Distribution et Parallélisation de Simulations Orientées Agents

Nicolas Sébastien

Département de Mathématiques et Informatique **École Doctorale Interdisciplinaire**
UFR Sciences et Technologies

Distribution et Parallélisation de Simulations Orientées Agent

THÈSE

présentée et soutenue publiquement le 10 Novembre 2009

pour l'obtention du

Doctorat de l'Université de La Réunion

(spécialité informatique)

par

Nicolas Sébastien

Composition du jury

Président :	Frédéric Mesnard	Professeur des Universités Université de la Réunion
Rapporteurs :	Jacques Ferber	Professeur des Universités Université de Montpellier 2
	Eric Ramat	Professeur des Universités Université du Littoral
Examinateur :	Fabien Michel	Maître de Conférences Université de Montpellier 2
Directeur de thèse :	Rémy Courdier	Professeur des Universités Université de La Réunion
co-Directeur de thèse :	Marc-Philippe Huget	Maître de Conférences Université de Savoie

Laboratoire d'Informatique et de Mathématiques, IREMIA — EA 2525

Mis en page avec la classe thloria.

Remerciements

Je tiens à remercier mon directeur de thèse, Rémy Courdier, pour m'avoir donner l'opportunité de traiter ce sujet de thèse. Mes remerciements vont également à mon co-directeur de thèse, Marc-Philippe Huget, dont les remarques tout aussi pertinentes que cinglantes ont été sources de motivation. J'ai apprécié l'autonomie et la liberté qu'ils m'ont laissé ainsi que leur soutien sans faille tout au long de mon travail de recherche.

Je remercie Jacques Ferber et Eric Ramat pour m'avoir fait l'honneur de relire et estimer mes travaux. Leurs remarques ont éclairé ces derniers sous de nouvelles perspectives.

J'ai eu un grand plaisir à accueillir Fabien Michel dans mon jury. Je le remercie tout autant pour le professionnalisme dont il a fait preuve dans l'évaluation de mes travaux que pour ses grandes qualités humaines et les discussions ouvertes que nous avons pu avoir.

Je remercie Frédéric Mesnard, directeur de l'Iremia durant ma thèse, d'avoir accepté de présider mon jury de thèse. J'ai pu apprécier sa rigueur et l'excellence de sa recherche.

Une grande partie de ma thèse n'aurait pu se faire sans Denis Payet. Ses travaux au sein de l'équipe Smart m'ont servi de base pour le développement et l'évaluation de mes algorithmes. J'ai trouvé en lui une oreille attentive branchée sur un cerveau bouillonnant. Je le remercie pour toutes ces heures passées à construire et déconstruire nos idées.

Durant ces années de doctorat, j'ai eu la chance d'être entouré de nombreuses personnes qui, chacun à sa manière, m'ont aidé à façonner ma vision de la recherche. Je remercie ici Daniel David, mon colocataire de bureau et attentif ami, Tiana Ralambondrainy, jeune chercheur exemplaire dont le rôle en réunion était souvent salvateur, Didier Hoarau qui m'a beaucoup aidé à améliorer ma rédaction et Yasine Gangat, doctorant au brillant avenir scientifique.

Je souhaite remercier tous les membres du laboratoire et de l'université qui m'ont tant apporté et m'ont soutenu durant ces années : Noël Conruyt, Marie-Catherine Daniel-Vatonne, Charles-André Payet, Jean Diatta, Gisèle Ulderic, Fabiola Celerine, Judex De Louise, France-Anne Longin, Philippe Bannet, Sylvie Desqué.

Cette thèse n'aurait pu se faire sans mes amis qui m'ont soutenu et m'ont partagé leur vision des *Sciences*. Je remercie notamment Evelyne, Tony, Karel, Marcus, Nirmal, Thierry, Christine, Gabriel, Lou, Laurent, Maëlle, Marc-Antoine, Emmanuel, Gaëlle, Alice, Nathalie, Emmanuelle, Karine, Flavie, Audrey, Laëtitia, Carole, Maéva, Jérémy, Caroline, Héléna ainsi que tous les membres de l'Association des Docteurs et Doctorants de la Réunion. Je remercie Christelle d'avoir eu la bonne idée de rédiger sa thèse vétérinaire en même temps que moi : je me remémorerai avec plaisir nos éprouvantes journées de rédaction où nous nous sommes soutenus mutuellement.

Enfin, je remercie ma famille pour son soutien permanent et ses nombreux encouragements tout au long de mes études.

“Rien n’est plus puissant que le cerveau humain,
Si ce n’est plusieurs cerveaux travaillant ensemble.”
— Anonyme

Table des matières

Introduction **1**

1 Contexte 1

2 Problématique 4

3 Objectif 5

4 Cadre et limites de ce travail de recherche 6

5 Organisation de la thèse 7

Chapitre 1 Cadre préalable et état de l'art **9**

1.1 Systèmes Multi-Agents pour la simulation de systèmes complexes 9

1.2 Travaux et réalisations de l'équipe SMART 13

1.3 La distribution sur les plates-formes de SOA courantes 14

1.4 Infrastructures d'exécution distribuée 25

1.5 Conclusion 31

Chapitre 2 Cadre conceptuel agent pour la réalisation d'une infrastructure pour la SOA distribuée **33**

2.1 Aborder la SOA distribuée sous l'angle du Cloud Computing 34

2.2 Caractéristiques de la distribution de SOA 35

2.3 Structuration en couche de l'architecture de distribution 38

2.4 Système Multi-Agents réparti pour la Simulation Orientée Agent distribuée . . . 42

2.5 Synthèse 48

Chapitre 3 Modèle d'ordonnancement parallèle multi plates-formes **53**

3.1 Représentation du temps dans les simulations 54

3.2 Ordonnancement et parallélisation 56
3.3 Identification des liens de causalité avec le modèle à Temporalité 62
3.4 Mise en oeuvre de l'ordonnanceur global 65
3.5 Discussion 71
3.6 Synthèse 79

Chapitre 4 Optimisation de SOA distribuées par équilibrage de charges 81

4.1 Cadre d'utilisation 82
4.2 Approche générale 83
4.3 Algorithme d'équilibrage de charges 86
4.4 Description des phases de l'algorithme 90
4.5 Adaptation à l'ordonnancement distribué séquentiel du modèle à Temporalité . . 97
4.6 Application à l'ordonnancement distribué parallèle du modèle à Temporalité . . 103
4.7 Synthèse 107

Chapitre 5 Prototypage et expérimentations 111

5.1 Intérêt du prototypage 112
5.2 Usages de la simulation d'exécution parallèle et distribuée de SOA 113
5.3 Simulation d'exécution parallèle et distribuée de SOA 115
5.4 Expérimentations 126
5.5 Discussions 139
5.6 Synthèse 142

Conclusion 145

1 Synthèse des propositions 145
2 Perspectives 147

Bibliographie 149

Table des figures 159

Introduction

Sommaire

1	**Contexte**	**1**
2	**Problématique**	**4**
3	**Objectif**	**5**
4	**Cadre et limites de ce travail de recherche**	**6**
5	**Organisation de la thèse**	**7**

1 Contexte

Les scientifiques ont à cœur de comprendre le fonctionnement de l'univers qui nous entoure. Informatique, Biologie, Physique, Chimie et Mathématiques sont qualifiées de sciences du « Comment » par analogie à la Philosophie qui se présente comme la science du « Pourquoi ». La compréhension de notre univers dans ses échelles aussi bien gigantesques (constellations) qu'infinitésimales (atomes) et dans son évolution tant spatiale que temporelle est la source de multiples enjeux : améliorer les conditions de vie, protéger l'environnement et la biodiversité, comprendre nos origines, anticiper les événements, etc.

Les avancées scientifiques et techniques donnent aux chercheurs des outils toujours plus pratiques et performants pour affiner leurs recherches. Le raisonnement scientifique, permettant d'acquérir de nouvelles connaissances, se base sur plusieurs mécanismes cognitifs dont les mieux connus sont :

– la déduction : démarche intellectuelle partant de prémisses et aboutissant à une conclusion[1]. D'après [Thiry, 1998], la déduction « part des jugements généraux pour aller vers les jugements particuliers », « elle se contente d'utiliser des données et d'en tirer le maximum de profit ». La valeur de vérité des conclusions obtenues par déduction dépend directement de la valeur de vérité des prémisses : des trois mécanismes cognitifs, c'est le seul permettant d'obtenir des conclusions dont la vérité universelle est prouvée.

[1] Définition de Déduction issue du dictionnaire *Le Petit Larousse 2008*.

– l'induction : forme d'inférence par laquelle on passe d'un ensemble fini d'observations particulières à une conclusion générale, et qui n'est pas de nature démonstrative[2]. Ce mécanisme cognitif permet d'établir une règle générale à partir de phénomènes observés. La règle est considérée comme vraie tant que tous les phénomènes observés se produisent en accord avec la règle.
– l'abduction : processus permettant d'expliquer un phénomène ou une observation à partir de certains faits, événements ou lois[3]. Pour Pierce ([Peirce *et al.*, 1966]), l'abduction part d'un phénomène observé et non expliqué, et vise à établir une règle hypothétique qui expliquerait la règle. Par exemple si les faits suivants sont avérés *« Les billes de cette urne sont rouges »* et *« Les billes qui se trouvent sur la table sont rouges »*, l'abduction mène à la conclusion que *« Les billes sur la table proviennent de l'urne »*.

De ces trois définitions, il ressort que l'observation d'un phénomène de la réalité, considéré comme occurrence particulière d'une loi générale que le chercheur souhaite exhiber, intervient dans deux mécanismes cognitifs, induction et déduction. L'observation du réel est donc une étape majeure dans l'acquisition de nouvelles connaissances scientifiques ; c'est mûs par ce besoin de mieux connaître les phénomènes de la réalité que les laboratoires poursuivent leur campagne d'instrumentation : les équipements et outils des scientifiques permettent d'isoler et de mettre en exergue les phénomènes que les scientifiques souhaitent observer et appréhender.

Malgré les progrès techniques et les efforts d'instrumentations, de nombreux phénomènes restent encore hors de portée de l'observation scientifique : expérimenter directement sur le système étudié n'est pas systématiquement possible ou optimal. En effet, selon l'expérience à mener, il peut y avoir un problème d'échelle spatiale (systèmes trop grands ou trop petits) ou temporelle (temps de réactions trop courts ou trop lents pour être humainement observables et analysables), et les considérations tant légales qu'éthiques ou économiques constituent autant de freins potentiels à l'expérimentation.

De plus, l'introduction d'un observateur peut induire un biais dans le phénomène. Par exemple, dans le domaine de la physique quantique, on peut citer l'expérience de pensée du chat de Schrödinger [Schrödinger, 1935] où la vie d'un chat est liée à l'état d'un atome d'un corps radioactif tous deux enfermés dans une boîte : en l'absence d'observateur constatant l'état (entier ou désintégré) de l'atome, la physique quantique considère que les deux états sont vrais et donc que le chat est à la fois mort et vivant tant que l'observateur ne regarde pas dans la boîte.

La simulation se pose alors comme une alternative pertinente à l'expérimentation directe. Une simulation est une représentation par un modèle physique ou mathématique d'un phénomène complexe, du comportement d'un appareil ou de l'évolution d'un système à des fins d'étude, de

[2]Définition d'Induction issue du dictionnaire *Dictionnaire d'histoire et philosophie des sciences 1999.*
[3]Définition d'Abduction issue du dictionnaire *Dictionnaire d'histoire et philosophie des sciences 1999.*

mesure ou d'essai[4]

La simulation informatique est un type de simulation particulier dans la mesure où celle-ci reproduit virtuellement le phénomène à observer : elle se base pour cela sur des techniques de modélisation permettant la représentation du système dans une forme exécutable par un ordinateur. La simulation informatique propose plusieurs approches dont les plus courantes sont :

- l'approche mathématique où le système est modélisé sous forme d'équations différentielles,
- la simulation discrète où le système de grande taille subissant les contraintes est subdivisé en cellules sur lesquelles seront appliquées et calculées les contraintes,
- la simulation à événements discrets où le système évolue en fonction des événements qui sont générés au fur et à mesure de la simulation,
- la simulation par automates cellulaires où le système est composé de cellules ayant leurs propres contraintes et interagissant de proche en proche,
- la simulation orientée agent (SOA) où le système est modélisé par les entités qui le composent et qui interagissent entre elles.

Puissance de modélisation et coûts de développement peu élevés font de ces solutions de simulation des outils incontournables dans les domaines aussi bien scientifiques qu'économiques ou sociaux. Les différentes approches en simulation informatique ont chacune leurs avantages et inconvénients et couvrent des champs d'application très variés. Parmi elles, la simulation orientée agent offre une grande puissance de modélisation et devient une approche de plus en plus pertinente dans de nombreux projets. Ces travaux de recherche s'inscrivent dans la recherche sur la Simulation Orientée Agent et prennent pour cible l'exécution parallèle et distribuée de simulations orientées agent.

Un des atouts majeurs de la simulation informatique réside dans le fait que le modélisateur peut définir une échelle temporelle sans rapport avec le temps de la réalité, c'est-à-dire qu'il lui est possible de définir un écoulement du temps propre à la simulation : celle-ci permet de ramener l'occurrence de phénomènes à des échelles temporelles observables par l'homme.

La simulation informatique, en fournissant une représentation virtuelle du système à observer, permet aussi de virtualiser les instruments d'observation. Il est alors possible de rendre le système insensible à ces instruments d'observation et donc éviter que ces derniers ne viennent perturber le système. Cet élément constitue également un défaut : la simulation informatique permet de créer des modèles très variés dont certains peuvent être aberrants (ne pouvant jamais se produire dans la réalité) ou basés sur une finesse de représentation trop grossière pour représenter la réalité du système.

[4]Définition de Simulation issue du dictionnaire *Le Petit Larousse 2008*.

2 Problématique

La simulation orientée agent est une branche spécifique des Systèmes Multi-Agents (SMA) qui découle elle-même de l'Intelligence Artificielle Distribuée (IAD). L'IAD vise la répartition de l'Intelligence Artificielle sur un réseau d'entités : ceci permet d'obtenir un système intelligent complexe en se focalisant sur des entités, sous-systèmes, plus simples à définir. Un système multi-agents est constitué de plusieurs agents interagissant entre eux et avec leur environnement. Un agent se définit comme une entité réelle ou virtuelle, qui se comporte de manière autonome, aux buts propres, aux perceptions et capacités d'action limitées, et capable d'interaction avec d'autres agents ainsi qu'avec l'environnement dans lequel il évolue (voir [Ferber, 1995] et [Wooldridge and Jennings, 1995] pour plus de détails). Les principales caractéristiques des systèmes multi-agents sont décrites dans [Sycara, 1998] :

1. Chaque agent est doté d'une perception incomplète de son environnement et porte des capacités d'action elles aussi limitées. Il n'a qu'un point de vue partiel sur la résolution du problème global.
2. Il n'y a pas de contrôle centralisé du système.
3. Les informations sont diffuses.
4. L'exécution est asynchrone.

Les systèmes multi-agents ont été éprouvés sur des *toys problems* (par exemple *Block Worlds* [Gupta and Nau, 1992] originellement décrit dans [Nilsson, 1982], ou encore *Tileworld* [Pollack and Ringuette, 1990]) dans un premier temps. Ils sont déployés dans de nombreux domaines : réalisation d'architectures informatiques [Foster *et al.*, 2004], contrôle de systèmes industriels [Azaiez *et al.*, 2007], étude comportementale [Beer *et al.*, 1990], jeux vidéos [Mamei and Zambonelli, 2004], etc.

La simulation orientée agent reprend les caractéristiques des SMA en y adjoignant un temps virtuel propre à la simulation : les comportements des agents, notamment leurs interactions, s'établissent en respect de règles temporelles basées sur ce temps virtuel et non sur le temps réel. Les SOA sont couramment utilisées pour la validation d'hypothèses scientifiques (par exemple, l'apparition d'organisations dans les colonies de fourmis [Drogoul *et al.*, 1995]), la réalisation d'outils de prospection et d'aide à la décision (par exemple, l'évolution du mode d'occupation des sols à la Réunion avec [David *et al.*, 2007]), *etc.*

Nous nous intéressons plus particulièrement à la simulation de systèmes complexes naturels ou sociaux. La réalisation de telles simulations implique la modélisation d'environnements très riches, contenant toutes les données et lois d'évolution internes nécessaires à l'exécution des agents. Elle nécessite également un nombre important d'agents en interaction et aux comportements variés. L'objectif de telles simulations est de comprendre les interactions entre les différents acteurs et l'environnement, et l'impact que peut avoir ces interactions sur certaines

valeurs, objets ou phénomènes critiques. En prenant pour exemple le projet Biomas [Courdier *et al.*, 2002], qui s'intéresse aux transferts de matières organiques entre éleveurs et agriculteurs, un phénomène à éviter est le déversement de lisier dans les ravines (pollution des eaux).

La modélisation de systèmes réels en vue de leur simulation informatique est une tache délicate. Cependant, une fois le modèle créé, il est simple et avantageux de le réutiliser pour effectuer plusieurs simulations. Ainsi, le modèle Biomas fait intervenir plusieurs centaines d'agents cognitifs sur un espace géographique restreint (la zone de Grand-Ilet à la Réunion), sa généralisation à l'ensemble de l'île nécessite l'emploi de plusieurs milliers d'agents.

La pertinence des simulations multi-agents pour la modélisation de systèmes complexes et la qualité des résultats fournis par ces simulations rendent cette approche très attrayante. Nous constatons ainsi deux phénomènes :

- une augmentation de la précision des modèles : ils intègrent de plus en plus de paramètres et de phénomènes afin de permettre une analyse multi-échelles.
- une augmentation de l'échelle des simulations : le système réel à modéliser couvre une zone de plus en plus étendue.

Les modèles développés sont de plus en plus complexes du fait de l'augmentation du nombre d'agents et de leurs interactions. L'utilisation de bases de données et autres Systèmes d'Information Géographique permet de stocker les données nécessaires à la simulation. Cependant, la simulation de systèmes large-échelle soulève de nombreuses interrogations : comment concevoir de telles simulations ? Comment supporter et optimiser leur exécution ? Comment observer et analyser les résultats ? Il faudrait revoir l'intégralité du processus de simulation dans cette optique de simulation large-échelle. Mais avant même de se focaliser sur ce sujet, il est nécessaire de définir le cadre conceptuel et les outils génériques sur lesquels appuyer la simulation large-échelle afin de mieux la supporter dans un premier temps, puis d'en faciliter l'usage.

3 Objectif

Nous nous inscrivons dans la recherche sur les simulations large-échelle orientées agent. Les travaux présentés dans ce mémoire ont pour objectif le support de l'exécution de telles simulations avec des performances optimisées.

Avec la généralisation des réseaux, nous nous intéressons aux solutions de type *cloud computing* [Hayes, 2008] pour asseoir la simulation sur une architecture d'exécution distribuée. Ce choix permet de bénéficier de l'aisance de déploiement et de la modularité des ressources propres à ce type de solutions tout en conservant la possibilité d'exploiter les machines multi-cœurs et multi-processeurs.

Notre objectif est donc de proposer un support d'exécution pour le fonctionnement optimisé de la simulation orientée agent sur un réseau ouvert. Souhaitant offrir les meilleures performances

en simulation, nous nous orientons vers une solution spécifique à la simulation orientée agent.

4 Cadre et limites de ce travail de recherche

La définition d'une architecture distribuée dédiée à la simulation orientée agent nécessite des travaux à différents niveaux d'abstraction :

- Au niveau agent, il s'agit dans un premier temps d'assurer la continuité sur le réseau des outils de simulation et d'observation classiques en SOA. Dans un second temps, c'est tout le processus de simulation distribuée qu'il convient de redéfinir : la distribution ne supporte pas que l'exécution de la simulation mais apporte aussi de nouvelles possibilités en terme d'édition, supervision, observation et exploitation de simulations.
- Au niveau des unités de simulation composant l'infrastructure d'exécution, il s'agit d'optimiser l'exécution de la simulation sur chacune d'elle afin de tirer au mieux parti de leur puissance.
- Au niveau réseau, ce dernier doit supporter les outils et services mis en place dans le cadre de l'architecture distribuée aussi bien que de tenir la charge de messages induits par l'exécution de la simulation. Le choix d'un protocole de communication est particulièrement délicat : fiabilité et vitesse de communication sont tous deux nécessaires dans le cadre d'une SOA distribuée. L'exécution sur des réseaux ouverts demande aussi la mise en place de politiques de sécurité.

Dans ce mémoire, nous proposons un cadre conceptuel permettant la réalisation d'une infrastructure pour la SOA distribuée et nous définissons des algorithmes pour l'optimisation de l'exécution parallèle et distribuée des agents de la SOA.

Ayant pour priorité la mise en place de simulations orientées agent distribuées, nous ne nous sommes intéressés qu'à la seule étape d'exécution de la simulation. Nous nous sommes de plus focalisés uniquement sur les optimisations possibles du point de vue agent : l'environnement reste un point crucial à traiter dans le cadre de l'exécution distribuée mais nécessite un travail trop spécifique pour être traité ici.

Nous avons aussi considéré le réseau comme un environnement maîtrisé, c'est-à-dire que celui-ci est fiable et qu'il n'y a aucun problème de sécurité : nous ne nous sommes pas intéressés aux mécanismes à mettre en place en cas de déconnexions intempestives des unités de simulation et nous n'avons pas géré la mise en place de politiques de sécurité. Et si nous sommes conscients de l'intérêt que peuvent revêtir les nouveaux protocoles réseaux (XCP notamment [Katabi *et al.*, 2002]), nous n'avons pas axé nos travaux sur ces thèmes et nous sommes concentrés sur des protocoles de plus haut niveaux (niveau application dans le modèle OSI [Zimmermann, 1980]).

Nos travaux visent l'exécution parallèle et distribuée des agents participant à la simulation d'un système complexe. Dans cette optique, nous définissons des concepts pour la réalisation d'une infrastructure de distribution. Considérant les aspects temporels de la simulation, nous

proposons des algorithmes pour l'ordonnancement parallèle de la simulation distribuée et pour un équilibrage de charges dynamiques fonctionnant sur la base de transferts d'agents.

5 Organisation de la thèse

Le premier chapitre présente notre étude bibliographique. Nous nous intéressons dans un premier temps à la place de la distribution dans différentes plates-formes de simulation orientée agent. Cette étude nous permet d'identifier les limites conceptuelles et logicielles des approches actuelles. Nous considérons par la suite quelques architectures de distribution reconnues et analysons leur potentiel quant à la distribution de la simulation orientée agent.

Notre étude bibliographique ayant abouti à la nécessité de concevoir une architecture pour la distribution de la SOA, nous définissons au chapitre 2 un cadre conceptuel agent pour la réalisation d'une infrastructure pour la SOA. Nous proposons de constituer une *plate-forme virtuelle distribuée* en appuyant sa réalisation sur un système multi-agent réparti. Nous abordons aussi bien la description du SMA réparti que les aspects logiciels nécessaires à son intégration dans l'infrastructure de distribution.

Le chapitre 3 vise la réalisation d'un modèle d'ordonnancement parallèle pour l'exécution distribuée de la simulation. Analysant les différentes approches proposées dans la simulation parallèle à événements discrets, nous optons pour une approche conservative. Notre idée de base est de proposer un ordonnancement parallèle de la simulation en exploitant les informations du modèle simulé : ceci permet d'exécuter en parallèle tous types de modèle. Nous nous appuyons sur un modèle temporel se présentant comme une généralisation des modèles temporels usuels (le modèle à pas de temps constant et le modèle évènementiel) pour identifier les liens de causalité entre les instants d'activation des agents.

Le deuxième mécanisme d'optimisation de l'exécution de la SOA que nous proposons est un mécanisme d'équilibrage de charges. Nous décrivons au chapitre 4 un algorithme simple à mettre en œuvre et permettant le transfert d'agents de la simulation en cours de simulation par négociation entre agents du SMA réparti. Nous étudions de plus le couplage entre notre ordonnancement parallèle et notre algorithme d'équilibrage de charges dynamique.

Nous présentons au chapitre 5 une mise en œuvre de nos algorithmes d'optimisation de l'exécution des agents de la simulation au travers d'une simulation orientée agent. Cette solution nous permet de nous affranchir des barrières techniques de la réalisation de l'architecture distribuée réelle et permet de tester nos algorithmes sans les perturbations induites par des éléments que nous n'avons pu traiter dans nos travaux (par exemple, la distribution de l'environnement). Les agents du SMA réparti sont reproduits dans notre prototype et simulent l'exécution d'une simulation représentée sous la forme d'un environnement.

Chapitre 1

Cadre préalable et état de l'art

Sommaire

1.1 Systèmes Multi-Agents pour la simulation de systèmes complexes . 9
1.2 Travaux et réalisations de l'équipe SMART 13
1.3 La distribution sur les plates-formes de SOA courantes 14
1.3.1 CORMAS 14
1.3.2 MASON 16
1.3.3 SWARM 18
1.3.4 RePAST Symphony 19
1.3.5 JEDI 20
1.3.6 MadKit 21
1.3.7 Synthèse 23
1.4 Infrastructures d'exécution distribuée 25
1.4.1 Grid Computing 25
1.4.2 High Level Architecture 28
1.4.3 Agents mobiles 29
1.5 Conclusion 31

Nous décrivons dans ce chapitre le cadre de nos travaux. Nous présentons pour cela les concepts majeurs dans le domaine de la simulation orientée agent. Nous nous intéressons ensuite à la distribution dans le domaine des SOA et dans un cadre plus général pour tenter d'identifier une approche distribuée adaptée à la SOA.

1.1 Systèmes Multi-Agents pour la simulation de systèmes complexes

Commme nous l'avons souligné en introduction, l'observation du monde réel est une source d'acquisition de connaissances, qu'elles soient scientifiques ou non. Cependant de nombreuses bar-

rières viennent limiter cette observation : grandeurs du système (temporelles ou spatiales), coût, éthique, lois, éloignement géographique, précision des outils de mesure, perturbations induites au moment de la mesure, complexité du système, etc. La simulation consiste alors à reproduire le phénomène que l'on souhaite observer dans un environnement plus maîtrisable. En ce sens, la simulation informatique est un outil particulièrement intéressant. Ce mode de simulation n'impose aucune contrainte sur le modèle à créer ce qui permet de modéliser le système considéré en choisissant et définissant une à une les influences auxquelles il est soumis.

La simulation orientée agent offre notamment de nombreuses possibilités en terme de modélisation : un agent est une entité autonome possédant sa propre vision locale de l'environnement dans lequel il évolue et des capacités d'interaction avec cet environnement et les autres agents. Un agent peut donc représenter une entité dans le système réel ; reste ensuite à définir l'environnement dans lequel les agents évoluent pour reproduire de manière virtuelle le système réel.

Nous nous intéressons plus particulièrement aux parties du monde réel faisant intervenir l'environnement naturel et les groupes sociaux. Ces simulations sont utilisées dans différents domaines : écologie, gestion des ressources, prévention des risques, sociologie, sécurité, etc. Dans ces domaines, les agents représentent les différentes entités composant le système et reproduisent ce système par les interactions qu'ils développent entre eux et avec leur environnement. Les enjeux de ces simulations sont alors de comprendre les phénomènes émergents qui peuvent apparaître, analyser les interactions à différentes échelles, ou mesurer l'impact d'une modification sur le système (introduction d'un nouvel élément). Pour réaliser de telles simulations, il est nécessaire d'isoler la partie que l'on souhaite modéliser du reste du monde réel. Dans le cadre de l'emploi de la SOA, cette partie isolée constitue le plus souvent un système complexe.

Il existe plusieurs définitions complémentaires des systèmes complexes. Nous pouvons citer [Simon, 1962] qui définit un système complexe comme un système constitué d'un nombre important de parties qui interagissent d'une manière non simple. [Moigne, 1990] caractérise la complexité des systèmes par l'imprévisibilité possible et l'émergence du nouveau et du sens plausible. Nous adoptons la définition donnée dans [Ralambondrainy, 2009] :

Définition 1.1.1 Système Complexe *: « une sous-partie du monde réel composée d'éléments en interactions, qui fait l'objet de modélisation et de simulation à des fins de compréhension du monde réel, la complexité pouvant provenir du nombre important de parties du système qui interagissent d'une manière non simple, et conduisant à de l'imprévisibilité possible, de l'émergence du nouveau et du sens plausible ».*

Une fois le système complexe identifié et délimité dans le monde réel, il est nécessaire de lui donner corps dans le monde virtuel servant à la simulation : c'est l'étape de modélisation. La définition de modèle la plus communément admise est celle donnée par [Minsky, 1965].

Définition 1.1.2 Modèle *: « pour un observateur B, un objet A* est un modèle d'un objet A dans la mesure où B peut utiliser A* pour répondre aux questions qui l'intéressent à propos de A ».*

A partir de cette définition de modèle, nous proposons deux types de modèles que nous manipulerons tout au long de ce mémoire : modèle de simulation (que nous désignerons aussi par modèle informatique) et modèle simulé.

Définition 1.1.3 Modèle de Simulation *: représentation informatique du système complexe à partir des connaissances des spécialistes et des concepts informatiques dont disposent les modélisateurs.*

Nous plaçant directement dans le cadre de la simulation informatique, un *modèle de simulation* est donc un modèle décrit sous une forme exécutable par un ordinateur.

Définition 1.1.4 Modèle Simulé *: instance du modèle de simulation paramétrée et initialisée.*

Le *modèle de simulation* décrit le système complexe de manière générale, le modèle simulé vise lui à modéliser précisément un cas particulier du monde réel. Dans le cadre de la simulation orientée agent, le modèle de simulation va définir les comportements et les structures de données associées aux entités composant le système complexe ainsi que la description de l'environnement dans lequel il évolue (notamment les objets qu'ils manipulent). Dans le modèle simulé, les entités sont instanciées : les agents sont créés et leurs structures de données sont initialisées, de même pour l'environnement : le modèle simulé peut être exécuté.

Définition 1.1.5 Simulation *: évolution du modèle simulé suivant un temps virtuel.*

La simulation consiste à faire évoluer les entités décrites dans le modèle simulé suivant un temps virtuel. Ce temps virtuel n'est pas forcément explicité : par exemple, dans [Holcombe *et al.*, 2006] les agents interagissent entre eux au fil de leur exécution sans imposer de règles temporelles particulières. Ce type de simulation n'est pas adapté aux systèmes où interviennent plusieurs dynamiques couplées aux caractéristiques temporelles différentes. Pour cette raison, nous nous plaçons dans le cadre de simulations où le temps simulé est défini de manière explicite : les comportements des agents et leurs interactions se font en respect de règles temporelles définies dans le modèle de simulation.

Devant les résultats prometteurs affichés par l'approche agent pour la simulation de systèmes complexes, les simulations orientées agent intègrent de plus en plus d'agents, développant de nombreuses interactions afin d'affiner le modèle de simulation ou d'étendre la taille du système complexe réel considéré. On parle ainsi de simulation large échelle (ou grande échelle). Nous adoptons la définition de [Kubera *et al.*, 2008].

Définition 1.1.6 Simulation large échelle *: « simulation dans laquelle l'environnement contient un grand nombre d'agents (on parle alors de large échelle en termes de calculs) ou une simulation pour laquelle le nombre de familles d'agents et/ou le nombre de comportements par famille d'agents est important (on parle alors de large échelle en terme de représentation des connaissances ».*

A l'aide des définitions précédentes, nous pouvons alors préciser la notion de processus de simulation.

Définition 1.1.7 Processus de simulation *: suite d'opérations à réaliser pour obtenir une simulation d'une partie du monde réel.*

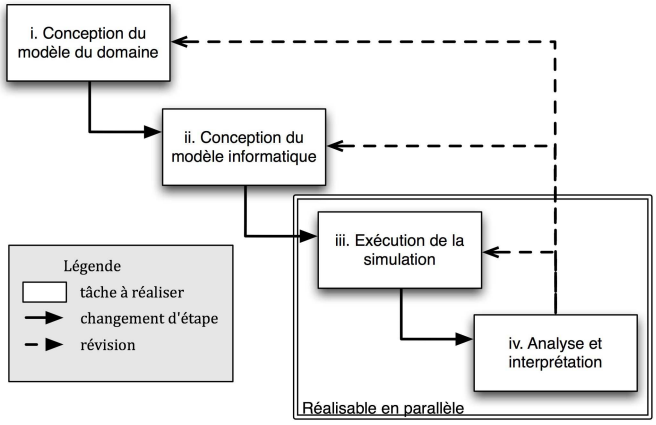


FIG. 1.1 – Le processus de simulation en quatre étapes.

La description du processus de simulation n'est pas unique : selon les auteurs ce processus peut comporter différentes étapes ([Drogoul *et al.*, 2003], [Ramat, 2006]). La différence se trouve en fait plus sur la granularité de la description que sur les opérations à réaliser en elles-mêmes. Ainsi, partageant la vision de [Ralambondrainy, 2009], nous considérons quatre étapes principales (voir figure 1.1) :

i Conception du modèle du domaine : délimitation de la partie du monde réel et modélisation de celui-ci sous la forme d'un modèle du domaine que l'on souhaite étudier.

ii Conception du modèle informatique : transposition du modèle du domaine sous une forme exécutable. Les plates-formes de simulation proposent notamment une approche facilitant la réalisation de tels modèles.

iii Exécution de la simulation : déploiement et exécution du modèle simulé. Les outils d'observation sont notamment configurés et sollicités pour extraire des données de la simulation.

iv Analyse et interprétation des résultats produits par le modèle simulé : comparaison avec les données réelles et extraction de connaissances.

Le modèle du domaine est conçu par les thématiciens : ce sont les experts du domaine qui posent une question à laquelle la simulation tente de répondre. La conception du modèle informatique est elle à la charge du modélisateur informatique. Il est courant que thématiciens et modélisateurs informatique collaborent pour passer directement du monde réel au modèle informatique.

Le processus de simulation est itératif : les connaissances obtenues (ou au contraire les données manquantes) au cours de l'exécution du modèle simulé amènent les concepteurs à améliorer la simulation, c'est-à-dire modifier le modèle du domaine, le modèle informatique ou le modèle simulé. Cette mise à jour permet de raffiner la simulation et obtenir de nouvelles connaissances. Le processus de simulation peut prendre la forme d'un processus en spirale comme le décrit [Courdier *et al.*, 1998].

1.2 Travaux et réalisations de l'équipe SMART

Les travaux de recherche présentés dans ce mémoire ont été réalisés au sein de l'équipe SMART[5] du Laboratoire d'Informatique et de Mathématiques LIM-IREMIA[6] (EA2525) de l'Université de la Réunion.

Notre équipe rapproche deux problématiques informatiques liées autour de la notion de communication en réseau en considérant d'une part la vue système technologique de télécommunication s'attachant aux services sur les réseaux et d'autre part la vue applicative au travers d'outils de simulations multi-agents.

Au niveau des systèmes technologiques, l'équipe s'intéresse au cas du réseau Internet et des réseaux sans fil. Dans le cas d'Internet, celui-ci offre un service de transfert *best effort*. Des pertes dues aux encombrements peuvent venir perturber le service. Dans ce genre de situation, quelle que soit la source, celle-ci doit répondre à la congestion par une diminution de son débit d'émission. De nos jours avec la diversité des supports de transmission, toutes les pertes ne sont pas dues à la congestion. La problématique du thème porte sur l'identification fiable des congestions dans un flot.

Au niveau application, l'équipe s'intéresse à la problématique d'ingénierie de conception de système de simulation par approche multi-agents avec une démarche d'intégration de cette technologie comme réponse à des besoins émanant de partenaires scientifiques. La simulation multi-agents permet de construire de véritables micro-mondes artificiels dont on peut contrôler tous les paramètres (quantitatifs ou qualitatifs) et cela à tous les niveaux du système, qu'il s'agisse de l'entité, du groupe, de la société ou au niveau des effets externes sur l'environnement.

Notre activité dans le domaine du multi-agents est structurée selon 3 axes :

[5] Systèmes Multi-Agents et Réseaux de Télécommunication
[6] Institut de REcherche en Mathématiques et Informatique Appliquée, ancien nom du laboratoire

- Développement d'une plate-forme multi-agents dédiée à la simulation de systèmes complexes naturels et sociaux GEAMAS (version NG actuellement) : [Soulié *et al.*, 1998], [Payet *et al.*, 2006a].
- Réalisation d'expérimentations et développement d'applications « partenaires » : Biomas [Courdier *et al.*, 2002], DOMINO-SMAT [David *et al.*, 2007].
- Proposition de démarches méthodologiques en ingénierie de conception de SMA pour la simulation : [Marcenac and Giroux, 1998], [Vally and Courdier, 1999], [Payet *et al.*, 2006b].

Les travaux actuels de l'équipe portent sur la simulation de systèmes large échelle : notre objectif est de faire de GEAMAS-NG une plate-forme de référence dédiée à la simulation large échelle . Plus particulièrement, les recherches de Ralambondrainy portent sur l'observation dans les simulations large échelle [Ralambondrainy, 2009] et David se focalise sur la place de l'émergence dans la modélisation et la simulation orientée agent [David and Courdier, 2009]. Ainsi, la plate-forme GEAMAS-NG est une base d'expérimentation à double titre : elle est utilisée pour réaliser des outils d'aide à la décision et nous l'utilisons pour confronter nos propositions aux utilisateurs. Elle dispose d'une architecture logicielle modulaire qui permet l'activation/-désactivation de certains modules en fonction du type de simulation que l'utilisateur souhaite lancer.

1.3 La distribution sur les plates-formes de SOA courantes

Nous nous intéressons ici aux aspects distribution des plates-formes de SOA. Plusieurs équipes de recherche dans les SMA ont relevé l'intérêt de la simulation distribuée et ont fait des propositions techniques en ce sens. Nous faisons un état de l'art de la distribution dans les plates-formes de SOA qui s'imposent dans notre domaine d'étude.

1.3.1 CORMAS

Présentation de la plate-forme

CORMAS (*COmmon Resources Multi-Agent System*) se définit comme un environnement de simulation s'appuyant sur les systèmes multi-agents et implémenté en *Smalltalk*, à l'aide de la librairie VisualWorks [Bousquet *et al.*, 1998]. Elle est gratuite mais protégée par les droits d'auteur. Il est possible de modifier la plate-forme, mais les modifications ne doivent pas être diffusées. Développée au CIRAD[7] dans le cadre de recherches scientifiques sur l'environnement et le développement, elle est téléchargeable sur le site *http ://cormas.cirad.fr*.

CORMAS est utilisée pour mieux comprendre les interactions complexes entre les dynamiques sociales et naturelles dans la gestion des ressources renouvelables. Elle fournit une base multi-

[7]Centre de coopération internationale en recherche agronomique pour le développement

agents qui est utilisée pour simuler les interactions entre un groupe d'agents et un environnement partagé contenant des ressources naturelles. L'objectif principal de CORMAS est de fournir des outils pour aider les utilisateurs à développer de nouvelles façons de penser plutôt que de réaliser des outils de prédictions. Elle est notamment utilisée pour la réalisation de jeux de rôles afin de faire connaître différents points de vues aux acteurs du système considéré. La conception d'une application CORMAS repose sur trois phases : la définition des entités et interactions, le contrôle et l'ordonnancement du modèle et la définition des outils d'observation. La plate-forme offre un mode de simulation à évènements discrets : à chaque pas de simulation, la plate-forme déclenche le comportement de chacun de ces agents.

