur. Soit N le nombre de secteurs qui intéressent le trader. On a choisi :

$$P_{reserve} = Liquidite \times \frac{\alpha}{N} \qquad (6)$$

où la *Liquidite* est l'argent liquide en la possession du trader et α est la dimension 4.b du profil. A cet investissement venant des liquidités du trader vient s'ajouter l'argent dégagé par la vente de titres dans ce même paquet EQUAL. Notons Ω l'ensemble des titres vendus du paquet (pouvant être vide). La somme *MontantAchat* qui va servir à l'achat est donc :

$$MontantAchat = P_{reserve} + \sum_{i \in \Omega} |q_i| \times p_i \qquad (7)$$

où p_i et $|q_i|$ sont le prix et la quantité possédée du titre à vendre i. Ainsi, en répartissant de manière égale cet investissement sur chacun des titres à acheter, la quantité positive q_i demandée pour un titre i à acheter dans ce paquet est :

$$q_i = \frac{Montant Achat}{nbAchat \times \hat{p}_i} \qquad (8)$$

où *nbAchat* est le nombre d'achats dans le paquet. $\hat{p}_i$ est l'évaluation de l'actif i compte tenu de l'effet de complémentarité. En effet, un participant combinant deux actifs dans un même paquet doit être prêt à donner plus que la somme des deux prix. Le $\hat{p}_i$ est donc égal à p_i plus une perturbation aléatoire positive.

Prix soumis à l'encanteur

Il est naturel de penser que le prix de réserve est une information confidentielle, et que le participant va essayer de spéculer pour obtenir mieux que son évaluation personnelle du paquet : il cherche à obtenir un gain sur son opération et non pas seulement un échange équitable d'une somme d'argent contre des valeurs financières. Ainsi, le prix soumis vaut :

$$p_{soumis} = p_{res} - |p_{res}| \times \beta \qquad (9)$$

β est le paramètre 4.c du profil du participant : le facteur de spéculation, compris entre 0 et 1. Ainsi, la perturbation diminue le prix proposé s'il s'agit d'un achat (prix réserve positif), et augmente en valeur absolue le prix négatif demandé s'il s'agit d'une vente (prix de réserve négatif).

6-8 Développement de la fédération

Les profiles utilisés ont été générés en utilisant les valeurs des dimensions données dans le tableau 6-1.

Les fichiers sources et les exécutables de la librairie et des fédérés du marché financier se trouvent dans l'arborescence commençant au répertoire */home/khemila/projet/source* du réseau CRT. Le contenu de chaque sous-répertoire est donné dans le tableau 6-2.

Dimension	Valeur
Montant d'argent initial	10000
Pourcentage de ce montant investi dans le portefeuille initial	0.7
Pourcentage des liquidités restantes investies à chaque nouvelle enchère	0.1
Valeurs minimales et maximales pour le facteur de spéculation	0.01 0.09
Valeurs minimales et maximales pour la perturbation des cotations	0.1 0.4
Nombre minimum et maximum de titres différents dans le portefeuille initial	5 8
Nombre minimum et maximum de secteurs désirés par participant	5 10
Fréquence minimale et maximale d'achat vente combinés	0.1 0.4
Fréquence minimale et maximale des XOR	0.5 1
Amplitude minimale et maximale des XOR	3 7
Amplitude minimale et maximale des EQUAL	1 4
Fréquence minimale et maximale des bornes inférieures	0 0.05
Valeurs minimales et maximales des bornes inférieures	0 1

Tableau 6-1 Valeurs des dimensions de profiles utilisées pour le test.

sous-répertoire	Contenu
source/include/	Fichiers include (.h) de la librairie.
source/src/	Fichiers sources (.cpp) et objet (.o) de la librairie.
source/data/	Fichier FED (.fed) de la fédération ainsi que le fichier market.omd contenant les modèles de données (FOM et SOM) de la fédération.
source/MF/	Arborescence contenant les fichiers sources et exécutables de la fédération du marché financier.
source/MF/CPP/	Fichiers sources (.cpp) et objet (.o) des fédérés PA et PM du marché financier.
source/MF/INCLUDE/	Fichiers include (.h) des fédérés PA et PM du marché financier.
source/MF/BIN/	Fichiers exécutables des fédérés PA et PM et du moniteur. Programme de génération des profiles (*GenererProfils*). Fichiers contenant les profiles des participants. Fichier contenant la description du vocabulaire des mises du marché financier. Script SQL de création de la base de données du fédéré PM.

Tableau 6-2 Arborescence contenant la fédération du marché financier.

Le tableau 6-3 donne une idée sur le nombre de lignes de code des différents modules développés.

Module	Nombre de lignes
Librairie et moniteur	14233
Fédéré PA	1390
Fédéré PM	1207

Tableau 6-3 Nombre de lignes du code développé.

6-9 Résultats de la fédération

La fédération a été testée avec une seule place de marché et une population de 10 fédérés PA, ayant chacun un profile différent généré par le programme de génération de profiles *GenererProfiles*, et s'exécutant chacun sur une machine différente. Le moniteur, à son tour, s'exécute sur une machine différente. Le test a été fait avec une seule place de marché puisque les participants proposés, utilisés dans la fédération, ne peuvent participer qu'à une seule place de marché à la fois.

Le fichier généré par le moniteur est trop volumineux. Nous n'en présentons donc que les extraits représentés dans la figure 6-5. La description de la structure de chaque ligne est donnée dans l'annexe A.

Notons, tout d'abord, que le mécanisme d'allocation aboutit, dans un grand nombre de cas, à un résultat où aucun titre n'est alloué. En effet, la construction du portefeuille initial, et par conséquent la stratégie des mises, des participants dépend de facteurs aléatoires. Ce qui peut aboutir à des mises qui ne satisfont pas le modèle d'équilibre du mécanisme d'allocation, décrit plus haut dans ce chapitre. Le problème a lieu, en particulier, lorsque les titres déclarés pour vente et ceux déclarés pour achat ne correspondent pas. Il faut donc exécuter la fédération plusieurs fois avec des participants

de différents profiles jusqu'à aboutir à une configuration où l'enchère réalise des allocations.