Distribution de la simulation

Dans CORMAS, tous les agents de la simulation sont exécutés sur la même plate-forme. Elle peut cependant être utilisée dans un cadre distribué. En effet, un des objectifs principaux de CORMAS étant de servir d'outil pour la réalisation de jeux de rôles collaboratifs, Guyot a développé une bibliothèque en JAVA appelée *Simulación* permettant la réalisation d'interfaces déportées pour la simulation participative sur CORMAS [Guyot and Honiden, 2006]. Simulación a la charge de gérer toutes les problématiques liées à la distribution pendant la simulation, elle gère aussi la localisation de l'interface.

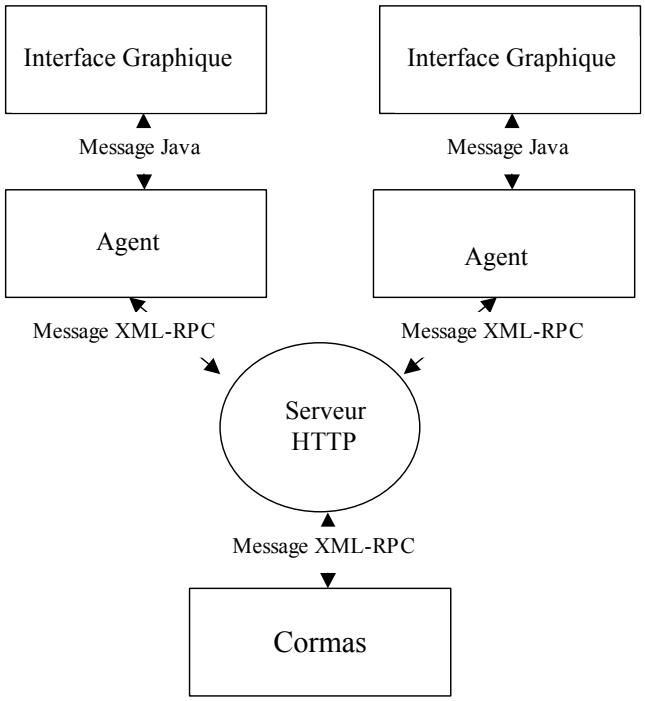


FIG. 1.2 – Architecture d'une simulation CORMAS avec Simulación, extrait de [Taillandier and Drogoul, 2005].

Le système distribué fonctionne en mode *client-serveur*. Chaque participant au jeu de rôle

ne dispose que d'une interface Simulación sur leur poste. Via cette interface, il prend le contrôle d'un agent de la simulation : Simulación se charge de transmettre les commandes de l'utilisateur au noyau de CORMAS et affiche ensuite à l'utilisateur l'évolution de la simulation qui en résulte (voir figure 1.2). La simulation reste donc centralisée : un seul ordinateur exécute véritablement la plate-forme CORMAS qui gère l'intégralité des agents. La bibliothèque Simulación reste cependant très intéressante : elle permet de contrôler et d'observer à distance la simulation en fournissant à l'utilisateur une interface entièrement personnalisable.

1.3.2 MASON

Présentation de la plate-forme

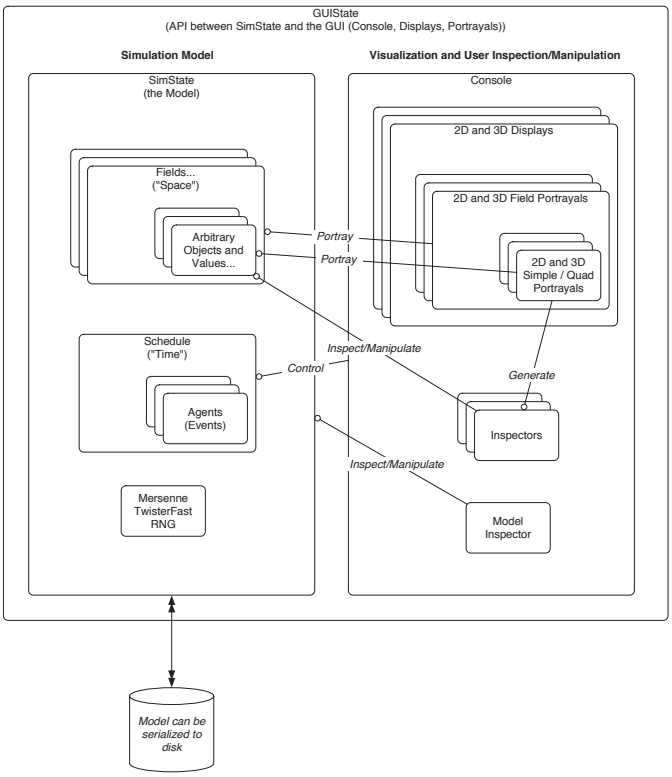


FIG. 1.3 – Architecture de MASON, issue de la documentation en ligne.

MASON (*Multi-Agent Simulator Of Neighborhoods... or Networks ... or something*) est une plate-forme de simulation orientée agent développée en Java [Luke *et al.*, 2005]. Elle est gratuite et *open-source* et est née des travaux du George Mason University's Computer Science Department et du George Mason University Center for Social Complexity. La plate-forme est disponible à l'adresse *http ://www.cs.gmu.edu/ eclab/projects/mason/.*

Conçue pour être performante, facilement extensible et offrant un mode de simulation à évènements discrets, elle fournit les bases nécessaires à la réalisation d'un large spectre de simulations

en se spécialisant toutefois sur les simulations impliquant un très grand nombre d'agents (plusieurs millions). Mason est constituée de deux couches (layers) clairement séparées : le modèle de simulation, et les outils de visualisation 2D et 3D (voir figure 1.3). Un système de checkpoint sauvegarde l'état du modèle de manière périodique, de manière à restaurer l'état du modèle ou à le visualiser. Le modèle de simulation intègre un ordonnanceur d'événements discrets qui gère l'exécution temporelle des agents : la simulation avance par pas de simulation *step*, un pas pouvant contenir des sous-pas de simulation.

Distribution de la simulation

Dans [Luke *et al.*, 2005], les auteurs soulignent que MASON n'intègre aucune capacité réseau. Leur choix est intentionnel et vient du fait qu'une architecture où la simulation serait exécutée en parallèle sur plusieurs processeurs en réseau est trop différente du cas mono-poste mono-processeur. Leur position a toutefois évoluée au cours du temps.

Dans sa version 13, MASON n'offre toujours pas de connexion réseau, néanmoins elle met à disposition un certain nombre de classes Java facilitant le développement de simulation incluant des processus parallèles. La simulation fonctionnant sur la base de pas de temps simulé, un kit de développement fourni les classes suivantes pour l'exécution parallèle :

- *ParallelSubstep*. Cette classe permet de créer des sous pas de simulations qui seront exécutés en parallèle. Lorsque le pas de simulation (*step*) va être exécuté, tous les *steppables* (sous-pas de simulation) qu'il contient seront lancés dans des *threads* en parallèle. Quand tous les *threads* auront terminé leur sous-pas d'exécution, la plate-forme va supprimer ces *threads* et le pas de simulation dans lequel ils étaint contenus va se terminer.
- *AsynchronousSteppable*. Cette classe est utilisée pour lancer un *thread* qui s'exécute indépendamment de la simulation. L'ordonnanceur de MASON n'attend pas la fin d'exécution de ce *thread* pour faire progresser la simulation.
- *Stopper*. Cette classe permet de stopper un *AsynchronousSteppable*. Etant *steppable*, elle peut être insérée dans un pas de simulation pour forcer l'arrêt d'un *AsynchronousSteppable* à un instant précis de la simulation.

Les auteurs soulignent que l'utilisation de ces classes nécessite des précautions dans la programmation, notamment au niveau des accès concurrents aux services de la plate-forme (le générateur de nombres aléatoires par exemple). Ce problème d'accès concurrent peut même survenir dans l'environnement dans la mesure où la simulation repose sur un modèle perception/action et non perception/influence (le modèle le plus communément admis et décrit dans [Ferber, 1995]) : lorsqu'un agent s'exécute il modifie directement l'environnement ; les agents étant exécutés séquentiellement, le premier agent exécuté est donc prioritaire sur les autres. En cas d'exécution parallèle deux agents peuvent donc modifier simultanément l'environnement et provoquer un conflit. L'identification et la résolution de ce genre de conflit n'est pas nativement pris en charge

par la plate-forme et reste à la charge du modélisateur.

1.3.3 SWARM

Présentation de la plate-forme

Créée en 1996, *SWARM* est une plate-forme de simulation de systèmes complexes adaptatifs [Minar *et al.*, 1996]. L'unité de base de la plate-forme SWARM est le *swarm*, une collection d'agents qui exécutent des actions planifiées. SWARM propose de modéliser les systèmes complexes sous la forme de hiérarchisations où un agent peut être constitué de *swarms* contenant plusieurs agents : la plate-forme elle-même est modélisée sous forme de *swarm* où existe un *swarm* contenant le modèle simulé, et un autre contenant les outils d'observation (voir figure 1.4). SWARM fournit les bibliothèques de composants réutilisables pour la création de modèle, l'observation et le pilotage d'expérimentation. Développé par le Santa Fe Institute en *Objective C* (avec des adaptateurs pour *Java*), elle est distribuée en licence GNU-GPL à l'adresse *http ://www.swarm.org*.

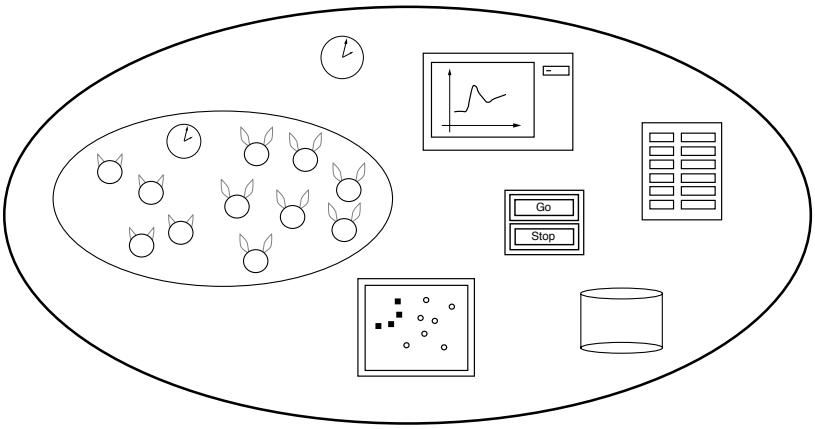


FIG. 1.4 – Architecture de SWARM, extrait de [Minar *et al.*, 1996].

La réalisation d'une application SWARM repose sur la spécification de trois types d'éléments représentés sur la figure 1.4 :

- les *swarmobjects* (les ellipses et les agents de la figure). Ils modélisent les entités de la simulation. Un *swarmobject* pourra être encapsulé dans un *swarm* (qui est lui-même un *swarmobject*) pour créer les hierarchies préconisées dans l'approche de la plate-forme.
- les *activities* (les horloges de la figure). Ils sont au cœur même des mécanismes de simulation dans la mesure où il s'agit d'ordonnanceurs. Chaque *swarm* dispose de son propre composant *activity* auprès duquel chaque agent du *swarm* enregistre ses actions et les *timestamps* associés. Les *activity* transmettent ensuite les contraintes temporelles à leur swarm de plus haut niveau jusqu'à obtenir l'ordonnancement complet de la simulation.

– les *simtools* (les objets restant de la figuree 1.4). Ce sont les outils utilisés pour observer et contrôler la simulation. Ils offrent notamment la possibilité de lancer la simulation dans deux modes : un mode graphique où l'utilisateur peut interagir avec la simulation durant son exécution et un mode console pour lancer des collectes de données sur plusieurs simulations.

Distribution de la simulation

Dans [Minar *et al.*, 1996], les auteurs avancent le fait que les concepts de leur plate-forme permettent un fonctionnement sur plusieurs processeurs en parallèle. En effet, la hiérarchisation des *swarms*, et notamment le fait que chacun d'entre eux possède son propre ordonnanceur, nous semble adapté à un fonctionnement distribué.

Il n'y a cependant aujourd'hui aucun outil fourni avec SWARM permettant de réaliser une simulation distribuée. Un utilisateur souhaitant réaliser une simulation distribuée doit alors développer lui même les interconnexions réseau nécessaires, gérer les échanges entre *activities*, l'éventuelle mise en commun de certains *swarmobjects*. Si les concepts de la plate-forme ont été pensés en vue d'une exécution parallèle distribuée, il manque les services et composants permettant d'opérationnaliser ce mode d'exécution.

1.3.4 RePAST Symphony

Présentation de la plate-forme

RePAST (REcursive Porous Agent Simulation Toolkit) Symphony est une plate-forme de modélisation et de simulation orientée agent développée en Java par l'Argonne National Laboratory (USA), gratuite et *open-source* [Tatara *et al.*, 2006]. Il s'agit d'une version avancée de la plate-forme RePAST originale décrite dans [Collier, 2003] qui elle même existe en trois version : *RePAST for Java*, *RePAST for Microsoft .NET framework* et *RePAST for Python Scripting* [North *et al.*, 2006]. Elle est disponible sur *http ://repast.sourceforge.net/*.

RePAST vise trois objectifs principaux : fournir les composants génériques pour la simulation multi-agents, permettre l'extensibilité de la plate-forme et fournir des performances correctes. RePAST Symphony ajoute à cela une séparation stricte entre modèles, stockage des données et visualisation. L'exécution de la simulation est basée sur des évènements déposés par les agents. Ils définissent des *ticks*, instants du temps simulé où ils mèneront des actions, ils y associent des préconditions, une action et un état final.

Distribution de la simulation

RePAST Symphony ne possède pas nativement d'outils pour l'exécution distribuée de la simulation. Cependant, dans la version courante 1.2 de la plate-forme, il est possible de lancer une simulation sur un réseau de plate-forme en utilisant un middleware appelé *Terracotta*[8]. Ce middleware offre un fonctionnement distribué basé sur le partage de mémoire entre *Machines Virtuelles Java* (JVM) : il permet d'interconnecter différentes JVM en partageant leurs mémoires pour ainsi créer une JVM virtuelle dont l'exécution s'appuie sur le réseau de JVM créé et donc sur différents ordinateurs.

L'intérêt de cette solution réside dans le fait que la plate-forme s'affranchit de toute considération sur la simulation distribuée. Dans la mesure où la distribution est totalement transparente pour la plate-forme de simulation, il n'y a aucune adaptation à faire des concepts de la plate-forme ou des modèles de simulation pour passer d'une exécution mono-poste mono-processeur à une exécution distribuée. En contrepartie, cela signifie que la distribution est faite sans aucune considération sur la plate-forme : aucune optimisation n'est faite sur les différents composants de la plate-forme (notamment sur le moteur d'exécution). De même, il n'est pas possible de paramétrer la distribution des agents sur le réseau d'ordinateurs : la distribution effectuée par Terracotta reste opaque et ne tient pas compte *a priori* des relations entre agents.

1.3.5 JEDI

Présentation de la plate-forme

JEDI (Java Environment for the Design of agent Interactions) est une plate-forme de simulation orientée agent écrite en Java, implémentant l'approche IODA (Interaction-Oriented Design of Agent simulations) et distribuée en *open-source* sur le site *http ://www2.lifl.fr/SMAC/projects/ioda/jedi/* [Kubera *et al.*, 2008]. IODA place les interactions au centre de la création de la simulation : un agent sera défini par quelques variables internes modifiables lors d'interactions mais surtout par les interactions qu'il pourra mener avec chaque autre type d'agents et l'environnement. Ainsi, la création d'agent passe par la spécification d'une matrice d'interactions (voir figure 1.5). La force de JEDI réside dans cette matrice : il suffit de la modifier pour modifier le comportement des agents ou pour ajouter des interactions avec un autre type d'agents.

Distribution de la simulation

Nous avons fait le choix d'aborder cette plate-forme dans cette partie car, outre son approche originale, les concepteurs nous ont fait part de leur intérêt et de leur vision de la distribution de

[8] *http ://www.terracotta.org/*

source \ target	∅	Grass	Sheep	Goat	Wolf
Grass	+(Grow;0)				
Animal	+(Die;3) +(Move;0)				
Herbivore		+(Eat;2;1;0)			
Sheep:Animal,Herbivore			+(Breed;1;1;1)		
Goat:Animal,Herbivore				+(Breed;1;1;1)	
Wolf:Animal	*((Die;3),Die,4)		+(Eat;2;1;0)	+(Eat;3;1;0)	+(Breed;1;1;1)

FIG. 1.5 – Matrice d'Interactions d'un modèle proies/prédateurs à 4 entités, extrait de [Kubera *et al.*, 2008].

la plate-forme JEDI (un sujet de thèse de doctorat sur ce thème avait notamment été proposé en 2009).

Dans sa version 1.0.2, JEDI ne permet de lancer de simulations qu'en solo, c'est-à-dire en mono-poste mono-processeur. Néanmoins, l'approche IODA ne pose aucun problème conceptuel dans le cadre d'un fonctionnement distribué. Qui plus est, même si à l'étape de conception la matrice d'interactions apparaît à l'utilisateur dans sa globalité, elle est ensuite découpée et les actions que chaque agent peut accomplir ne sont connues que de lui seul : il n'y a plus qu'à la distribuer avec les agents sur l'infrastructure d'exécution.

Toutefois le fonctionnement de l'ordonnanceur de JEDI obscurcit cette vision distribuée. En effet, l'ordonnanceur actuel exécute séquentiellement les agents. Les actions d'un agent modifient immédiatement l'environnement ou les agents avec lesquels il interagit. Dans un cadre distribué, plusieurs agents s'exécutent en même temps (dans le temps réel) et peuvent avoir des actions aboutissant à une situation conflictuelle : il n'y a actuellement pas de méthode dans JEDI pour gérer ces conflits. Si l'ordonnancement de la simulation reste à revoir dans un cadre distribué, les concepteurs soulignent un autre problème : la distribution de l'environnement.

1.3.6 MadKit

Présentation de la plate-forme

MadKit (Multi-Agent Development Kit) est une plate-forme générique de conception et d'exécution de systèmes multi-agents [Gutknecht and Ferber, 2000]. Bien que n'étant pas dédiée à la simulation, elle est utilisée dans des applications variées et est reconnue dans la communauté Multi-Agents. MadKit est développée en Java au sein du LIRMM[9] et est distribuée, sous licence LGPL pour son noyau et GPL pour les autres composants, sur le site *http ://www.madkit.org/*.

Le modèle AGR (Agent, Groupe, Rôle) [Gutknecht, 2001] est au centre de la plate-forme et sert aussi bien d'outil de modélisation et de conception pour les développeurs de SMA que de principe d'architecture de la plate-forme elle-même. En effet, les services de la plate-forme sont réalisés par des agents spécialisés afin d'obtenir un maximum de flexibilité. Le noyau de MadKit

[9]Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier

est donc particulièrement léger et n'assure que trois services : la Gestion des Groupes et Rôles, le Moteur Synchrone et les échanges de Messages Locaux (voir figure 1.6).

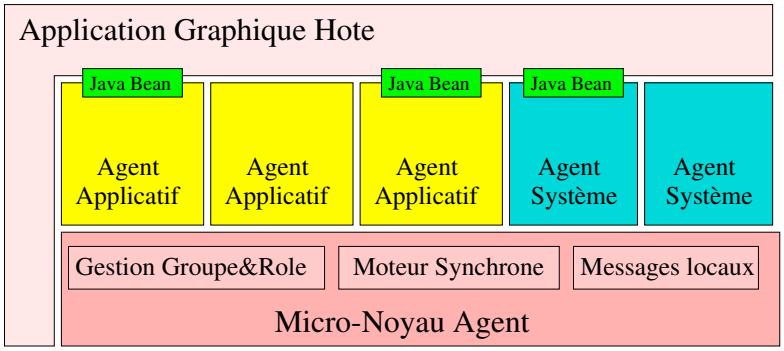


FIG. 1.6 – Architecture de MadKit, extrait de [Gutknecht and Ferber, 2000].

Dans le cadre d'une application agent, MadKit associe un *thread* d'exécution à chaque agent. Ce mode d'exécution induirait de nombreux *threads* pour une SOA, entraînant une forte perte de performances voire l'impossibilité de supporter la large échelle. Un mode alternatif permet de créer des agents de MadKit externes à la simulation qui, possèdant leur *thread* d'exécution, vont exécuter séquentiellement les agents de la simulation pour les faire évoluer dans le temps simulé. Cette technique permet de limiter fortement le nombre de *threads* utilisés par la simulation et donc d'en améliorer les performances.

Distribution de la simulation

Le noyau de MadKit étant particulièrement léger, il n'intègre pas nativement d'éléments utiles à la distribution. Cependant, les concepteurs de la plate-forme ont mis en place des mécanismes appelés *kernel hooks* permettant d'étendre les capacités du noyau. On distingue :

- les *monitor hooks*. Basés sur un mécanisme de *Publish/Subscribe*, un mécanisme de diffusion de messages à tout agent abonné à la liste de diffusion [Eugster *et al.*, 2003], ils permettent à des agents membres du groupe système et abonnés à la liste de recevoir des informations sur l'activité et les évènements reçus par le noyau.
- les *interceptor hooks*. Un seul agent peut s'inscrire par *interceptor hook*. Ce faisant, il désactive la méthode du noyau associée à l'*interceptor hook* et devient lui même responsable du service désactivé. Cette technique permet d'étendre simplement les fonctionnalités du noyau notamment dans le cadre de la distribution.

A l'aide de ces mécanismes, les concepteurs de la plate-forme livrent avec celle-ci un agent appelé *Communicator* et permettant d'interconnecter plusieurs noyaux MadKit entre eux. Chaque plate-forme MadKit devant participer au réseau exécute un agent *Communicator*, une fois cet agent connecté à l'agent *Communicator* d'une autre plate-forme, il récupère les informations de toutes les plates-formes connectées afin d'éviter un fonctionnement centralisé. Les groupes de

chaque plate-forme peuvent alors être partagés ce qui permet d'établir des communications entre agents de plates-formes distantes. Les communications se font en mode *Point à Point*.

Dans sa version 4, MadKit intègre un nouvel outil de communication réseau inter plates-formes plus complet appelé *NetComm*. Cet outil est constitué de plusieurs agents pour améliorer la flexibilité. Il permet notamment d'opter pour différents protocoles de communication et peut utiliser le broadcast pour détecter les plates-formes à interconnecter.

L'outil de communication réseau fournit par MadKit peut être utilisé comme support pour la simulation distribuée interactive. Dans [Michel *et al.*, 2002], MadKit est employé pour la simulation distribuée interactive. Deux approches sont considérées :

- Approche client/serveur. Une plate-forme MadKit fait office de serveur : elle exécute à elle seule l'intégralité de la simulation et possède des agents transmettant et recevant des informations des plates-formes clientes. Cette approche est similaire à ce qu'offre Simulación à CORMAS.
- Approche *Dupliqua*. Toutes les plates-formes partagent le modèle simulé, seules les interactions des utilisateurs sont échangées pour garder la cohérence de simulation sur l'infrastructure d'exécution. Une plate-forme est prise pour référence pour régler les conflits lors des synchronisations.

Communicator et *NetComm* semblent adaptés à la réalisation de SMA distribués. Ils supportent de plus la simulation distribuée pour la réalisation de jeux de rôles. Dans le cadre d'utilisation proposé par [Michel *et al.*, 2002], chaque plate-forme simule les mêmes agents. La simulation peut donc être légèrement différente d'une plate-forme à une autre. Selon les usages de la SOA, ce biais n'est pas toujours acceptable. Ainsi, le cadre le plus large de la simulation distribuée nécessite la création d'un agent ordonnanceur gérant l'évolution global de la simulation sur l'infrastructure d'exécution. Cet agent doit interagir avec les agents ordonnanceurs mais aussi gérer les échanges que peuvent avoir les agents avec l'environnement (transmission éventuelle des requêtes de perception ou d'influence sur le réseau) et pouvoir résoudre ou éviter les éventuels conflits de concurrence entre agents.

1.3.7 Synthèse

Nous n'avons retenu dans notre analyse que les plates-formes possédant des bases pour une exécution distribuée. Il existe d'autres plates-formes de simulation orientée agent reconnues dans la communauté, mais que nous n'avons pas présenté ici car elles n'intègrent pas encore d'outils pour la distribution de SOA. Par exemple, la plate-forme *NetLogo* [Tisue and Wilensky, 2004], réputée pour sa simplicité et son accessibilité, ne fonctionne pas encore en distribuée. Toutefois, dans la documentation de sa version 4, les auteurs indiquent que les aspects distribués feront partie des prochaines fonctionnalités.

Nous soulignons aussi les travaux réalisés autour de la plate-forme *MIMOSA* [Müller, 2004].

Cette plate-forme permet de spécifier le méta-modèle utilisé et donc d'implémenter des modèles variés. Même si elle ne supporte pas encore une exécution distribuée, le découplage des composants de la plate-forme ainsi que la possibilité de formaliser les concepts à la base de la simulation, lui confère la capacité d'évoluer vers une version permettant la distribution des simulations qui auront été réalisées.

Cette analyse nous a permis de relever un certain paradoxe. Alors même que les SMA sont considérés comme une branche de l'IAD, il n'existe que peu d'outils intégrés aux plates-formes permettant de distribuer la simulation orientée agent de manière efficace et respectueuse du paradigme agent.

Nous constatons toutefois un engouement naissant pour l'exécution distribuée de SOA. Divers travaux ont été entrepris ou sont en cours (JEDI, NetLogo), même si aucun d'entre eux n'a pour l'instant abouti à une solution aisément déployable et adaptée à la SOA. Nous avons pu relever différentes approches : réalisation de simples interfaces distantes (CORMAS), éléments de génie logiciel pour l'exécution parallèle (MASON, SWARM), connexion réseau avec support de communication (MadKit) ou virtualisation de l'architecture distribuée (RePAST Symphony).

Partant du fait que toutes ces approches n'ont pour l'instant abouti qu'à des solutions incomplètes, nous déduisons que la simulation orientée agent distribuée nécessite :

- un méta-modèle respectueux des concepts agents. La plate-forme doit notamment garantir l'autonomie des agents, découpler les outils d'observation de la simulation et délimiter correctement agents et environnement.
- une architecture réseau robuste et simple à utiliser. A la manière de MadKit où des agents assurent la connexion entre toutes les plates-formes et gèrent l'échange de messages sur le réseau, l'accès au réseau dans la plate-forme doit pouvoir se faire aisément dans la mesure où la SOA distribuée implique de nombreux échanges de message et que différents services de la plate-forme seront eux aussi amenés à utiliser le réseau.
- des services facilitant le déploiement et l'exécution de la simulation. L'objectif premier de ces services est de coordonner les différentes unités de simulation mises à disposition par l'architecture réseau afin de supporter l'exécution distribuée. Il s'agit dans un deuxième temps de développer des services spécifiques permettant de tirer parti des atouts d'une architecture d'exécution distribuée.

Les plates-formes présentées ici devraient évoluer rapidement en terme d'outils pour la distribution de SOA dès lors que leurs concepteurs identifieront plus précisément les problématiques de recherche liées à la distribution et veilleront à considérer la distribution de SOA à tous les niveaux d'abstraction de leur plate-forme. Nous espérons que ce travail pourra ainsi être exploité par l'ensemble des concepteurs de plates-formes de SOA pour mieux supporter la distribution de simulations.

1.4 Infrastructures d'exécution distribuée

Nous avons vu que les outils de distribution des plates-formes de SOA les plus classiques sont encore limités. Le domaine de l'informatique distribuée est quant à lui très vaste. La généralisation des réseaux et des équipements informatiques ont notamment engendré une recherche très active dans ce domaine. Nous nous intéressons à plusieurs architectures de distribution réputées et analysons leur potentiel en tant que support pour la distribution de simulations orientés agent.

1.4.1 Grid Computing

Présentation

Ces dix dernières années ont vu l'essor d'ambitieux programmes de recherche nécessitant d'importantes ressources informatiques. Ces programmes ont souvent conduit à la constitution de solutions *ad hoc*, avec par exemple SETI@HOME [Anderson *et al.*, 2002] ou Folding@Home [Larson *et al.*, 2009]. Dans l'optique de généraliser ce genre d'approches, les chercheurs en informatique se sont concentrés sur la réalisation d'infrastructure permettant la recherche collaborative (qualifiée d'*e-Science* [Hey and Trefethen, 2002]) ou d'autres usages plus généraux [Grimshaw and Wolf, 1996]. Ce nouveau domaine de recherche est appelé *Grid Computing*.

Plusieurs définitions de la *Grille* (en anglais *Grid*) existent (voir [Buyya and Venugopal, 2005], [Foster, 2002], [Gentzsch, 2002]) selon qu'elles mettent en avant l'infrastructure elle-même ou ses services. Nous reprenons une des définitions proposées dans [Buyya and Venugopal, 2005] :

Définition 1.4.1 Grille *: système parallèle et distribué qui gère le partage, la sélection et l'aggrégation dynamiques de ressources « autonomes » géographiquement distribuées pour optimiser l'exécution d'une tâche selon la disponibilité, la capacité, les performances, le coût ou la qualité de services des ressources présentes dans le système.*

Il existe deux approches complémentaires dans les travaux sur la *Grille* : l'approche orientée infrastructure, qui se focalise sur la réalisation technique de la *Grille* et l'approche orientée services, qui s'intéresse aux services à mettre en place pour que les utilisateurs puissent profiter de la *Grille*. Ces deux approches s'illustrent dans deux technologies :

– *Globus* [Foster and Kesselman, 1997]. Cette plate-forme a deux objectifs : offrir des mécanismes de bas niveau pouvant être réutilisés pour fournir des services de haut niveau et mettre en place des techniques permettant d'observer et de piloter l'utilisation de ces mécanismes. Elle définit plusieurs modules visant à identifier et agréger les ressources afin de constituer un ordinateur virtuel et rendre son utilisation transparente à l'utilisateur (qui doit s'authentifier dans le système). D'un point de vue technique, elle s'appuie sur

différents protocoles de communication afin de permettre l'agrégation de ressources très diverses.

– *Gridbus* [Buyya and Venugopal, 2004]. Cette plate-forme s'intéresse aux couches supérieures de la *Grille* : les outils de contrôle de l'exécution des tâches, d'équilibrage des accès aux ressources et de gestion des comptes utilisateurs (notamment une possibilité de tarification). Elle repose sur différents composants qui s'articulent autour d'un *Grid Service Broker* auquel l'utilisateur soumet les tâches à effectuer.

Ces deux approches sont complémentaires et devraient amener à une structuration de la Grille en plusieurs couches comme le montre la figure 1.7.

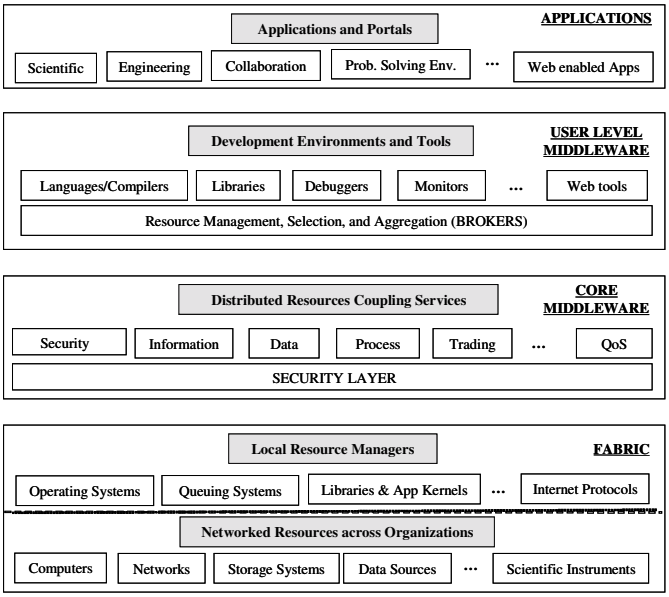


FIG. 1.7 – Architecture en couches de la *Grille*, extrait de [Buyya and Venugopal, 2005].

Grille et Distribution de Simulations Orientées Agent

Le rapprochement entre les travaux sur la *Grille* et les Agents a déjà été analysé. Ian Foster souligne dans [Foster *et al.*, 2004] l'importance de l'approche agent dans la mise en place de cette architecture. La convergence s'illustre par le fait que la *Grille*, constitue « la musculature » et les agents « le cerveau » d'une solution d'exécution parallèle et distribuée globale. En effet, pour l'auteur, la Grille assure une architecture solide dans laquelle les agents peuvent établir des interactions fiables. Les agents visent eux à fournir tous les services permettant de rendre la *Grille* fonctionnelle. Un premier effort d'architecture de *Grille* à base de système multi-agents est ainsi décrit dans [Roy *et al.*, 2005]

Dans [Jonquet *et al.*, 2006], les auteurs s'engagent dans cette voie en proposant d'intégrer le système multi-agents comme un ensemble de services de la *Grille* en définissant un vocabulaire commun aux deux domaines. La Grille fournit alors plusieurs services très basiques et les agents

fournissent eux des services de plus haut niveau. Les agents étant libres d'interagir entre eux, ils ont de plus la possibilité de transformer leurs actions en services pour les mettre directement à la disposition des utilisateurs.

La convergence entre *Grille* et Agents se focalise donc sur la mise en œuvre de la *Grille*. Dans notre cadre d'étude, il s'agirait d'appuyer l'exécution de la SOA sur cette architecture : la *Grille* est souvent présentée comme un ordinateur virtuel sur lequel s'exécute les applications de nombreux utilisateurs. La SOA pourrait bien sûr être exécutée sur une telle architecture mais il faut vérifier quels seraient les avantages de ce support d'exécution.

La *Grille* permet l'exécution distribuée et parallèle de tâches. Dans le cadre de projets de type e-Science, ces tâches se caractérisent le plus souvent par un certain nombre de traitements à appliquer à une grande quantité de données (typiquement des applications du genre SETI@HOME). Ce type d'application est facilement parallélisable et bénéficie réellement des atouts de la *Grille* : les données peuvent être acquises par les outils de mesures intégrés à l'architecture, stockées dans des bases de données distribuées et les traitements sont exécutés sur différents processeurs. L'accès à des outils de mesure ou de stockage de données est un élément intéressant pour la simulation orientée agent mais l'exécution de SOA sur la *Grille* n'est pas conforme à la philosophie de cette dernière.

La simulation orientée agent met en avant les interactions entre agents et avec l'environnement et nécessite l'échange de nombreux messages sur le réseau. Les interactions entre les parties de simulations distribuées sur le réseau limitent fortement l'exécution parallèle.

La *Grille* se charge normalement de distribuer et d'exécuter en parallèle des tâches présentant un couplage faible : elles peuvent nécessiter le séquencement des traitements mais n'imposent pas de synchronisations comme le fait la SOA distribuée. Dans le cas des SOA, la *Grille* exécuterait donc de manière distribuée un seul pas de simulation, puis regrouperait la simulation sur un unique ordinateur pour que ce dernier procède à la résolution des conflits et à la mise à jour de l'environnement (pour plus d'informations sur l'ordonnancement de la simulation, se référer au paragraphe 3.2.3). L'exécution de la SOA est alors découpée en pas de temps, chacun constituant une tâche distribuable et exécutable en parallèle, et en points de synchronisation, tâches centralisant l'ensemble des informations générées par le dernier pas de simulation et chargé de leurs données une cohérence. Ce mécanisme est peu adapté à la SOA puisqu'il implique un redéploiement systématique de la simulation sur la *Grille* à chaque pas de temps.

Nous retenons que l'intégration de matériels hétérogènes offert par la *Grille* étend certes les possibilités de la SOA, mais que la philosophie de cette architecture est plutôt de permettre l'exécution parallèle d'applications traitant de gros volumes de données et ayant un faible couplage entre elles. Ainsi, les mécanismes qu'elle propose supportent la simulation orientée agent distribuée mais ne permet pas réellement de tirer parti de cette distribution.

1.4.2 High Level Architecture

Présentation

High Level Architecture (HLA) est une initiative du département américain de la défense lancée il y a plusieurs années et aujourd'hui adoptée en tant que standard IEEE (IEEE 1516, 2000). Cette spécification vise à uniformiser les outils de simulations utilisés dans les domaines militaires et de la recherche afin de les rendre inter-opérables et de pouvoir réaliser des simulations distribuées [Kuhl *et al.*, 1999]. Dans HLA, un élément de simulation est appelé *federate*. A la manière de Swarm, les *federates* peuvent être regroupés dans des organisations appelées *federations* qui constituent des unités de simulation de plus haut niveau.

HLA définit trois spécifications :

- *Interface Specification*. Cette spécification détaille les interactions entre les *federates* et la *Run-Time Infrastructure* (RTI). La RTI est un *middleware* central dans HLA : il se comporte comme un système d'exploitation pour les *federates*. Il assure la mise en place de tous les services nécessaires à l'exécution, la synchronisation et la communication des *federates*.
- *Object Model Template* (OMT). L'OMT fournit une définition formelle des messages échangés entre *federates*. A partir de cette définition, chaque *federate* construit un *Simulation Object Model* (SOM) qui présente les entités et les interactions dont il a besoin pour fonctionner ou qu'il peut fournir. Ces SOM servent à définir des *Federation Object Model* qui définissent toutes les informations échangeables entre *federations*.
- *HLA Rules*. Ces règles synthétisent les principes clés de la spécification HLA : il existe 5 règles ayant trait aux *federations* et 5 autres ayant trait aux *federates*.

HLA se présente avant tout comme une spécification. Pour cette raison, il n'y a pas une unique plate-forme de simulation HLA et le choix de la *Run-Time Infrastructure* est laissé libre aux concepteurs.

HLA et Distribution de Simulations Orientées Agent

Les différences entre la distribution de simulations orientées agent que nous souhaitons proposer et les travaux sur HLA se situent aussi bien au niveau conceptuel qu'au niveau technique.

Au niveau conceptuel : l'objectif de notre travail de recherche est de réaliser une infrastructure d'exécution dédiée à la SOA distribuée. Les bénéfices attendus sont le support de la large échelle avec des performances optimisées, l'exécution sur un réseau ouvert et extensible, l'observation distribuée et dans un second temps la modélisation et l'analyse collaborative. HLA satisfait à ces critères dans une certaine mesure mais vise avant tout l'interopérabilité de modèles de simulation hétérogènes [Boer *et al.*, 2009]. Ainsi, la spécification se limite à la définition de modèles pour

la structuration de la simulation dans sa globalité et pour la communication entre simulateurs (*federates*). Il va de soit que cette communication entre *federates* génère moins de messages que des agents s'exécutant dans le cadre d'une simulation : HLA ne fournit aucune directive en ce qui concerne l'optimisation des performances en simulation.

Au niveau technique : comme nous l'avons souligné, HLA ne fournit pas de *Run-Time Infrastructure*, RTI qui est à considérer comme le pendant de la plate-forme dans le domaine de la SOA. Cela signifie que HLA suppose qu'il existe une architecture réseau et notamment un ordonnanceur distribué capable de faire avancer le temps dans l'ensemble des *federates* présents. Or, à la section 1.3, nous avons souligné qu'il n'existait pas encore d'architecture réseau implémentée dans les plates-formes de SOA et adaptée à l'activité de simulation agent. Notre objectif est donc bel et bien de travailler à la réalisation d'un « RTI » dédié à la distribution de SOA.

Pour ces raisons, même si HLA est une spécification intéressante à la fois pour sa capacité d'intégration de modèles de simulation hétérogènes et pour sa reconnaissance dans les communautés scientifiques et militaires, nous jugeons préférable de nous focaliser dans un premier temps dans une solution de simulation distribuée dédiée à la SOA. Une telle approche vise à améliorer les capacités des SOA sans pour autant verrouiller toute tentative de mise en conformité des SOA avec HLA : elle pourrait même aboutir vers la réalisation d'une nouvelle *Run-Time Infrastructure* où les *federates* et autres *federations* seraient représentés par des agents.

1.4.3 Agents mobiles

Présentation

Dans la communauté agent, le réseau a souvent été abordé dans les SMA avec la notion d'Agents Mobiles. Les agents mobiles sont des agents ayant la possibilité de se déplacer d'une plate-forme d'exécution à une autre. Leur migration est une action volontaire de leur part qui rentre le plus souvent dans leur démarche pour atteindre leurs buts. Les agents mobiles constituent un thème de recherche à part entière, notamment en ce qui concerne leurs communications [McCormick *et al.*, 2000], [Padovitz *et al.*, 2003] et surtout les politiques de sécurités à définir pour leurs déplacements et leur intégrité [Jansen, 2000], [Vigna, 1998].

Un des champs d'application des agents mobiles est la recherche de ressources disponibles sur un réseau. Plusieurs travaux ([Montresor *et al.*, 2002] ou [Schaerf *et al.*, 1995]) se basent sur cette capacité des agents mobiles pour l'exécution parallèle de tâches sur un réseau d'ordinateurs. Dans ce genre de systèmes, un agent se voit attribué l'exécution d'une tâche et se charge de trouver la meilleure machine sur laquelle l'exécuter. L'agent peut adopter des stratégies différentes : il peut par exemple cartographier une partie de l'infrastructure d'exécution et retourner le plus souvent sur les machines qui ont été les plus performantes (fonctionnement par apprentissage/exploitation), il peut également communiquer avec d'autres agents mobiles pour

adopter une véritable stratégie de groupe (il s'agit le plus souvent d'un partage d'information plutôt que de l'élaboration d'une stratégie collective avancée).

Agents Mobiles et Distribution de Simulations Orientées Agent

L'emploi d'agents mobiles pour la distribution de SOA serait à considérer de deux manières :

- les agents de la simulation pourraient être eux-mêmes des agents mobiles et donc se déplacer sur l'infrastructure d'exécution et communiquer entre eux.
- les agents mobiles pourraient être chargés chacun de la simulation d'une partie des agents de la simulation. Cet axe permet d'utiliser les travaux menés sur l'emploi d'agents mobiles pour la répartition de tâches.

La première approche consistant à transformer tous les agents de la simulation en agents mobiles pose des problèmes aussi bien conceptuels que techniques.

D'un point de vue conceptuel, les déplacements des agents mobiles sont une conséquence même de leur comportement. Ainsi, les agents auraient alors un comportement propre à la simulation et un autre, indépendant, propre à la stratégie de placement sur l'infrastructure d'exécution : la conception et la maintenance de tels agents est particulièrement délicate et limite fortement les possibilités de portage de simulations distribuées vers des simulations mono-poste mono-processeur.

D'un point de vue technique, nous envisageons la distribution de la SOA pour s'affranchir des problématiques de large échelle. S'il est réaliste d'attendre de l'infrastructure d'exécution le support de dizaines ou centaines de milliers d'agents de simulation complexes et en interaction, il en est tout autre dans le cas d'agents mobiles « intelligents ». En effet, contrairement aux agents de simulation, les agents mobiles sont exécutés dans un *thread*[10] qui leur est propre. Ainsi, il est impensable d'avoir plusieurs centaines d'agents mobiles sur une même plate-forme : ceci limite fortement le nombre d'agents mobiles sur l'infrastructure d'exécution. Il n'est actuellement pas possible de conférer aux agents de la simulation les capacités des agents mobiles.

Dans la deuxième approche, un agent mobile aurait la charge de l'exécution de plusieurs agents de la simulation pour éviter les critiques de la première approche. L'idée est alors d'employer les mécanismes de recherche de contextes d'exécution proposés par la communauté Agents Mobiles. On notera toutefois que ces mécanismes ne fournissent pas forcément la répartition optimale mais une solution considérée satisfaisante. Dans le cas de la SOA distribuée, dans la mesure où l'infrastructure d'exécution est entièrement dédiée à l'activité de simulation, nous souhaiterions opter pour une répartition plus efficace.

[10] Dans la suite de ce mémoire, nous désignons par le terme technique *thread*, les concepts plus génériques des fils d'exécution.

Mais si nous n'avons pas retenu cette deuxième approche c'est avant tout pour des raisons d'optimisation des échanges réseaux. En effet, l'emploi d'agents mobiles pour la SOA distribuée solliciterait le réseau pour :

- les déplacements d'agents mobiles. Cette activité serait coûteuse en débit dans la mesure où les agents mobiles peuvent alors être considérés comme des plates-formes de simulation minimales et embarquent des agents de simulation.
- les échanges de messages entre agents mobiles. Ces échanges servent à la fois à optimiser leurs déplacements d'une plate-forme à une autre (partage d'informations, élaboration de stratégies) et à transmettre toutes les informations nécessaires à la simulation et à sa supervision (synchronisation des agents de simulation sur le réseau, contrôle et observation de la simulation).
- les nombreux échanges de messages entre agents de la simulation.

Cette réflexion sur l'emploi d'agents mobiles nous permet d'envisager le déplacement d'agents de simulation pour l'optimisation des performances en simulation. Cependant, l'emploi d'agents mobiles introduit une complexité supplémentaire dans l'architecture que nous souhaitons réaliser. En effet, considérant qu'à un instant donné, il est possible d'avoir une vue sur l'intégralité de l'infrastructure réseau, il n'est pas nécessaire d'employer des mécanismes aussi complexes que ceux mis en œuvre dans les agents mobiles.

1.5 Conclusion

Nous avons montré dans ce chapitre que la question de la distribution de simulations orientées agent est une problématique actuelle et paradoxalement délicate à gérer : alors que le paradigme agent garantit l'autonomie des agents de simulation, les contraintes liées à l'exécution de simulations (tout particulièrement les contraintes temporelles et d'équité entre agents) obligent à fournir un cadre d'exécution très strict qui est délicat à distribuer.

Conscients des intérêts et enjeux de la distribution pour le développement de SOA, les concepteurs de plates-formes adoptent des comportements inégaux vis-à-vis de cette problématique : certains font appel à des solutions techniques extérieures à leur plate-forme quand d'autres proposent des solutions partielles. S'il n'est pas impossible de réaliser une simulation distribuée, la majeure partie du travail de conception de l'architecture distribuée ou de synchronisation des agents distribués reste à la charge de l'utilisateur.

Il existe bien sûr des travaux que nous n'avons pas abordé ici. Par exemple, Wang propose dans [Wang *et al.*, 2004] d'adapter la SOA pour une utilisation couplée avec HLA. [Scheutz and Schermerhorn, 2006] offre un cadre formel pour la définition de modèles de simulations orientées agent, cette description du modèle permet ensuite de compiler une simulation distribuée. Michel s'intéresse dans [Michel *et al.*, 2002] à une approche de simulation participative distribuée en

interconnectant des plates-formes entre elles et en nommant une plate-forme maître : en cas de conflit sur l'état de la simulation, la plate-forme maître sert de référence.

Toutefois la portée de ces travaux pour la distribution de SOA reste assez limitée : ces travaux visent en général soit l'intégration de la SOA dans une architecture de niveau supérieur (par exemple HLA) au détriment de la réflexion pure sur la SOA distribuée, soit ne s'intéressent qu'à un aspect précis de la SOA distribuée (par exemple, l'interaction entre plusieurs acteurs et la simulation dans le cadre de jeux de rôles) ou encore se placent dans un cadre trop restreint (modèle de simulation trop contraignant).

Ce constat nous a amené à considérer différentes architectures distribuées reconnues et à analyser si la SOA distribuée pourrait s'appuyer sur celles-ci. Nous avons constaté que la simulation pourrait certes s'appuyer sur ces architectures mais qu'elle n'en tirerait pas vraiment parti. Dans le cas de HLA, cette architecture vise avant tout l'intégration de modèles de simulations différents et non une optimisation particulière de la simulation. Dans le cas de la *Grille* ou des Agents Mobiles, ces architectures sont adaptées à l'exécution de tâches purement parallèles. Ces architectures n'offrent aucun outil pour la distribution et l'exécution parallèle de tâches fortement couplées comme peut l'être la simulation orientée agent.

Nous retenons donc de ce chapitre qu'aucune proposition ne fait actuellement loi en ce qui concerne la distribution de SOA. Une des raisons de ce constat vient du fait que peu d'approches considèrent la distribution à la fois sur le plan de l'infrastructure d'exécution, de la modélisation de la simulation (voire du méta-modèle de simulation) et des services à déployer pour superviser la simulation. A la recherche d'une infrastructure d'exécution pour la SOA distribuée, l'analyse d'architectures distribuées variées a fait ressortir le besoin d'une solution avant tout dédiée à cette activité.

Chapitre 2

Cadre conceptuel agent pour la réalisation d'une infrastructure pour la SOA distribuée

Sommaire

2.1 Aborder la SOA distribuée sous l'angle du Cloud Computing 34
2.2 Caractéristiques de la distribution de SOA 35
2.2.1 Une architecture à l'interface de plusieurs dynamiques 35
2.2.2 Qualité de simulation 36
2.2.3 Les exigences de la SOA distribuée 37
2.3 Structuration en couche de l'architecture de distribution 38
2.3.1 Interconnexion de plates-formes 38
2.3.2 Communications réseau 38
2.3.3 Système Multi-Agents réparti pour la supervision de la simulation distribuée 40
2.3.4 Synthèse 41
2.4 Système Multi-Agents réparti pour la Simulation Orientée Agent distribuée 42
2.4.1 Description générale 42
2.4.2 Composition des agents macro de distribution 43
2.4.3 Organisation des agents du SMA de distribution 45
2.4.4 Interactions agents du SMA de distribution 45
2.4.5 Environnement du SMA de distribution 45
2.4.6 Notion de space pour l'implémentation d'agents systèmes dans la plate-forme de SOA 46
2.5 Synthèse 48

Nous avons constaté au chapitre précédent que les solutions de distribution de SOA actuelles sont incomplètes : elles ne traitent pour la plupart que d'un seul niveau d'abstraction (niveau méta-modèle, niveau service ou niveau réseau). Nous proposons dans ce chapitre de définir un ensemble d'outils et de concepts pour architecturer une solution de distribution adaptée à l'activité de simulation orientée agent.

2.1 Aborder la SOA distribuée sous l'angle du Cloud Computing

Nous avons vu au paragraphe 1.3 que les plates-formes de simulation orientée agent actuelles proposent des solutions de distribution très différentes. Cette différence est encore plus marquée lorsque l'on s'intéresse à l'aisance de déploiement de simulations distribuées. Par exemple, dans *MadKit* il est nécessaire de développer un agent chargé de l'ordonnancement de la simulation distribuée alors que dans *RePAST Symphony* il n'y a aucune considération à faire sur la distribution, le problème dans ce dernier cas est que la distribution échappe complètement aux utilisateurs (et même à la plate-forme). Cette approche n'en demeure pas moins intéressante : une fois l'architecture mise en place (le déploiement de *Terracotta* dans le cas de *RePAST Symphony*), les utilisateurs n'ont rien à gérer pour bénéficier de l'exécution distribuée. Le passage de la SOA exécutée en local à la SOA distribuée se fait très simplement ; une telle solution contribue à ce que plus de thématiciens à travailler dans un cadre distribué.

Pour cette raison, nous considérons que l'exécution distribuée de SOA doit être abordée sous l'angle du *Cloud Computing*[11] en terme de déploiement. Le concept du *Cloud Computing* est en effet d'abstraire l'infrastructure d'exécution à l'utilisateur : celui-ci n'a pas besoin de gérer le déploiement de la simulation sur l'infrastructure d'exécution ni même de savoir quelles sont les machines participant à la simulation ; de son point de vue tout se passe comme si la simulation s'exécutait sur sa propre machine. Dans cette approche, le travail de distribution devrait aboutir à la réalisation d'une *plate-forme de SOA virtuelle* que l'utilisateur pourrait utiliser de la même manière qu'il utilise une plate-forme sur son ordinateur. Cette « virtualisation » de l'infrastructure d'exécution est une approche de plus en plus courante. La Grille, à son niveau infrastructure [Foster and Kesselman, 1997], repose notamment sur ce concept en cherchant à constituer un *ordinateur virtuel*.

Le terme de *Fire and Forget* décrit bien le fonctionnement attendu de la plate-forme de SOA virtuelle. Ce terme est utilisé dans le monde militaire pour caractériser un guidage de missiles : une fois le missile lancé, il n'a pas besoin d'un maintien de la désignation pour atteindre sa cible. De la même manière, une fois le fonctionnement distribué activé, il se chargerait de toutes les considérations liées à la distribution (détection de l'infrastructure de simulation, partage

[11] aussi nommé « *nuage informatique* » ou « *informatique dans les nuages* ». Nous préférons ici utiliser le terme anglophone « *Cloud Computing* » qui est le terme d'usage dans le monde scientifique francophone.

d'informations, répartition de la charge, *etc.*) ; l'utilisateur n'ayant alors plus à considérer la distribution dans son activité.

L'approche *Cloud Computing* implique une idée forte : la portabilité des modèles de simulation, c'est-à-dire leur capacité à fonctionner dans différents environnements. En effet, dans la mesure où le thématicien utilise une plate-forme virtuelle, il n'a pas à prendre en compte le contexte d'exécution. Le modèle de simulation n'est pas bruité par des considérations liées à la distribution ; un modèle de simulation créé sur la plate-forme virtuelle peut être exécuté sur une plate-forme mono-poste. Réciproquement, la plate-forme virtuelle supporte l'exécution distribuée de modèles de simulation conçus pour une utilisation mono-poste.

2.2 Caractéristiques de la distribution de SOA

2.2.1 Une architecture à l'interface de plusieurs dynamiques

La distribution est ici abordée comme un moyen de s'affranchir des problématiques opératoires liées à la large échelle dans les SOA. Une autre alternative serait le recours à un super-calculateur, mais cette solution nécessite de la part des utilisateurs de posséder de coûteuses installations dont l'extensibilité est limitée.

Au travers de la distribution de SOA, nous souhaitons donner aux utilisateurs la possibilité d'étendre dynamiquement les ressources disponibles pour la simulation. Ainsi, l'infrastructure d'exécution se compose de plusieurs ordinateurs aux ressources différentes communiquant sur un réseau. Le nombre de machines participant à la simulation peut évoluer au cours du temps (connexion, déconnexion d'ordinateurs) et les performances de chaque ordinateur sont variables (un utilisateur peut lancer d'autres applications sur son ordinateur). Avec un réseau dont le débit peut aussi fluctuer au cours du temps, l'infrastructure d'exécution possède elle-même une dynamique complexe non prédictive. Comme indiqué dans l'introduction au paragraphe 4, nous ne considérerons pas le cas de déconnexions accidentelles d'ordinateurs : une machine devant quitter l'infrastructure a suffisamment de temps pour exécuter les procédures liées à son départ.

D'un point de vue computationnel, la simulation orientée agent possède sa propre dynamique. En effet, le modèle simulé est une représentation du système réel que les thématiciens souhaitent étudier (voir définition 1.1.4). La simulation orientée agent, reproduisant les interactions complexes au sein du système à étudier, intègre ces différentes dynamiques du système. Ces dernières ont un impact sur le modèle puisque le nombre d'agents de simulation peut fortement varier au cours de l'exécution. De même, les interactions entre agents peuvent aussi évoluer, les organisations changer et des phénomènes émergents modifier la topographie de la simulation. Tous ces éléments contribuent à faire de la simulation un système possédant sa propre dynamique.

Une architecture de SOA adaptée à la simulation distribuée et proposant une approche de

type *Cloud Computing* doit donc travailler à l'interface entre la *dynamique d'infrastructure* et la *dynamique de simulation*. Dans une vision « plate-forme virtuelle », elle doit abstraire la dynamique d'infrastructure à la simulation pour lui fournir un contexte d'exécution stable.

2.2.2 Qualité de simulation

Les simulations orientées agent sont aujourd'hui utilisées en production : ayant dépassé le cadre d'outils expérimentaux, les thématiciens et les décideurs se reposent sur les données qu'elles produisent pour analyser un système et prendre des décisions s'y rapportant. La distribution de SOA, telle que nous l'abordons, vise à étendre les capacités de représentation et d'exécution des plates-formes pour servir des projets de grande envergure. L'exécution de la simulation en environnement distribué doit alors fournir des résultats aussi fiables que ceux obtenus par la simulation mono-poste mono-processeur.

Nous notons toutefois que Rao a montré, dans [Rao *et al.*, 1998], qu'un assouplissement de cette fiabilité pouvait conduire à des résultats relativement proches de ceux d'une exécution stricte du modèle simulé. Mais l'exécution de SOA n'est pas seulement exploitée pour le résultat final qu'elle produit. Elle l'est également pour les différents états dans lesquels le système peut se retrouver au cours de la simulation. Et dans le cadre de systèmes complexes, le moindre biais « accidentel » peut induire un impact fort sur l'évolution de la simulation même si le résultat final reste encore cohérent. Ainsi, la reproductibilité des résultats est une caractéristique capitale dans les SOA : hormis les biais induits par les éventuelles valeurs aléatoires définies dans le modèle simulé, la SOA doit, à tout moment de la simulation, conduire aux mêmes états du système.

Le terme de *qualité de la simulation* désigne le fait que celle-ci ne doit pas amener de biais dans les modèles comportementaux. Cela signifie que, considérant une exécution mono-poste mono-processeur et une exécution distribuée d'un même modèle de simulation, un utilisateur ne peut différencier l'exécution locale de l'exécution distribuée autrement qu'en se basant sur des considérations informatiques (performances, activité réseau, *etc.*).

La simulation orientée agent est un domaine de recherche très actif où de nouveaux outils, concepts ou approches sont proposés. L'architecture de distribution doit supporter le développement et l'intégration de ces nouveaux éléments. Qui plus est, offrant un cadre distribué à l'activité de simulation, elle sera elle-même à l'origine de nouvelles contributions scientifiques, notamment des contributions ayant trait au processus de simulation.

La satisfaction de cette contrainte d'enrichissement ne passe pas uniquement par des solutions de type Génie Logiciel. Ces solutions sont certes nécessaires pour garantir l'extensibilité de l'architecture de distribution, mais il faut aussi considérer l'impact des nouvelles fonctionnalités sur les performances globales. Il est bien évidemment impossible de mesurer un tel impact à l'heure actuelle, nous considérons donc que l'architecture doit être la plus « économe » possible en terme de coût d'exécution et d'échanges réseau.

La distribution de la SOA étend les capacités des plates-formes actuelles vis-à-vis de la large échelle mais les mécanismes de distribution contraignent l'exécution de la simulation sur l'infrastructure d'exécution et induise alors un impact sur les performances d'exécution. Il convient donc d'étudier ces mécanismes, afin de minimiser leur impact sur les performances, et d'exploiter au mieux les capacités de chaque machine de l'infrastructure pour offrir les meilleures performances de simulation.

2.2.3 Les exigences de la SOA distribuée

Proposer une solution technique pour l'exécution distribuée de SOA n'est pas suffisant pour encourager les efforts dans ce domaine. Nous prônons une approche de type *Cloud Computing* où l'utilisateur n'a pas à se soucier du contexte d'exécution de la simulation. Mais derrière la simplicité d'utilisation se cache une importante technicité : pour être efficace, l'architecture de distribution doit prendre en compte la dynamique d'infrastructure d'une part et la dynamique de simulation d'autre part.

Cette architecture doit être conçue autour des exigences suivantes :

- *qualité de simulation.* La simulation distribuée doit fournir la même qualité de données que la simulation exécutée en mono-poste. Cela signifie que la distribution n'induit aucun biais dans le déroulement de la simulation.
- *portabilité des modèles.* L'exécution distribuée d'une simulation ne doit pas nécessiter une réécriture du modèle de simulation : un modèle simulé conçu pour une exécution mono-poste doit pouvoir tirer parti de la distribution. Réciproquement, un modèle qui serait conçu sur la plate-forme virtuelle doit pouvoir s'exécuter en mono-poste.
- *flexibilité de l'infrastructure.* L'infrastructure doit supporter l'ajout ou la suppression d'ordinateurs pendant la simulation. Elle doit aussi tenir compte des importantes variations de performances que peuvent rencontrer les machines.
- *extensibilité des fonctionnalités.* La recherche dans les SOA conduit à de nombreuses innovations qu'il faut pouvoir intégrer à la SOA distribuée. De plus, le contexte distribué devrait permettre d'apporter de nouveaux outils, méthodologies et usages qu'il faudra intégrer au fur et à mesure.
- *optimalité de l'exécution et des performances générales.* Les performances peuvent être très variables dans un cadre distribué. Il faut cependant veiller à les optimiser tant que possible de manière à ne pas pénaliser le recours à la simulation distribuée. Il faut donc minimiser l'impact de l'architecture de distribution sur les performances de chaque plate-forme autant qu'optimiser l'exécution de la simulation.

La conciliation de toutes ces exigences est une tâche ambitieuse qui nécessite des efforts aussi bien techniques que conceptuels. Nous nous focalisons sur les solutions les plus génériques afin de proposer un cadre conceptuel permettant la réalisation d'une architecture pour réaliser une

infrastructure dédiée à la SOA distribuée. Il peut être réifié de différentes manières pour s'adapter aux spécificités de chaque plate-forme de SOA.

2.3 Structuration en couche de l'architecture de distribution

2.3.1 Interconnexion de plates-formes

L'infrastructure sur laquelle s'appuie la SOA distribuée est composée d'ordinateurs connectés en réseau. Que doivent exécuter ces ordinateurs pour supporter la SOA distribuée ? Il existe plusieurs possibilités :

- une interface distante à la manière de CORMAS (voir paragraphe 1.3.1). Cette solution ne nous convient pas puisque les ordinateurs ne peuvent alors exécuter la simulation, une unique machine jouant le rôle de serveur.
- un noyau d'exécution. Cette solution est couramment employée en simulation (par exemple dans SWARM) où elle permet de lancer des batteries de simulations en parallèle. Cette solution est à coupler avec la première pour obtenir une infrastructure où certaines machines se chargent de l'exécution de la simulation tandis que d'autres sont utilisées pour le pilotage et l'observation par les utilisateurs.
- une plate-forme de SOA. L'infrastructure d'exécution est composée de plates-formes de SOA interconnectées. Chaque plate-forme traite une partie de la simulation et peut superviser l'exécution globale de la SOA.

Nous optons pour cette dernière solution pour des raisons aussi bien techniques qu'ergonomiques. En effet, d'un point de vue technique, l'interconnexion de plates-formes fournit un avantage conséquent pour la transposition des services existants de la SOA (par exemple, service d'observation, de réification de l'émergence, *etc.*) dans le cadre distribué. L'exécution d'un service sur l'infrastructure distribuée revient à coordonner l'exécution local de ce service sur les plates-formes connectées.

D'un point de vue ergonomique, le déploiement de plates-formes de SOA s'intègre bien à la vision *Cloud Computing* que nous prônons : l'utilisateur retrouve le confort de la plate-forme à laquelle il est habitué tout en travaillant dans un contexte distribué. Les outils de ces plates-formes peuvent par la suite être étendus pour mieux définir le processus de simulation dans un cadre distribué.

2.3.2 Communications réseau

La SOA distribuée s'appuie sur le réseau pour des échanges d'informations diverses :

- des informations liées à la simulation. Dans la mesure où la simulation est exécutée de

manière distribuée, l'infrastructure doit se charger de l'acheminement d'une plate-forme à une autre des messages et autres requêtes des agents simulés et de l'environnement.
– des informations liées à l'architecture distribuée. Ces informations peuvent être variées allant de la connexion/déconnexion de plates-formes à la transmission de requêtes liées à la configuration ou l'emploi des services distribués.

Ainsi, plus que de choisir un protocole réseau, il s'agit ici de définir quels sont les modes de communication les plus adaptés à la transmission de ces informations. De nombreux *middleware* réseau existent et fournissent des modes de communication :

– *unicast* : communication point à point.
– *multicast* : communication multi-points, c'est-à-dire qu'un expéditeur peut envoyer le même message à plusieurs destinataires.
– *broadcast* : communication en diffusion, le message envoyé par un expéditeur est transmis à toutes les machines connectées. Nous appuyant sur les modes de communication proposés par des middleware, ce broadcast logiciel correspond à du multicast en terme d'échanges réseau : il consiste à envoyer un message réseau à toutes les machines employant le middleware et non à émettre un message sur l'adresse IP de diffusion.

Nous proposons l'emploi d'un middleware fonctionnant en broadcast pour plusieurs raisons. Tout d'abord, le broadcasting est une solution permettant d'éviter la présence d'un nœud central[12] au sein de l'infrastructure : toutes les machines partageant la même information, il n'y a aucune hiérarchie entre les plates-formes. Nous notons ici que MadKit permet un fonctionnement sans nœud central mais avec un mode de communication unicast : une plate-forme peut rejoindre le réseau en se connectant à une autre plate-forme qui va lui transmettre les adresses des autres plates-formes connectées. Mais la communication réseau en diffusion permet de plus de minimiser le nombre d'informations que doivent posséder les plates-formes. Notamment, il n'y a pas besoin de tenir un annuaire de localisation des agents de simulation : pour faire parvenir un message de simulation à son destinataire, il suffit de le diffuser sur le réseau. Ce fonctionnement sans annuaire est nécessaire pour s'affranchir réellement des problématiques large échelle ; dans un tel cadre, chaque plate-forme doit avoir une connaissance partielle de la simulation, cette connaissance se limite à la partie de simulation qu'elle doit gérer.

Un troisième argument en faveur de la diffusion se trouve dans les perspectives qu'elle apporte, notamment en terme d'écoute flottante. Ce mécanisme, utilisé notamment dans les systèmes multi-agents [Balbo *et al.*, 2004], consiste à écouter et exploiter des messages qui ne nous sont pas directement destinés. Un exemple d'utilisation dans notre infrastructure est l'analyse de l'état des plates-formes : le fait de recevoir un message d'une plate-forme, qu'il contienne des informations exploitables par la plate-forme « écoutant » le message ou non, indique quand même que la plate-forme émettrice est encore connectée et en état de communiquer.

[12] nous entendons par « nœud » un ordinateur participant à l'infrastructure d'exécution

Enfin, la communication en diffusion permet de minimiser le nombre de messages réseau que s'échangent les plates-formes, par exemple en établissant des stratégies de regroupement d'informations variées pour toutes les transmettre au sein d'un seul message réseau. Un message peut alors contenir aussi bien des requêtes de configuration d'un service, des messages des agents simulés à destination d'agents se trouvant sur une autre plate-forme et des influences sur l'environnement. Ce message sera ensuite exploité différemment selon les plates-formes, chacune tirant le maximum d'informations qui lui est possible d'exploiter (requête de configuration d'un service). Cette possibilité sera d'autant plus intéressante lorsque les équipements réseau (routeurs et autres switches) offrant des possibilités de multicasting matériel se seront démocratisés : cela entraînera une diminution drastique du nombre de messages échangés sur le réseau.

Optant pour la communication en diffusion, nous nous sommes orientés vers une communication orientée messages (*Message-Oriented Middleware* – MOM) et un fonctionnement *publish/subscribe* [Eugster *et al.*, 2003]. L'approche *publish/subscribe* repose sur l'emploi de *topics*. Un *topic* est à considérer comme une *mailing-list* à laquelle peuvent s'abonner des clients. Lorsqu'un client envoie un message sur le *topic*, tous les abonnés reçoivent ce message (même l'expéditeur du message si ce dernier s'est abonné à la liste). Pour réaliser l'infrastructure d'exécution, nous avons considéré l'usage d'un unique *topic* auquel toutes les plates-formes s'inscrivent.

2.3.3 Système Multi-Agents réparti pour la supervision de la simulation distribuée

Nous avons vu au paragraphe 2.2.1 que l'architecture de distribution se situe au carrefour de la dynamique de l'infrastructure d'exécution et de la dynamique de simulation : elle doit réifier la notion de *plate-forme virtuelle* au sein d'un environnement hautement dynamique. Les systèmes multi-agents sont particulièrement adaptés aux environnements dynamiques et apportent une réponse pertinente aux exigences aussi bien techniques que conceptuelles de notre architecture.

En effet, nous avons proposé au paragraphe 2.3.1 que l'infrastructure d'exécution soit composée de plates-formes qui gardent ainsi leurs capacités et possèdent une certaine autonomie. Ces plates-formes peuvent être modélisées par des agents, le réseau étant leur support de communication. Ce modèle convient d'autant mieux que la communication réseau se fait par le biais d'échanges de messages. Enfin, la conception des agents selon un cycle incrémental (défini dans [Boehm, 1986] et transposé dans le cadre de la SOA dans [Courdier *et al.*, 1998]) garantit l'extensibilité des capacités de l'architecture.

La réalisation d'un SMA de distribution réparti sur les plates-formes de simulation offre une solution pour satisfaire à plusieurs exigences énoncées au paragraphe 2.2, à savoir qualité de simulation, flexibilité de l'infrastructure, extensibilité des fonctionnalités et optimalité de l'exécution et des performances générales. De plus, le cadre que nous avons défini précédemment s'accorde parfaitement avec cette approche : les agents du SMA de distribution réparti permettent

de réifier chaque plate-forme au sein de l'infrastructure d'exécution ; et le réseau, via le choix d'un middleware orienté messages, se présente comme support naturel des communications entre agents.

2.3.4 Synthèse

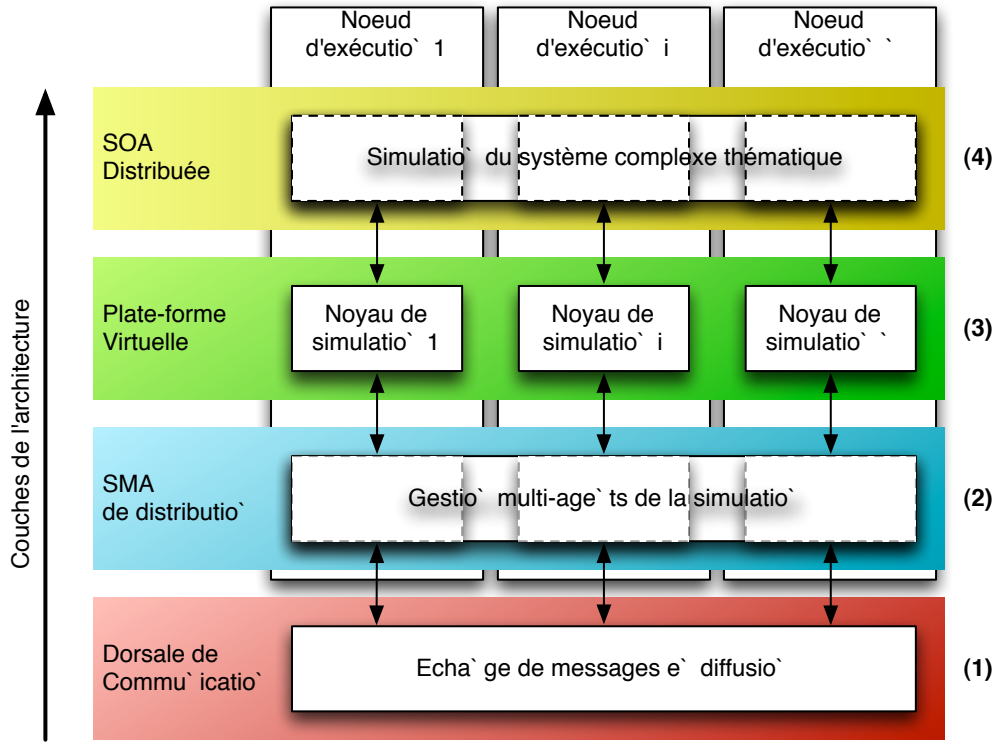


FIG. 2.1 – Structuration en couches de l'architecture de distribution.

L'infrastructure d'exécution dédiée à la SOA que nous proposons repose sur quatre couches représentées à la figure 2.1 :

1. *Dorsale de communication.* Etablie au niveau réseau, elle constitue le médium de communication entre les nœuds de simulation. Nous préconisons une dorsale employant un mode de communication en diffusion de messages. Les middleware proposant ce mode de fonctionnement offrent notamment la possibilité d'utiliser différents protocoles (*socket tcp*, *http*, *rmi*, *etc.*) ainsi que divers niveaux de qualité de service au sens des réseaux (QoS).
2. *SMA de distribution.* Réparti entre les nœuds de simulation, il assure la continuité du point de vue réseau de tous les services nécessaires à l'activité de simulation distribuée. Son objectif principal est bien évidemment d'assurer la qualité de simulation, mais le paradigme agent fait de cette couche la plus apte à gérer la flexibilité de l'infrastructure, l'extensibilité des fonctionnalités ainsi que l'optimalité de l'exécution et des performances générales. Ce SMA joue donc un rôle central dans la mise en œuvre de l'architecture de distribution.

3. *Plate-forme virtuelle.* Chaque nœud de simulation exécute une plate-forme de SOA. Les plates-formes sont interconnectées entre elles au niveaux des couches inférieures. Au niveau logique, le SMA de distribution assure le prolongement des services associés à la SOA et, au niveau physique, la dorsale de communication diffuse les messages aux plates-formes connectées. L'exécution d'un service de la SOA dans l'infrastructure d'exécution sollicite l'implémentation du service sur chaque plate-forme de l'infrastructure.
4. *SOA distribuée.* La simulation est répartie sur les nœuds de simulation. Chacun des nœuds dispose d'un noyau de simulation permettant l'exécution de la partie de simulation hébergée sur la plate-forme. Les couches précédentes permettent d'interconnecter les parties de simulation de manière à ce que la simulation soit vue dans toute sa richesse à ce niveau. La compréhension des phénomènes globaux et émergeant impliquant tout agents des différentes plates-formes est possible.

Nous appuyant sur une plate-forme de SOA et un middleware de communication réseau, le système multi-agents de distribution réparti est la clé de voûte vers la mise en œuvre de notre architecture. Assurant l'interconnexion des plates-formes à un niveau logique, il permet de passer de la problématique de la distribution d'une SOA large échelle possédant une dynamique non prédictive à un système multi-agents distribué et ainsi de bénéficier des propriétés de ce paradigme (intelligence collective, négociation de ressources, *etc.*).

2.4 Système Multi-Agents réparti pour la Simulation Orientée Agent distribuée

Le SMA de distribution et la SOA distribuée sont deux systèmes multi-agents dont la forme et les aboutissants divergent. Afin de marquer clairement la différence entre les agents appartenant à l'un ou l'autre, nous désignerons par *agents de simulation* (annotés A_S sur les figures) les agents participant à la SOA.

2.4.1 Description générale

Nous considérons que le SMA de distribution réparti réifie les plates-formes de simulation au sein de l'infrastructure d'exécution. A un niveau macro, chaque plate-forme est donc représentée dans le SMA de distribution réparti par un *Agent Macro de Distribution* (AM_D) qui dialogue avec les autres et gère une partie de la SOA ainsi que les ressources de sa plate-forme hôte (voir figure 2.2).

A la manière des *Swarms* proposés dans [Minar *et al.*, 1996], les agents macro de distribution sont composés d'agents à la granularité plus fine : chaque plate-forme exécute plusieurs *Agents Micro de Distribution* (Am_D) qui se partagent les tâches liées à l'activité de distribution sur la

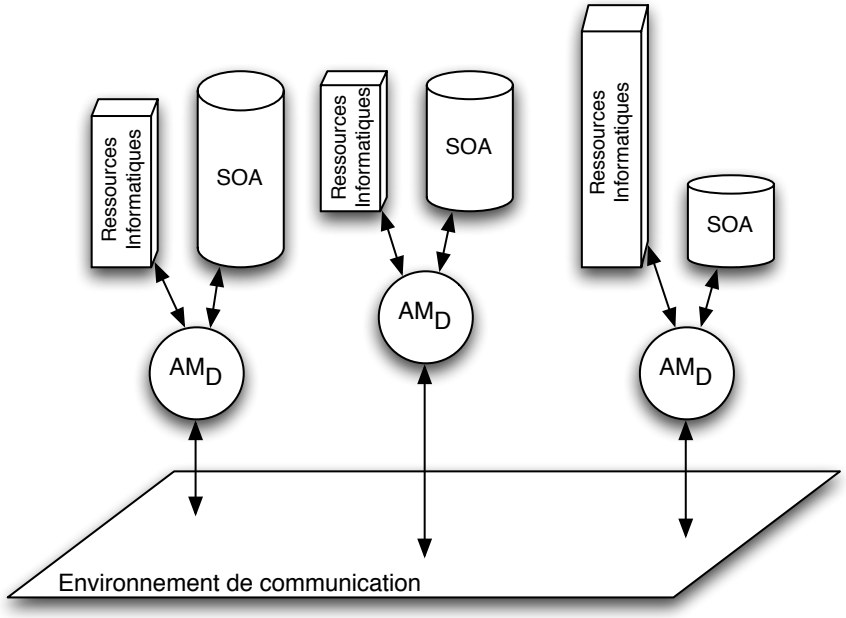


FIG. 2.2 – Vue générale du SMA de distribution réparti.

plate-forme. La définition du SMA de distribution se fait donc à deux échelles :

- au niveau de l'infrastructure d'exécution, les AM_D interagissent entre eux via la diffusion de messages sur le réseau. Ils peuvent faire transiter des données relatives à la simulation (échanges de messages entre agents de simulation ou perceptions/influences réalisées par ces agents) aussi bien que des données relatives à la distribution (connexion/déconnexion de plate-forme, bilan d'activités, équilibrage de charges, *etc.*).
- au niveau de chaque plate-forme, les AM_D sont composés de plusieurs Am_D (agents micro de distribution) répondant à divers besoins, leur nombre peut être étendu pour accroître les capacités des AM_D. Ce SMA de distribution embarqué sur chaque plate-forme est connecté aux SMA de distribution des autres plates-formes via le réseau.

Le comportement des Agents Macro de Distribution découlant de celui des Agents Micro de Distribution présents dans chaque plate-forme, nous nous focalisons sur la description du SMA de distribution d'une plate-forme. Nous adoptons pour cela une description suivant l'approche VOYELLE proposée dans [Demazeau, 1997].