Le fichier contient une ligne par événement. Notons que la figure ne représente qu'un extrait du fichier. Les lignes vides, représentées dans la figure, correspondent à une partie du fichier qui a été omise. Le fichier commence par des lignes contenant l'information sur les fédérés qui ont rejoint la fédération. La première ligne du fichier signifie que le fédéré dont le code est PA1 a rejoint la fédération à partir de la machine « oka ». La troisième ligne correspond à un signal d'ouverture du marché. Le code PM11/1 représente le code de l'événement, unique au niveau de la fédération. La valeur 1 représente le temps logique de l'événement. Normalement, l'ouverture du marché doit avoir lieu au temps t=0, mais puisque le fédéré utilise un lookahead égal à 1, l'ouverture a lieu au temps t=1. le nombre 8 entre parenthèses est le code du signal, qui correspond à un signal d'ouverture du marché. Les deux lignes commençant par le mot « Inscription » correspondent aux inscriptions des deux fédérés PA1 et PA2. Elles sont suivies par les deux accords retournés par le fédéré PM. Ensuite, au temps t=16 (donc 15+lookahead), un signal de début d'enchère est envoyé par le fédéré PM à tous les participants de l'enchère. Ensuite, vient une ligne qui représente une mise envoyée par le fédéré PA1. La mise est suivie d'un accord envoyé par la place de marché. Évidemment, le fichier original contient 10 mises, une mise par fédéré PA. Ensuite, vient une succession de lignes représentant les allocations envoyées par le fédéré PM aux fédérés PA. Les allocations sont suivies de trois signaux : un signal de fin de l'enchère, un signal de fermeture du marché qui correspond à la fin de la première occurrence et un signal d'ouverture du marché qui correspond à l'ouverture de la deuxième occurrence du marché. Notons que le signal d'ouverture du marché est au temps t=1441 (24*60+lokahead) qui correspond à 8h00 de la deuxième journée plus le lookahead. Suite à l'ouverture de la nouvelle occurrence, le même cycle que pour la première occurrence a lieu, dont nous ne représentons que les deux premières lignes qui correspondent aux inscriptions des deux fédérés PA1 et PA2 à la nouvelle occurrence du marché.

6-10 Evaluation

Nous n'avons pas développé le programme qui génère les statistiques à partir du fichier résultat du moniteur. En effet, notre tâche se limite à donner le moyen de calculer les statistiques, c'est-à-dire le fichier généré par le moniteur. En l'absence d'objectifs précis au niveau de l'évaluation de ce marché, nous n'avons pas développé un modèle d'analyse du fichier résultat. Nous laissons ce soin aux personnes directement concernées par le calcul des statistiques et leur interprétation.

Le premier objectif de notre outil, celui de développer un outil générique, a été atteint dans certaines limites. L'outil permet, en effet, de développer la simulation d'un marché tant que ce dernier est conforme à la description que nous avons décrite dans le chapitre IV et au prototype que nous avons proposé dans le chapitre V. L'outil présente une limitation importante en imposant un prototype générique à tous les marchés. Notons, cependant, que ce choix nous a permis de développer un scénario générique pour les simulations de marchés et donc de gagner beaucoup en généricité. De même, la limitation de l'outil aux vocabulaires de mises conformes à l'architecture décrite dans [Abrache02a] nous a fait gagner en généricité, puisque cette limitation nous a permis de définir une structure unique pour les mises et les annonces et d'implanter, une fois pour toutes, les procédures de vérifications des mises et des annonces émises par les participants.

Le second objectif, celui de permettre l'évaluation des marchés à travers les simulations distribuées, a également été atteint dans certaines limites. En effet, le fichier généré par le moniteur contient une trace des données dynamique de la fédération et offre donc le moyen de calculer une panoplie de statistiques servant à l'évaluation du marché simulé. Toutefois, d'autres statistiques spécifiques désirées peuvent dépendre de données dynamiques qui ne sont pas fournies dans le fichier généré par le moniteur et ne peuvent donc pas être calculées à travers notre outil. L'outil, ainsi conçu, gagne en généricité et perd en fonctionnalité.

Enfin, l'objectif d'encapsuler, par l'outil, les aspects reliés à l'implantation des techniques de simulation a été largement atteint. En effet, la gestion des événements, des interactions entre les fédérés, des compteurs statistiques et, en grande partie, du temps simulé est complètement prise en charge par l'outil. Deux tâches reliées à l'implantation de la gestion du temps, à savoir le choix du rapport de projection entre le temps physique et le temps logique et le choix de la valeur du lookahead des fédérés, ne sont pas implantées par l'outil et nous avons donc laissé à l'utilisateur le soin de les faire. La raison en est que ces deux aspects dépendent largement de la nature du système simulé et doivent donc être réétudiés pour chaque nouveau cas de marché.

```
FEDERE:PA1:5:oka
FEDERE:PA2:7:porto10

Signal:PM11/1:1:PM11:<>:(8)

Inscription:PA1/1:3:PA1:<PM11>
Inscription:PA2/1:5:PA2:<PM11>

Accord:PM11/2:4:<PA1>:PA1/1
Accord:PM11/3:6:<PA2>:PA2/1

Signal:PM11/12:16:PM11:<PA1,PA2,PA3,PA4,PA5,PA6,PA7,PA8,PA9,PA10>:(1)

Mise:PA1/2:17:PA1:<PM1>:[OR,0,0,[XOR,0,0,{EQUAL,12729.9,1,(205.5,0,-278,0,MONOPRIX),(39.36,0,751,0,CREDIT-
LYONNAIS),(40.87,0,723,0,BNP-PARIBAS)},{EQUAL,12729.9,1,(56.7,0,-1007,0,AVENTIS),(40.87,0,723,0,BNP-
PARIBAS),(39.36,0,751,0,CREDIT-LYONNAIS)},{EQUAL,12729.9,1,(39.36,0,142,0.213472,CREDIT-
LYONNAIS),(40.87,0,137,0.213472,BNP-
PARIBAS)}],[XOR,0,0,{EQUAL,12729.9,1,(14.3,0,931,0,EADS)}],[XOR,0,0,{EQUAL,12729.9,1,(42.32,0,-
1350,0,LVMH),(23.65,0,-
2416,0,TF1),(40.48,0,3144,0,CARREFOUR)},{EQUAL,12729.9,1,(40.48,0,328,0.260845,CARREFOUR)},{EQUAL,127
0.48,0.328,0,CARREFOUR)},{EQUAL,-358619,1,(55.2,0,-1035,0.291219,SEITA),(23.65,0,-2416,0.291219,TF1),(19.72,0,-
2897,0.291219,EURONEXT),(40.05,0,-1426,0.291219,LAGARDERE),(135.7,0,-421,0.291219,AIR-LIQUIDE),(42.56,0,-
1342,0.291219,PHILLIP-MORRIS)},{EQUAL,12729.9,1,(55.2,0,-1035,0,SEITA),(19.72,0,-
2897,0,EURONEXT),(40.48,0,3144,0,CARREFOUR)}],[XOR,0,0,{EQUAL,12729.9,1,(0.17,0,-336134,0,CRYO-
INTERACTIVE),(78.9,0,-724,0,ADIDAS-SALOMON),(43.74,0,2910,0,PECHINEY)},{EQUAL,12729.9,1,(0.17,0,-
336134,0.240265,CRYO-INTERACTIVE),(43.74,0,1607,0.240265,PECHINEY)},{EQUAL,12729.9,1,(0.17,0,-
336134,0.260441,CRYO-INTERACTIVE),(19.95,0,-
2864,0.260441,KAUFMAN&BROAD),(43.74,0,2910,0.260441,PECHINEY)}],[XOR,0,0,{EQUAL,12729.9,1,(44.5,0,12
ATE-PALMOLIVE),(74,0,75,0,PROCTER&GAMBLE)},{EQUAL,12729.9,1,(44.5,0,125,0.225332,COLGATE-
PALMOLIVE),(74,0,75,0.225332,PROCTER&GAMBLE)},{EQUAL,12729.9,1,(74,0,75,0,PROCTER&GAMBLE),(44.5
COLGATE-PALMOLIVE)},{EQUAL,12729.9,1,(23.5,0,-2431,0,PHILIPS),(44.5,0,664,0,COLGATE-
PALMOLIVE),(74,0,399,0,PROCTER&GAMBLE)}],[XOR,0,0,{EQUAL,12729.9,1,(4.78,0,-
11954,0.233886,ORANGE),(21.17,0,3321,0.233886,PROVIMI)},{EQUAL,12729.9,1,(4.78,0,-
11954,0,ORANGE),(21.17,0,3321,0,PROVIMI)},{EQUAL,12729.9,1,(4.78,0,-
11954,0,ORANGE),(21.17,0,3321,0,PROVIMI)},{EQUAL,12729.9,1,(4.78,0,-
11954,0,ORANGE),(21.17,0,3321,0,PROVIMI)}],[XOR,0,0,{EQUAL,12729.9,1,(49.58,0,112,0,ATOS),(22.92,0,244,0,A
}],[XOR,0,0,{EQUAL,12729.9,1,(40,0,93,0,RENAULT),(42.1,0,88,0,PEUGEOT),(12.51,0,298,0,FORD)},{EQUAL,127
1,0,133,0,PEUGEOT),(40,0,140,0,RENAULT)},{EQUAL,12729.9,1,(12.51,0,298,0,FORD),(42.1,0,88,0,PEUGEOT),(4(
ENAULT)}],[XOR,0,0,{EQUAL,12729.9,1,(0.44,0,8488,0.291418,BULL),(68.3,0,54,0.291418,IBM),(23.1,0,161,0.291418,ST-
MICROELECTRON)}],[XOR,0,0,{EQUAL,12729.9,1,(44.8,0,125,0,CATERPILLAR),(27.88,0,200,0,MECATHERM)},{
12729.9,1,(44.8,0,125,0,CATERPILLAR),(27.88,0,200,0,MECATHERM)},{EQUAL,12729.9,1,(44.8,0,125,0,CATERP
7.88,0,200,0,MECATHERM)},{EQUAL,12729.9,1,(27.88,0,200,0.236895,MECATHERM),(44.8,0,125,0.236895,CATE
)},{EQUAL,12729.9,1,(27.88,0,200,0,MECATHERM),(44.8,0,125,0,CATERPILLAR)},{EQUAL,12729.9,1,(44.8,0,125
CATERPILLAR),(27.88,0,200,0.117573,MECATHERM)}]]

Accord:PM11/14:18:<PA1>:PA1/2


Allocation:PM11/23:61:PM11: <PA1,PA2,PA3,PA4,PA5,PA6,PA7,PA8,PA9,PA10>:{(10.000000;5.000000;LAGARDERE)}


Signal:PM11/36:61:PM11: <PA1,PA2,PA3,PA4,PA5,PA6,PA7,PA8,PA9,PA10>:(5)
Signal:PM11/38:61:PM11: <PA1,PA2,PA3,PA4,PA5,PA6,PA7,PA8,PA9,PA10>:(9)

Signal:PM11/39:1441:PM11: <PA1,PA2,PA3,PA4,PA5,PA6,PA7,PA8,PA9,PA10>:(8)

Inscription:PA1/3:1442:PA1:<PM11>
Inscription:PA2/3:1442:PA2:<PM11>
```

Figure 6-5 Extraits du fichier généré par le moniteur.

6-11 Evaluation de la plate-forme de simulation

Dans le cadre de l'évaluation de notre outil, nous n'avons pas fait d'études comparatives entre la simulation du marché financier, développée avec notre outil, et la simulation séquentielle du même marché, développée avec un outil de simulation traditionnel. Nous nous reposons toutefois sur les recherches décrites dans les littératures pour affirmer que de façon générale, la simulation distribuée d'un système traitant un grand nombre d'événements réduit considérablement le temps d'exécution par rapport à la simulation séquentielle du même système.

Au niveau de la conception et du développement de notre outil, l'utilisation de la plate-forme *High Level Architecture* nous a considérablement facilité la tâche. En effet, nous avons constaté ceci à plusieurs niveaux :

- La complexité de l'application est réduite. En effet, une fois le FOM et les scénarios de la simulation développés, chaque fédéré peut être conçu et développé individuellement, indépendamment des autres fédérés. Bien que les fédérés PA et PM du marché financier ont été développés et testés séparément, leur intégration a été très facile et n'a pas posé de problèmes particuliers.

- La transparence de la gestion du temps de la simulation au niveau de chaque fédéré facilite également la conception et l'implantation des fédérés. Généralement, dans les environnements de simulation distribuée, la gestion du temps de la simulation au niveau de chaque fédéré dépend de celle des autres fédérés, ce qui engendre un niveau de corrélation élevé entre les différents fédérés. Dans HLA, par contre, la gestion du temps d'un fédéré ne dépend pas de celle des autres fédérés. Dans la même fédération HLA, chaque fédéré gère son temps de manière transparent aux autres fédérés, ce qui nous a permis de concevoir et d'implanter la gestion du temps de chaque fédéré de la simulation de marchés de façon indépendante.

- La ré-utilisabilité d'un fédéré dans des fédéréations différentes, qui forme l'un des objectifs de notre projet, a été facilement atteinte puisque cette caractéristique est assurée par l'environnement HLA. En effet, comme nous l'avons mentionné dans le chapitre III, l'un des objectifs principaux de HLA est de permettre la ré-utilisabilité des fédérés dans le cadre de fédérations différentes, ceci avec un minimum d'effort d'intégration. Ainsi, si nous désirons simuler un marché financier avec des participants ayant les caractéristiques de ceux que nous avons développés dans le cadre de notre preuve de concept, nous n'avons pas à re-développer ces fédérés mais nous pouvons re-utiliser les fédérés existants. Cependant, ces fédérés PA ont été développés pour participer dans des places marchés particulières. Dans le cas où nous désirerions simuler des marchés financiers avec des places de marchés différentes de celles développées dans le cadre de la preuve de concept, les fédérés PA développés doivent être modifiés ou d'autres développés.

Chapitre 7
Conclusion

Le but de ce mémoire a été de développer un outil permettant de développer des simulations distribuées de marchés électroniques basés sur les enchères. En absence, sur le marché, d'outils de simulations distribuées dédiés au domaine des enchères électroniques, notre outil fournit un moyen plus direct et plus simple à utiliser que les outils de simulation distribuée à utilité générale. En effet, avec notre outil, l'utilisateur n'a pas à se soucier des détails reliés à la technique de simulation, mais plutôt des aspects reliés au domaine d'application.