2.4.2 Composition des agents macro de distribution

Chaque agent macro de distribution est composé de plusieurs agents micro de distribution. Ces Am_D gèrent au niveau de la plate-forme des aspects différents de la distribution et interagissent entre eux pour définir globalement le comportement de la plate-forme dans le cadre distribué. Nous focalisant sur l'exécution distribuée des agents de simulation, nous définissons

ici trois Am_D pour réaliser cette tâche. D'autres agents micro de distribution pourront bien sûr être développés et interagir avec ceux que nous définissons pour traiter d'autres problématiques liées à la SOA distribuée.

Agent Réseau : A_R

Plus que d'assurer la connexion réseau, il centralise l'accès réseau sur la plate-forme et peut notamment appliquer différentes politiques de gestion de cette ressource. Tout agent souhaitant diffuser un message sur le réseau s'adresse donc à l'A_R. De même, à la réception d'un message réseau, c'est ce dernier qui a la charge d'adresser le message réseau au destinataire local.

Il joue un rôle majeur dans la garantie de la qualité de la simulation en assurant la transmission des messages réseau liés au déroulement de la simulation. Il peut être étendu avec des outils tels que ceux proposés dans la détection des congestions TCP [Anelli *et al.*, 2008] pour adapter la qualité de service et exploiter au mieux le réseau.

Agent de Cohérence : A_C

Cet agent a pour charge première de garantir la qualité de la simulation. En tant que tel, il coordonne l'ordonnancement local de la simulation afin que ce dernier reste cohérent avec ceux des autres plates-formes et il assure aux agents de simulation leur contexte d'exécution en diffusant si nécessaire perceptions et influences.

C'est au travers de cet agent que nous assurons le déploiement sur le réseau des services des plates-formes. Par exemple, quand un utilisateur d'une plate-forme ptf_1 fait une requête d'observation sur un agent de simulation se trouvant sur une plate-forme ptf_2, l'agent de cohérence de ptf_1 diffusera la requête sur le réseau. A sa réception, l'agent de cohérence de ptf_2 traitera la requête en la faisant exécuter par le service d'observation de ptf_2, il enverra ensuite les résultats sur le réseau de manière à ce que l'agent de cohérence de ptf_1 transmette les résultats de l'observation à l'utilisateur de ptf_1.

Il intervient enfin dans la stratégie d'optimisation de la simulation : étant garant de la qualité de la simulation, il peut accepter ou rejeter les propositions d'optimisation que lui fait l'agent de flexibilité défini au paragraphe suivant.

Agent de Flexibilité : A_F

L'agent de flexibilité, comme son nom l'indique, gère principalement la flexibilité de l'infrastructure de simulation, au niveau de sa plate-forme hôte. Cela signifie qu'il peut négocier avec l'agent de cohérence l'accueil d'une nouvelle partie de la simulation (c'est-à-dire des agents de simulation et une partie de l'environnement) qui proviendrait d'une plate-forme se déconnectant.

En cas de déconnexion de sa plate-forme, il va négocier avec les autres agents de flexibilité le transfert de la partie de simulation dont sa plate-forme avait la charge.

Cet agent met en place différentes stratégies dans le but d'optimiser la simulation. Intégrant des mécanismes de négociation de transferts d'agents de simulation, il peut optimiser la répartition de charges en cours d'exécution de la simulation : il négocie des échanges d'agents de simulation avec l'agent de cohérence de sa plate-forme et avec les agents de flexibilité des autres plates-formes.

2.4.3 Organisation des agents du SMA de distribution

Pour éviter la création de nœud central dans l'infrastructure d'exécution, il n'y a aucune hiérarchie ni organisation particulière entre les plates-formes. Cela se traduit par une absence d'organisation entre les agents macro de distribution. Un AM_D est constitué de trois agents micro de distribution autonomes (A_R, A_C et A_F) et traitant de tâches différentes, il n'y a pas non plus d'organisation spécifique entre ces trois agents.

2.4.4 Interactions agents du SMA de distribution

Dans ce SMA de distribution réparti, les agents micro de distribution interagissent entre eux via des échanges de messages et avec l'environnement sous la forme de perceptions/influences.

Nous détaillons les interactions entre agents et environnement au paragraphe 2.4.5. Pour ce qui est des échanges de messages entre agents micro de distribution, deux types d'échanges de messages peuvent avoir lieu :

- échange de messages entre les agents de cohérence, de flexibilité ou réseau d'une même plate-forme. Ces agents se trouvant sur la même plate-forme, ils s'échangent des messages « locaux » en s'appuyant sur les moyens de communication mis à leur disposition.
- échange de messages entre agents micro de distribution situés sur des plates-formes distantes. D'un point de vue conceptuel, cet échange n'est pas directement réalisable puisqu'un agent micro de distribution ne peut percevoir que l'agent macro de distribution d'une autre plate-forme. Ainsi, l'agent de cohérence A_{Cptf1} ne peut s'adresser qu'à l'agent macro AM_{Dptf2} de ptf_2. Pour cela, il envoie son message à l'agent réseau de sa plate-forme A_{Rptf1} qui transmet le message à AM_{Dptf2}. Au niveau de ptf_2, A_{Rptf2} reçoit le message qui est alors transmis à A_{Cptf2}.

2.4.5 Environnement du SMA de distribution

L'environnement des agents micro de distribution d'une plate-forme est très diversifié et ne peut être décrit dans son intégralité dans la mesure où le SMA de distribution réparti doit pouvoir

étendre son support à tout service développé sur la plate-forme.

Nous considérons ainsi l'environnement sous l'angle du multi-environnement tel que décrit dans [Soulié, 2001]. Ainsi, nous pouvons identifier plusieurs environnements :

- la SOA. Il s'agit de la partie de simulation (agents de simulation et environnement) exécutée par la plate-forme. Parmi les trois agents micro de distribution de la plate-forme, l'agent de cohérence est celui qui développe le plus d'interactions avec cet environnement puisque ce dernier doit transmettre certaines interactions des agents de simulation aux autres plates-formes. L'agent de flexibilité interagit avec cet environnement pour modifier la partie de simulation hébergée par la plate-forme.
- le moteur de simulation et plus particulièrement l'ordonnanceur de la plate-forme. L'agent de cohérence observe et pilote l'ordonnanceur local de la plate-forme afin d'assurer la qualité de la simulation.
- le réseau. L'agent réseau est le seul à interagir avec cet environnement. Il le sollicite pour la diffusion de messages mais veille aussi à minimiser le trafic induit par la simulation.
- les services de la plate-forme. Il s'agit là de disposer d'un environnement propre à chaque service et au travers duquel l'agent de cohérence peut solliciter localement le service pour répondre à une demande extérieure à la plate-forme ou observe le service pour étendre ces capacités sur l'infrastructure d'exécution.

L'approche multi-environnement joue un rôle majeur dans l'extensibilité des capacités du SMA réparti. Lorsqu'un service est ajouté à la plate-forme, il suffit de lui associer un environnement et de définir les règles d'interaction entre cet environnement et les agents micro de distribution concernés. Un nouvel agent micro de distribution peut être ajouté si la distribution de ce service le nécessite.

2.4.6 Notion de space pour l'implémentation d'agents systèmes dans la plate-forme de SOA

Les éléments de conception que nous proposons pour la réalisation d'une architecture de distribution se concentrent autour de la réalisation d'un système multi-agents de distribution réparti sur les plates-formes. Dans un cadre plus large, l'introduction d'un SMA au sein même de la plate-forme de SOA ouvre la voie à de nombreuses possibilités : agents d'observation adaptant l'interface homme-machine en fonction de l'utilisateur, agents « explorateurs » de simulations testant différentes configurations de simulation pour optimiser certains indicateurs (à la manière de LEIA [Gaillard *et al.*, 2008]), *etc.*. Nous nommons *Agents Systèmes* A_{SYS} les agents d'une plate-forme ne faisant pas partie du modèle de simulation ; les agents micro de distribution sont des agents systèmes. Nous introduisons la notion de *space* pour identifier et délimiter les différents groupes d'agents systèmes.

Définition 2.4.1 Space *: ensemble d'agents systèmes collaborant entre eux à une tâche complexe, possédant leurs propres règles d'activation temporelle, et interagissant via perceptions et influences avec « l'extérieur » du space, c'est-à-dire la simulation, les autres spaces et les services de la plate-forme.*

Cette approche a pour avantage d'offrir un faible couplage entre les spaces et les services de la plate-forme. L'ajout ou la suppression d'un space ne déstabilise pas la plate-forme : les services minimaux garantissant le fonctionnement de la plate-forme sont implémentés indépendamment de tout space et leur fonctionnement n'est pas modifié par la création de spaces.

L'emploi de spaces s'appuie fortement sur la notion de multi-environnement. Cette représentation des composants (*i.e.* des services et spaces) de la plate-forme sous forme d'environnements confère aux spaces plusieurs avantages :

- assurer le faible couplage entre spaces. C'est une conséquence directe du fait de percevoir les autres spaces sous forme d'environnements.
- uniformiser et faciliter l'accès aux différents composants de la plate-forme. Les agents systèmes peuvent manipuler tout composant de la plate-forme avec les seules requêtes de perception ou d'influence.
- contrôler les actions des agents systèmes. Les agents systèmes ne peuvent qu'émettre des influences vers les autres composants de la plate-forme : ces influences ne se traduisent pas systématiquement par l'action attendue.

Dans le cas des spaces, la notion d'environnement n'est pas abordée uniquement sur le plan conceptuel. Elle suppose la création d'une couche logicielle à l'interface entre le paradigme agent des A_{SYS} et l'architecture objet de la plate-forme (voir figure 2.3). Se présentant aux agents sous la forme d'un environnement, cette couche convertit les requêtes de perception/influence des agents systèmes en actions à produire sur le service ou le space subissant l'interaction. Cette couche peut notamment vérifier que la requête soit correcte avant d'en effectuer les actions associées ou limiter les actions autorisées à certains A_{SYS}.

Afin de supporter le fonctionnement distribué, les spaces doivent tous pouvoir répondre à certaines requêtes relatives à l'ajout ou la suppression d'agents de simulation. Par exemple, lorsque l'agent de flexibilité d'une plate-forme ptf_1 demande le déplacement d'agents de simulation vers une plate-forme ptf_2, toutes les règles d'observation, et de manière générale tout élément de configuration de la plate-forme relatif aux agents de simulation à déplacer, doivent être transférés à la nouvelle plate-forme hôte.

Pour réaliser aisément une telle action dans un contexte où le nombre et les fonctions des spaces sont inconnus, le space de distribution de ptf_1 diffuse un message à tous les spaces de la plate-forme. Ces spaces transmettent sous la forme d'influences de configuration toute leur configuration relative aux agents à déplacer. L'agent réseau de ptf_1 transmet alors tous les agents de simulations et les influences de configuration qui leurs sont associées à l'agent réseau

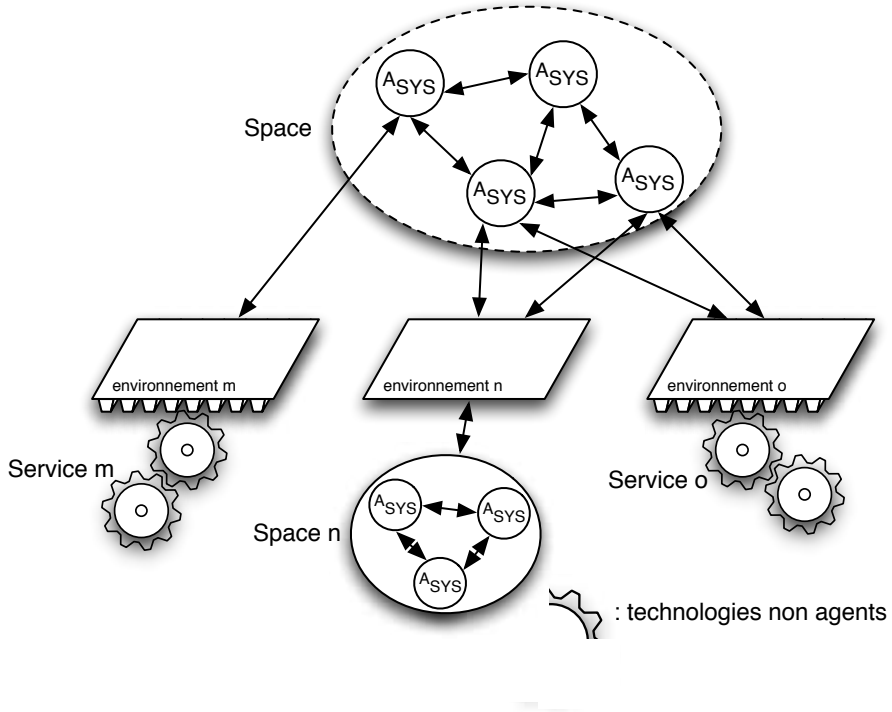


FIG. 2.3 – Les agents systèmes d'un space et les environnements qu'ils perçoivent.

de ptf_2. L'agent de cohérence de ptf_2 injecte enfin les agents de simulation dans la partie de simulation gérée par ptf_2 et transmet les influences de configuration aux spaces de ptf_2.

2.5 Synthèse

Dans ce chapitre, nous nous sommes focalisés sur un cadre conceptuel pour la réalisation d'une infrastructure dédiée à la SOA distribuée. Le choix d'une infrastructure distribuée, plutôt que de recourir à une solution de type super-calculateur, est principalement motivé par le coût et l'aisance de déploiement pour les utilisateurs. En accord avec cette volonté de produire un outil aisé à déployer, nous préconisons une approche de type *cloud computing*, où l'utilisateur n'a pas à se préoccuper de l'infrastructure pour en tirer parti. Nous introduisons la notion de plate-forme de SOA virtuelle : l'utilisateur interagit avec la SOA distribuée de la même manière qu'il interagit avec la SOA exécutée en mono-poste mono-processeur, la distribution et ses mécanismes complexes lui sont transparents.

La réalisation d'une telle infrastructure est une tâche complexe. Il faut en effet satisfaire différentes exigences tout en prenant en compte les contraintes imposées par l'infrastructure d'exécution et la dynamique de simulation. Nous avons identifié cinq exigences : qualité de simulation, portabilité des modèles, flexibilité de l'infrastructure, extensibilité des fonctionnalités et optimalité de l'exécution et des performances générales.

Pour satisfaire ces exigences, nous détaillons à la figure 2.4 l'architecture en couches proposée au paragraphe 2.3 :

- La dorsale de communication est une couche de communication applicative s'appuyant sur le principe de la diffusion de messages.
- Le SMA de distribution fonctionne dans un cadre multi-environnement. Interagissant avec les environnements couplés aux plates-formes, les agents macro de distribution accèdent aux services des plates-formes. Ils établissent des stratégies globales de répartition des agents et d'ordonnancement (synchronisations entre plates-formes, ordonnancement parallèle, *etc.*) au travers d'un environnement de communication qui donne accès à des services portés par la dorsale de communication.
- La plate-forme virtuelle encapsule une plate-forme de SOA non-distribuée (par exemple GEAMAS-NG, SWARM, *etc.*).
- La SOA distribuée accueille l'ensemble des agents participant au système complexe thématique ; ces agents étant répartis sur les différents nœuds d'exécution par les couches inférieures de l'architecture.

L'intégration d'un SMA pour conférer des fonctionnalités avancées à la plate-forme de SOA est une tâche qui doit être cadrée. Pour cela, nous introduisons la notion de space : regroupement d'agents systèmes participant entre eux à la réalisation d'une tâche liée à l'activité de simulation. Les spaces permettent d'établir un faible couplage entre spaces et avec les services de la plate-forme. Les agents systèmes d'un space interagissent avec tout autre composant de la plate-forme par le biais de perceptions et influences. Ceci implique le développement d'une couche logicielle (représentée à la figure 2.4) traduisant les requêtes des agents en actions sur les services ou spaces sollicités.

Les différentes couches de notre architecture répondent aux exigences que nous avons identifiées de la manière suivante :

- qualité de simulation : c'est le cœur de métier de l'agent de cohérence.
- portabilité des modèles : elle est assurée par l'emploi de plates-formes de SOA comme nœud de l'infrastructure d'exécution.
- flexibilité de l'infrastructure : l'agent de flexibilité est dédié à cette tâche.
- extensibilité des fonctionnalités : les agents micro de distribution considèrent tout composant de la plate-forme comme des environnements. Cette vision agent confère un grand potentiel d'extensibilité aux agents qui peuvent alors gérer de nouvelles fonctionnalités.
- optimalité de l'exécution et des performances : l'architecture complète doit être implémentée de manière à impacter le moins possible les ressources de l'infrastructure. L'agent réseau traite spécifiquement des performances du réseau. Les agents de cohérence et de flexibilité sont chargés de l'optimisation de l'exécution de la simulation.

Notons que l'architecture de distribution proposée dans ce chapitre n'aborde l'optimalité de l'exécution et des performances que sous l'angle de la consommation de ressources. Il convient

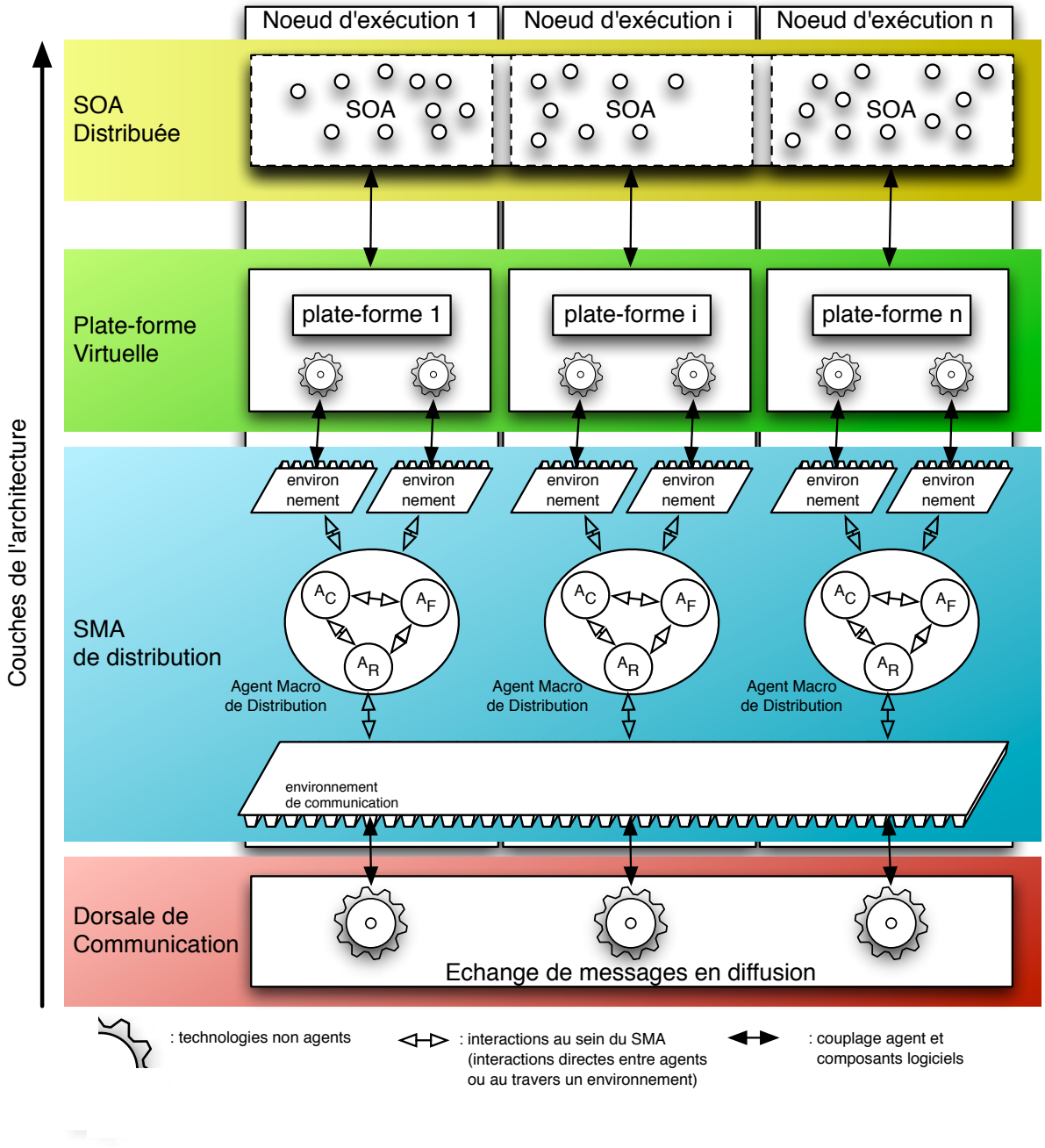


FIG. 2.4 – Architecture de distribution détaillée.

maintenant de réaliser des propositions permettant de tirer profit de l'infrastructure distribuée. Ainsi, après avoir proposé un cadre général conceptuel d'architecture, nous présentons les contributions théoriques au centre de notre travail et visant l'exécution de SOA en exploitant au mieux les ressources partagées par notre architecture distribuée.

Chapitre 3

Modèle d'ordonnancement parallèle multi plates-formes

Sommaire

3.1 Représentation du temps dans les simulations 54
3.1.1 Double sémantique du temps dans les simulations 54
3.1.2 Représentation virtuelle du temps 55
3.2 Ordonnancement et parallélisation 56
3.2.1 Le modèle à Temporalité 56
3.2.2 Ordonnancement du modèle à Temporalité 58
3.2.3 Exécution parallèle 59
3.3 Identification des liens de causalité avec le modèle à Temporalité . 62
3.3.1 Définition des liens de causalité dans notre modèle de simulation 62
3.3.2 Recherche des liens de causalité 63
3.3.3 Formalisation 64
3.4 Mise en oeuvre de l'ordonnanceur global 65
3.4.1 Table de dépendances 66
3.4.2 Complexité de calcul de la table de dépendances 67
3.4.3 Fonctionnement en simulation 68
3.4.4 Exécutabilité de la simulation 70
3.5 Discussion 71
3.5.1 Intérêt de notre approche : exemple de couplage de modèles simulés . . 71
3.5.2 Temps simulés multiples 74
3.5.3 Simulation parallèle conservative 76
3.5.4 Détection de liens de dépendance 77
3.6 Synthèse 79

La simulation permet de modéliser des systèmes dynamiques, évoluant au cours du temps. Ainsi, que les concepts temporels soient explicités ou non dans le modèle de simulation, il est possible de définir un ordonnancement des événements se produisant dans la simulation. Le moteur de simulation de la plate-forme est donc utilisé pour « faire avancer le temps dans la simulation ».

Dans un cadre distribué, nous disposons de plusieurs moteurs de simulation aux performances variées. On pourrait être tenté de les synchroniser de manière à se ramener à un temps unique dans la simulation distribuée : la qualité de simulation serait préservée au détriment des performances (les moteurs de simulation sont fortement contraints). Nous montrons dans ce chapitre que la non-uniformisation du temps est en fait un avantage.

Nous nous intéressons dans ce chapitre aux concepts temporels pour proposer un algorithme d'ordonnancement parallèle de la simulation. Cet algorithme permet notamment de casser la croissance monotone du temps au sein d'une plate-forme.

3.1 Représentation du temps dans les simulations

3.1.1 Double sémantique du temps dans les simulations

Le temps est une notion fondamentale dans la simulation. Il n'est pourtant pas systématiquement explicité dans tous les modèles de simulation. Par exemple, Holcombe s'intéresse dans [Holcombe *et al.*, 2006] à une simulation distribuée sans aborder la question du temps : les agents s'exécutent en parallèle et produisent des événements au fil de leur exécution sans respecter de règles temporelles particulières. A l'inverse, un modèle de simulation comme Biomas [Courdier *et al.*, 2002] propose une description temporelle très complète des activités des agents.

Cette différence s'explique par le fait que le temps dans les simulations est utilisé de deux manières : il sert de *dimension* et de *grandeur*. Par *dimension* nous entendons l'ordre dans lequel les événements occurrent. La simulation est donc exécutée sans associer de sémantique à un pas de temps : la valeur du pas de temps n'ayant pas de sens particulier, l'utilisateur s'intéressera plutôt à « l'avant » ou « l'après » d'un événement.

Dans de nombreuses simulations de systèmes réels naturels ou complexes, il est intéressant d'associer au temps simulé des *grandeurs* de temps réelles : l'utilisateur peut alors travailler la simulation en considérant des dates et durées caractéristiques du système réel plutôt que les valeurs de temps simulé internes au moteur de simulation.

Ainsi, dans sa vision la plus riche, le temps dans la simulation exprime à la fois la succession des événements (*dimension*) et le *mapping* du temps simulé sur le temps réel[13] (*grandeur*). Nous

[13]Dans ce chapitre nous considérons le sens premier du terme temps réel : il s'agit du temps dans lequel

notons toutefois que la dimension est directement liée au moteur de simulation tandis que la grandeur est liée à l'interprétation de l'utilisateur : elle n'est pas utilisée pour le séquencement de la simulation. Cela signifie que toute simulation implique une dimension temporelle mais pas nécessairement une interprétation du temps simulé en tant qu'image d'un temps de la réalité. Cette interprétation peut se faire à différents instants du processus de simulation et n'impacte pas l'exécution de la simulation. Nous nous focalisons donc sur l'aspect « dimension » du temps dans nos travaux.

3.1.2 Représentation virtuelle du temps

Simuler un système a pour avantage d'en offrir une meilleure compréhension, c'est-à-dire autoriser une observation de sa dynamique dans un temps adapté à une interprétation scientifique. Ainsi, que le temps dans une simulation soit abordé ou non sous l'angle de sa grandeur, il possède sa propre règle d'évolution par rapport au temps de la réalité ; la courbe d'évolution du temps simulé par rapport au temps réel n'est pas contraint à la fonction identité.

Dans certaines SOA, l'approche consistant à associer un *thread* par agent n'apporte qu'une solution au problème de la gestion des conflits d'accès aux ressources. Depuis la proposition du modèle influence/réaction, la quasi-totalité des simulations orientées agent reposent sur un ordonnanceur gérant l'activation des agents. Les agents de simulation sont censés être activés en même temps. Pour réaliser cela, il est nécessaire d'arrêter l'écoulement du temps simulé, le temps « réel » d'exécuter tous les agents simulés devant l'être à cet instant de la simulation. Cela signifie que le temps simulé est discrétisé d'une part (l'ordonnanceur n'amène le temps qu'aux instants définis dans le modèle) et évolue par créneaux par rapport au temps réel d'autre part (voir figure 3.1). Le temps évolue sous la forme d'une fonction continue par morceaux croissante monotone.

Au niveau de la distribution, nous avons proposé au paragraphe 2.3.1 de fonder l'exécution distribuée sur les moteurs de simulation des plates-formes connectées. Cela signifie que la simulation distribuée s'appuie sur plusieurs ordonnanceurs. En l'absence de mécanismes spécifiques, la distribution de l'exécution de SOA produit deux phénomènes sur l'ordonnancement de la simulation :

- un décalage du temps simulé entre les plates-formes. Selon la charge et les performances de chaque plate-forme, certaines plates-formes simuleront plus vite que d'autres et afficheront donc un temps simulé « plus avancé » que sur les autres plates-formes.
- un retard dans certaines interactions. Pour réaliser certaines interactions entre agents simulés ou avec un environnement se trouvant sur des plates-formes différentes, il est nécessaire de transmettre les requêtes de perception/influence sur le réseau. Le temps de transfert du

nous vivons et non le terme informatique *temps réel* qui désigne une contrainte temporelle dans l'exécution de traitements informatiques

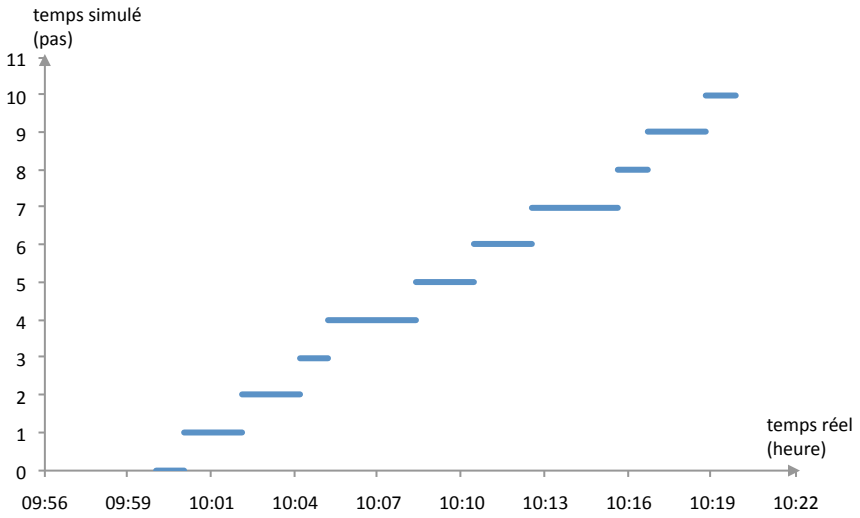


FIG. 3.1 – Exemple d'une progression du temps simulé par rapport au temps réel.

message perturbe alors l'exécution de la simulation.

Ces deux phénomènes induisent un biais dans l'exécution du modèle simulé. Ce biais n'est pas systématiquement problématique dans la mesure où plusieurs modèles introduisent eux-même des retards, notamment pour tester la robustesse des solutions proposées dans le modèle. Nous souhaitons toutefois assurer la qualité de la simulation en supprimant toute distance entre le modèle de simulation et son exécution : l'exécution distribuée n'introduit aucun biais dans la simulation.

Dans cette démarche, une première étape consiste à synchroniser les ordonnanceurs des plates-formes de l'infrastructure d'exécution à chaque pas de temps. Mais nous allons voir qu'il est possible de définir un algorithme permettant de minimiser le nombre de synchronisations nécessaires. Nos travaux se basent ici sur le modèle à Temporalité proposé dans [Payet *et al.*, 2006a].

3.2 Ordonnancement et parallélisation

3.2.1 Le modèle à Temporalité

Dans l'absolu, le temps englobe trois concepts distincts : la simultanéité, la succession et la durée [Klein, 2004]. La gestion du temps dans une simulation doit s'exprimer pour chacun de ces concepts : au niveau de la simultanéité on doit définir un mécanisme qui permet de résoudre les problèmes de concurrence des actions ; au niveau de la succession on doit établir l'ordre dans lequel les actions seront réalisées ; et, au niveau de la durée on doit spécifier les intervalles de temps durant lesquels les actions seront menées. Couramment dans les simulateurs multi-agents le niveau simultanéité est lié à la gestion de l'environnement (car c'est le milieu

dans lequel les actions concurrentes se réalisent) et est fortement dépendant du modèle, alors que les niveaux de succession et de durée sont gérés au sein d'un processus plus autonome qu'on appelle l'ordonnanceur.

Le *modèle à Temporalité* a été introduit comme modèle temporel pour la SOA. Ses atouts sont multiples avec notamment la possibilité de clairement identifier dans le temps les différentes dynamiques auxquelles participe chaque agent. En effet, le modèle à temporalité confie à l'agent la tâche de décrire sa propre dynamique temporelle, *i.e.* expliciter les instants où il souhaite s'activer. L'ordonnanceur est centré sur les besoins directement exprimés par les agents qui composent le modèle simulé. De tels besoins sont exprimés à l'aide d'une structure de données appelée *temporalité* et qui est définie par $t = \{id, d, f, p, v\}$ avec :

- id : l'identifiant de la temporalité auquel l'agent pourra associer un comportement spécifique.
- $[d, f]$: l'intervalle de temps durant lequel la temporalité peut être déclenchée.
- p : la période qui définit la durée entre deux déclenchements de la temporalité (p=0 si l'action est ponctuelle).
- v : la variabilité, c'est-à-dire la précision en dessous de laquelle l'occurrence temporelle reste valide.

Une temporalité provoquera le déclenchement du comportement de l'agent à toutes les dates x vérifiant : $x = d + p \times k$, avec k un entier tel que $0 \leq k \leq n$, et n le plus grand entier vérifiant $(d + p \times n) \leq f$. On nomme cet ensemble de dates : dates de déclenchement de la temporalité.

L'établissement des temporalités est effectué en phase d'initialisation de la simulation par l'invocation d'une méthode particulière sur chacun des agents. Cette étape indique aux agents que la simulation va commencer et qu'ils sont donc invités à définir l'ensemble des temporalités qui vont régir leurs déclenchements au cours du temps. Durant le déroulement de la simulation, les agents, dans le cadre de leurs activités, pourront à tout moment redéfinir ou créer de nouvelles temporalités afin d'ajuster leur dynamique comportementale en fonction de la situation. En fonction des moyens mis à disposition par la plate-forme d'exécution (par défaut un appel système), l'agent pourra transmettre à l'ordonnanceur les modifications qu'il souhaite apporter à sa dynamique.

Au niveau du moteur, nous considérons deux structures de données supplémentaires (voir figure 3.2) :

- *Slot Temporel* : élément d'ancrage correspondant à un point de l'axe du temps sur lequel le simulateur va être amené à déclencher l'exécution du comportement d'un agent.
- *Tempo* : structure regroupant l'ensemble des temporalités qui, à un instant donné, sont situées sur un même slot temporel et qui ont une même valeur de période. C'est cette valeur de période qui caractérise le tempo en traduisant le fait que les temporalités qu'il comporte ont toutes le même rythme. La variabilité v des temporalités est considérée lors

du regroupement des temporalités en tempos : deux temporalités t_1 et t_2 peuvent être située sur le même tempo si $p_1 = p_2$ et $[(d_1 + k \times p_1) - v_1, (d_1 + k \times p_1) + v_1] \bigcap [(d_2 + l \times p_2) - v_2, (d_2 + k \times p_2) + v_2] \neq \emptyset$ avec k et l deux entiers naturels.

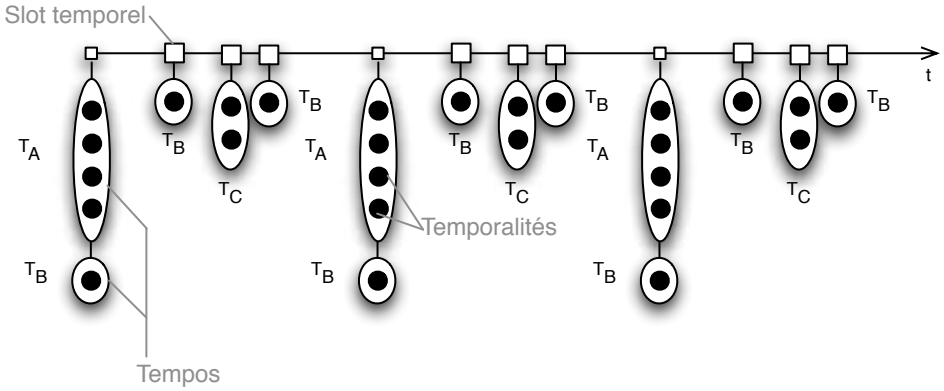


FIG. 3.2 – Structures de données associées au modèle à Temporalité.

3.2.2 Ordonnancement du modèle à Temporalité

Nous considérons dans un premier temps l'ordonnancement d'une simulation s'exécutant sur une unique plate-forme en mono-processeur.

L'ordonnanceur fait progresser le temps simulé en l'emmenant successivement sur chacune des positions signalées par un slot temporel. Sur chaque slot temporel, tous les tempos sont traités en produisant l'exécution de tous les comportements associés aux temporalités qu'ils comportent. Durant son exécution, un agent peut percevoir et influencer son environnement ainsi que définir de nouvelles temporalités et modifier ses temporalités existantes. Les influences produites n'entrainent aucune modification de l'environnement pendant le traitement d'un slot temporel.

Après l'exécution de tous les traitements associés à un tempo, et grâce à la valeur de la période de ce tempo, nous pouvons déterminer le prochain slot temporel sur lequel le rattacher. Ainsi, en une seule opération, la prochaine date de déclenchement de toutes les temporalités portées par le tempo est obtenue.

Une fois que tous les tempos rattachés à un slot temporel ont été traités, l'ordonnanceur fait progresser le temps simulé jusqu'au prochain slot temporel. Avant le traitement des tempos associés à ce slot temporel, l'environnement est mis à jour en fonction de ses propres lois d'évolution (en s'appuyant sur la mesure du temps simulé s'étant écoulé entre le slot temporel précédent et celui qui va s'exécuter) et des influences faites par les agents lors de leur exécution au slot temporel précédent. C'est notamment à ce moment de l'exécution de la simulation, qu'il est possible de résoudre les éventuels problèmes de simultanéité : le nouvel état de l'environnement relatera la priorité donnée à certaines influences. Passée cette phase de mise à jour de l'environnement,

l'ordonnanceur déclenche l'exécution des tempos associés au slot temporel en cours. Ce processus est réitéré jusqu'à la fin de la simulation.

Considérons maintenant une distribution *a priori* des agents sur des noeuds de simulation. Pour garantir une exécution cohérente du modèle simulé, il est nécessaire d'avoir une entité spécialement chargée du pilotage de chaque noeud d'exécution : nous nommerons cette entité *l'ordonnanceur global* du fait que sa portée s'étend à l'ensemble des noeuds de simulation. Sur chaque noeud, un ordonnanceur local sera chargé de l'activation des agents de manière séquentielle et en conformité avec le modèle temporel. Nous désignerons par ordonnanceur, le groupe constitué de l'ordonnanceur global et des ordonnanceurs locaux qu'il supervise.

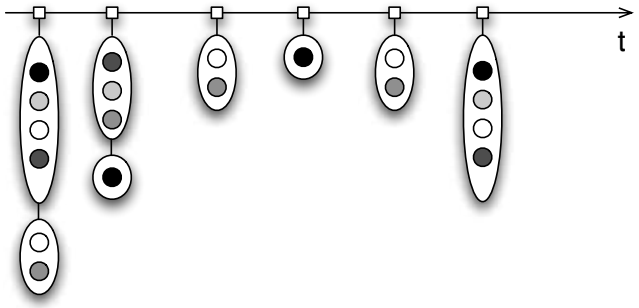


FIG. 3.3 – Axe temporel simulé du modèle à Temporalité

Les agents déposant leurs temporalités avant le lancement de la simulation, il est possible d'avoir une première image de l'axe temporel simulé que suivra la simulation. La figure 3.3 présente un axe temporel des ordonnanceurs locaux pour le fonctionnement distribué : les agents situés sur des noeuds de simulation différents ont des temporalités de couleur différente.

Nous considérons ici que l'environnement est morcelé sur les plates-formes disponibles : quand une plate-forme se connecte, respectivement se déconnecte, elle reçoit une portion d'environnement à simuler, respectivement transmet les morceaux d'environnements aux plates-formes restantes. Durant la simulation, les plates-formes se chargent de fournir aux agents un environnement virtuel tel que décrit dans [Mertens *et al.*, 2004] : les agents interagissent avec l'environnement sans avoir à connaître sur quelles plates-formes se situent les parties d'environnement qu'ils manipulent. les perceptions des agents sont directement traitées, même si elles nécessitent des échanges réseaux au cours de l'exécution du slot temporel, tandis que les influences ne sont transmises qu'à la fin de celui-ci. Nous ne détaillerons pas plus les mécanismes associés à l'environnement distribué dans la mesure où ils dépassent le cadre de notre étude.