Dès le départ, un des objectifs les plus importants que nous nous sommes fixés a été la création de l'outil le plus générique possible. Ceci ne signifie pas que l'outil doit permettre de développer tous les prototypes de marché possibles, mais doit permettre de simuler des marchés construits conformément au prototype de marché paramétrable que nous avons eu à développer en premier lieu de notre travail. Cet objectif a été atteint dans une grande mesure : notre outil permet à l'utilisateur de jouer sur les paramètres du prototype pour instancier la simulation d'un prototype particulier de marché. Ainsi, l'utilisateur peut introduire les spécificités de son cas particulier telles que les règles du marché, le vocabulaire utilisé, les mécanismes d'enchère, etc.. Le développement d'un outil plus ouvert qui offre le moyen de développer des simulations de différents prototypes de marché était possible mais, un tel outil offrirait des fonctionnalités de plus bas niveau que le notre et rendrait plus fastidieuse la tâche de l'utilisateur.

L'objectif de dégager au maximum l'utilisateur des efforts de conception et d'implantation des aspects reliés aux techniques de simulation, a également été atteint dans certaines limites. En effet, certains de ces aspects dépendent largement des spécificités du cas particulier de marché simulé, et doivent donc être réétudiés pour chaque nouveau cas de marché. Nous citons, en particulier, le choix de la valeur du lookahead pour les deux fédérés PA et PM, et la simulation du comportement interne de

l'aviseur. Toutefois, l'outil encapsule en grande partie les aspects reliés aux techniques de simulation. En particulier, l'utilisateur n'a pas à se soucier de l'avancement du temps des fédérés, des interactions entre les fédérés et, en grande partie, de la gestion des événements.

Le test de l'outil, réalisé dans le cas particulier du marché financier, nous a montré que l'utilisation de l'outil est relativement facile et aboutit aux résultats désirés. En effet, il nous a suffi de développer le code du mécanisme d'allocation, de l'aviseur et des règles du marché pour obtenir l'application de simulation du marché.

Certains aspects n'ont pas été implantés par l'outil et peuvent donc faire l'objet d'améliorations futures. Parmi ces dernières, citons, en particulier, la prise en charge d'autres vocabulaires de mises que celui développé dans le cadre du projet TEM, et le support de la simulation continue pour les aviseurs. Notons, enfin, que l'outil peut être amélioré au niveau de l'implantation des règles. Ainsi, par exemple, la définition des règles deviendrait beaucoup plus facile pour l'utilisateur, si on arrive à développer une grammaire qui permet à l'utilisateur d'exprimer ses règles dans certaines dimensions prédéfinies en utilisant un ensemble d'opérateurs prédéfinis, au lieu de développer des classes C++ pour implanter ces règles.

Bibliographie

[Abrache01a] Abrache, J., Crainic, T.G. et Gendreau, M. *Auction Mechanisms for Freight Transportation*. Proceedings of the Fourth Triennial Symposium on Transportation Analysis (TRISTAN-IV), Sao Miguel, Portugal, June 2001.

[Abrache01b] Abrache, J., Crainic, T.G. et Gendreau, M. *Design Issues for Combinatorial Auctions*. In Proceedings of the 4th International Conference on Electronic Commerce Research (ICECR-4), Dallas, TX, 2001.

[Abrache02a] Abrache, J., Bourbeau, B., Crainic, T.G. et Gendreau, M. *A New Bidding Framework for Combinatorial E-Auctions*. Publication CRT-2002-19, Centre de Recherche sur les Transports, Université de Montréal, Canada, 2002.

[Abrache02b] Abrache, J., Crainic, T.G. et Gendreau, M. *Combinatorial Auctions for Portfolio Bundle Trading*. Working paper, Centre de Recherche sur les Transports, Université de Montréal, Canada, 2002.

[Babin01] Babin, G., Crainic, T.G., Gendreau, M., Keller, R., Kropf, P.G, et Robert, J. *Towards Electronic Market-places: A Progress Report*. ICECR-4, Dallas, TX, pp. 637-648, November 2001.

[Bennett95] Bennett, B.S. *Simulation Fundamentals*. Prentice Hall, London, 1995.

[CIRANO00] *Centre Interuniversitaire de Recherche en Analyse des Organisation, http://www.cirano.qc.ca/.*

[DMSO00] *Defense Modeling and Simulation Office (DMSO)*, http://www.dmso.mil.

[Ferscha94] Ferscha, A. et Tripathi, S. K. *Parallel and distributed simulation of discrete event systems, Technical Report*. CS.TR.3336, University of Maryland, 1994.

[Fujimoto00] Fujimoto, R.M. *Parallel and distributed simulation systems*. Wiley, 2000.

[Fujimoto96] Fujimoto, R. M. et Weatherly, R. M. *HLA Time Management and DIS*. 14th Workshop on Standards for the Interoperability of Distributed Simulation, March 1996.

[Fujimoto98] Fujimoto, R.M. *Time Management in the High Level Architecture*. Simulation Special Issue on High Level Architecture, vol. 71, no. 6, 388-400, 1998.

[Fujimoto99] Fujimoto, R.M. *Parallel and Distributed Simulation*. Proceedings of the 1999 Winter Simulation Conference, 1995.

[Ghost00] Ghost, S. Lee, T.S. *Modeling and Asynchronous distributed simulation: Analyzing complex systems*. IEEE Press, 2000.

[IEEE99a] IEEE. *IEEE P1516/D4 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules.* IEEE Press, April 1999.

[IEEE99b] IEEE. *IEEE P1516.1/D4 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federate Interface Specification.* IEEE Press, April 1999.

[IEEE99c] IEEE. *IEEE P1516.2/D4 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Object Model Template (OMT).* IEEE Press, April 1999.

[IEEE99d] *Defense Modeling and Simulation Office (DMSO), RTI 1.3 Next Generation Programmer's guide.* IEEE Press, October 1999.

[IEEE99e] IEEE. *IEEE P1516.3/D4 Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Federation Development and Execution Process (FEDEP).* IEEE Press, April 1999.

[Law00] Law, A.M. Kelton, W.D. *Simulation Modeling and Analysis.* McGraw-Hill, 2000.

[Low99] Low, Y., Lim, C., Cai, W., Huang, S., Hsu, W., Jain, S. et Turner, S.J. *Survey of Languages and Runtime Libraries for Parallel Discrete-Event Simulation.* Simulation 72:3, 170-186, 1999.

[McLeod00] *McLeod Institute of Simulation Sciences,* http://www.ecst.csuchico.edu/~mcleod/.

[Misra86] Misra J. *Distributed Discrete-Event Simulation.* Computing Surveys, Vol. 18, No. 1, March 1986.

[Mitre] *Distributed SimJava - Mitre Technologies,* http://ms.ie.org/websim/simjava.

[ModSim] *Modeling & Simulation - ModSim,* http://www.modsim.com.

[Morse97] Morse, K.L. et Steinman, J.S. *Data Distribution Management in the HLA.* Spring Simulation Interoperability Workshop, 1997.