3.2.3 Exécution parallèle

Nous cherchons à définir un ordonnanceur optimisant l'exécution parallèle de la simulation et s'appuyant sur le modèle à Temporalité. Notre préoccupation première est de minimiser la

durée globale des simulations. Nous employons les informations temporelles pour définir un ordonnanceur global indépendant de la sémantique de l'application [Sébastien *et al.*, 2008b].

Cet ordonnanceur doit permettre d'activer les tempos au plus tôt dans le temps réel tout en garantissant que cette exécution au plus tôt n'introduira pas d'incohérence vis-à-vis du modèle simulé.

Il faut en effet prendre en compte certains phénomènes pour garantir la cohérence de la simulation. Considérons l'exemple suivant : une cuve se remplit en permanence, un agent transporteur T a pour charge de récupérer le contenu de la cuve, un agent superviseur S observe la cuve et déclenche une procédure d'urgence si celle-ci déborde. Le superviseur et la cuve sont sur une plate-forme $P1$ et le transporteur sur une plate-forme $P2$. Sur $P1$, S voit la cuve déborder à l'instant ts et déclenche la procédure d'urgence. $P1$ simule alors la procédure d'urgence à l'instant $ts + 1$ quand elle reçoit un message de $P2$ et daté de l'instant ts indiquant que T a vidé la cuve : la cuve n'aurait pas dû déborder à l'instant ts et S n'aurait pas dû déclencher la procédure d'urgence. L'état du système devient incohérent avec le modèle simulé.

L'ordonnanceur global a la charge de garantir la cohérence temporelle du modèle simulé, c'est-à-dire qu'il doit assurer que les agents disposent de toutes les informations à jour pour leur exécution. Outre les données internes à l'agent, il s'agit de fournir à cet agent une vision à jour de l'environnement. Ainsi, l'ordonnanceur global va imposer aux ordonnanceurs locaux un point de synchronisation de manière à ce que les influences produites par les agents sur tous les noeuds de simulation soient diffusées et le nouvel état de l'environnement déterminé.

Pour détailler le fonctionnement des points de synchronisation, nous posons Ptf l'ensemble des plates-formes connectées et participant à la simulation ; cet ensemble est connu de toutes les plates-formes grâce à la diffusion de messages sur le réseau. $Slot(i, j)$ désigne le dernier slot temporel exécuté sur i et $t_{Slot}(i, j)$ l'instant simulé qui lui est associé, le prochain slot temporel exécuté sur i est alors noté $Slot(i, j + 1)$ et a pour instant simulé $t_{Slot}(i, j + 1)$. Un point de synchronisation correspond à une pause dans l'exécution de la simulation. Quand la plate-forme i termine l'exécution du slot temporel $Slot(i, j)$, elle diffuse un message de synchronisation à toutes les plates-formes et se met en attente, au lieu d'exécuter $Slot(i, j+1)$ comme elle le ferait en fonctionnement non distribué. Ce message de synchronisation contient :

- $t_{Slot}(i, j)$, le temps simulé du dernier slot temporel exécuté par i,
- $t_{Slot}(i, j + 1)$, le temps simulé du prochain slot temporel qu'exécutera i,
- les influences produites par les agents au cours du traitement de $Slot(i, j)$ et devant impacter la partie de l'environnement non-exécutée sur i.

La plate-forme i ne pourra poursuivre son exécution que lorsque toute les autres plates-formes auront terminé l'exécution des slots temporels dont le temps simulé est inférieur à $t_{Slot}(i, j + 1)$. Ainsi, la condition d'activation de $Slot(i, j + 1)$ est donnée par $\forall k \in Ptf, t_{Slot}(i, j + 1) \leq t_{Slot}(k, j + 1)$. Pour éviter que les plates-formes restent en attente d'une plate-forme ayant subi

une déconnexion accidentelle, il est possible de fixer un temps d'attente maximal au-delà duquel elles reprendront l'exécution ou préviendront les utilisateurs.

L'ordonnancement global repose entièrement sur la diffusion de messages de synchronisation et l'application par chaque plate-forme de la règle d'activation ci-dessus. Cette solution a pour avantage de supporter un ordonnancement complexe sans pour autant nécessiter un composant assurant le rôle d'ordonnanceur global : l'ordonnancement se fait sans nœud central.

Les influences étant transmises dans les messages de synchronisation, l'état de l'environnement pourra être mis à jour dès la réception des messages de synchronisation : l'adjonction des influences dans les messages de synchronisation permet de garantir que toutes les influences ont été prises en compte (il n'y a aucune influence qui n'aurait pas encore été délivrée par le réseau).

Pour optimiser l'exécution de la simulation dans le cadre distribué, il est nécessaire de limiter le nombre de points de synchronisation tout en prenant garde au maintien de la cohérence de la simulation. Ce problème de maintien de cohérence au sein de l'ordonnanceur global fait référence aux nombreux travaux sur la causalité des évènements au sein d'un système distribué. Le principe de causalité affirme que si un phénomène (appelé *cause*) produit un autre phénomène (appelé *effet*) alors *l'effet ne peut précéder la cause.*

A partir de cette définition du principe de causalité, l'exécution d'un agent au plus tôt dans le temps réel se traduit par : "Un agent peut être activé par l'ordonnanceur dès que celui-ci peut déterminer qu'il n'existe aucune relation de causalité entre cet agent et ceux, encore non-activés, qui le précèdent sur l'axe temporel simulé."

La considération des liens de causalité entre entités donne naissance à deux types d'ordonnanceurs dans le cadre de l'exécution parallèle de simulations à évènements discrets (PDES : *Parallel Discrete Event Simulation*) : les ordonnanceurs conservateurs et les optimistes [Quesnel *et al.*, 2003], [Fujimoto, 1990], [Ferscha and Tripathi, 1994]. En PDES, la simulation, considérée comme une suite d'évènements auxquels sont attachés des *timestamps*, est distribuée sur des *Logical Processes* (LP) qui assurent l'exécution d'une partie de la simulation et communiquent entre eux par l'envoi de messages réseau.

L'approche conservative consiste à favoriser le parallélisme sans violer le principe de causalité. La synchronisation pas à pas des LP, où un ordonnanceur global désigne l'unique *timestamp* que les LP peuvent exécuter, constitue la technique la plus basique de cette approche. De manière plus générale, les travaux dans le cadre de cette approche consistent à définir pour chaque LP le *timestamp* maximal qu'il pourra traiter sans entrainer de violation du principe de causalité. Ainsi, Lamport propose dans [Lamport, 1978] de définir une horloge logique propre à chaque LP et qui permet d'identifier les liens de causalité existant entre évènements exécutés sur deux LP distincts.

L'approche optimiste, introduite par Jefferson dans [Jefferson, 1985], n'impose aucune règle de synchronisation entre les LP : un LP ayant fini de traiter les évènements associés à un *timestamp*

passera automatiquement au traitement des évènements du *timestamp* suivant. Dans une telle configuration, il est possible qu'un LP reçoive un message, et donc un événement, possédant un *timestamp* inférieur à celui qu'il est en train de traiter. Afin d'éviter cette violation du principe de causalité, Jefferson propose de restaurer l'état du LP à la date à laquelle il aurait dû traiter le message afin que l'événement soit bien pris en compte dans la simulation. Cette restauration de l'état passé est appelée *rollback*. Il existe de nombreux travaux visant à optimiser la charge mémoire ou processeur induite par un *rollback* ou minimiser le nombre de *rollbacks* [Tand *et al.*, 2005], [Jefferson and Reiher, 1991], [Akyildiz *et al.*, 1993].

Considérant la large échelle, la simulation est composée de centaines de milliers d'agents en interaction entre eux et avec l'environnement. Ceci implique donc un nombre élevé de *rollbacks* potentiels dans le cas d'une approche optimiste. Considérant la complexité des algorithmes à déployer avec une telle approche, les ressources qu'elle nécessite et la probabilité élevée d'avoir un *rollback*, nous optons pour un ordonnanceur global basé sur l'approche conservative. Ce choix nous conduit à identifier les liens de causalité existant entre agents.

3.3 Identification des liens de causalité avec le modèle à Temporalité

3.3.1 Définition des liens de causalité dans notre modèle de simulation

Considérant un modèle simulé conçu sur le modèle à Temporalité, nous montrons ici que les liens de causalité entre agents sont à rechercher dans l'environnement et que ces liens peuvent se ramener à des liens de causalité entre temporalités.

Dans notre approche, les agents interagissent entre eux uniquement via l'environnement. Ils perçoivent leur environnement et y déposent des influences qui peuvent modifier ce dernier [Ferber, 1995]. Un agent a est influencé par un agent b si la perception de l'environnement faite par a diffère en fonction de l'activation antérieure de l'agent b. Si cette perception est différente, cela signifie que l'agent b a déposé une influence i qui a modifié une partie du monde perçu par a. Les mécanismes de perception/influence établissent des liens de causalité entre agents. Les perceptions et influences des agents sont réalisées dans le cadre de l'activation de temporalités. Ainsi, l'influence i a été produite lors de l'exécution d'une temporalité de b. De même, la perception faite par a est réalisée dans le cadre de l'activation d'une de ses temporalités. Le lien de causalité existant entre a et b se traduit par un lien de causalité entre temporalités associées à ces deux agents.

3.3.2 Recherche des liens de causalité

Les liens de causalité entre agents se traduisent par des liens de causalité entre temporalités. Ces liens entre temporalités prennent leurs sources dans deux faits :

(i) Les deux temporalités considérées ont été posées par le même agent. Les comportements associés à ces temporalités mettent en jeu les mêmes données de l'agent.

(ii) L'agent activé par la temporalité la plus avancée sur l'axe temporel simulé perçoit une partie de l'environnement qui a été influencée par l'agent activé par la temporalité antérieure.

Le fait (i) traduit le lien de causalité qui peut exister entre deux temporalités du même agent : l'exécution de la temporalité la moins avancée sur l'axe temporel simulé peut avoir un impact sur les données rattachées à l'agent qu'il utilisera lors de l'activation de la temporalité la plus avancée. Ne pouvant déterminer l'impact de l'exécution d'une temporalité sur les données internes d'un agent, nous généralisons le fait (i) en considérant qu'il existe un lien de causalité entre deux temporalités à partir du moment où ces temporalités ont été posées par le même agent.

Le fait (ii) permet d'établir les liens de causalité existant entre temporalités via l'environnement. Un agent, au cours d'une temporalité t, peut produire une influence conduisant à une modification de l'environnement. Toutes les temporalités qui seront sensibles à cette modification présentent une dépendance de causalité avec la temporalité t.

Pour déterminer ces liens de causalité, il faudrait donc connaître les zones d'influence et de perception associées à chaque influence ou perception de chaque agent. A moins que cette information ait été renseignée dans le modèle simulé, il est difficile de déterminer *a priori* les zones d'influence ou de perception. Nous pouvons toutefois élargir les conditions de détection des liens de causalité induits par le fait (ii) en considérant l'approche multi-environnement [Soulié, 2001], [Payet *et al.*, 2006b]. Cette approche consiste à décrire le monde dans lequel les agents évoluent par un ensemble d'environnements, chacun d'eux possédant des lois d'évolution propres. Un agent peut percevoir et influencer plusieurs environnements durant l'activation d'une temporalité. Disposant de plusieurs environnements, nous majorons les zones de perceptions et d'influences en considérant qu'elles s'étendent à l'intégralité de l'environnement support de la requête. Ceci nous permet de ramener le problème de l'étude des zones de perception et d'influence à des considérations sur les environnements.

Un des atouts de l'approche multi-environnement est une meilleure segmentation des différentes dynamiques du modèle : un environnement modélise une dynamique du système réel. Ainsi, à une temporalité donnée, au sein de laquelle l'agent traitera une dynamique particulière, le nombre d'environnements perçus et le nombre d'environnements influencés seront restreints. Par exemple, dans le modèle Biomas [Courdier *et al.*, 2002], un agent agriculteur aura une temporalité relatant son activité agricole et d'autres temporalités associées à ses négociations avec

la centrale de traitement.

Ainsi, l'approche multi-environnement, et notamment la modélisation orientée dynamique, nous permet de simplifier le fait (ii) en : "Il existe un lien de causalité entre deux temporalités à partir du moment où l'agent activé par la temporalité la plus avancée sur l'axe temporel simulé effectue une perception sur un environnement qui aura été influencé par un agent activé par une temporalité antérieure."

3.3.3 Formalisation

La connaissance des liens de causalité entre temporalités nous permet, en simulation, d'exécuter une temporalité dès que les temporalités dont elle dépend ont elles-mêmes été exécutées [Sébastien *et al.*, 2008a]. Le modèle à Temporalité et l'approche multi-environnement permettent de simplifier la recherche de liens de causalité. Nous considérons donc l'approximation suivante :

" Il existe un lien de causalité entre deux temporalités quand un des phénomènes suivants est constaté :

(i) Les deux temporalités considérées ont été posées par le même agent.

(ii) L'agent activé à la temporalité la plus avancée sur l'axe temporel simulé effectue une perception sur un environnement qui aura été influencé par un agent activé par la temporalité antérieure."

Les liens de causalité induits par le fait (i) sont simples à identifier : il suffit d'observer les temporalités déposées par les agents et identifier toutes celles posées par le même agent.

La condition (ii) nécessite de connaître des informations complémentaires sur le comportement des agents : les environnements potentiellement perceptibles et potentiellement influençables au cours de l'activation d'une temporalité d'un agent. Soit $E = \{E_1, E_2, ..., E_n\}$ l'ensemble des environnements du modèle simulé, on appelle :

- $E_R(a, t)$ l'ensemble des environnements perceptibles par l'agent a à la temporalité t (l'indice R est mis pour *read*).
- $E_W(a, t)$ l'ensemble des environnements influençables par l'agent a à la temporalité t (l'indice W est mis pour *written*).

Avec ces informations, nous pouvons formaliser la condition (ii). Soit t_{si} la valeur du temps simulé correspondant à l'activation courante de la temporalité t_i. On définit t_{sg} comme étant la valeur de temps simulé maximale telle que toutes les temporalités t_l qui vérifient $t_{sl} \leq t_{sg}$ ont été exécutées. En notant A l'ensemble des agents et $TE(a)$ l'ensemble des temporalités posées par l'agent a, une temporalité pourra être exécutée en respect avec la condition (ii) dès que le t_{sg} sera tel que

$$E_R(a,t_i) \cap \left(\bigcup_{(k \in A, t_j \in TE(k) \cap \{t_l, t_{sg} < t_{sl} < t_{si}\})} E_W(k,t_j) \right) = \oslash$$

Notons que l'ensemble $\{t_k, t_{sg} < t_{sk} < t_{si}\}$ correspond à l'ensemble des temporalités susceptibles de ne pas avoir encore été exécutées et possédant un temps inférieur à celui de la temporalité considérée.

L'emploi du modèle à Temporalité et de l'approche multi-environnement permet d'écrire les deux inégalités suivantes :

- $\forall a \in A, \forall t \in TE(a), \text{card}(E_R(a,t)) \leq \text{card}(E)$
- $\forall a \in A, \forall t \in TE(a), \text{card}(E_W(a,t)) \leq \text{card}(E)$

La modélisation orientée dynamique favorise donc la minimisation de $\text{card}(E_R(a,t))$ et $\text{card}(E_W(a,t))$. Plus ces cardinaux sont faibles, plus il sera possible de paralléliser l'exécution du modèle simulé au regard de la condition (ii).

3.4 Mise en oeuvre de l'ordonnanceur global

Devant le grand nombre de temporalités qu'une simulation à grande échelle implique (il y a au moins une temporalité par agent), il serait coûteux de gérer les liens de causalité entre temporalités. Nous reprenons alors toute la démarche précédente mais en considérant ici les tempos. On pose alors pour le tempo T :

- $E_R(T) = \bigcup_{a \in A, t_l \in T \cap TE(a)} E_R(a,t_l)$ l'ensemble des environnements perceptibles au cours de l'exécution du tempo T.
- $E_W(T) = \bigcup_{a \in A, t_l \in T \cap TE(a)} E_W(a,t_l)$ l'ensemble des environnements influençables au cours de l'exécution du tempo T.

On définit pour le tempo T_i, T_{si} comme étant la valeur du temps simulé de l'activation courante du tempo T_i. Nous définissons également T_{sg} comme étant la valeur maximale de temps simulé telle que tous les tempos T_l qui vérifient $T_{sl} \leq T_{sg}$ ont déjà été exécutés. On a de façon immédiate que si $t_j \in T_i$ alors $t_{sj} = T_{si}$; de là, nous avons également $T_{sg} = t_{sg}$. Ainsi, le tempo T_i pourra être déclenché en accord avec la condition (ii) lorsque le t_{sg} sera tel que

$$E_R(T_i) \cap \left(\bigcup_{T_j \in \{T_l, t_{sg} < t_{sl} < t_{si}\}} E_W(T_j) \right) = \oslash$$

Dans le modèle à temporalité, les agents sont invités à déposer leurs temporalités avant le lancement de la simulation. Cette phase déclarative conduit à une organisation de l'axe temporel simulé correspondant à l'agencement initial des slots temporels et des tempos. Celle-ci va nous

permettre d'établir une table des dépendances entre temporalités qui, au cours de la simulation, devra être révisée si un agent modifie ses temporalités.

3.4.1 Table de dépendances

La table de dépendances s'établit à partir des tuples $\langle E_R(T)|E_W(T)\rangle$ de chaque tempo. Elle est construite à l'aide de l'algorithme suivant :

```
pour chaque tempo Tk faire
        pour chaque environnement El perçu
        au cours de l'exécution de Tk faire
                identifier les tempos Ti
                au cours desquels El est influencé
        fin pour
fin pour
```

Cet algorithme permet de déterminer les liens de causalité existants entre tempos via les environnements. La table de dépendances est ensuite complétée en y intégrant les liens de causalité existant entre tempos contenant des temporalités appartenant aux mêmes agents.

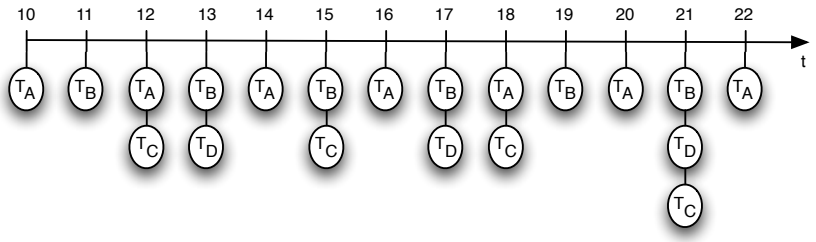


FIG. 3.4 – Exemple d'exécution asynchrone

Nous considérons l'axe temporel de la figure 3.4, sur lequel nous n'avons représenté que les tempos. Dans cet exemple, chaque agent ne possède qu'une seule temporalité (il n'y a donc pas de lien de causalité induit par le fait (i)). Nous désignons par les tuples $\langle E_R(T)|E_W(T)\rangle$ les environnements perçus et influencés à chaque tempo :

T_A : $\langle env_1|env_1\rangle$

T_B : $\langle env_2|env_2\rangle$

T_C : $\langle env_2|env_1\rangle$

T_D : $\langle env_1, env_2|env_1\rangle$

La nature des deux environnements considérés ici importe peu. En effet, les mécanismes présentés auparavant n'imposent aucune connaissance sur la nature de l'environnement. Nous déduisons la table de dépendances de l'axe temporel donné en exemple, la dépendance étant notée par le symbole "$\rightarrow$" :

$T_A \rightarrow T_C \wedge T_D$

$T_B \rightarrow \oslash$

$T_C \rightarrow T_B$

$T_D \rightarrow T_A \wedge T_B \wedge T_C$

L'exécution d'un tempo T_k sera possible quand les tempos dont il dépend auront été exécutés de telle manière que leur prochain instant d'activation sera supérieur ou égal à l'instant d'activation T_{sk} du tempo considéré : dans l'exemple figure 3.4, l'exécution de T_D à $t = 17$ dépend de l'exécution de T_A à $t = 16$, T_B à $t = 15$ et T_C à $t = 15$.

La table restera valide tant qu'aucun agent ne modifiera ses temporalités. La question de la manipulation des temporalités des agents est critique pour l'ordonnancement. Considérons un exemple : l'ordonnanceur local propose l'activation d'une temporalité t_2 percevant l'environnement E_1 alors qu'une temporalité t_1, avec $t_{s1} < t_{s2}$ n'influençant pas E_1 doit encore être exécutée. Plus tard, au cours de l'exécution du comportement associé à t_1, l'agent pose une temporalité t_3 avec $t_{s3} < t_{s2}$ et au sein de laquelle il va influencer E_1. Il y a ici une violation du principe de causalité : t_3 aurait dû être activée avant t_2.

Ce problème de détection induit par la modification des temporalités peut être résolu en déterminant les temporalités au sein desquelles un agent pourra modifier ses temporalités.

Dans notre plate-forme de simulation agent, GEAMAS-NG, le modèle à Temporalité est représenté par un *environnement temporel*. Ainsi, les agents posent leurs temporalités en influençant cet environnement. L'activation d'une temporalité constitue, elle, une perception de l'environnement temporel. Le problème induit par la modification des temporalités se ramène à la détection de liens de causalité via un environnement. Les liens de causalité qui peuvent être induits par la modification de l'environnement temporel sont donc identifiés de la même manière que pour les autres environnements. La table des dépendances prend donc naturellement compte des perceptions/influences sur l'environnement temporel.

3.4.2 Complexité de calcul de la table de dépendances

La table de dépendances est construite en deux étapes distinctes :

1. identification des liens de causalité existants entre tempos au travers des environnements (fait ii).
2. identification des liens de causalité existants entre deux tempos contenant des temporalités appartenant à un même agent (fait i).

Le calcul de la complexité de chacune de ces étapes est fonction du nombre d'environements perçus/influencés au cours de chaque tempo et du nombre d'agents devant être activés à chaque tempo. Dans notre cadre théorique, nous nous proposons de définir la borne supérieure de cette

complexité : nous considérons que tous les environnements sont perçus/influencés à chaque tempo et que tous les agents sont activés à chaque tempo. Il faut noter que dans un tel cas la détermination de la table de dépendances n'a aucun intérêt puisque tous les tempos dépendent alors les uns des autres.

On pose :

- n le nombre de tempos.
- $l = \text{card}(E)$ le nombre d'environnements.
- $g = \text{card}(A)$ le nombre d'agents

Ceci nous permet de donner la borne supérieure de la complexité à chaque étape de la construction de la table de dépendances :

1. identification des liens de causalité existants entre tempos au travers des environnements : n^2l^2.
2. identification des liens de causalité existants entre deux tempos contenant des temporalités appartenant à un même agent : n^2g^2.

Ainsi l'algorithme de construction de la table de dépendances est de complexité polynomiale et son majorant est $n^2(l^2 + g^2)$.

Le fait que cette complexité fasse intervenir g^2, c'est-à-dire le nombre d'agents au carré, semble en contradiction avec notre objectif d'utiliser cet algorithme dans le cadre de simulations large échelle. Il faut cependant considérer les faits suivants :

- Le terme g^2 correspond à la comparaison des identifiants des agents d'un tempo avec ceux d'un autre : dans notre calcul, nous avons considéré que tous les agents étaient exécutés à chaque tempo. En pratique le terme g^2 correspond au produit du nombre d'agents dans chacun des tempos considérés.
- Cette comparaison peut avantageusement être remplacée par celles proposées par l'environnement de programmation (bibliothèques spécifiques) ou par des algorithmes spécialisés.
- La table de dépendances est définie en début de simulation et reste valide tant que l'environnement temporel ne subit pas d'influences. La modification de l'environnement temporel est donc à limiter dans la conception du modèle de simulation.

3.4.3 Fonctionnement en simulation

Nous considérons un ordonnanceur global permettant l'exécution distribuée et parallèle. Un tempo peut être exécuté sur plusieurs noeuds : les temporalités qui lui sont associées sont réparties sur l'infrastructure d'exécution. Au cours de la simulation, les plates-formes de SOA composant l'infrastructure diffusent des messages de fin d'exécution de tempo.

On appelle ligne de dépendances d'un tempo T_y l'ensemble des tempos dont T_y dépend.

Ainsi, une ligne de dépendances correspond à une ligne de la table de dépendances. La ligne de dépendances d'un tempo T_y est satisfaite si et seulement si pour tout tempo T_z de cette ligne de dépendances, on a $T_{sy} \leq T_{sz}$. Aussi, on dit qu'un tempo T_z valide la ligne de dépendances d'un tempo T_y si et seulement si T_y dépend de T_z et $T_{sz} \geq T_{sy}$.

En reprenant la table des dépendances établie pour l'axe temporel de la figure 3.4, nous détaillons le fonctionnement attendu. Les cases en gris clair représentent les tempos qui valident une dépendance. Les tempos en gris foncé dans la première colonne sont les tempos en cours d'exécution. Considérant l'axe temporel à partir de $t_{sg} = 10$, certaines dépendances de la table sont déjà validées.

A $t_{sg} = 10$

A cet instant de la simulation, le prochain instant de déclenchement de chaque tempo est le suivant : $T_{sA} = 10$, $T_{sB} = 11$, $T_{sC} = 12$ et $T_{sD} = 13$. La table de dépendances est donc la suivante :

T_A	T_C	T_D	
T_B			
T_C	T_B		
T_D	T_A	T_B	T_C

Ayant supposé que tous les tempos T_k avec un $T_{sk} < 10$ ont été exécuté, on a $T_{sC} = 12$ et $T_{sD} = 13$ tous les deux supérieurs à $T_{sA} = 10$, T_C et T_D valident donc la ligne de dépendances de T_A qui est alors exécuté. Dans la mesure où T_B ne dépend d'aucun tempo, T_B peut être exécuté en parallèle de l'exécution de T_A. Le t_{sg} passera directement à $t_{sg} = 12$ puisque le slot temporel à $t = 11$ ne contenait que T_B qui a déjà pu être exécuté.

A $t_{sg} = 12$

Le prochain instant d'activation de chaque tempo est maintenant le suivant : $T_{sA} = 12$, $T_{sB} = 13$, $T_{sC} = 12$ et $T_{sD} = 13$. La table de dépendances est donc la suivante :

T_A	T_C	T_D	
T_B			
T_C	T_B		
T_D	T_A	T_B	T_C

Ayant $T_{sB} > T_{sA}$ et $T_{sC} \geq T_{sA}$, T_A peut être exécuté. De même, la précédente exécution de T_B valide la ligne de dépendances de T_C qui est maintenant exécuté. Enfin, T_B peut lui aussi être exécuté.

A $t_{sg} = 13$

Le prochain instant d'activation de chaque tempo est maintenant : $T_{sA} = 14$, $T_{sB} = 15$, $T_{sC} = 15$ et $T_{sD} = 13$. La table de dépendances est donc la suivante :

T_A	T_C	T_D	
T_B			
T_C	T_B		
T_D	T_A	T_B	T_C

Les dépendances de T_D sont toutes validées et conduisent à son exécution. De plus, t_B ayant été exécuté de nombreuses fois, son T_sB est suffisamment élevé pour que T_C puisse être exécuté avec $T_sC = 15$ (on a bien $T_sB \geq T_C$). T_B est lui aussi exécuté.

A $t_{sg} = 14$

Le prochain instant d'activation de chaque tempo est maintenant le suivant : $T_{sA} = 14$, $T_{sB} = 17$, $T_{sC} = 18$ et $T_{sD} = 17$. La table de dépendances est donc la suivante :

T_A	T_C	T_D	
T_B			
T_C	T_B		
T_D	T_A	T_B	T_C

La ligne de dépendances de T_A étant satisfaite, ce tempo sera exécuté. Comme T_B avec $T_sB = 15$, le t_{sg} passera directement à $t_{sg} = 16$.

Nous avons pu constater sur cet exemple que le tempo T_B ne dépend pas des autres tempos et peut donc être exécuté quelque soit l'état de la simulation. Néanmoins, les influences qu'il aura produites au cours de ses exécutions seront estampillées et mises en attente : elles ne seront traitées par les environnements concernés que lorsque ceux-ci seront perçus aux instants simulés où elles seront censées avoir produit leurs effets.

3.4.4 Exécutabilité de la simulation

La table de dépendances explicite les liens de dépendance existant entre tempos. Cette table est exploitée en simulation : elle empêche l'exécution d'un tempo tant que ses dépendances ne sont pas satisfaites. Dans la mesure où le simple accroissement du temps simulé ne semble plus être suffisant pour provoquer l'exécution d'un tempo, on pourrait se demander s'il n'existe pas de situations où aucun tempo ne peut être exécuté, bloquant alors l'exécution de la simulation.

Nous montrons ici qu'une telle situation est impossible dans la mesure où les dépendances explicitées dans la table de dépendances constituent un sous-ensemble des dépendances temporelles qu'implique un ordonnancement « usuel » (l'ordonnancement présenté au paragraphe 3.2.2) de la simulation.

Afin de faciliter la compréhension, nous considérons l'axe temporel de la figure 3.4. Pour activer l'exécution du tempo T_D à $t = 17$, l'ordonnanceur usuel attend la fin de l'exécution des tempos se trouvant à $t = 16$. De la même manière, l'exécution des tempos à $t = 16$ dépend de l'exécution de ceux se trouvant à $t = 15$ et ainsi de suite. Il est possible d'établir la table des dépendances usuelles associée à cet ordonnancement de la simulation : l'exécution d'un tempo

dépend de l'exécution de tous les tempos le précédant. Dans notre exemple, nous avons donc la table des dépendances usuelles suivante :

T_A	T_B	T_C	T_D
T_B	T_A	T_C	T_D
T_C	T_A	T_B	T_D
T_D	T_A	T_B	T_C

Notre algorithme de construction permet d'éliminer des dépendances de la table de dépendances usuelles. Notons de plus que, par construction de la table, un tempo ne peut dépendre que de tempos dont l'instant d'activation est strictement inférieur au sien. Nous montrons alors par récurrence qu'à tout instant simulé t_{sg} donné, il est possible d'exécuter au moins les tempos du prochain slot temporel en utilisant notre algorithme d'exécution parallèle.

A $t_{sg} = 0$, toutes les dépendances sont satisfaites (l'initialisation a été faite), le premier slot temporel peut donc être exécuté. Considérons maintenant que la simulation ait pu être exécutée jusqu'à $t_{sg} = t_n$, cela signifie que tous les tempos T_i tels que $T_{si} \leq t_n$ ont été exécutés. Ces tempos exécutés, ils satisfont les dépendances des tempos devant s'activer à t_{n+1} dans la table de dépendances usuelles et donc nécessairement dans la table de dépendances que nous proposons (celle-ci ne contenant qu'un sous-ensemble des dépendances de la première). Ceci nous permet d'affirmer qu'avec notre algorithme, à tout instant t_{sg}, il est possible d'exécuter au moins les tempos du prochain slot temporel. Notre ordonnancement parallèle ne peut donc pas conduire à un blocage de la simulation.

3.5 Discussion

3.5.1 Intérêt de notre approche : exemple de couplage de modèles simulés

Nous nous plaçons dans le cadre de la simulation Biomas [Courdier *et al.*, 2002]. Ce modèle, permet de simuler les flux de transfert de matières organiques entre exploitations agricoles localisées au sein d'un territoire. L'objectif est d'aider les acteurs agricoles à expérimenter de multiples options de gestion collective par le biais de simulations, à l'instar du projet Farmscape mené par le CSIRO australien [McCown, 2002].

Après une phase préliminaire de mise au point et de test du modèle sur différents cas d'école, nous avons utilisé ce SMA pour aider les conseillers agricoles de Grand-Ilet (cirque de Salazie, La Réunion) à proposer des solutions de gestion collective. Cette localité présente, en effet, une situation particulièrement critique du point de vue de l'environnement car elle rassemble sur une superficie agricole très réduite ($187ha$) le quart de la production porcine réunionnaise générant $17.000m^3$/an de lisier. Nous avons aussi employé ce SMA pour la zone du Petit Tampon, productrice de cannes à sucre, et donc potentiellement consommatrice de lisier. Ainsi, nous souhaitons étudier les échanges de matières organiques qui peuvent être établis entre ces zones

géographiques. D'un point de vue simulation, nous avons donc deux simulations autonomes qui n'interagissent entre elles que pour négocier des échanges de nitrate : ces négociations ont lieu dans la période de plantation de la canne à sucre.

Pour exécuter un tel système, nous disposons de deux noeuds de simulation, chargés respectivement de la simulation de la zone de Grand-Ilet et la simulation de la zone du Petit Tampon. Ces deux modèles disposent chacun de leur propre environnement, qui se résume à un unique environnement à deux dimensions géographiquement délimité (la communication n'est pas représentée sous forme d'environnement dans cette version de Geamas). Les échanges entre les agents des deux noeuds de simulation se fait par envoi de messages.

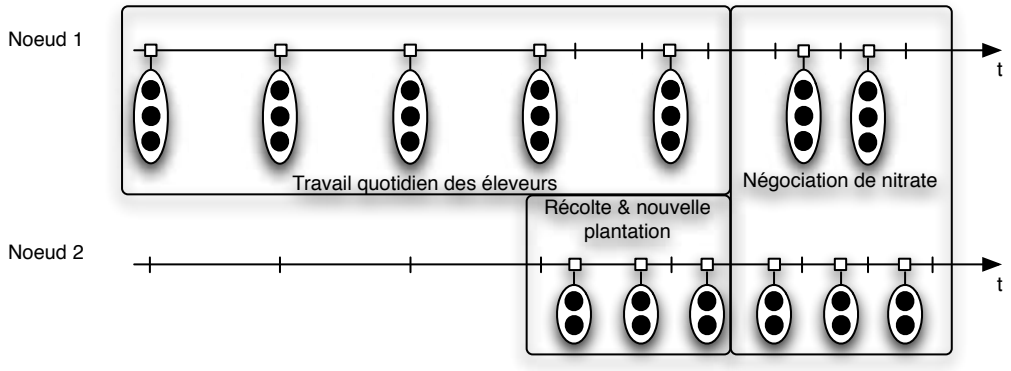


FIG. 3.5 – Axes temporels du couplage de Biomas sur Grand-Ilet et le Petit Tampon

Les éleveurs de Grand Ilet possèdent des temporalités journalières représentant leur travail quotidien. Au niveau des agriculteurs du Petit Tampon, l'activité n'est intense qu'en période de récolte et nouvelle plantation de la canne. Ceci nous conduit aux axes temporels simulés de la figure 3.5.

Pour mieux percevoir l'intérêt de notre approche, nous présentons graphiquement les durées d'exécution de chaque noeud de simulation. Nous prenons comme référence une première exécution basée sur le modèle temporel à pas de temps constant (figure 3.6). La synchronisation des plates-formes est faite après chaque pas de temps. Nous illustrons ensuite sur la figure (figure 3.7) l'exécution de cette même simulation en utilisant le modèle à Temporalité avec la synchronisation décrite à la section 3.2.3. Enfin, nous considérons en figure 3.8 le fonctionnement obtenu avec l'ordonnanceur parallèle que nous proposons.

Légende :

Nœud de simulation exécutant des agents éleveurs

Simulation en pause : plate-forme en attente ou en diffusion de messages

Nœud de simulation exécutant des agents agriculteurs

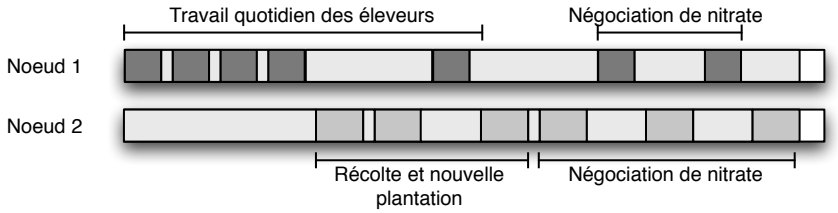


FIG. 3.6 – Modèle à pas de temps constant, synchronisation pas par pas.

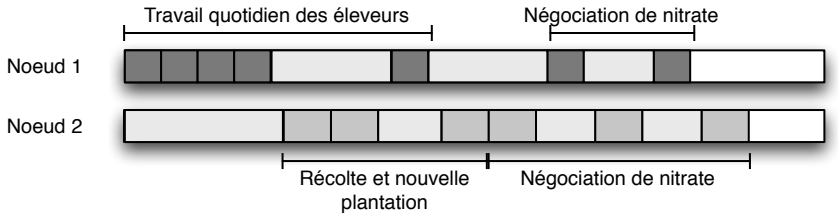


FIG. 3.7 – Modèle à Temporalité, synchronisation minimale.

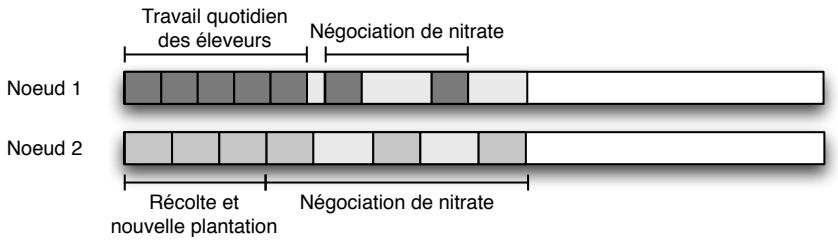


FIG. 3.8 – Modèle à Temporalité, fonctionnement parallèle.

Dans cet exemple, la synchronisation à chaque pas, couplée à l'utilisation du modèle à pas de temps constant, (figure 3.6) conduit à de nombreuses pauses dans l'exécution de la simulation : en début de simulation, seule la plate-forme 1 a des traitements à effectuer, mais elle marque quand même une pause pour se synchroniser avec la plate-forme 2 (il s'agit d'un simple échange de messages réseaux où chacune signifie à l'autre qu'elle a terminé le pas de temps constant).

La synchronisation proposée à la section 3.2.3 offre des performances légèrement meilleures (figure 3.7) : la plate-forme 2 diffuse en début de simulation le premier instant simulé où elle aura un slot temporel à exécuter. La plate-forme 1 peut faire avancer le temps simulé jusqu'à cette valeur de temps simulé en diffusant au fur et à mesure les nouveaux messages de fin d'exécution de slot temporel. Le gain de vitesse est minime puisqu'il permet juste de réduire le nombre d'attentes de synchronisation.

Dans cet exemple, l'algorithme d'exécution parallèle que nous avons défini se montre très efficace (voir figure 3.8) : il n'existe des temps d'attente que lorsque les deux nœuds de simulation voient leurs agents entamer des négociations.

Mais si notre algorithme d'exécution parallèle ne nécessite aucune considération particulière pendant l'étape de conception, les performances de l'algorithme dépendent beaucoup du modèle à exécuter. En effet, dans l'exemple présenté ici, agriculteurs et éleveurs ont des dynamiques distinctes et leurs interactions sont localisées dans le temps. Pour tirer au mieux parti de l'ordonnancement parallèle que nous proposons, il faut donc veiller à identifier et isoler au mieux les dynamiques du système à modéliser, notamment en suivant l'approche proposée dans [Payet *et al.*, 2006b].

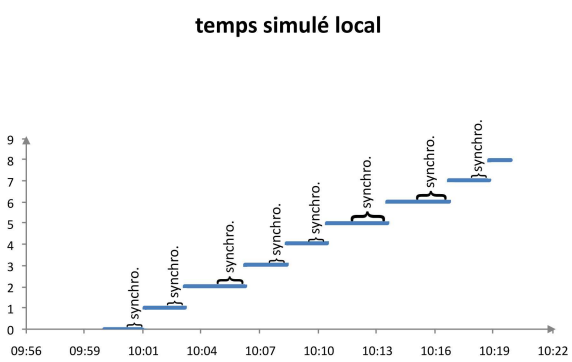
Cet exemple montre de plus que notre proposition s'adapte très aisément à des simulations qui n'emploient pas l'approche multi-environnement. L'algorithme que nous avons décrit repose sur le modèle à Temporalité et nécessite de borner la portée des perceptions et influences : l'approche multi-environnement ne constitue qu'une méthode permettant de déterminer ces bornes. Toute plate-forme supportant le modèle à Temporalité peut donc proposer ce fonctionnement parallèle.

3.5.2 Temps simulés multiples

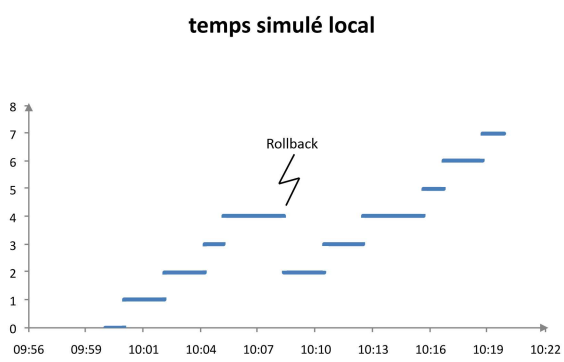
Une simulation orientée agent exécutée en mono-poste et mono-processeur définit un unique temps simulé croissant au cours du temps réel (voir figure 3.1). L'ordonnanceur de la plate-forme gère l'évolution de ce temps.

Dans un cadre distribué, l'exécution de la simulation s'appuie sur plusieurs ordonnanceurs locaux synchronisés par un ordonnanceur global. Ces ordonnanceurs locaux possèdent chacun un temps simulé et l'ordonnanceur global, dans sa stratégie de synchronisation des plates-formes, peut lui aussi définir un temps simulé qui lui est propre. L'ordonnanceur global a pour rôle de gérer l'avancée du temps de simulation global, il peut contraindre la valeur de temps simulé sur chaque plate-forme selon sa stratégie.

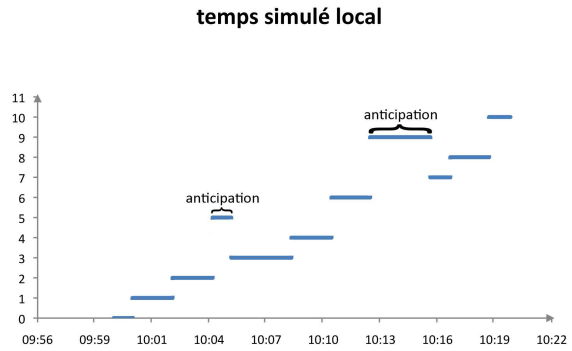
Par exemple, pour le fonctionnement synchronisé basique présenté au paragraphe 3.2.3, l'ordonnanceur global empêchera la plate-forme i d'exécuter le slot temporel $Slot(i, j+1)$ (et donc d'avancer au temps simulé $t_{Slot}(i, j+1)$) tant que les autres plates-formes simuleront des instants simulés antérieurs. A l'inverse, dans les approches optimistes, l'ordonnanceur global ne contraint que faiblement le temps simulé sur chaque plate-forme : chaque plate-forme exécute la simulation librement, incrémentant alors leur temps simulé. Ce n'est que lorsqu'une violation de causalité est détectée que l'ordonnanceur global va imposer un temps simulé à la plate-forme où la violation de causalité se produit (ce temps simulé sera bien sûr inférieur au temps simulé auquel la plate-forme était arrivée).



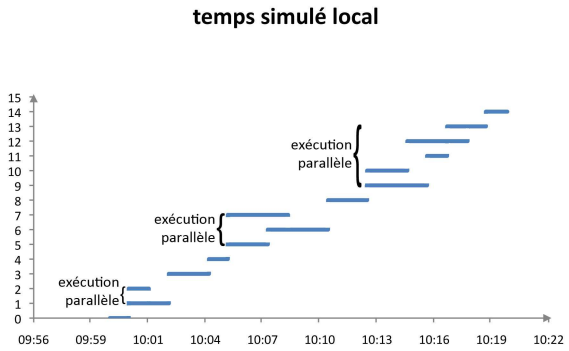
(a) Ordonnancement conservatif



(b) Ordonnancement optimiste



(c) Ordonnancement que nous proposons exécuté localement sur un seul *thread*



(d) Ordonnancement que nous proposons exécuté localement sur deux *threads*

FIG. 3.9 – Evolution du temps simulé par rapport au temps réel selon le type d'ordonnancement global

Il est intéressant de considérer l'impact de la stratégie d'ordonnancement global sur l'évolution du temps simulé d'une plate-forme au cours du temps. Pour une synchronisation basique, le temps simulé local est croissant (voir figure 3.9(a)) ; la synchronisation induit des temps d'attente. Les approches optimistes permettent de faire évoluer la simulation au rythme propre de chaque plate-forme mais l'ordonnanceur global force des retours vers des temps simulés passés (*rollbacks*) quand une violation du principe de causalité est détectée : l'évolution du temps simulé est non

monotone, plus précisément croissant par parties (voir figure 3.9(b)). Notre proposition est considérée comme une approche conservative au sens où nous évitons toute violation du principe de causalité mais permet une évolution non monotone du temps simulé sur une plate-forme. Comme plusieurs tempos possédant des instants d'activation différents peuvent être exécutables en même temps, deux cas illustrant notre approche sont à considérer : dans celui de la figure 3.9(c), la plate-forme ne dispose que d'un seul *thread* pour exécuter la simulation tandis que dans le cas de la figure 3.9(d), plusieurs *threads* participent à l'exécution locale de la simulation. Dans ce dernier cas, cela signifie qu'il peut exister plusieurs temps simulés sur la même plate-forme.

L'évolution du temps sur une plate-forme participant à la simulation distribuée constitue en elle-même un phénomène digne d'observation. Cependant, il y a autant de temps simulés qu'il y a de plates-formes participant à la simulation. La prise en compte de ces multiples temps de simulation est délicate pour les utilisateurs. Ainsi Mattern s'intéresse dans [Mattern, 1989] à la définition de *snapshots* : ces snapshots permettent à l'utilisateur d'avoir une vue complète de l'état de la simulation à un instant donné même si certains ordonnanceurs locaux ont déjà simulé des instants supérieurs.

Dans notre proposition où nous permettons l'exécution « anticipée » de certains tempos, nous rencontrons le même phénomène. Il n'est possible d'afficher l'état complet de la simulation à l'instant simulé t que lorsque toutes les plates-formes auront exécuté tous les tempos T_i tels que $T_{si} \leq t$. Cette condition correspond au T_{sg} défini au début du paragraphe 3.4. Ce T_{sg} est donc considéré comme le *temps simulé global*, temps simulé associé à l'ordonnanceur global. Ce temps simulé global est inférieur ou égal aux temps simulés des ordonnanceurs locaux et, contrairement à ces derniers, il est croissant dans le temps réel comme l'est le temps simulé dans le cadre d'une simulation mono-poste mono-processeur.

3.5.3 Simulation parallèle conservative

Comme souligné en section 3.2.3, il existe plusieurs travaux visant la réalisation d'ordonnanceurs parallèles conservatifs. Si [Lamport, 1978] constitue une référence en la matière, nous pouvons citer ici les travaux sur Parsec [Bagrodia *et al.*, 1998] où les auteurs décrivent des variables servant aussi bien une stratégie conservative qu'optimiste :

- Le EOT, *Earliest Output Time*, est l'instant simulé le plus petit auquel un Logical Process sera susceptible d'envoyer un message.
- Le EIT, *Earliest Input Time*, est l'instant simulé le plus petit auquel le LP sera susceptible de recevoir un message.
- Le *lookahead* est l'intervalle temporel simulé au sein duquel le LP peut déterminer les évènements qu'il va générer.

Avec ces variables, un LP peut traiter ses évènements dont le *timestamp* est inférieur au EIT sans risque de violation de la causalité (sachant qu'aucun message n'arrivera avant l'EIT, le LP

possède tous les évènements à traiter jusqu'à l'EIT). La mise à jour du EIT en cours de simulation se fait via la connaissance des EOT des autres LP. Chaque LP va donc propager ses changements et provoquer des mises à jour des EIT et EOT des autres LP : ceci permet de faire avancer la simulation. Cette approche présente un premier point commun avec notre proposition : elle se base sur des éléments du modèle simulé pour déterminer les variables nécessaires au fonctionnement distribué.

Cependant, dans ces travaux, comme dans [Lamport, 1978], la prise en compte de la distribution doit se faire dès la conception du modèle de simulation, ce qui surcharge le modèle avec des caractéritisques ne provenant pas du système que l'on souhaite étudier. L'intérêt de notre approche réside dans le fait qu'elle s'appuie sur la connaissance du modèle simulé pour établir, sans modification de ce dernier, les liens de causalité.

Lubachevsky définit dans [Lubachevsky, 1988] la notion de fenêtre temporelle (*Time Windows*). Les fenêtres temporelles ont pour but d'améliorer le parallélisme dans les simulations conservatives en tirant partie, par exemple, des architectures multi-processeurs. Un LP possède ici plusieurs fenêtres temporelles qui sont elles-mêmes constituées d'évènements devant être traités par le LP. Ces fenêtres temporelles sont définies de manière à ce qu'elles soient indépendantes : l'exécution d'une fenêtre temporelle d'un LP n'a pas d'influence sur l'exécution d'une autre fenêtre du même LP.

Ce découplage entre groupes d'évènements d'un même LP fait échos aux liens de dépendance que nous établissons entre tempos. A partir des liens de dépendance, nous pouvons effectivement déterminer des séquences de tempos : les tempos d'une telle séquence devront s'exécuter l'un après l'autre. De telles séquences présentent une certaine analogie aux fenêtres temporelles.

Dans la communauté Agent, certains travaux fournissent un cadre favorable à la définition d'ordonnanceurs conservatifs. Nous pouvons notamment citer [Piunti *et al.*, 2007] où les auteurs proposent d'utiliser des algorithmes d'apprentissage pour que les agents identifient des comportements récurrents chez les autres afin de coordonner leurs actions. Cette approche peut conduire à la réalisation d'un ordonnanceur global proche de celui que nous proposons à la différence près que cet algorithme serait basé sur un apprentisage propre à chaque agent sans pour autant pouvoir garantir la parfaite exécution conservative (un agent changeant son comportement pourrait induire un biais dans la simulation).

3.5.4 Détection de liens de dépendance

Dans nos travaux, nous avons montré que les influences et perceptions faites par les agents sur les environnements pouvaient entrainer des liens de causalité entre temporalités [Sébastien *et al.*, 2009]. Lees, faisant le même constat dans [Lees *et al.*, 2004], définit des sphères d'influence. S'inspirant des LP, il propose de spécifier des ALP (*Agent Logical Process*), ELP (*Environment Logical Process*) et CLP (*Communication Logical Process*) hiérarchisés dans une structure d'arbre. Les

ELP et ALP constituent les feuilles de l'arbre, les noeuds étant des CLP. Cette structure d'arbre permet de décomposer les variables partagées de l'environnement en LP. Un agent souhaitant percevoir/influencer une variable de l'environnement va s'adresser au CLP auquel il est rattaché : c'est ce CLP qui sera garant de la cohérence de la variable.

Cette approche constitue une des premières visant la distribution des environnements. En effet, Lees utilise la connaissances des sphères d'influences pour répartir dynamiquement les LP (ALP, ELP et CLP) sur différentes plates-formes. Dans le cadre de nos travaux, nous n'avons pas retenu cette approche car elle limite trop la définition des environnements qui sont alors réduits à des espaces de variables partagées. Elle peut toutefois être utilisée en complément de l'approche multi-environnement pour raffiner la détection des liens de causalité sur les environnements qui peuvent être modélisés sous forme d'espaces de variables partagées.

Dans [Holcombe *et al.*, 2006], Holcombe propose une approche intéressante pour la distribution d'environnements de communication et pour borner les zones de perceptions et d'influences dans ces environnements. Il s'intéresse à la notion de relation de communication : il définit des listes de messages localisées dans l'environnement. Les agents étant proches d'une même zone communiquent entre eux en utilisant la liste associée à la zone. Un agent se trouvant en bordure d'une zone adresse alors ses messages à la fois à la zone dans laquelle il se trouve encore mais aussi à celle dont il se rapproche.

Si cette proposition impose une topologie spécifique à l'environnement de communication, elle peut servir de base à un travail sur l'identification de réseaux de communication entre agents. La connaissance de ces réseaux de communication peut alors être utilisée dans notre algorithme d'exécution parallèle. De manière similaire aux travaux de Lees, cette approche est très pertinente pour les simulations dont les interactions sont essentiellement basées sur des échanges de messages entre agents mais reste très limitée dans un cadre général où les interactions ne se réduisent pas au simple échange de messages.

Bien que ne s'inscrivant pas dans un contexte agent, les travaux de Quaglia [Quaglia and Baldoni, 1999], sont proches de ceux présentés dans ce chapitre. Quaglia définit la notion de WEak Causality (WEC) qui modélise le parallélisme intra-objet. De manière très simple, WEC décrit l'existence d'une causalité interne à un objet si le non-respect de l'ordre temporel dans le traitement de deux évènements du même objet produit un état différent de celui attendu. Décrivant ses objets comme des ensembles de variables. Les évènements font évoluer l'état d'un objet. Lors de l'exécution d'un événement, certaines variables sont lues et d'autres mises à jour. Quaglia identifie les ensembles de variables lus et mises à jour à chaque événement et établit ainsi une liste de dépendances entre évènements. A partir de cette liste, il est possible de proposer plusieurs ordres d'exécution des évènements.

Nous pouvons établir des parallèles entre ces travaux et notre proposition : les évènements sont à rapprocher de nos temporalités, et l'observation faite sur les états de l'objet est semblable à l'observation faite sur les environnements de la simulation. Nos travaux s'inscrivent néanmoins

dans un cadre plus large que les relations entre évènements d'un même objet : les liens de dépendance établis dans notre proposition permettent l'exécution conservative de l'ensemble de la simulation. De plus, nous nous inscrivons dans un cadre beaucoup plus flexible dans la mesure où nous supportons l'ajout/suppression de temporalités via la manipulation de l'environnement temporel.

3.6 Synthèse

Si l'infrastructure distribuée n'a pas pour but premier de maximiser les performances de simulation mais bien de supporter un grand nombre d'agents en large échelle, elle s'appuie sur un ordonnancement global permettant d'améliorer les performances. Ainsi, nous avons défini un ordonnanceur global permettant de faire évoluer le temps simulé sur chaque plate-forme participant à la simulation. Cet ordonnanceur permet l'exécution parallèle de la simulation au travers d'une approche de type conservative (il n'y a pas de violation du principe de causalité). Il respecte de plus les exigences énoncées au paragraphe 2.2.3, notamment en ce qui concerne la qualité de la simulation et la portabilité des modèles.

Les approches conservatives usuelles n'offrent que peu de place à la parallélisation (synchronisation à chaque pas de temps) ou nécessitent une réécriture du modèle de simulation qui est en contradiction avec notre volonté de portabilité des modèles. Nous définissons donc notre propre ordonnanceur en se basant notamment sur les concepts temporels introduits par le modèle à Temporalité. Ce modèle temporel offre une description riche des différents comportements (périodiques ou non) que peuvent adopter les agents et admet pour cas limites le modèle à pas de temps constant et l'approche type événementiel.

Dans notre approche, toute interaction est réalisée par le biais d'influences/perceptions : les liens de causalité entre agents sont donc à identifier au travers de ces mécanismes d'une part, et au travers des modifications des états internes des agents (pour les liens de causalité entre deux temporalités du même agent) d'autre part. Comme il est délicat de délimiter les zones d'influences et de perceptions de chaque agent, nous exploitons l'approche multi-environnement (où le temps est lui-même représenté comme un environnement temporel dans lequel les agents manipulent leurs temporalités) et les informations associées aux temporalités pour établir une table de dépendances entre instants d'activation des agents. Cette table est exploitée durant la simulation pour permettre aux agents de s'exécuter le plus tôt possible dans le temps réel. Notre ordonnanceur dépasse ainsi le cadre usuel des ordonnanceurs conservatifs : l'évolution du temps simulé par rapport au temps de la réalité n'est plus une fonction monotone croissante dans la mesure où un tempo sera exécuté sur une plate-forme dès que ses dépendances seront vérifiées, indépendamment de la valeur de son prochain instant d'activation.

L'ordonnanceur que nous avons proposé exploite les informations et connaissances intégrées au modèle simulé pour permettre l'exécution parallèle. Il respecte donc la contrainte de portabilité

des modèles. Toutefois, les performances de l'ordonnanceur dépendent fortement de la rigueur avec laquelle le modèle de simulation aura été établi : le modélisateur doit isoler au mieux les différentes dynamiques du système qu'il modélise pour bénéficier au mieux du parallélisme entre ces dynamiques. Enfin, l'ordonnanceur ne fait aucune hypothèse sur l'architecture parallèle utilisée et peut donc être déployé sur des architectures multi-processeurs.

A travers l'exécution parallèle de tempos ayant des instants (simulés) d'activation différents, nous cherchons à maximiser localement l'exécution de la partie de simulation se trouvant sur une plate-forme. Cependant, maximiser les performances sur chaque plate-forme ne suffit pas à maximiser les performances de la SOA distribuée : la répartition de la charge de simulation sur l'infrastructure d'exécution joue un rôle critique. L'établissement d'une répartition optimale ou dynamique de la simulation constitue donc une étude complémentaire à celle que nous avons présentée dans ce chapitre.

Chapitre 4

Optimisation de SOA distribuées par équilibrage de charges

Sommaire

4.1 Cadre d'utilisation 82
4.2 Approche générale 83
4.2.1 Cadre usuel des algorithmes d'équilibrage de charges 83
4.2.2 Cadre général pour la définition d'un algorithme d'équilibrage de charges pour la SOA distribuée 85
4.3 Algorithme d'équilibrage de charges 86
4.3.1 Notion d'agents déplaçables 87
4.3.2 Un algorithme en trois phases 87
4.3.3 Exécution de la simulation à l'équilibre des charges 89
4.4 Description des phases de l'algorithme 90
4.4.1 Diffusion d'informations 91
4.4.2 Détection de surcharge 91
4.4.3 Transfert d'agents 92
4.4.4 Exécution de l'équilibrage de charges 94
4.4.5 Pistes de réflexion pour une sélection optimisée des agents de simulation à transférer 96
4.5 Adaptation à l'ordonnancement distribué séquentiel du modèle à Temporalité 97
4.5.1 Axe temporel simulé distribué 98
4.5.2 Identification de l'*unité logique de simulation* à distribuer 98
4.5.3 Anticipation de la charge 102
4.6 Application à l'ordonnancement distribué parallèle du modèle à Temporalité 103
4.7 Synthèse 107

Dans notre approche de simulation distribuée, l'utilisateur manipule le système global sous la forme d'une plate-forme virtuelle. Il fait notamment abstraction de toute considération liée à la répartition des agents de simulation sur l'infrastructure d'exécution. Ainsi, ces derniers sont répartis sur l'infrastructure d'exécution sans aucune considération sur les performances.

Nous proposons dans ce chapitre de définir un algorithme d'équilibrage de charges qu'emploieront les agents de flexibilité de notre architecture pour modifier dynamiquement la répartition des agents de simulation et améliorer ainsi les performances d'exécution de la simulation. Notre démarche dans ce chapitre sera de présenter notre algorithme d'équilibrage de charges dans le cadre d'un modèle temporel à pas de temps constant avant de nous intéresser à son adaptation dans le cadre de l'utilisation du modèle à temporalité et de l'ordonnancement parallèle que nous avons proposé.

4.1 Cadre d'utilisation

Nous avons vu au paragraphe 2.2.1 que l'infrastructure d'exécution sur laquelle nous souhaitons appuyer la simulation orientée agent distribuée possède sa propre dynamique : les performances des machines et du réseau la constituant peuvent varier et l'infrastructure peut être dynamiquement enrichie comme appauvrie en nombre de plates-formes participant à la simulation.

Afin de prendre en compte cette dynamique, nous proposons d'introduire un système multi-agents de distribution réparti sur les plates-formes de simulation (voir paragraphe 2.4). Ce SMA définit des agents non-mobiles s'exécutant sur chaque plate-forme et assurant toutes les fonctionnalités nécessaires à l'activité de simulation distribuée. Notamment, la prise en compte de la dynamique de l'infrastructure est assurée par l'agent de flexibilité (A_F) de chaque plate-forme : cet agent a pour but d'optimiser les performances de la plate-forme sur laquelle il se situe tout en tenant compte des performances générales de l'infrastructure d'exécution.

Les ressources systèmes allouées à une plate-forme sont limitées par sa machine hôte. L'agent de flexibilité ne peut alors faire varier les performances de son hôte qu'au travers de la variation de la charge de simulation. L'agent de flexibilité d'une plate-forme négocie donc avec les autres agents de flexibilité des transferts de parties de la simulation (agents et/ou environnements).

Les agents de flexibilité ont la capacité de modifier la répartition des agents de la simulation sur les plates-formes composant l'infrastructure d'exécution et ont pour charge d'optimiser les performances de l'infrastructure durant l'exécution de la simulation. Ils sont donc adaptés à la gestion de la dynamique d'infrastructure ; quand la topologie de l'infrastructure change, ils peuvent déterminer une nouvelle répartition de la simulation.

Ainsi, les agents de flexibilité assurent deux tâches :

– ils optimisent les performances globales de l'infrastructure pour l'exécution de la SOA distribuée.
– ils assurent la flexibilité de l'infrastructure d'exécution en révisant la répartition de la simulation en cas de connexion ou de déconnexion de plates-formes.

Ils réalisent ces deux tâches en utilisant le même mécanisme : ils observent d'une part leur plate-forme hôte et d'autre part l'infrastructure dans sa globalité et, à la suite de cette observation, proposent une nouvelle répartition de la simulation sur les plates-formes disponibles. Ainsi, nous proposons dans ce chapitre de définir un algorithme d'équilibrage de charges que les agents de flexibilité appliqueront pour assurer à la fois l'optimisation globale des performances et le support des connexions/déconnexions de plates-formes.

4.2 Approche générale

4.2.1 Cadre usuel des algorithmes d'équilibrage de charges

L'équilibrage de charges sur des réseaux d'ordinateurs constitue un thème de recherche à part entière. Il existe de nombreux travaux dans ce domaine qui touchent des types de problèmes variés. On retrouve ainsi de nombreux travaux visant à équilibrer la charge de calcul induite par l'analyse numérique multi dimensionnelle [Williams, 1991].

Par exemple, dans [Lan *et al.*, 2001] les auteurs se placent dans le cadre de la simulation par analyse numérique et propose un algorithme d'équilibrage de charges basé sur le découpage et la répartition des grilles de calcul : l'algorithme consiste à déplacer une des grilles de l'ordinateur le plus chargé vers l'ordinateur le moins chargé (manœuvre permise seulement si le déplacement ne fait pas dépasser le seuil d'utilisation normal à l'ordinateur le moins chargé). Si ce premier mécanisme ne suffit pas à équilibrer le système, l'ordinateur le plus chargé a la possibilité de fragmenter sa grille de calcul pour en transférer un fragment à l'ordinateur le moins chargé.

Mais ce type de simulation par analyse numérique est trop distant de la simulation orientée agent pour que nous puissions réutiliser ces travaux. En effet, le temps n'intervient pas dans l'application présentée ici alors qu'il constitue un élément fondamental dans nos simulations. De plus la charge est pour eux une simple maille : le réseau d'interactions entre les nœuds de la maille est parfaitement connu est maîtrisé alors que, dans les SOA, les agents de simulation jouissent d'une grande liberté dans leurs interactions. Le type et la finalité des travaux réalisés en analyse numérique différant grandement de ceux réalisés en SOA, les techniques d'équilibrage de charges dédiées à ce domaine peuvent difficilement être reprises dans le cadre de la SOA distribuée.

Le domaine de l'équilibrage de charges couvre également la répartition de tâches dans le cadre d'architectures telles que la *Grille* (voir paragraphe 1.4.1). Il est intéressant de constater que les algorithmes d'équilibrage de charges dans ce domaine font souvent intervenir des agents

intelligents mobiles.

Dans [Schaerf *et al.*, 1995], les agents mobiles sont chargés de l'exécution de tâches sur un réseau d'ordinateurs disponibles. Les ordinateurs acceptent systématiquement les tâches que les agents peuvent leur soumettre : un ordinateur performant peut donc voir ses performances chuter si tous les agents lui affectent des tâches. Un agent a donc deux stratégies : il peut adopter un comportement d'exploration où il va confier une tâche à un ordinateur dont il n'a jamais éprouvé les performances ou adopter un comportement d'exploitation en confiant une tâche à un ordinateur dont il estime les performances optimales. Les performances de ce système de base peuvent être améliorées en conférant aux agents des capacités additionnelles comme le partage de leur expérience (la connaissance des performances des machines) ou l'interrogation des machines.

L'algorithme *Messor* proposé dans [Montresor *et al.*, 2002] fait aussi appel à des agents mobiles intelligents échangeant des tâches sur un mode de fonctionnement de type *peer-to-peer*. Ces agents sont comparables à des fourmis et les ordinateurs à des nids. Les ordinateurs exécutent les tâches, certains d'entre eux étant surchargés par le nombre de tâches à traiter. Les agents fourmis se déplacent de nid en nid et laissent comme trace les identifiants des ordinateurs les plus proches de chaque nid (ce qui permet d'obtenir une topologie partielle de l'infrastructure d'exécution). Dans un premier temps les agents se dirigent vers les nids qu'ils estiment les plus chargés en tâches et se saisissent d'une tâche. Ils vont ensuite chercher le nid le moins chargé du réseau qu'ils perçoivent pour y déposer la tâche puis se remettent à la recherche du nid le plus chargé.

Ces algorithmes d'équilibrage de charges sont difficilement applicables à notre approche. En effet, ces derniers font intervenir des agents systèmes mobiles tandis que l'architecture que nous préconisons repose sur des agents systèmes attachés à chaque plate-forme : nos agents doivent rester associés à une plate-forme car ils ne sont pas dédiés uniquement à l'équilibrage de charges mais sont au centre de la distribution (ils assurent de nombreux services afin de répondre aux exigences que nous avons énoncées au paragraphe 2.2.3). S'il serait alors possible d'envisager l'adjonction d'agents mobiles dans notre architecture distribuée, cette solution n'aurait pas grand intérêt car le cadre d'utilisation de l'algorithme *Messor* diffère du nôtre. En effet, cet algorithme est intéressant à employer lorsque la topologie du réseau est inconnue. Or nous avons fait le choix d'utiliser un protocole de communication en diffusion entre les plates-formes : toutes les plates-formes sont connues, il n'y a donc pas besoin de considérer de phases exploratoires.

Le choix d'un protocole de communication en diffusion nous positionne sur un cas particulier d'équilibrage de charges puisque toutes les plates-formes ont connaissance de la topologie de l'infrastructure d'exécution. [Corradi *et al.*, 1999] étudie l'impact de la stratégie de communication entre plates-formes sur les performances et en conclut que la définition d'un algorithme de charges reste spécifique au type de problème adressé.

Une différence majeure de notre approche par rapport aux cadres des algorithmes présentés ici réside également dans le type de charges manipulées. Ces algorithmes sont appliqués à la

gestion de tâches indépendantes pouvant donc être exécutées de manière parallèle. La simulation orientée agent met certes en jeu des agents de simulation offrant une granularité facilitant la distribution, les agents n'en restent pas moins en interaction dans un temps propre à la simulation. Les dépendances entre agents sont telles que le déplacement trop fréquent d'agents pénalise l'exécution de la simulation, ces déplacements devant être fait lorsque la simulation est en pause (voir paragraphe 4.2.2). Ainsi, les algorithmes nécessitant de nombreux déplacements de tâches pour l'obtention d'une situation optimale sont à éviter.

4.2.2 Cadre général pour la définition d'un algorithme d'équilibrage de charges pour la SOA distribuée

L'équilibrage de charges d'une simulation orientée agents distribuée est une tâche complexe car les agents de la simulation à distribuer sur l'infrastructure d'exécution sont contraints par l'ordonnancement global de la simulation et développent de nombreuses interactions entre eux.

La modification de la répartition des agents de simulation sur l'infrastructure d'exécution ne devrait pas avoir d'impact sur la qualité de la simulation. Pourtant le déplacement d'un agent de simulation d'une plate-forme à une autre est une étape critique : l'agent « disparaît » de la simulation (ou est dupliqué) pendant la durée du transfert. Un agent pourrait donc ne pas être exécuté à cause de son déplacement. Pour éviter ce phénomène, il ne convient alors de procéder au transfert d'agents de simulation que lorsque la simulation n'est pas exécutée, c'est-à-dire durant les points de synchronisation et les temps d'attente. Ainsi, un agent est localisable sur une seule et unique plate-forme de l'infrastructure d'exécution à tout instant du temps simulé.

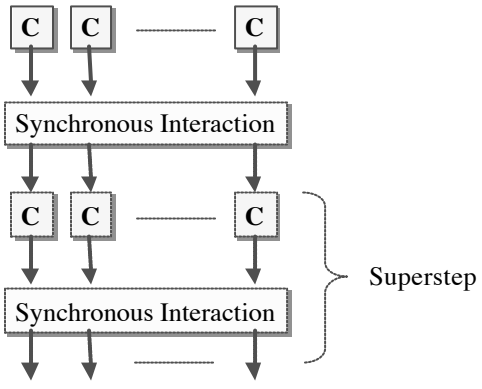


FIG. 4.1 – Exécution parallèle d'une application de type *phase parallel*, extrait de [Osman and Ammar, 2002].

Nous nous basons sur la classification proposée dans [Osman and Ammar, 2002] pour positionner le type d'algorithmes d'équilibrage de charges adapté à l'exécution de SOA sur notre architecture distribuée. L'exécution de la simulation orientée agent sur une infrastructure distribuée constitue une problématique de type *Phase parallel* : les plates-formes de simulation font

progresser la simulation par *superstep* (voir figure 4.1). Chaque *superstep* contient une phase purement parallèle, l'exécution des agents de la simulation, et une phase synchrone, la synchronisation des plates-formes pour mise à jour des environnements. Notons ici que l'exécution des agents de simulation est une opération purement parallèle dans la mesure où les environnements sont les supports des interaction. Les environnements n'étant mis à jour qu'à la phase de synchronisation, ils constituent une source d'informations statiques au cours de l'exécution des agents.

Les auteurs de [Osman and Ammar, 2002] suggèrent pour ce type de problème un algorithme aux caractéristiques suivantes :

- initialisation : événementiel, à chaque étape de synchronisation. Cela signifie que l'algorithme d'équilibrage de charges doit être déclenché après chaque étape de synchronisation.
- type d'organe d'équilibrage de charges : centralisé ou distribué synchrone. Souhaitant un fonctionnement non-centralisé, nous nous orientons vers une solution distribuée synchrone.
- échange d'information : devrait être global pour observer les différentes charges. La communication en diffusion que nous employons est donc le support idéal de communication dans ce cadre.
- identification de la charge à transférer : dépend de l'application.

Au chapitre 2, nous avons introduit plusieurs concepts pouvant être exploités pour l'élaboration d'un algorithme d'équilibrage de charges performant. Nous soulignons notamment la communication par diffusion qui facilite l'échange d'informations entre plates-formes. Les agents micro de distribution d'une plate-forme constituent aussi un atout à exploiter dans la stratégie d'équilibrage de charges dans la mesure où ces derniers permettent d'obtenir de nombreuses informations sur la plate-forme et la partie de simulation qu'elle exécute et qu'ils peuvent agir sur les différents composants de la plate-forme.

A partir des préconisations précédentes et des spécificités de notre architecture, nous nous orientons vers la définition d'un algorithme de type *Simple Diffusion Method* (SDM) [Luque *et al.*, 1995]. Dans cet algorithme, une plate-forme compare ses performances à la moyenne de ses voisins et effectue donc des transferts de charge pour équilibrer localement sa charge. Nous basant sur un protocole de communication en diffusion, ce sont toutes les autres plates-formes qui constituent le voisinage de la plate-forme considérée. Nous décrivons dans un premier temps le fonctionnement général de l'algorithme avant de nous intéresser aux détails des étapes clés.

4.3 Algorithme d'équilibrage de charges

Nous nous focalisons dans ce chapitre uniquement sur la répartition des agents de simulation sur l'infrastructure d'exécution : l'équilibrage de charges se fait par déplacement d'agents de la simulation. Nous ne modifions pas la répartition de l'environnement. Cette problématique est

certes d'une grande importance mais nécessite une étude spécifique.

4.3.1 Notion d'agents déplaçables

Nous avons vu au paragraphe 1.4.3 que les agents de simulation ne pouvaient être des agents mobiles. En effet, il pourrait être tentant de donner à ces agents la capacité de se déplacer sur l'infrastructure d'exécution pour qu'ils optimisent eux-même l'exécution de la simulation. Mais les agents de simulation embarqueraient alors des considérations aussi bien thématiques (liées à la simulation) que techniques (liées à leur exécution) ce qui complexifie grandement leur création aussi bien que leur exécution.

Le déplacement des agents de simulation d'une plate-forme à une autre reste cependant un mécanisme tout à fait pertinent pour réaliser l'équilibrage de charges. En effet, l'autonomie des agents facilite leur déplacement et leur granularité permet d'adapter finement la charge de simulation sur chaque plate-forme. Qui plus est, les mécanismes inhérents à la plate-forme virtuelle permettent d'assurer à l'agent de simulation de retrouver le même environnement dans lequel il évolue quelque soit la plate-forme qui l'exécute.

Si les agents de simulation ne peuvent être assimilés à des agents mobiles, nous pouvons les qualifier d'*agents déplaçables*. Ce terme désigne le fait qu'ils peuvent être déplacés d'une plate-forme à une autre sans induire aucune modification dans leur comportement : n'interviennent dans leur comportement que des considérations thématiques et non sur leur contexte informatique d'exécution.

Cela signifie notamment que les éventuels appels à des ressources informatiques extérieures à la plate-forme doivent être réalisés par le biais de mécanismes agents (*i.e.* par perception/influence) à la manière de ce que nous proposons dans le cadre du déploiement des *spaces* (voir paragraphe 2.4.6). Un agent de simulation n'a donc pas « conscience » qu'il existe quoi que ce soit en dehors des agents et des environnements décrits dans le modèle simulé. Si l'on se rapporte aux définitions proposées dans [Mertens *et al.*, 2004] qui explique qu'un environnement logique peut être découpé et exécuté sur plusieurs processeurs sous la forme d'environnements physiques, un agent de simulation n'évolue et ne perçoit que les environnements sous leur forme logique. C'est à la plate-forme virtuelle de garantir la représentation logique de l'environnement en interconnectant les environnements physiques qui le composent.

4.3.2 Un algorithme en trois phases

Un algorithme d'équilibrage peut être décomposé en plusieurs phases dont le nombre varie selon les références bibliographique. Nous considérons la décomposition en cinq étapes proposée dans [Willebeek-LeMair and Reeves, 1993] :

– Evaluation de la répartition. Cette étape vise à identifier si la répartition actuelle est une

répartition non-optimale.
- Détermination de la profitabilité. Un déséquilibre dans la répartition de la charge ayant été détecté, l'équilibrage de charges n'aura lieu que si le coût de l'équilibrage de charges est inférieur au coût induit par le déséquilibre de charges actuel.
- Dimensionnement de la charge à déplacer. A partir des mesures effectuées à la première étape, la quantité de charge idéale à transférer est déterminée.
- Sélection des tâches. A partir de la quantité idéale déterminée à l'étape précédente, l'algorithme sélectionne les tâches à déplacer.
- Migration des tâches. Les tâches sélectionnées sont transmises aux destinataires.

Considérant les activités à mener au cours des différentes étapes de l'algorithme d'équilibrage de charges, nous identifions trois phases distinctes :

1. *Diffusion d'informations.* Pour pouvoir déterminer un déséquilibre de performances entre les plates-formes il est nécessaire de disposer d'une information sur les performances générales des plates-formes participant à l'exécution de la simulation. C'est à partir des informations diffusées à cette phase que l'agent de flexibilité de chaque plate-forme pourra décider si un transfert d'agents de simulation pourrait améliorer les performances générales. Cette phase est préliminaire aux étapes décrites par Willebeek-LeMair et Reeves.

2. *Détection de surcharge.* La phase de détection est la phase durant laquelle l'agent de flexibilité d'une plate-forme analyse et juge les performances de sa plate-forme hôte vis-à-vis des indicateurs de performances transmis par les autres plates-formes à l'étape de diffusion d'informations. A l'issue de la phase de détection, si les performances de sa plate-forme ne lui semblent pas satisfaisantes, il exécutera la phase 3 pour procéder à un rééquilibrage de la charge. Cette phase correspond aux étapes d'évaluation de la répartition et de détermination de la profitabilité.

3. *Transfert d'agents.* Au cours de cette phase, l'agent de flexibilité d'une plate-forme négocie un transfert d'agents de simulation avec une autre plate-forme pour améliorer les performances d'exécution de sa plate-forme hôte. Il exécute les trois dernières étapes du processus d'équilibrage de charges proposé par Willebeek-LeMair et Reeves.

Cet algorithme procède à l'équilibrage de la charge de simulation par des échanges de type *point à point.* Il n'y a pas d'entité qui observe la répartition de la simulation dans sa globalité et qui procède à une redistribution complète des agents de simulation sur l'infrastructure d'exécution. Ce sont les agents de flexibilité qui négocient entre eux des transferts d'agents de simulation et, en optimisant localement la performance de leur plate-forme hôte, vont optimiser globalement les performances en simulation. Une telle approche est la seule envisageable dans le cadre de la large-échelle où nous considérons que la simulation ne peut être représentée sur une unique plate-forme.

Ayant opté pour un mode de communication en diffusion, chaque plate-forme a connaissance des performances de toutes les plates-formes composant l'infrastructure d'exécution. Un agent de flexibilité d'une plate-forme devant réduire la part de simulation qu'elle exécute peut donc directement négocier avec l'agent de flexibilité de la plate-forme la plus apte à accueillir une charge de simulation supplémentaire. Contrairement aux algorithmes que nous avons présentés au paragraphe 4.2.1 (ceux décrits dans [Schaerf *et al.*, 1995] et [Montresor *et al.*, 2002]), notre algorithme vise à identifier le meilleur transfert à effectuer plutôt que de procéder par « propagation » de la charge.