[TWOS] *Time Warp Operating System - Laboratory for Advanced Systems research, http://ficus-www.cs.ucla.edu/ficus-members/reiher/Time_Warp.html.*

[WARPED] WARPED: *a time warp simulation kernel - University of Cincinnati,* *http://www.ececs.uc.edu/~paw/warped*.

[Weatherly99] Kuhl, F. Weatherly, R. et Dahmann, J. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture.* Prentice Hall, 1999.

Annexe A

Structure du fichier généré par le moniteur

Le moniteur écrit toutes les informations dans le fichier *moniteur.txt*, au format texte plat que nous décrivons dans ce paragraphe. Le fichier représente un événement de la simulation par ligne. Les lignes sont séparées par un retour à la ligne (LF). Dans cet annexe nous décrivons la structure de ce fichier.

Les fédérés qui rejoignent la simulation

Syntaxe :

FEDERE:*code*:*handle*:*host*

Description :

Cette ligne correspond à un fédéré qui a rejoint la fédération.

Paramètres :

- *code* : c'est le code du fédéré donné par l'utilisateur lors du lancement du fédéré. En d'autres termes, il représente le code d'une place de marché ou le code d'un participant.
- *handle* : c'est le handle du fédéré qui lui est attribué par la RTI lorsque le fédéré rejoint la fédération.
- *host* : c'est le nom du hôte où est lancé le fédéré en question.

Les fédérés ayant quitté la fédération

Syntaxe :

RESIGN :*code*

Description :

Cette ligne indique qu'un fédéré a quitté la fédération

Paramètres :

- *code* : c'est le code du fédéré en question.

Les caractéristiques d'un marché

Syntaxe :

CARACTERISTIQUES:*code,codeEvenement :code :codeTypePM :(Type1,...,Typen,:[RèglesDebutEnchere] :.... :[RèglesArretMarché],nombreDePhases,(Phase1),..... (Phase n)*

Où chaque Phase a la syntaxe suivante :

Mecanisme,[RèglesArretPhase],[RèglesAcitivte],[ReglesAdmissibilite],[ReglesArretRondes]

Description :

Cette ligne indique que les caractéristiques d'un marché ont été diffusées aux participants à travers la mise à jour d'un objet de la classe *caracteristiquesPM*. Le format utilisé pour communiquer les caractéristiques de données n'a pas été développé. En effet, il faut définir une grammaire qui nous permet de spécifier les caractéristiques. Pour cette version de l'outil, les places de marchés envoient les caractéristiques du marché à titre indicatif, c'est à dire qu'il n'est pas utilisé par les participants qui les reçoivent. Ces caractéristiques ont été introduites pour les besoins futurs. Les participants utilisent le code contenu dans le paramètre *codeTypePM*, qui donne une idée globale sur les caractéristiques de la place de marché. En effet, l'utilisateur affecte un code type pour chaque fédéré PM qu'il développer. Chaque fédéré de ce type publie ce code type à travers le paramètre CodeTypePM de la classe *caracteristiquesPM*. De sa part, un participant doit reconnaître les places de marché auxquelles il est capable de participer, à travers leur *codeTypePM*, qu'il découvre dynamiquement.

Paramètres :

- *code* : code de la place de marché ayant diffusé cette information.
- *CodeTypePM* : code du type de la place de marché.
- *CodeEvenement* : code de l'événement.
- *Type1,...Typen* : les types des objets qui peuvent être transigées dans le marché. La structure de cette information est spécifique et est définie par l'utilisateur lors de l'instanciation d'un nouveau marché.
- *RèglesDebutEnchere* : l'ensemble des règles de début de l'enchère. Pour l'instant ils sont en format texte en français.
- *RèglesArretMarche* : l'ensemble des règles d'arrêt du marché.. Pour l'instant ils sont en format texte en français.
- *nombreDePhases* : nombre de phases dans l'enchère.
- *Mecanisme* : code du mécanisme de la phase. On a supposé ici que dans les versions futures de l'outil les places de marchés pourront communiquer les codes des mécanismes qu'ils utilisent. Ces codes doivent être connus aux participants.
- *RèglesArretPhase* : c'est l'ensemble des règles d'arrêt de la phase.. Pour l'instant ils sont en format texte en français.
- *RèglesActivité* : c'est l'ensemble des règles d'activité de la phase.. Pour l'instant ils sont en format texte en français.
- *RèglesAdmissibilité* : c'est l'ensemble des règles d'admissibilité de la phase.. Pour l'instant ils sont en format texte en français.
- *RèglesArrêtRondes* : c'est l'ensemble des règles d'arrêt de ronde de la phase.. Pour l'instant ils sont en format texte en français.

Les Inscriptions

Syntaxe :

Inscription :codeEvenement :temps :*code*:<*dest*>

Description :

Cette ligne représente une demande d'inscription envoyée par un participant à une place de marché.

Paramètres :

- *CodeEvenement* : code de l'événement. Il est unique au niveau de la fédération et créé par le fédéré expéditeur.
- *temps* : temps logique de l'envoi de la demande.
- *code* : c'est le code du participant en question.
- *dest* : place de marché destinatrice de la demande.

Les Accords

Syntaxe :

Accord:codeEvenement:temps :*code*:<*dest*>:*referenceEvenement*

Description :

Cette ligne indique qu'une place de marché a envoyé un accord à un participant indiquant l'acceptation d'une mise ou une inscription.

Paramètres :

- *codeEvenement* : code de l'événement. Il est unique au niveau de la fédération et créé par le fédéré expéditeur.
- *temps* : temps logique de l'envoi de la demande.
- *code* : c'est le code du participant en question.
- *dest* : place de marché destinatrice de la demande.
- *referenceEvenement* : référence sur l'événement sur lequel porte l'accord.

Les Refus

Syntaxe :

Refus:codeEvenement:temps:*code*:*<dest>*:*referenceEvenement:*(*motif*)

Description :

Cette ligne indique qu'une place de marché a envoyé un refus à un participant indiquant le rejet d'une mise ou une inscription.

Paramètres :

- *codeEvenement* : code de l'événement. Il est unique au niveau de la fédération et créé par le fédéré expéditeur.
- *temps* : temps logique de l'envoi de la demande.
- *code* : c'est le code du participant en question.
- *dest* : place de marché destinatrice de la demande.
- *referenceEvenement* : référence de l'événement sur lequel porte l'accord.
- *motif* : c'est le motif du rejet. Les conventions sur le format de ce champ sont spécifiques et doivent être faites par l'utilisateur pour chaque nouveau marché développé.

Les signaux

Syntaxe :

Signal:codeEvenement:temps:*code*:*<dest>*:*(tpyeSignal)*

Description :

Cette ligne indique qu'une place de marché a envoyé un signal aux participants. Un signal indique un changement d'étape de la place de marché.