4.3.3 Exécution de la simulation à l'équilibre des charges

L'objectif des agents de flexibilité est avant tout d'optimiser localement les performances de leur plate-forme hôte vis-à-vis des performances des autres plates-formes. Un transfert d'agents de simulation d'une plate-forme à une autre se traduit donc par une augmentation de la vitesse de simulation d'un pas de simulation pour la plate-forme qui demande le transfert de charge et une diminution de la vitesse de simulation d'un pas de simulation pour la plate-forme qui accepte le transfert et qui voit donc sa charge d'agents augmenter. Notons que dans le cadre du modèle à Temporalité, la notion de pas de temps telle qu'elle est utilisée ici est à rapporter aux tempos associés aux agents de simulation qui ont été transférés.

Légende :

Nœud de simulation exécutant la simulation.

Nœud de simulation en attente de synchronisation.

Nœud de simulation en synchronisation.

(a) Exécution de la simulation sans équilibre des charges de simulation.

(b) Exécution de la simulation avec un équilibre des charges (cas idéal).

FIG. 4.2 – Objectif de l'équilibrage de charges : minimiser les pauses sur chaque plate-forme.

Un transfert d'agents de simulation a pour effet d'accélérer l'exécution de la simulation sur certaines plates-formes et de la ralentir sur d'autres. Ce ralentissement local de la simulation est pourtant acceptable dans la mesure où l'exécution distribuée de la simulation impose des temps

d'attente entre les plates-formes. En effet, nous positionnant dans le cadre d'un ordonnancement conservatif (voir chapitre 3), les plates-formes doivent régulièrement se synchroniser. Ainsi, une plate-forme exécute la simulation aussi vite que possible jusqu'à arriver à un instant simulé où une synchronisation est nécessaire. Arrivé à ce temps simulé, elle met la simulation en pause jusqu'à réception des messages de synchronisation des autres plates-formes. L'idée de l'équilibrage de charges consiste à minimiser le temps d'attente des messages de synchronisation : au lieu de se retrouver à faire une pause dans la simulation, la plate-forme simule plus d'agents de simulation ce qui décharge les autres plates-formes et réduit donc le temps réel nécessaire à atteindre le point de synchronisation.

La figure 4.2 illustre cet objectif : le graphe de la figure 4.2(a) présente deux plates-formes exécutant ensemble la simulation distribuée avec une répartition arbitraire de la charge. La figure 4.2(b) montre l'exécution de la simulation après équilibrage de charges. Le nœud 1 simule plus d'agents de simulation que précédemment et prend plus de temps réel pour faire avancer le temps simulé jusqu'au point de synchronisation. Le nœud 2 exécutant moins d'agents de simulation, il fait avancer le temps simulé plus rapidement au point de synchronisation. Dans le cas idéal qu'est celui de la figure, les deux plates-formes arrivent au point de synchronisation en même temps et peuvent donc immédiatement se synchroniser. Dans un cas plus réaliste, il subsistera toutefois de petits temps d'attente pouvant notamment être induits par des variations ponctuelles et aléatoires de la puissance d'une machine.

4.4 Description des phases de l'algorithme

Pour procéder à l'équilibrage de charges entre les plates-formes de l'infrastructure d'exécution, il est nécessaire de pouvoir fragmenter l'exécution de la simulation en « séquences de simulation » : l'algorithme d'équilibrage de charges doit alors optimiser la répartition de la charge de manière à ce que toutes les plates-formes simulent chaque séquence à des vitesses similaires. Nous nommons *unité logique de simulation* la séquence d'exécution de la simulation sur la base de laquelle l'algorithme d'équilibrage de charges s'appliquera. Cette notion est proche de la notion de *superstep* présentée au paragraphe 4.2.2 mais diffère sur le fait que le *superstep* implique une synchronisation de toutes les plates-formes et pas l'*unité logique de simulation*. Cette dernière peut de plus contenir une alternance de plusieurs phases asynchrones et synchrones.

Nous considérons dans un premier temps la mise en œuvre de notre algorithme d'équilibrage de charges sur une simulation distribuée basée sur un ordonnancement de type pas à pas où les plates-formes de simulation se synchronisent après chaque pas de simulation. L'*unité logique de simulation* est alors le pas de simulation (elle est ici égale au *superstep*).

4.4.1 Diffusion d'informations

La phase de diffusion d'informations vise à diffuser toutes les informations nécessaires à la réalisation de deux autres phases de l'équilibrage de charges. A la fin de l'exécution de chaque *unité logique de simulation*, chaque plate-forme procède à cette phase et diffuse un message contenant :

- id_{uls} : l'identifiant de l'*unité logique de simulation*. Dans le cas présent il s'agit de la valeur du temps simulé associé au pas de temps considéré.
- $d_{ptf}(id_{uls})$: la durée de temps réel mis par la plate-forme à exécuter l'*unité logique de simulation* considérée.
- $P_{ptf}(id_{uls})$: un indicateur de la puissance d'exécution actuelle de la plate-forme. Plus la valeur de l'indicateur sera élevée, plus la plate-forme aura une grande puissance de traitement

Nous avons identifiés ces informations comme étant le tuple minimal à connaître pour exécuter notre algorithme d'équilibrage de charges. D'autres informations pourraient être ajoutées pour optimiser le fonctionnement de l'équilibrage de charges.

L'indicateur de puissance d'exécution, $P_{ptf}(id_{uls})$, est utilisé pour dimensionner la part des agents de simulation à déplacer au cours du transfert d'agents. Nous ne détaillons pas ici son évaluation mais notons qu'il peut s'agir directement d'informations techniques (modèle de processeur et quantité de mémoire) ou d'une durée d'exécution d'un traitement caractéristique commun à toutes les plates-formes : l'évaluation de la puissance doit être commun à toutes les plates-formes pour que les agents de flexibilité puissent établir des rapports de puissance.

La synchronisation des plates-formes a lieu après chaque pas de simulation. L'agent réseau étant chargé de l'optimisation des communications réseau (voir paragraphe 2.4.2), il transmet les informations de la phase de diffusion d'informations dans le message de synchronisation : un unique message permet de préparer l'exécution du prochain pas de temps sur le plan thématique (mise à jour des environnements au cours de la synchronisation des plates-formes) aussi bien que sur le plan technique (mise à jour de la répartition des agents de la simulation par équilibrage de charges).

4.4.2 Détection de surcharge

L'agent de flexibilité exécute la phase de détection de surcharge lorsqu'il a reçu tous les messages de diffusion d'informations relatifs à l'exécution d'une *unité logique de simulation*, c'est-à-dire qu'il a reçu un nombre de messages égal au nombre de plates-formes ayant participé à l'exécution de l'*unité logique de simulation*.

En phase de détection de surcharge, l'agent de flexibilité vérifie sa condition de satisfaction vis-à-vis de l'exécution de l'*unité logique de simulation*. S'il n'est pas satisfait, il exécutera la

phase de transfert d'agents pour alléger sa charge de simulation. S'il est satisfait, il ne prendra pas l'initiative de proposer un transfert d'agents à un autre agent de flexibilité et restera en attente.

La vitesse globale d'exécution de la simulation dépend de l'infrastructure d'exécution et notamment de la puissance des plates-formes qui la compose. Ainsi, l'agent de flexibilité établi son niveau de satisfaction relativement aux performances des autres plates-formes. Notant $\overline{d}(id_{uls})$ la durée moyenne d'exécution de l'*unité logique de simulation* id_{uls}, la condition de satisfaction de la plate-forme ptf s'écrit :

$$d_{ptf}(id_{uls}) - \overline{d}(id_{uls}) \leq tolerance$$

L'objectif d'un agent de flexibilité est d'amener le temps d'exécution de chaque *unité logique de simulation* sur sa plate-forme au temps moyen d'exécution sur l'infrastructure d'exécution. Cependant chercher à obtenir la relation stricte $d_{ptf}(id_{uls}) = \overline{d}(id_{uls})$ est utopique dans un contexte où la durée d'exécution d'une *unité logique de simulation* peut varier très légèrement. Pour éviter qu'une variation non-symptomatique d'un déséquilibre de charges ne produise un transfert d'agents, nous introduisons le terme *tolerance* dans la condition de satisfaction : c'est la valeur de temps en dessous de laquelle l'agent de flexibilité estime qu'il n'est pas nécessaire de procéder à un transfert d'agents. Nous nous intéressons aux phénomènes induits par l'absence de la tolérance au paragraphe 5.4.2.

4.4.3 Transfert d'agents

Cette phase n'est exécutée que par les agents de flexibilité dont la condition de satisfaction n'a pas été remplie. Ces agents de flexibilité négocient durant cette phase un transfert d'agents avec d'autres agents de flexibilité afin de décharger leur plate-forme hôte et donc en améliorer les performances. Nous considérons ici l'agent de flexibilité A_{Fi} de la plate-forme ptf_i dont la condition de satisfaction n'a pas été remplie.

La phase de transfert d'agents est elle-même constituée de plusieurs étapes :

- identification de la plate-forme la plus apte à accueillir la charge de simulation.
- sélection des agents de la simulation qui devront être transférés.
- négociation du transfert.

Identification d'une plate-forme destinatrice

Au travers des informations collectées en phase de diffusion d'informations, l'agent de flexibilité connaît les durées d'exécution $d_{ptf}(id_{uls})$ de chaque plate-forme et peut donc positionner la performance de sa plate-forme par rapport aux autres : il sait notamment que les agents de

flexibilité de toutes les plates-formes affichant un temps inférieur au temps de sa plate-forme effectueront un transfert de charges.

Si tous les agents de flexibilité souhaitant effectuer un transfert d'agents s'adressent à l'agent de flexibilité de la plate-forme ayant affiché la plus courte durée d'exécution de id_{uls}, la plupart des négociations échoueront faute de capacité d'accueil. Ainsi, la stratégie que nous proposons consiste à identifier le meilleur destinataire en se basant sur le classement des plates-formes vis-à-vis de la durée d'exécution. L'agent de flexibilité de la plate-forme la plus lente durant l'exécution de id_{uls} négociera avec l'agent de flexibilité de la plus rapide, celui de la $k^{\text{ième}}$ plus lente avec celui de la $k^{\text{ième}}$ plus rapide : c'est un algorithme de type *Rendez-Vous* comme le définit [Osman and Ammar, 2002].

Sélection des agents de la simulation qui devront être transférés

Une fois la plate-forme destinatrice ptf_{dest} identifiée, il convient de dimensionner le transfert d'agents de la simulation à effectuer de manière à ce que l'agent de flexibilité A_{Fi} puisse se rapprocher de sa condition de satisfaction. Ainsi, A_{Fi} évalue le nombre d'agents à transférer de manière à obtenir $d_{ptf_i}(id_{uls+1}) = d_{ptf_{dest}}(id_{uls+1})$, c'est-à-dire égaliser les durées d'exécution de la prochaine *unité logique de simulation* de ptf_i et ptf_{dest}.

Pour atteindre cet objectif, l'agent de flexibilité doit sélectionner des agents de simulation de manière à réduire son temps d'exécution de la simulation de la durée $d_{transfer}(ptf_i, ptf_{dest})$ donnée par :

$$d_{transfer}(ptf_i, ptf_{dest}) = (d_{ptf_i}(id_{uls}) - d_{ptf_{dest}}(id_{uls})) \times \frac{P_{ptf_{dest}}(id_{uls})}{P_{ptf_i}(id_{uls}) + P_{ptf_{dest}}(id_{uls})}$$

Pour cela, nous considérons que les agents de flexibilité mesurent en simulation la durée d'exécution des agents de simulation et calculent une moyenne par famille d'agents. Ainsi pour une famille d'agents de simulation Fa, chaque agent de flexibilité peut établir un coût d'un agent de la famille :

$$cost_{ptf_i}(Fa) = \overline{simTime}_{a \in Fa}(id_{uls}) \times P_{ptf_i}(id_{uls})$$

L'agent de flexibilité sélectionne donc les agents de simulation à déplacer de manière à approcher au mieux la relation

$$\frac{\sum_{Fa_k}(n_{Fa_k} \times cost_{ptf_i}(Fa_k))}{P_{ptf_i}(id_{uls})} = d_{transfer}(ptf_i, ptf_{dest})$$

n_{Fa_k} représente alors la quantité d'agents de la famille Fa_k devant être transférés de ptf_i à ptf_{dest}. L'identification précise des agents de simulation est une question stratégique qui est abordée au paragraphe 4.4.5.

Négociation du transfert

La dernière étape de la phase de transfert d'agents est la négociation proprement dite du transfert. A_{Fdest} peut refuser le transfert que propose A_{Fi} pour plusieurs raisons :

- il a une demande de transfert d'agents prioritaire. L'agent de flexibilité d'une plate-forme se déconnectant de l'infrastructure d'exécution lui a par exemple transmis une forte charge de simulation.
- la plate-forme pft_{dest} est en train de se déconnecter.
- la dynamique de la simulation a conduit à la création de nombreux agents sur ptf_{dest} qui grèveront l'exécution de la prochaine *unité logique de simulation*.
- la plate-forme ptf_{dest} n'est pas en état de recevoir de nouveaux agents de simulation : elle a repris l'exécution de la simulation.

Si la négociation du transfert échoue avec la plate-forme ptf_{dest}, A_{Fi} a la possibilité de reprendre les étapes du transfert de charges en identifiant une nouvelle plate-forme destinatrice. Cependant, notons que l'équilibrage de charges s'exécute entre deux pas de simulation : la simulation ne pourra reprendre qu'une fois les transferts d'agents négociés et éventuellement effectués.

La phase de transfert d'agents se conclut par un message contenant les agents de simulation de ptf_i à destination de ptf_{dest}. Ce message a donc une taille importante qui, diffusé dans l'infrastructure d'exécution, peut perturber le trafic réseau. Dans la mesure où ce message n'implique que ptf_i et ptf_{dest}, l'agent réseau A_{Ri} établit une connexion point à point avec l'agent réseau A_{Rdest} pour transmettre ce message sans le dupliquer pour diffusion.

4.4.4 Exécution de l'équilibrage de charges

La figure 4.3 détaille le comportement d'un agent de flexibilité dans sa tâche d'optimiser la charge de simulation que doit traiter sa plate-forme hôte. Les étapes en gris clair sont les étapes qui ne sont pas assurées par l'agent de flexibilité.

Après l'exécution de chaque *unité logique de simulation*, les agents de flexibilité de toutes les plates-formes participant à l'exécution de la simulation effectuent les phases de diffusion d'informations et de détection de surcharge mais seuls les agents de flexibilité dont la condition de satisfaction n'est pas remplie procéderont à la phase de transfert d'agents de simulation.

Nous avons expliqué au paragraphe 4.3.2 que les déplacements d'agents devaient être effectués alors que la simulation n'a pas encore repris. Un équilibrage de charges, partant de la phase de diffusion d'informations jusqu'au transfert effectif d'agents de simulation, est réalisé au cours

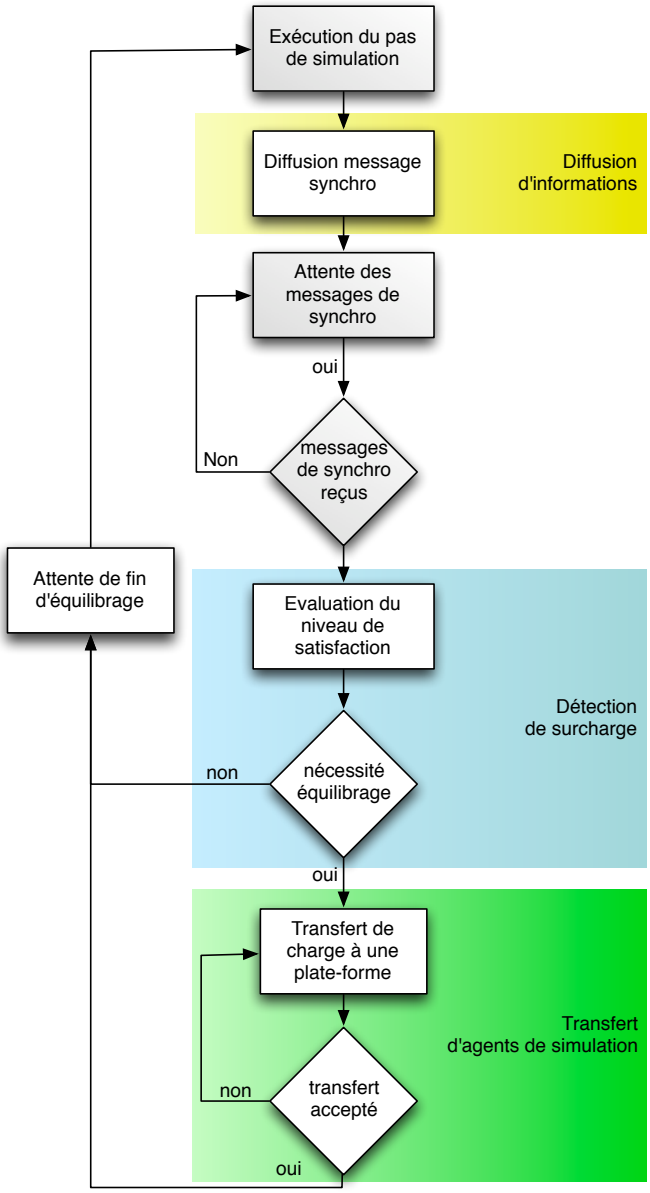


FIG. 4.3 – Comportement d'un agent de flexibilité pour l'équilibrage de charges.

d'une pause entre deux pas de simulation. La figure 4.4 présente l'enchaînement des différentes phases de l'algorithme d'équilibrage de charges partant de la fin de l'exécution d'une *unité logique de simulation* au lancement de l'*unité logique de simulation* suivante. Le nœud 1 ayant montré de bonnes performances n'initie pas de transfert de charges tandis que le nœud 2 réalise l'intégralité des phases de l'équilibrage de charges. L'abréviation « d.s. » est mise pour « Détection de surcharge », « transfert a. » pour « Transfert d'agents » et « négo. » pour « négociation ».

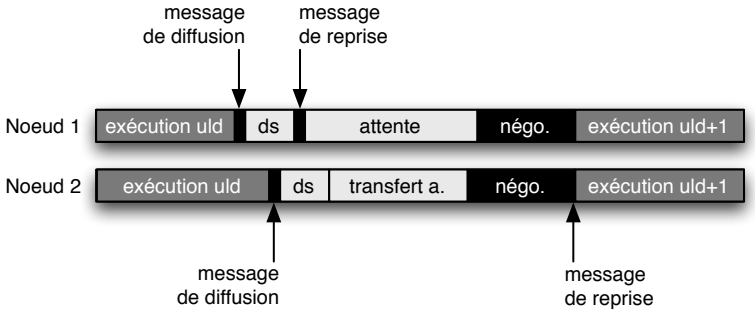


FIG. 4.4 – Illustration des différentes séquences de l'équilibrage de charges.

Les agents de flexibilité n'effectuant pas de transfert d'agents transmettent un message de reprise de la simulation pour indiquer leur disposition à reprendre la simulation sans pour autant lancer l'exécution du prochain pas de simulation. Ce n'est que lorsque tous les transferts d'agents auront eu lieu, c'est-à-dire lorsque toutes les plates-formes auront diffusé leur message de reprise, que l'exécution de la simulation reprendra sur toutes les plates-formes.

4.4.5 Pistes de réflexion pour une sélection optimisée des agents de simulation à transférer

L'identification des agents de simulation à sélectionner pour un transfert d'agents revêt un aspect stratégique dès lors que l'on s'intéresse à la charge réseau qu'ils induisent. En effet, une des caractéristiques fondamentales du paradigme agent réside dans les interactions que les agents établissent entre eux. Dans un cadre distribué, les interactions de deux agents hébergés sur des plates-formes différentes devront transiter via le réseau. Avec le modèle perception/influence, les environnements sont les supports de ces interactions : deux agents en interaction peuvent être hébergés sur la même plate-forme mais voir leurs interactions transiter sur le réseau pour peu que la part d'environnement[14] qu'ils perçoivent ou influencent ne se trouve pas sur la même plate-forme qu'eux.

Dans la mesure où la répartition de la charge de simulation sur l'infrastructure a un impact sur le nombre de communication réseau, lors de l'étape de transfert d'agents, il serait pertinent d'étudier la meilleure localisation de chaque agent de la plate-forme émettrice ptf_i, entre ptf_i et

[14]Nous considérons dans ce paragraphe que les environnements sont distribués sur les plates-formes de simulation.

la plate-forme destinatrice ptf_{dest}. Nous appuyant sur le modèle perception/influence, il s'agit donc d'étudier les liens qu'ont les agents de simulation de ptf_j avec les parts d'environnements hébergés sur ptf_j et ptf_{dest}.

Une première solution serait de comptabiliser le nombre de perceptions et d'influences adressées aux parts d'environnements hébergés sur chaque plate-forme : la localisation idéale de l'agent de simulation serait alors la plate-forme à laquelle il a adressé le plus de requêtes. Mais cette solution est difficilement applicable puisqu'elle impose à chaque plate-forme de tenir à jour un compteur de requêtes de perception/influence pour chaque agent et à destination de chacune des plates-formes de l'infrastructure d'exécution. De plus, considérant ici la distribution des environnements, ce système ne serait pas applicable dès lors que les environnements peuvent être dynamiquement partitionnés et redéployés sur l'infrastructure d'exécution.

Une piste de réflexion nous semble tenir dans la généralisation de la définition de position d'un agent dans un environnement. Les environnements spatiaux définissent une métrique et un positionnement de l'agent : si l'environnement est fragmenté sur plusieurs plates-formes, la localisation optimale de l'agent sur l'infrastructure d'exécution sera la plate-forme hébergeant la part de l'environnement incluant sa position. En généralisant le concept de positionnement d'un agent vis-à-vis de tous les environnements qu'il perçoit, les agents de flexibilité peuvent mesurer la distance des agents de simulation à leur plate-forme et sélectionner les agents les plus éloignés de la plate-forme émettrice et proches de la plate-forme destinatrice. De cette manière, le transfert d'agent aura pour effet de diminuer le nombre de messages réseau supports de requêtes de perception ou d'influence, ou du moins d'en minimiser l'accroissement.

Nous soulignons aussi les travaux autour de l'approche AGR [Ferber *et al.*, 2004a]. Le modèle AGR constitue une approche organisationnelle pour la réalisation d'applications agent : des *Agents* ne peuvent interagir entre eux que dans le cadre d'un *Groupe* où chacun d'eux assure au moins un *Rôle*. Les groupes constituent alors des ensembles d'agents et permettent de partitionner les organisations du système. L'observation des groupes est tout à fait pertinente pour sélectionner les agents de simulation à déplacer d'une plate-forme à une autre. Qui plus est, le modèle AGR possède différentes extensions qui enrichissent les possibilités de modélisation agent tout en conservant cette faculté à identifier simplement les interactions entre agents ; nous retenons notamment le modèle AGRE [Ferber *et al.*, 2004b] qui uniformise les considérations sur les environnements spatiaux et les environnements organisationnels.

4.5 Adaptation à l'ordonnancement distribué séquentiel du modèle à Temporalité

Nous considérons maintenant une simulation basée sur l'emploi du modèle à Temporalité et dont l'ordonnancement distribué se fait selon la méthode présentée au paragraphe 3.2.3. Chaque

plate-forme déclare l'instant simulé associé au prochain slot temporel qu'elle va exécuter : une synchronisation a lieu dès qu'une plate-forme Ptf_j a exécuté des agents entre deux slots temporels successifs de la plate-forme Ptf_i.

4.5.1 Axe temporel simulé distribué

Présenté au paragraphe 3.2.1, le modèle à Temporalité permet à chaque agent de définir, au travers des temporalités, les instants, périodiques ou non, auxquels il souhaite être activé (il peut de plus associer un comportement spécifique à chacune de ses temporalités). Par rapport au modèle temporel à pas de temps constant, ce modèle possède deux caractéristiques à considérer spécifiquement dans l'adaptation de l'équilibrage de charges :

- à un instant simulé donné, seuls seront exécutés les agents de la simulation qui auront défini une temporalité devant être activée à l'instant simulé considéré.
- ce modèle permet d'identifier sur l'axe temporel simulé les instants pertinents au regard de la simulation. Le temps simulé séparant deux instants d'activation est variable.

Ces deux éléments ont un impact sur l'exécution distribuée de la simulation dans la mesure où l'axe temporel simulé diffère entre deux plates-formes participant à la même simulation distribuée. La figure 4.5 illustre ce phénomène : une simulation est exécutée sur deux plates-formes, la répartition des agents est telle que les plates-formes ne partagent aucun slot temporel. Les points de synchronisation induits par l'ordonnanceur global utilisé sont représentés par des croix sur les deux axes temporels. Sur cet exemple, l'axe temporel de la simulation sur le nœud 1 diffère entièrement de celui du nœud 2. L'axe temporel complet de la simulation, que nous désignerons sous le terme *axe temporel simulé global*, est alors donné par la superposition des axes temporels de toutes les plates-formes participant à la simulation.

Notons que cet exemple n'est pas irréaliste : on peut imaginer deux thématiciens ayant abordé chacun un aspect différent de la même simulation (par exemple, un thématicien travaillant sur les coraux et un autre sur les poissons récifaux pour la simulation d'un lagon) et qui lancent la simulation distribuée en mettant en réseau leur ordinateur. Le couplage entre les deux dynamiques de simulation se fait au travers de l'environnement (échange de messages ou modification de l'environnement).

4.5.2 Identification de l'*unité logique de simulation* à distribuer

Au paragraphe 4.4, nous avons considéré les performances de chaque plate-forme sur chaque pas de simulation. Tous les agents étant exécutés à chaque pas de temps, la répartition des agents est stable *a priori* après exécution des premiers pas.

S'appuyant sur la méthode de synchronisation des plates-formes décrite précédemment, la simulation ne progresse pas de la même manière sur l'ensemble de l'infrastructure d'exécution.

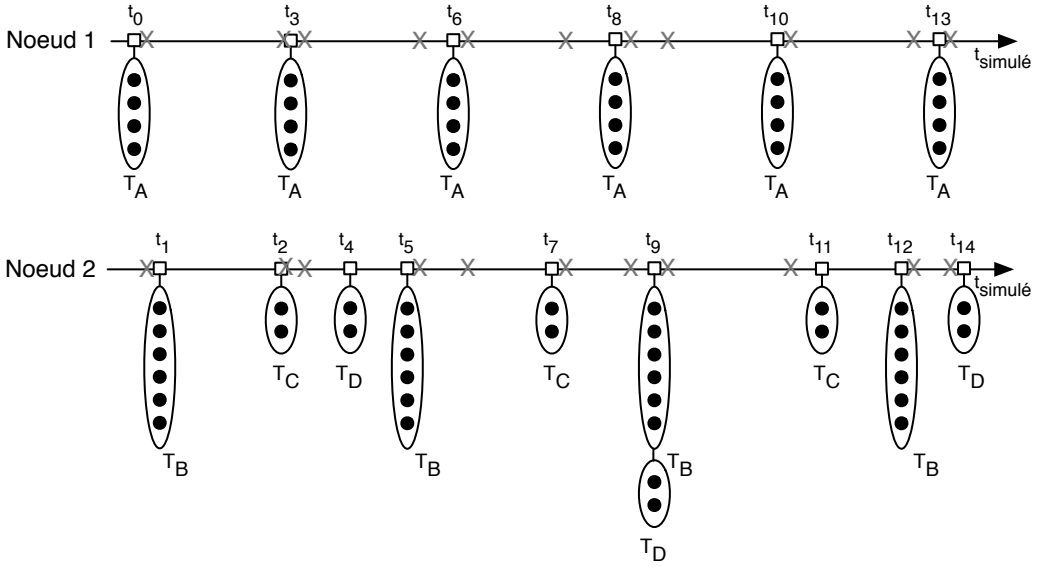


FIG. 4.5 – Des axes temporels simulés différents pour la même simulation distribuée.

En effet, deux plates-formes n'exécuteront la simulation en parallèle que si elles partagent un même slot temporel. Dans le cas « extrême » de la figure 4.5, les deux nœuds ne partagent aucun slot temporel. Cela signifie que le nœud 2 sera en pause pendant toute la durée de l'exécution de t_0 par le nœud 1. Puis le nœud 1 se mettra en pause pendant l'exécution de t_1 et t_2 sur le nœud 2 et ainsi de suite.

Cette exécution ne tire aucunement parti de l'infrastructure d'exécution. Pour ce faire, il faudrait en effet que les plates-formes disposent chacune d'une charge de simulation à exécuter en parallèle entre deux points de synchronisation. Dans le cadre d'une simulation basée sur le modèle à Temporalité et exécutée de manière distribuée à l'aide d'une synchronisation basique, l'équilibrage de charges a pour but de conférer une charge de simulation à toutes les plates-formes participant à la simulation de manière à ce que chacune ait un temps de simulation similaire entre deux points de synchronisation.

Si nous disposons d'un algorithme d'équilibrage de charges, nous devons identifier sur la base de quelle *unité logique de simulation* l'employer. Le modèle à Temporalité définit plusieurs concepts temporels : nous repassons en revue leur capacité à servir de base pour le déploiement de notre algorithme d'équilibrage de charges.

Equilibrage de charges sur la base temporelle des points de synchronisation

Les slots temporels ne conviennent pas comme *unité logique de simulation*. La première raison vient du fait qu'il est difficile de déterminer l'impact d'un agent de simulation sur l'exécution de la simulation entre deux points de synchronisation : certains agents de simulation peuvent avoir été exécutés plusieurs fois alors que d'autres n'auront pas été activés. Le choix des agents

de simulation à transférer devient particulièrement délicat.

La deuxième raison est liée à l'identification de la séquence de simulation exécutée entre deux points de synchronisation. Supposons qu'à la suite d'un point de synchronisation, une répartition optimale de la simulation ait été déterminée. Cette répartition n'est optimale que pour l'exécution des slots temporels ayant été exécutés avant la dernière synchronisation. Par exemple, en suivant l'axe temporel de la figure 4.5, la répartition des agents exécutés à t_0 n'optimise aucunement l'exécution des agents à t_1 (qui sont exécutés après synchronisation des deux plates-formes).

Equilibrage de charges sur la base temporelle des slots temporels

Le deuxième concept temporel que nous considérons est le slot temporel. En répartissant sur l'ensemble de l'infrastructure les agents de simulation devant être exécutés à un slot temporel donné, nous permettons l'exécution parallèle du slot temporel même si en retour nous forçons la réalisation d'une synchronisation entre les plates-formes à la fin de l'exécution du slot temporel.

Si l'équilibrage de charges est réalisé en se basant sur les slots temporels de l'axe temporel simulé global, les temporalités (et bien évidemment les agents possédant ces temporalités) associées au slot temporel sont distribuées sur l'infrastructure d'exécution de manière à équilibrer la charge de simulation sur chaque plate-forme et ce, indépendamment de toute considération sur les tempos auxquels elles appartiennent.

Si cette considération ne semble pas importante de prime abord, elle l'est en considérant le cas où deux tempos se retrouvent sur le même slot temporel. En effet, considérons que l'équilibrage de charges ait été réalisé après l'exécution du temps t_9 dans l'exemple de la figure 4.5. Supposant que l'équilibrage de charges ait mené à une situation où T_B s'exécute sur le nœud 2 et T_D sur le nœud 1, cette répartition de charges est certes optimale lorsque T_B et T_D sont exécutés au même slot, mais conduira à une situation où à T_{12} le nœud 2 assurera seul l'exécution du slot temporel et, en retour, le nœud 1 devra exécuter à lui seul le slot temporel à t_{14}. Ainsi, la considération des slots temporels peut conduire à une modification récurrente de la répartition des agents de simulation sur l'infrastructure d'exécution.

Equilibrage de charges sur la base temporelle des tempos

Pour obtenir une répartition des agents de simulation qui soit à la fois optimale et stable, c'est-à-dire qui ne doit pas être révisée de manière récurrente, il est nécessaire de considérer un équilibrage de charges basé sur les tempos. L'algorithme d'équilibrage de charges vise donc à équilibrer la répartition des agents de simulation sur l'infrastructure d'exécution de manière à ce que chaque tempo soit exécuté sur l'ensemble des nœuds de simulation à une vitesse similaire. En considérant les tempos, la répartition des agents reste stable tant qu'il n'y a pas de modification du nombre d'agents, et ce peu importe le nombre de tempos rattachés au même slot temporel.

En simulation, un agent de flexibilité diffuse donc, durant la phase de diffusion d'informations, le temps mis par la plate-forme à exécuter chaque tempo du dernier slot temporel exécuté. Sur la base de ces temps, chaque plate-forme évalue sa condition de satisfaction par rapport à l'exécution de chacun des tempos. Si son niveau de satisfaction est insuffisant sur l'exécution d'un tempo, un agent de flexibilité déclenche un transfert d'agents de simulation en ne considérant que les agents de simulation activés au cours de l'exécution du tempo dont le niveau de satisfaction est jugé insuffisant. L'algorithme reste donc très proche de celui décrit au paragraphe 4.4 : au lieu de considérer chaque pas de temps ainsi que tous les agents de la simulation, il se focalise ici sur les tempos et les agents de simulation activés durant ces tempos.

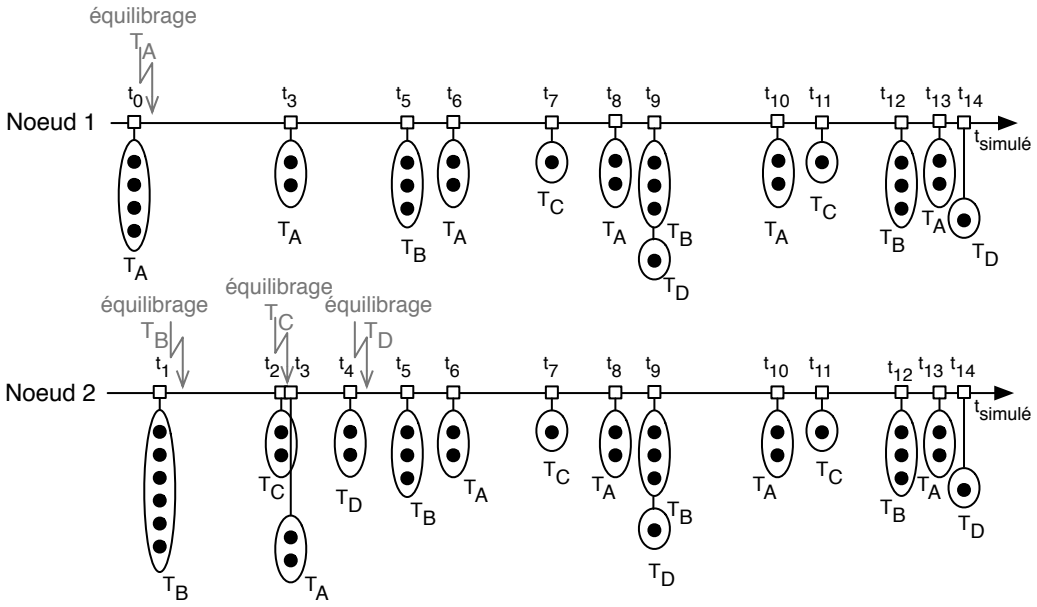


FIG. 4.6 – Equilibrage de charges basés sur les tempos.

Nous reprenons l'exemple de la figure 4.5 auquel nous appliquons l'algorithme d'équilibrage de charges en considérant que les deux nœuds de simulation possèdent une puissance de traitement équivalente et que les agents d'un même slot temporel nécessitent également des temps de traitement similaires. Nous obtenons les axes temporels de la figure 4.6. Nous n'avons pas représenté les points de synchronisation sur ce schéma dans la mesure où ceux-ci ont lieu après l'exécution de chaque slot temporel.

Dans cette configuration, il n'y a besoin de répartir les agents de simulation qu'une seule fois pour chaque tempo. Dans les simulations que nous traitons, les agents d'une même famille (basés sur le même modèle de comportement) partagent les mêmes condition d'activation et voient donc leurs activations regroupées dans les même tempos. Cela signifie qu'en optimisant la répartition des agents d'un même tempo, il est possible d'améliorer l'exécution d'un autre tempo où seraient activés les mêmes agents.

4.5.3 Anticipation de la charge

Un des atouts du modèle à Temporalité est d'offrir une vision *a priori* de l'axe temporel de la simulation. Les agents de simulation expriment leurs besoins futurs, en terme d'activation de comportement, au travers des temporalités. La structure même des temporalités, et notamment leur période, nous permettent de définir un axe temporel prévisionnel. Cet axe temporel peut bien sûr évoluer dans la mesure où les agents sont libres de modifier leurs temporalités,

Si cet axe temporel prévisionnel permet d'anticiper quels seront les prochains tempos qui seront exécutés, il est de plus possible d'évaluer quelles seront les performances des plates-formes lors de l'exécution de ces tempos.

Connaissant la puissance actuelle de la plate-forme ptf_i, $P_{ptf_i}t$, l'agent de flexibilité de ptf_i doit évaluer quel sera le coût d'exécution d'un tempo, c'est-à-dire, évaluer le temps que la plate-forme mettra à simuler le tempo considéré. Pour cela, l'agent de flexibilité doit se baser sur un historique d'exécution des tempos. Ainsi, il détermine la moyenne des temps d'exécution des comportements, associés à un même tempo, des agents d'une même famille d'agents de simulation. Cette mesure suppose qu'à un tempo donné, les agents d'une même famille adopteront un comportement similaire (par exemple, des agents poissons exécuteront leur comportement *« se nourrir »* au sein d'un même tempo).

L'anticipation de charges s'appuyant sur un historique d'exécution des tempos, elle ne peut être utilisée qu'après l'exécution de la première occurrence d'un tempo. Soient les notations suivantes :

- $t_{courant}$, l'instant simulé auquel se trouve la plate-forme.
- $T_k(i)$, la $i^{\text{ème}}$ occurrence du tempo T_k ; $T_{sk}(i)$ étant alors l'instant simulé auquel l'occurrence i du tempo T_k a lieu.
- $P(Tsk(i))$, la puissance de la plate-forme considérée à l'instant $T_{sk}(i)$.
- $F(T_k)$, l'ensemble des familles d'agents possédant une temporalité rattachée au tempo T_k.
- $n_{Fa_m}(T_k(i))$, le nombre d'agents de la famille Fa_m ayant une temporalité rattachée à l'occurrence i du tempo T_k.
- $ts_{Fa_m}(T_k(i))$, le temps moyen mis pour exécuter les agent de la famille Fa_m à l'occurrence i du tempo T_k.
- $ts(T_k(i))$, le temps d'exécution de l'occurrence i du tempo T_k.

Avec ses notations, l'évaluation de la durée de simulation du tempo T_k sera donné au temps $t_{courant}$ par

$$ts(T_k(i+1)) = \frac{P(Tsk(i))}{P(t_{courant})} \times \sum_{F(T_k)} n_{Fa_m}(T_k(i+1)) \times ts_{Fa_m}(T_k(i))$$

Notons que la différence entre cette évaluation et le temps réellement mis pour exécuter

le tempo considéré est induite par la différence de puissance ($\frac{P(Tsk(i))}{P(t_{courant})}$) et l'approximation $ts_{Fa}(T_k(i)) \simeq ts_{Fa}(T_k(i+1))$.