Paramètres :

- *codeEvenement* : code de l'événement, unique au niveau de la fédération.
- *temps* : temps logique de l'envoi de la demande.
- *code* : c'est le code de la place de marché en question.
- *dest* : destinataire du signal. Dans le cas d'un signal concernant l'activité d'un participant (typeSignal = 6 ou7) il indique le code du participant. Dans tous les autres cas *dest* est vide indiquant que le le signal est envoyé à tous les fédérés participants même s'ils ne participant pas dans la place de marché.
- *TypeSignal* : code numérique indiquant signification du signal. Les signaux suivants sont supportés
  - `1 : Signal de debut d'enchère`
  - `2 : Signal de debut de phase`
  - `3 : Signal de fin de la période des annonce et du début de la période des mises dans le cas d'une phase discrète avec acheteurs et vendeurs exclusifs.`
  - `4 : Signal d'une nouvelle ronde.`
  - `5 : Signal de fin de l'enchère.`
  - `6 : Signal d'avertissement concernant l'activité d'un participant.`
  - `7 : Signal d'exclusion d'un participant.`
  - `8 : Signal d'ouverture du marché.`
  - `9 : Signal de fermeture de marché.`

Les Allocations

Syntaxe :

Allocation:codeEvenement:temps :*code*:<*dest1*,...,*destn*>:`{Allocation1,…,AllocationN}`

Où *AllocationI* représente une allocation atomique ayant la forme :

(prix;Quantité;ReferenceObjet)

Description :

Cette ligne représente une allocation envoyée par une aux participants.

Paramètres :

- *codeEvenement* : code de l'événement. Il est unique au niveau de la fédération et créé par le fédéré expéditeur.
- *temps* : temps logique de l'envoi de la demande.
- *code* : c'est le code du participant en question.
- *dest,...,destn* : participants destinataires du message.
- quantité : quantité allouée de l'objet. Si l'enchère ne supporte pas de quantités. Ce champ contient 0.
- prix : prix à payer pour l'objet.
- referenceObjet : référence de l'objet alloué.

Les mises

Syntaxe :

Mise:codeEvenement :codeEvenement:temps:*code:<dest1,...,destn>:*`MISE_COMBINEE`

Où *MISE_COMBINEE* est définie récursivement et a la forme :

[OP_COMBINE,PRIX, LISTE_MISE_COMBINEE] | MISE_PARTIELLE,

LISTE_MISE_COMBINEE a la forme :

MISE_COMBINEE,LISTE_MISE_COMBINEE | Null,

MISE_PARTIELLE a la forme :

{OP_PARTIELLE,PRIX,LISTE_MISE_ATOMIQUE} | MISE_ATOMIQUE,

LISTE_MISE_ATOMIQUE a la forme :

MISE_ATOMIQUE,LISTE_MISE_ATOMIQUE | Null,

OP_COMBINE est de la forme :

Or | And | Xor | Select(Nl;Nu),

OP_PARTIELLE est de la forme :

Equal | Ordering | Simplex | Q-simplex(Beta) | Select(Kl,Ku),

MISE_ATOMIQUE est de la forme :

(PRIX, Quantite, BorneInferieure, ReferenceObjet),

PRIX est de la forme :

Prix,Significatif

Description :

Cette ligne indique qu'un participant a envoyé une mise. La structure d'une mise est telle que définie dans le document "A new Bidding Framework for Combinatorial Auctions".

Paramètres :

- *codeEvenement* : code de l'événement. Il est unique au niveau de la fédération et créé par le fédéré expéditeur.
- *temps* : temps logique de l'envoi de la demande.
- *code* : c'est le code du participant en question.
- *dest,...,destn* : participants destinataires du message.
- *prix* : prix à payer pour l'objet.
- *significatif* : =1 si le prix est significatif et 0 sinon.
- *quantité* : quantité minimale demandée pour l'objet
- *borneInférieure* : borne inférieur du pourcentage demandé.
- *referenceObjet* : référence de l'objet demandé.

Annexe B

Digrammes UML de la librairie

La librairie développée offre un ensemble de classes permettant de développer les fédérés PM et PA. Dans cet annexe, nous présentons les digrammes UML de conception de ces classes. Dans ces digrammes, nous présentons uniquement les attributs et méthodes les plus importantes des classes. En effet, les diagrammes originaux sont trop larges pour être présentés en entier dans ce mémoire. Les diagrammes originaux, développés avec l'outil *ArgoUML 0.14* sont disponibles dans le répertoire */home/khemila/projet/Uml* du réseau du Centre de Recherche sur les Transports CRT.

Dans cet annexe, nous ne décrivons pas en détail les classes représentées dans les diagrammes. En particulier, nous ne décrivons pas en détail les attributs et les méthodes de ces classes. Tous ces détails sont présentés dans le document de conception de la librairie, disponible dans le répertoire */home/khemila/projet/conception* du réseau du CRT.

Les diagrammes des classes d'événements

Les diagrammes UML représentant les classes qui implantent les événements de la simulation sont représentés dans les figures B-1 et B-2. La figure B-1 représente la classe abstraite *Evenement* et ses sous-classes. Les classes d'événements sont utilisées tant au niveau du fédéré PA que du fédéré PM puisque ces deux derniers échangent tous les types d'événements représentés dans le diagramme.

La classe *Evenement* représente un événement en général. C'est la classe racine de toutes les classes d'événements. Elle porte en particulier l'attribut *temps* qui représente le temps de l'événement. Les classes *Inscription, Allocation, Annonce, Reponse, Mise* et *Signal* représentent respectivement une inscription, une annonce, une réponse d'acception ou de refus, une mise et un signal. La classe *caracteristiquesPM* représente les

« caractéristiques » d'une place de marché qu'elle communique aux participants. Les caractéristiques des phases d'enchère utilisées par les places de marché sont modélisées par les classes représentées dans le diagramme de la figure B-2. Ces « caractéristiques » ne sont pas utilisées dans la version actuelle de la librairie. Elles ont été introduites pour permettre le développement d'aviseurs capables de s'adapter dynamiquement aux places de marchés suivant les caractéristiques de ces dernières.

Vue la complexité d'une mise dans le cadre du vocabulaire de mises utilisé, une mise est représentée par plusieurs classes : *Mise, MiseCombinee, MisePartielle, Mise Atomique, MisePartielleNonAtomique* et *MiseRecursive.*

Digramme des classes du fédéré PA

En plus des classes d'événements représentées dans les figures B-1 et B-2, Le fédéré PA utilise principalement les cinq classes représentées dans la figure B-3. La classe *Participant* représente un participant. Elle définit en particulier l'attribut *aviseur* qui est un objet de la classe *Aviseur* et qui représente l'aviseur du participant. La classe *Participant* implante également la méthode *executer()* qui lance la boucle principale du participant. La classe *Aviseur* utilise la méthode *consulter()* appelée par le participant pour « consulter » son aviseur dans l'objectif de recevoir une stratégie de mises. L'utilisation de cette méthode est décrite en détail dans le chapitre V de ce document.

La classe *interfaceRTI* implante l'interface entre le participant et la RTI. Elle utilise les deux classes *RTIAmb* et *FedAmb* qui implantent l'ambassadeur de la RTI et l'ambassadeur du fédéré (cf. chapitre III).

Diagramme des classes du fédéré PM

A part les classes d'événements décrits plus haut dans cette section, le fédéré PM utilise les classes représentées dans le diagramme UML de la figure B-4. Les classes représentées sont les suivantes :

- Les classes *Vocabulaire, InnerLevel* et *OuterLayer* représentent un vocabulaire de mises avec ses deux niveaux (cf. chapitre V). La classe *RegleVoca* implante une règle d'admissibilité des mises dont l'objectif est de vérifier que les mises émises par les participants sont conformes au vocabulaire utilisé.

- La classe *Phase* est une classe abstraite. Les classes *PhaseContinue, PhaseAchatVenteSimulatane* et *AchatVenteExclusif* héritent de la classe *Phase* et implantent une phase continue, une phase discrète avec position d'achats et ventes simultanées et une phase discrète avec position d'achats et ventes exclusives, respectivement. Chacune de ces classes utilise, en particulier, des attributs qui sont des vecteurs contenant les règles de la phase.

- Les classes représentées en couleur foncée, à l'exception des classes *calculAllocationPrix* et *calculAllocationPrixContinu*, modélisent les règles du marché. Toutes ces classes sont abstraites puisqu'elles définissent la méthode abstraite *verifier()* qui doit être implantée par l'utilisateur.

- Les classes *calculAllocationPrix* et *calculAllocationPrixContinu* modélisent les mécanismes d'allocation et de calcul de prix dans le cas d'une phase discrète et continue, respectivement. Elles sont des classes abstraites puisqu'elles définissent la méthode *allouer()* qui doit être implantée par l'utilisateur.

- La classe *Encanteur* implante l'encanteur. Elle définit, en particulier, la méthode *executer()* qui implante la boucle principale de l'encanteur.

- La classe *interfaceRTI* implante l'interface entre le participant et la RTI. Elle utilise les deux classes *RTIAmb* et *FedAmb* qui implantent l'ambassadeur de la RTI et l'ambassadeur du fédéré (cf. chapitre III).

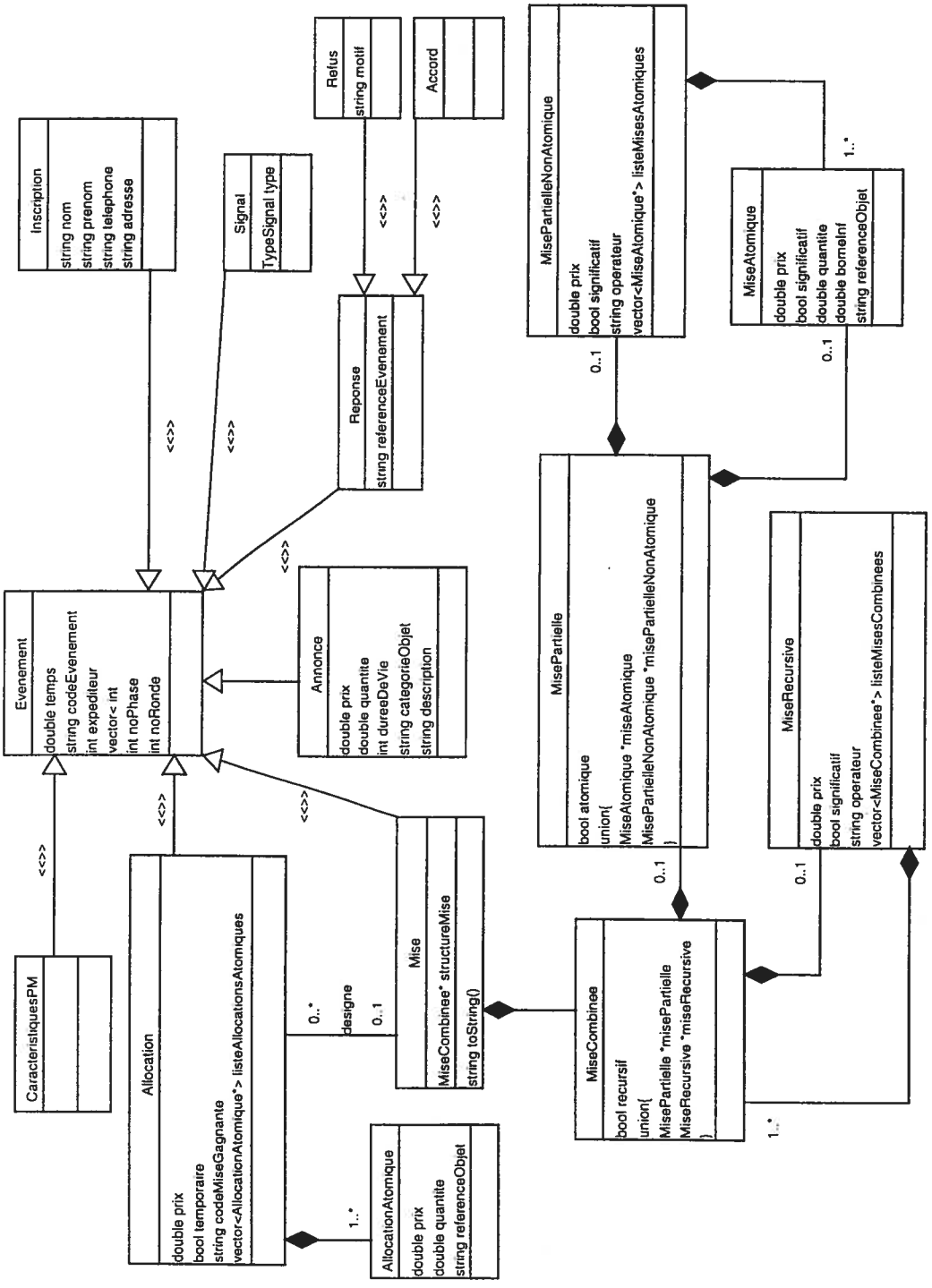

Figure B-1: Diagramme UML des classes d'événements.

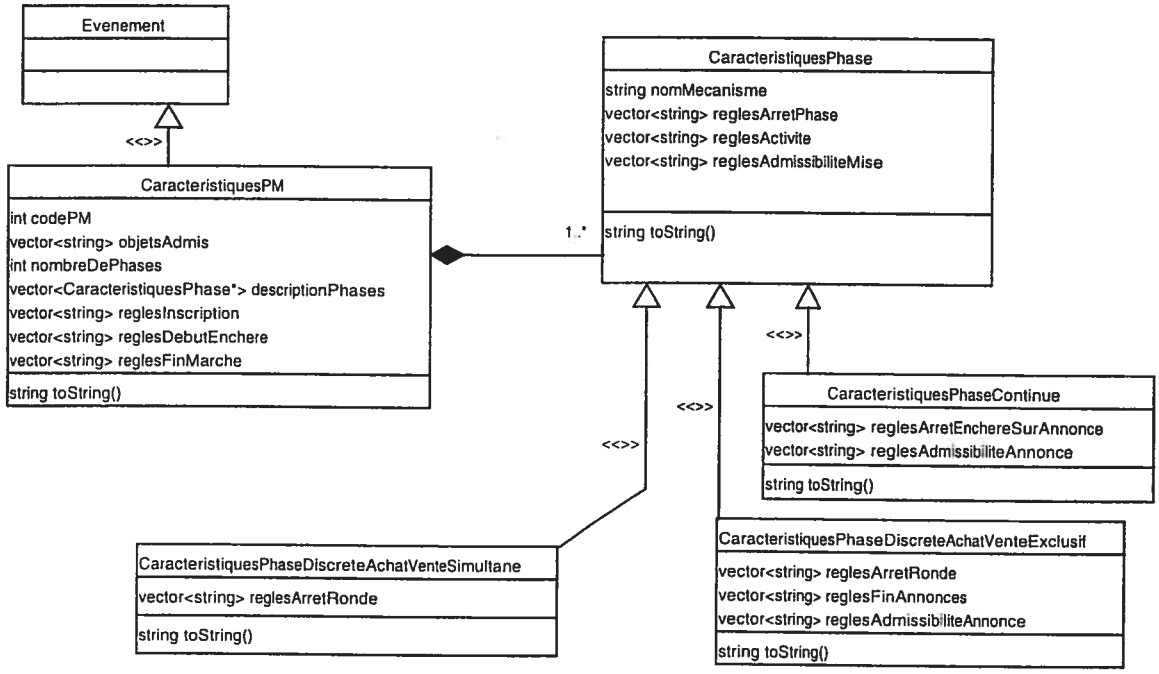

Figure B-2: **Diagramme UML de la classe caracteristiquesPM.**

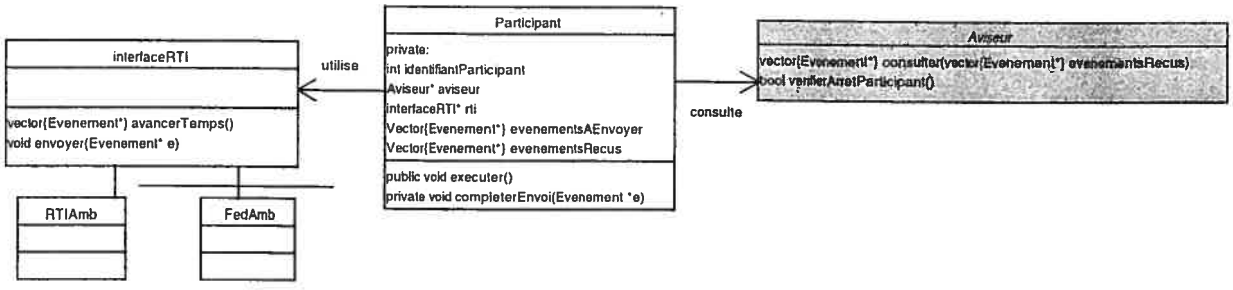

Figure B-3: **Diagramme UML des classes du fédére PA.**

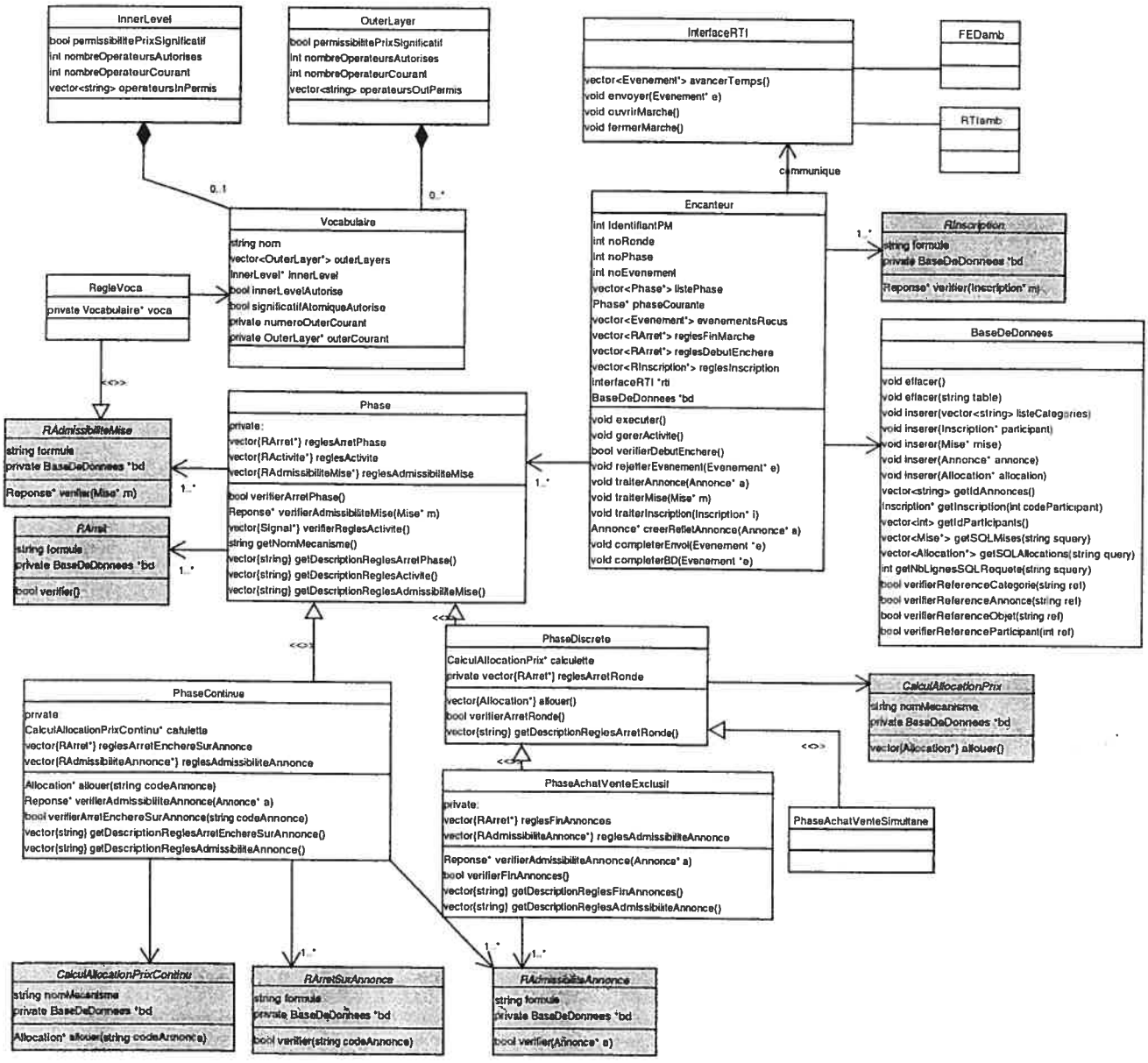

Figure B-4: Diagramme UML des classes du fédére PM.