Disposant d'une évaluation du temps d'exécution des prochains tempos, les agents de flexibilité peuvent enrichir les informations échangées pendant la phase de diffusion d'informations en y intégrant le temps d'exécution estimé des l prochains tempos (plus l'instant d'activation du tempo se trouve loin sur l'axe du temps moins l'estimation sera bonne puisque la simulation possède sa propre dynamique). Ainsi, les plates-formes vont pouvoir anticiper la charge qu'elles auront à exécuter. Chaque plate-forme peut évaluer le niveau de satisfaction qu'elle aura pour l'exécution des l prochains tempos et peut alors déclencher un transfert d'agents de la simulation devant être exécutés au cours du traitement d'un tempo dont elle estimera son niveau de satisfaction insuffisant.

Cette anticipation est intéressante car elle permet de réduire les temps de pause entre deux synchronisations. Nous avons vu au paragraphe 4.4.4 que l'équilibrage de charges introduit un échange de messages entre les plates-formes avant la reprise de la simulation (les agents de flexibilité valident la fin de l'équilibrage de charges). Dans le cadre d'un équilibrage de charges par anticipation, il n'est pas nécessaire de mettre toutes les plates-formes en attente. Une plate-forme peut en effet transférer l'exécution future de certains de ces agents sur une autre plate-forme tout en assurant l'exécution présente des agents de simulation non concernés par le transfert. L'équilibrage de charges par anticipation permet donc de limiter les équilibrages de charges devant être réalisés entre deux points de synchronisation en plus de mieux optimiser l'exécution de la simulation (en détectant par avance les situations de fort déséquilibre entre les plates-formes).

Nous rappelons qu'un agent de simulation se situe sur une seule et unique plate-forme à tout instant de la simulation : tant que la plate-forme destinatrice n'accuse pas réception des agents de simulation transférés, les agents de simulation en transfert seront encore considérés comme gérés par la plate-forme émettrice.

Pour permettre l'équilibrage de charges par anticipation, les agents de flexibilité font appel à leur mémoire (historique des temps moyens d'exécution) et évalue les temps d'exécution des futurs tempos. Les agents de flexibilité sont capables d'agir en fonction d'un futur qu'ils estiment : ils deviennent dans ce cadre des agents cognitifs.

4.6 Application à l'ordonnancement distribué parallèle du modèle à Temporalité

Ayant proposé au chapitre 3 un algorithme permettant l'ordonnancement parallèle de la simulation distribuée, nous exploitons à la fois l'exécution parallèle offerte par cet ordonnancement et l'algorithme d'équilibrage de charges que nous proposons.

L'algorithme d'ordonnancement parallèle que nous proposons s'appuie sur la seule connais-

sance du modèle de simulation pour identifier les liens de causalité entre tempos. Ayant dressé une table de dépendances entre tempos, il permet d'identifier à tout instant de la simulation l'ensemble des tempos pouvant être exécutés, indépendamment de leur prochain instant d'activation. Notre algorithme d'exécution parallèle n'impose donc pas un ordonnancement spécifique de la simulation mais offre la possibilité aux plates-formes d'anticiper[15] l'exécution de certains tempos.

Considérant une simulation basée sur le modèle à Temporalité, nous avons montré au paragraphe 4.5.2 que l'algorithme d'équilibrage de charges vise à optimiser l'exécution distribuée de chaque tempo en répartissant les agents de simulation devant y être activés. Utiliser le tempo comme *unité logique de simulation* sur laquelle appliquer l'équilibrage de charges permet d'une part d'identifier la charge de simulation à distribuer et d'autre part d'obtenir une répartition des agents qui ne soit pas dépendante du slot temporel auquel le tempo est rattaché.

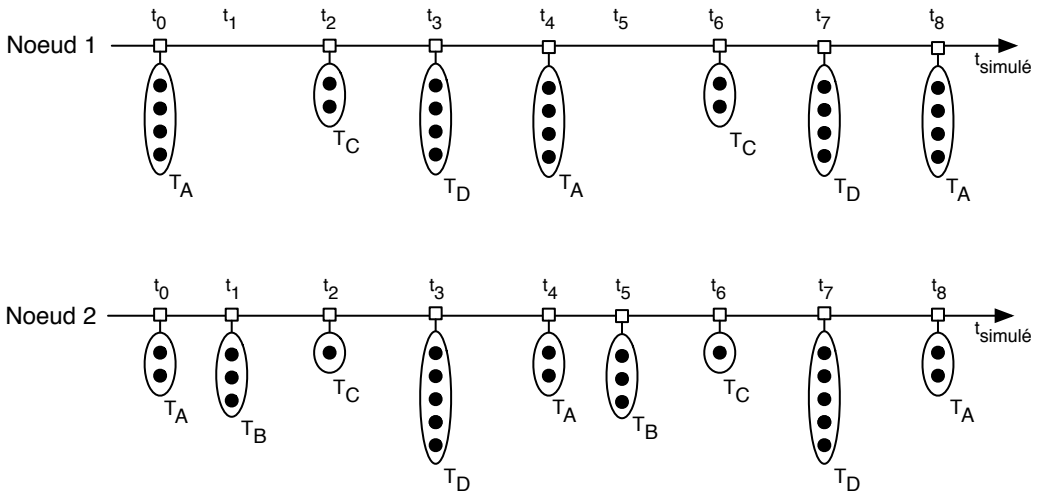


FIG. 4.7 – Exemple d'axe temporel où l'on suppose que T_D peut être exécuté juste après T_A.

Cependant, dans le contexte de l'exécution parallèle, une plate-forme peut procéder à l'exécution anticipée d'un tempo. Ainsi, à un moment donné de la simulation, il est possible d'avoir des tempos pour lesquels l'exécution sur certaines plates-formes sera déjà accomplie. Ce phénomène ne perturbe pourtant pas l'équilibrage de charges. Quand une plate-forme termine l'exécution, anticipée ou non, d'un tempo (*i.e.* elle termine l'exécution des agents de simulation qu'elle héberge et qui sont associés au tempo), elle émet le message de diffusion d'informations associé à la fin d'exécution du tempo avant de poursuivre l'exécution de la simulation, c'est-à-dire passer en attente de synchronisation si nécessaire ou exécuter un autre tempo.

Considérons l'exemple de la figure 4.7. Nous supposons que le nœud d'exécution numéro 2 est deux fois plus performant que le numéro 1. Nous supposons de plus que le tempo T_D ne dépend que du tempo T_A tandis que T_B dépend de T_A, T_C de T_B et T_A de T_D.

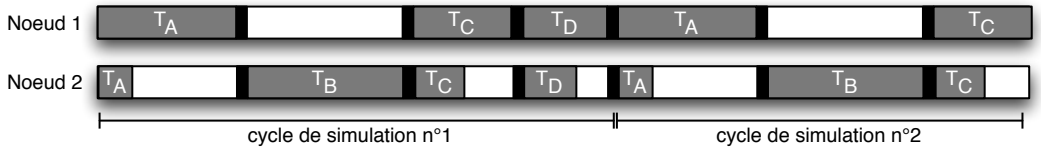
[15]Il y a anticipation de l'exécution d'un tempo dès lors que l'exécution du tempo est déclenchée sans attendre la fin du point de synchronisation le précédant.

Légende :

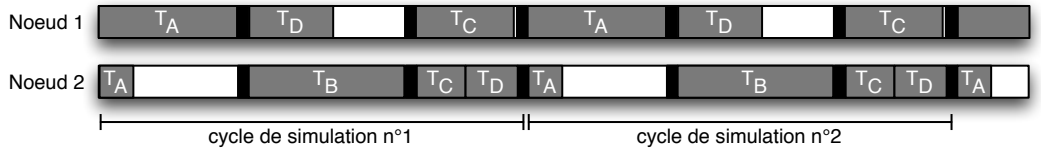
Nœud de simulation exécutant la simulation.

Nœud de simulation en attente de synchronisation.

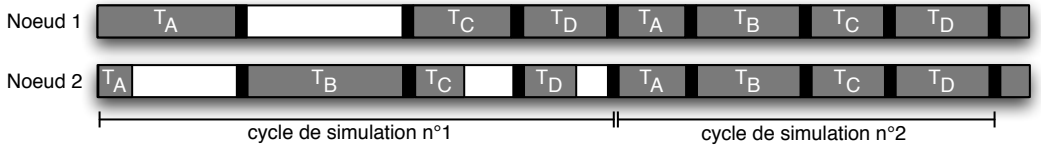
Nœud de simulation en synchronisation.



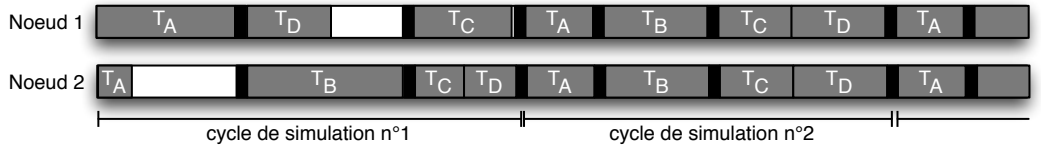
(a) Exécution classique de la simulation.



(b) Exécution parallèle de la simulation.



(c) Exécution de la simulation avec équilibrage des charges.



(d) Exécution parallèle de la simulation avec équilibrage des charges.

FIG. 4.8 – Les différents modes d'exécution de la simulation.

La figure 4.8 présente différentes barres de progression de la simulation au cours du temps réel en fonction de l'emploi ou non des algorithmes d'optimisation de la simulation. A la figure 4.8(b), l'exécution de la simulation se fait de manière parallèle en employant notre algorithme de détection des liens de causalité. Après l'exécution de T_A, le nœud 1 ne peut exécuter T_C car il doit attendre que l'exécution de T_B soit terminée sur le nœud 2, il entame donc l'exécution de T_D. On observe que cette méthode permet de réduire le temps d'exécution d'un cycle de simulation (*i.e.* le plus petit commun multiple des périodes des tempos) par rapport à la référence qui est l'exécution sans aucune optimisation de la simulation visible à la figure 4.8(a).

La figure 4.8(c) représente la progression de la simulation en utilisant l'algorithme d'équilibrage de charges. La première exécution de chaque tempo sur les nœuds de simulation permet d'équilibrer la répartition des agents de simulation exécutés à ces tempos si bien que tout temps

de pause, en dehors des synchronisations, est supprimé dès le deuxième cycle de simulation. Toutefois, les performances en début de simulation restent les mêmes que le cas de la figure 4.8(a).

Enfin, la figure 4.8(d) illustre l'exécution de la simulation avec le couplage des algorithmes d'exécution parallèle et d'équilibrage de charges. L'algorithme d'exécution ne se basant que sur les connaissances du modèle simulé, il se montre efficace dès le début de la simulation en permettant d'anticiper l'exécution de T_B sur le nœud 1. A la fin de la première exécution de T_B, ce nœud de simulation procède à la diffusion d'informations concernant l'exécution de T_B puis se mettra en attente de synchronisation avec le nœud 2. Ce n'est que lorsque le nœud 2 aura terminé lui aussi l'exécution de T_B et diffuser les informations relatives à son exécution que les nœuds 1 et 2 pourront se répartir la charge d'exécution relative à T_B. Tous les tempos ayant par la suite été exécutés une première fois et les agents de simulation leur étant rattachés ayant été idéalement distribués sur les nœuds de simulation, l'exécution du reste de simulation se fait sans avoir de temps d'attente entre les nœuds.

La figure 4.8(d) nous montre que l'exécution anticipée proposée par notre algorithme d'exécution parallèle a encore un intérêt lorsque les agents de simulation sont idéalement répartis sur l'infrastructure d'exécution. En identifiant les liens de dépendance entre tempos, nous pouvons identifier quels sont les points de synchronisation réellement nécessaires. Ainsi, dans l'exemple, l'indépendance de T_D par rapport à T_C permet d'effectuer la synchronisation (la mise à jour des environnements influencés au cours de l'exécution de T_C) pendant l'exécution de T_D : il n'y a aucun temps d'attente entre l'exécution de T_C et celle de T_D.

Toutefois, ce mécanisme d'exécution parallèle perturbe l'équilibrage de charges par anticipation. L'équilibrage de charges par anticipation permet de modifier la répartition des agents de simulation associés à un tempo T_i tel que $t_s < T_{si}$ (t_s étant le temps simulé du tempo en cours d'exécution). Or le mode d'exécution parallèle que nous proposons permet d'anticiper l'exécution d'un tempo. Il serait alors possible qu'une plate-forme reçoive des agents de simulation rattachés au tempo T_i alors même qu'elle ait anticipé l'exécution du tempo T_i et donc déjà traité les agents associés qu'elle hébergeait avant de recevoir de nouveaux agents de simulation. Pour éviter ce phénomène, les agents de flexibilité ne doivent pas effectuer d'équilibrage de charges par anticipation lorsque l'ordonnancement parallèle est en cours.

Les agents de flexibilité des plates-formes de simulation peuvent s'accorder pour activer/désactiver les différents mécanismes d'optimisation de la simulation. Ainsi, en début de simulation, l'équilibrage de charges par anticipation ne peut fonctionner (car nécessite un historique des performances et temps de simulation) et il est beaucoup plus intéressant de disposer d'un ordonnancement parallèle de la simulation fonctionnant de pair avec l'algorithme d'équilibrage de charges classique. Par la suite, à l'équilibre de charges, notre algorithme d'exécution parallèle peut devenir moins pertinent que la possibilité d'anticiper les équilibrages de charges : les agents de flexibilité peuvent décider de désactiver le premier mécanisme au profit du second.

4.7 Synthèse

Nous nous sommes intéressés dans ce chapitre à la définition d'un algorithme d'équilibrage de charges dynamique adapté à la simulation orientée agents. Cet algorithme est employé par les agents de flexibilité de chaque plate-forme pour optimiser dynamiquement la répartition des agents de simulation sur les plates-formes constituant l'infrastructure d'exécution. Cette approche permet notamment de conserver l'aspect purement thématique des agents de simulation : ils sont préservés de toute considération sur le support d'exécution (exécution mono-poste mono-processeur ou distribuée).

Cet algorithme fonctionne de manière distribuée : il est partagé entre les agents de flexibilité qui, à partir d'informations sur les performances de simulation de chaque plate-forme, négocient entre eux des transferts d'agents de simulation. Il sert de plus à satisfaire deux exigences, parmi celles énoncées au paragraphe 2.2.3 à savoir l'optimisation des performances en simulation et le support de la flexibilité de l'infrastructure d'exécution. En effet, considérant les seules déconnexions voulues par l'utilisateur, nous considérons que l'agent de flexibilité d'une plate-forme devant se déconnecter aura la capacité de se décharger de ces agents de simulation avant de se déconnecter. De même, une plate-forme venant de se connecter sera à même d'accueillir des agents de la simulation qui lui seront transmis par les agents de flexibilité des plates-formes déjà connectées.

Les algorithmes d'équilibrage de charges existants nous semblent peu adaptés au cadre de la simulation orientée agents distribuée dans la mesure où cette dernière nécessite une synchronisation régulière entre les plates-formes de simulation (les tâches à distribuer ne sont pas totalement indépendantes). Ainsi, le but de l'équilibrage de charges est ici de minimiser les temps d'attentes de synchronisation que peuvent rencontrer les plates-formes participant à la simulation. Souhaitant de plus tirer parti des solutions présentées dans notre approche de distribution, nous avons défini notre propre algorithme. Celui-ci repose sur trois phases :

- diffusion d'informations. S'appuyant sur la communication en réseau par diffusion, les agents de flexibilité de chaque plate-forme participant à la simulation fournissent un ensemble d'informations sur l'exécution de la simulation sur leur plate-forme. Ces informations servent à déterminer si un équilibrage de charges est nécessaire et à établir le meilleur transfert d'agents possible d'une plate-forme à une autre.
- détection de surcharge. A partir des informations diffusées, chaque agent de flexibilité vérifie leur condition de satisfaction vis-à-vis de l'exécution courante de la simulation. Il compare ainsi les performances de sa plate-forme par rapport aux autres. Si son niveau de satisfaction est suffisant, il n'entame aucune autre procédure. Mais si le niveau de satisfaction est insuffisant, l'agent de flexibilité exécute la dernière phase de notre algorithme.
- transfert d'agents. Un agent de flexibilité ayant estimé son niveau de satisfaction insuffisant quant aux performances de sa plate-forme par rapport aux autres, il va transmettre une

partie des agents de simulation dont il a la charge à une autre plate-forme de l'infrastructure. La connaissance de l'état des autres plates-formes lui permet d'identifier le meilleur interlocuteur et de dimensionner au mieux le transfert d'agents (le choix des agents de simulation à transférer reste une problématique que nous avons abordé au paragraphe 4.4.5).

Pour pouvoir bénéficier de l'optimisation apportée par cet algorithme il est nécessaire d'identifier une *unité logique de simulation* sur la base de laquelle comparer les performances en simulation chaque plate-forme. Dans le cas du modèle temporel à pas de temps constant et d'une synchronisation sur chaque pas de temps, le pas de simulation constitue tout naturellement cette *unité logique de simulation*. Nous nous intéressons plus particulièrement à la mise en œuvre de notre algorithme sur une simulation employant le modèle à Temporalité. Nous montrons que le tempo constitue le meilleur élément sur la base duquel évaluer les performances et répartir la charge de simulation.

La richesse de la description temporelle proposée par le modèle à Temporalité permettant d'établir un axe temporel prévisionnel, à partir duquel il est possible d'anticiper la charges de simulation d'un tempo et évaluer si un équilibrage de charges permettrait d'en optimiser l'exécution. Se basant sur l'historique d'exécution du tempo considéré, les agents de flexibilité peuvent alors évaluer la charge que représentera ce tempo et déclencher un équilibrage de charges anticipé. Cette technique est avantageuse puisqu'elle permet d'effectuer des transferts d'agents au cours de l'exécution d'autres tempos (elle ne ralentit pas les synchronisations) et, bien évidemment, contribue à réduire la durée d'exécution distribuée du tempo.

Nous intéressant à l'utilisation conjuguée de notre algorithme d'ordonnancement global parallèle et de notre algorithme d'équilibrage dynamique, nous montrons de manière théorique que ces deux algorithmes se complètent pour optimiser la simulation. L'ordonnancement parallèle ne se basant que sur les connaissances du modèle de simulation, il se montre efficace dès le début de la simulation. L'algorithme d'équilibrage de charges nécessite lui l'exécution d'au moins une occurrence d'un tempo pour optimiser la répartition des agents de simulation activés durant ce tempo. Réduisant drastiquemment les écarts de durée de simulation d'un tempo entre chaque plate-forme, il minimise le recours à l'ordonnancement parallèle après un certain temps de simulation (dans le cas où la dynamique de la simulation et celle de l'infrastructure sont toutes deux stables) : l'ordonnancement parallèle est alors avant tout utilisé pour identifier les liens de causalité entre tempos successifs et ainsi réduire le nombre de points de synchronisation nécessaires.

Si nous constatons que l'ordonnancement parallèle que nous proposons n'est pas compatible avec le mécanisme d'équilibrage de charges par anticipation, nous observons que ces deux outils d'optimisation se montrent les plus performants à des moments de la simulation différents. L'ordonnancement parallèle est surtout utile lorsque la répartition des agents de simulation n'est globalement pas optimale (ce que nous considérons être le cas en début de simulation) alors que l'équilibrage de charges par anticipation nécessite un historique d'exécution des tempos pour

fonctionner. Ces deux mécanismes peuvent donc être activés ou désactivés au cours de la simulation par les agents de flexibilité : compte tenu de l'état de la simulation, ces derniers se concertent pour déterminer quels sont les outils d'optimisation à employer pour réduire au mieux la durée de la simulation distribuée.

Chapitre 5

Prototypage et expérimentations

Sommaire

5.1 Intérêt du prototypage 112
5.2 Usages de la simulation d'exécution parallèle et distribuée de SOA 113
5.3 Simulation d'exécution parallèle et distribuée de SOA 115
5.3.1 Description générale 115
5.3.2 Description des agents 116
5.3.3 Description des environnements 117
5.3.4 La question du temps réel simulé 120
5.3.5 Le comportement des agents 122
5.4 Expérimentations 126
5.4.1 Vérification du fonctionnement général 127
5.4.2 Etude de l'équilibrage de charges 131
5.4.3 Exemple d'utilisation 137
5.5 Discussions 139
5.5.1 Limites du prototype actuel 139
5.5.2 Discussions 141
5.6 Synthèse 142

Nous avons établi au chapitre 2 une architecture flexible et générique, centrée sur un système multi-agents de distribution, et conçue de façon à supporter la distribution de l'activité de simulation. Nous nous sommes par la suite focalisés sur l'optimisation des performances d'exécution de la simulation et plus particulièrement sur l'optimisation de l'exécution des agents.

Conscients qu'il reste encore plusieurs étapes avant d'obtenir une solution de distribution totalement opérationnelle, nous proposons dans ce chapitre de valider nos propositions concernant l'optimisation de l'exécution parallèle et distribuée de simulation par le biais d'un prototype prenant la forme d'une simulation orientée agent. Nous montrons que ce prototype est aussi bien utilisable comme base d'expérimentation d'algorithmes d'exécution distribuée que comme outil

d'évaluation des modèles simulés. Il s'intègre enfin dans une démarche vers la mise en œuvre effective de notre proposition dans un cadre réel.

5.1 Intérêt du prototypage

La mise en œuvre de notre architecture dans un cas réel est une tâche de grande ampleur difficile à concilier avec le cadre du travail de thèse. Nous considérons ainsi une approche d'expérimentation de nos propositions par simulation. Nous avons présenté au paragraphe 2.4 un système multi-agents de distribution réparti sur l'infrastructure d'exécution ayant pour charge de gérer les échanges et communications entre les plates-formes pour supporter la distribution. Ce SMA, véritable noyau de notre solution de distribution, s'intègre aux plates-formes de simulation ainsi qu'au réseau par l'intermédiaire de couches d'abstraction, les *spaces* (voir paragraphe 2.4.6), qui servent d'interface entre le paradigme agent des agents micro distribution et les autres composants de la plate-forme. Par le biais des spaces, le SMA de distribution supervise la distribution de la SOA au travers d'interactions avec des environnements.

La supervision et l'optimisation de la distribution étant assurées par un SMA manipulant un environnement, c'est tout naturellement que nous proposons d'éprouver ce SMA en l'insérant dans une simulation en tant qu'application thématique : les environnements qu'il perçoit ne sont pas connectés à un système réel mais simulent ce dernier. Cette approche vise la validation de nos propositions mais peut aussi se révéler utile en vue de la réalisation du système réel.

L'expérimentation par simulation nous permet de construire un banc d'essai pour nos propositions d'optimisation. Reprenant la problématique de la distribution des environnements de la simulation, dans le cadre de notre simulation, nous pouvons tester nos algorithmes sans avoir à prendre en compte la surcharge de messages réseau induite par la non-distribution des environnements. Nous pouvons donc isoler et éprouver un aspect de notre solution de distribution avant de l'intégrer dans le fonctionnement global.

Les simulations orientées agent étant hautement paramétrables, il est très aisé de tester nos travaux sous différentes configurations. Ces configurations nous permettent de faire varier les modèles de simulation qui sont à exécuter de manière distribuée aussi bien que les différentes plates-formes et réseaux composant l'infrastructure d'exécution.

Notre prototype constitue un premier pas vers la mise en œuvre réelle de notre architecture. Le passage du système réel à la simulation revient à plonger le SMA de distribution dans des environnements entièrement virtuels : ces environnements représentent la simulation ainsi que les autres éléments avec lesquels interagit le SMA de distribution. Ainsi, les agents de notre simulation, représentant le SMA de distribution dans la réalité, adoptent un fonctionnement similaire à celui de ces derniers. L'implémentation des agents de notre simulation peut donc servir de base pour la conception du SMA de distribution.

Afin de nous dégager de toute ambiguïté concernant la simulation orientée agent dont il serait question, nous désignerons par le terme *simulation thématique* la simulation dont nous souhaitons assurer l'exécution distribuée. Notre prototype nous sert donc à simuler ce que seraient les performances d'exécution distribuée et parallèle de cette simulation thématique sur un réseau d'ordinateur.

5.2 Usages de la simulation d'exécution parallèle et distribuée de SOA

La simulation de l'exécution parallèle et distribuée de simulations orientées agent sert plusieurs objectifs. Dans nos travaux, le premier de ces objectifs est bien évidemment de pouvoir vérifier le fonctionnement de nos propositions en terme d'optimisation de l'exécution de la simulation. De manière plus générale, ce prototype modélise de manière synthétique une simulation orientée agent qui doit être exécutée sur plusieurs plates-formes. En tant que tel, il constitue une base expérimentale pour l'élaboration de divers mécanismes ayant trait à l'exécution distribuée de SOA.

Si les informaticiens considèrent les algorithmes des agents micro de distribution, les concepteurs de simulation peuvent aussi employer la simulation que nous proposons pour évaluer différents choix de conception ou de déploiement. Nos algorithmes d'optimisation exploitent des informations issues du modèle simulé, ces derniers sont donc sensibles à la façon dont est modélisé le système que souhaitent simuler les thématiciens. Nous montrons au paragraphe 5.5.2 que les règles de conception qu'ils exploitent, afin d'offrir les meilleures performances en exécution distribuée, aboutissent à la réalisation d'un modèle de simulation robuste et extensible. Notre prototype, en permettant de faire varier le nombre de plates-formes ou le réseau composant l'infrastructure d'exécution, permet d'évaluer les performances attendues du système réel et donc de dimensionner au mieux l'infrastructure d'exécution.

Il est possible d'envisager un autre usage pour cette simulation, usage dans lequel elle serait exécutée en marge du système réel. En effet, les agents micro de distribution exploitent les informations du modèle de simulation. Or, considérant une simulation thématique large-échelle distribuée sur une infrastructure dynamique, l'accès aux informations du modèle simulé complet peut être délicate. Une alternative serait alors d'exploiter la simulation nous servant de prototype pour déduire des actions permettant d'optimiser l'exécution réelle de la simulation thématique. Autrement dit, chaque plate-forme simule l'exécution distribuée en parallèle de la simulation thématique afin de déterminer les actions à entreprendre pour optimiser son exécution réelle.

Cette approche, consistant à observer en temps réel un modèle pour en piloter le système réel, est connue en automatique : pour les modèles représentés sous forme de représentation d'état, lorsque l'état d'un système n'est pas directement mesurable, il est possible d'en construire un

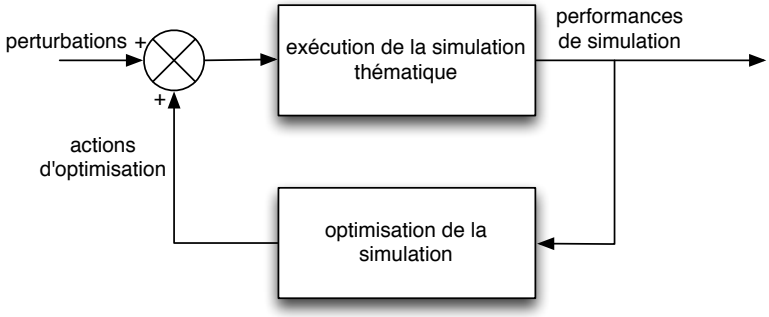


FIG. 5.1 – Représentation sous forme de *schéma-bloc* du mécanisme d'optimisation de la simulation thématique.

observateur qui pourra fournir cet état à partir d'un modèle du système dynamique et d'autres grandeurs accessibles à la mesure [Luenberger, 1964]. Pour mieux appréhender cette technique dans notre cas, nous tentons de représenter notre solution d'optimisation, que nous qualifions ici de « classique » sous forme de *schéma-bloc* (voir figure 5.1).L'exécution de la simulation constitue le système que nous souhaitons piloter. Il n'y a pas, à proprement parler, de grandeur d'entrée (de consigne) mais des perturbations en entrée sont à considérer, ces dernières représentant les fluctuations des performances de l'infrastructure d'exécution. Se basant sur la mesure de la performance de simulation, notre « correcteur », c'est-à-dire notre mécanisme d'optimisation de la simulation thématique observe celle-ci et produit des actions sur le moteur d'exécution de la simulation.

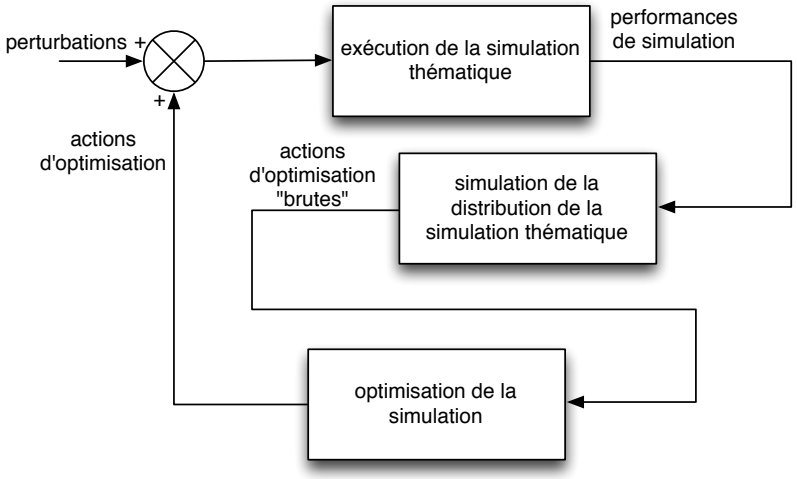


FIG. 5.2 – Emploi de la simulation d'exécution distribuée de simulation thématique pour l'optimisation de l'exécution distribuée réelle.

Considérant comme impossible ou pénalisante l'observation de certains éléments de la simulation thématique, nous pouvons construire notre équivalent d'un observateur de Kalman [Kalman and Bucy, 1973]. Ce qui conduit, dans notre cas, au *schéma-bloc* de la figure 5.2. Les performances

en simulation distribuée du système réel servent à ajuster l'état de la représentation du modèle de simulation thématique dans notre simulation (cet ajustement requiert peu d'informations sur le système réel et est donc moins coûteux que l'observation complète). Au travers de la simulation de l'exécution distribuée et parallèle, plusieurs actions d'optimisation de la simulation sont proposées au SMA de distribution réel : ce dernier reste libre de les appliquer ou non au système réel[16].

5.3 Simulation d'exécution parallèle et distribuée de SOA

5.3.1 Description générale

Notre prototype est développé sous la forme d'une simulation orientée agent implémentée sur la plate-forme GEAMAS-NG en Java 6. L'utilisation de cette plate-forme nous permet de bénéficier de plusieurs atouts :

- la gestion native du multi-environnements. Nous avons vu précédemment que le SMA distribué est en interaction avec différents environnements. Si nous ne nous focalisons ici que sur l'exécution de la simulation thématique, restreignant ainsi le nombre d'environnements à modéliser, il n'en reste pas moins nécessaire de gérer plusieurs environnements.
- la possibilité de changer de modèle temporel. Dans une simulation, le temps est abordé sous la forme d'un environnement où les agents définissent leurs règles d'activation. Le changement de modèle temporel ne nécessite alors qu'une révision des influences que les agents produiront sur l'environnement temporel.
- l'implémentation rigoureuse du modèle perception/influence. Dans le système réel, les agents micro de distribution ne peuvent être sûr que leurs influences auront les effets attendus. Le fait de pouvoir reproduire ce phénomène nous permet d'évaluer la réactivité de ces agents.
- le langage d'initialisation XELOC [David *et al.*, 2009]. Basé sur la syntaxe *XML*, Ce langage de script facilite la réalisation des tests en nous permettant de modifier très simplement la représentation du modèle de simulation thématique aussi bien que la composition et les performances de l'infrastructure d'exécution.

Nous inscrivons notre prototype dans un cycle de développement incrémental. Ce cycle nous permet de valider par étapes les techniques d'optimisation que nous proposons mais aussi de travailler à leur opérationnalisation. En effet, notre démarche tout au long de nos travaux étant d'exploiter les informations contenues dans le modèle simulé, nous concevons la simulation en détaillant au fur et à mesure le modèle de simulation thématique que les agents exploitent.

[16]Nous soulignons ici que, par système réel, nous entendons l'exécution distribuée et parallèle de la simulation thématique et non le système réel que cette simulation modélise

L'objectif de notre prototype est d'éprouver les algorithmes des chapitres 3 et 4 qui constituent pour nous les contributions centrales de ce travail de thèse. En ce sens, l'infrastructure d'exécution et la simulation thématique sont représentées de manière synthétique : leurs modèles ne visent qu'à permettre l'exécution de nos algorithmes. Ainsi, le modèle de simulation que nous considérons n'est pas suffisamment complexe et riche pour justifier le détail des Agents Macroscopiques Distribués en Agents de Cohérence, de Flexibilité et Réseau. Ces agents offrent un plus grand potentiel et feront l'objet de travaux futurs de l'équipe.

5.3.2 Description des agents

Virtualisant notre architecture, il est nécessaire de virtualiser les agents micro de distribution (A_R, A_C et A_F) et notamment leur moteur d'exécution. L'intégration d'un moteur d'exéction SMA au sein même du moteur d'exécution SOA de GEAMAS-NG n'est pas supporté par la plate-forme. Ainsi, à travers l'exécution d'un agent macro de distribution, le moteur d'exécution de GNG exécute les agents A_R, A_C et A_F qui communiquent alors de manière synchrone. Cette approche est cohérente avec les algorithmes présentés aux chapitres 3 et 4 dans la mesure où nous ne considérons pas ici de protocoles de négociation.

Au niveau du moteur d'exécution de GEAMAS-NG, n'apparaissent que les agents macro de distribution (les agents micro sont activés à l'intérieur de ces derniers). Ces agents adoptent tous un comportement similaire. Un agent gère l'exécution de la simulation sur une plate-forme ; son but est de faire exécuter par sa plate-forme hôte la part de simulation qui lui incombe et d'obtenir une durée globale de simulation proche de la moyenne des durées de simulation des autres plates-formes. Pour cela il dispose des deux mécanismes que nous avons proposés :

- un mécanisme d'exécution parallèle de la simulation. Nous considérons que la simulation thématique emploie le modèle à Temporalité et que les environnements perçus et influencés à chaque temporalité sont renseignés. A un instant donné de notre simulation, un agent AM_D a la possibilité de choisir le tempo qu'il va exécuter, indépendamment des tempos exécutés sur les autres plates-formes, pour peu que ce tempo puisse être exécuté sans violation du principe de causalité. Dans notre prototype, une plate-forme ne peut exécuter qu'un seul tempo à la fois (la plate-forme ne possède qu'un unique *thread* de simulation) ; une fois le tempo choisi il est exécuté dans son intégralité sans interruption.
- un mécanisme d'équilibrage de charges. Seuls les agents macro de distribution dont l'A_F constate de mauvaises performances de simulation utilisent ce mécanisme. Ce dernier permet de transférer une partie de la charge de simulation d'une plate-forme à une autre.

Chaque agent AM_D possède une temporalité au cours de laquelle il fait évoluer l'exécution de la part de simulation thématique hébergée sur sa plate-forme en employant les deux mécanismes précédents. Un agent AM_D n'a pas de mémoire ou d'attributs spécifiques. Il possède par contre des corps dans les différents environnements auxquels sont associés plusieurs attributs. Ainsi, le

fonctionnement des agents repose essentiellement sur les perceptions et influences qu'ils réalisent : le code des agents contient uniquement « l'intelligence » leur permettant d'utiliser à bon escient leurs capacités de perception et d'influence pour réaliser leur objectif. Cette caractéristique facilite le développement incrémental et la programmation de comportements variés.

5.3.3 Description des environnements

Le développement des agents est axé sur leur comportement ; toutes les données manipulées par les agents proviennent des environnements au travers du mécanisme de perception/influence. Ces derniers jouent donc un rôle clé dans la simulation. Au paragraphe 2.4 nous avions identifié plusieurs environnements à prendre en compte dans le mécanisme d'exécution de la simulation thématique :

- un environnement réseau. C'est le support des communications entre agents AM_D et entre agents de simulation se trouvant sur des plates-formes différentes.
- un environnement de plates-formes. Les performances d'une plate-forme peuvent varier au cours de la simulation ou cette dernière peut quitter la simulation. Si tel est le cas, nous considérons qu'elle transmet d'abord aux plates-formes restantes la charge de simulation qui lui incombe.
- un environnement représentant la simulation thématique. Les agents n'influent absolument pas sur la simulation thématique en tant que telle. Ils l'observent pour optimiser sa répartition sur les plates-formes et pour permettre l'exécution parallèle. Nous rappelons ici que nous nous sommes focalisés sur l'exécution des agents de simulation et non sur les environnements de la simulation thématique.

Les fonctions et informations de ces environnements sont scindées ou mutualisées dans trois environnements différents.

Environnement de communication

Le réseau est le support de communication entre plates-formes. Sa fonction est d'assurer les échanges de messages entre plates-formes mais il induit des retards dans la transmission de ces messages. Nous scindons ici ces deux faits : l'environnement de communication assure uniquement la transmission des messages d'une plate-forme à une autre et ne modélise donc pas les effets que le réseau induit sur l'exécution de la simulation.

Cet environnement de communication s'appuie sur celui par défaut de GEAMAS-NG qui définit une boîte aux lettres propre à chaque agent. La communication en diffusion est réalisée en adressant une copie du message envoyé à chaque boîte aux lettres présente dans l'environnement.

Le modèle d'interaction entre agents et environnements de GEAMAS-NG ne permet pas à l'environnement de communication de générer une influence sur l'agent pour l'avertir de l'arrivée

d'un message. C'est donc à l'agent lui-même de relever sa boite aux lettres. Si ce mode de fonctionnement peut aussi être utilisé dans le déploiement réel de notre solution de distribution, l'agent réseau peut influencer les autres agents micro de distribution de sa plate-forme pour le traitement de messages critiques.

Environnement réseau

L'environnement réseau sert ici uniquement à mesurer l'impact du réseau sur l'exécution de la simulation thématique, le transfert des messages entre agents étant assuré par l'environnement de communication. Ce découplage du fonctionnement normal d'un réseau en deux environnements nous permet de pouvoir estimer des temps de transfert réseau sans nécessairement produire de messages entre les agents. Par exemple, plusieurs messages liés aux agents de la simulation thématique doivent normalement transiter entre les plates-formes à chaque pas de simulation (ce sont les requêtes de perception ou d'influence des agents de la simulation thématique). Dans notre modèle il est intéressant de tenir compte de la charge réseau induite par ces messages sans pour autant devoir les diffuser réellement aux agents macro de distribution du prototype.

Le réseau est modélisé sur la base de deux informations :

- $delay$: le temps de réponse du réseau, il s'exprime en ms.
- $throughput$: le débit du réseau, il s'exprime en Ko/ms.

Il permet d'évaluer le temps nécessaire pour la diffusion d'une charge sur le réseau (émission d'un message à chaque plate-forme) comme le temps nécessaire pour un transfert point à point. La communication point à point est utilisée dans notre modèle pour les transferts d'agents de simulation thématique d'une plate-forme à une autre. A une charge $load$ donnée en ko, le temps de transfert point à point sera donné par $\frac{load}{throughput} + delay$ et le temps en diffusion par $nb_{ptf} \times (\frac{load}{throughput} + delay)$ (avec nb_{ptf} le nombre de plates-formes connectées).

Environnement de simulation

Il existe un lien fort entre les agents AM_D et les plates-formes de simulation : chaque plate-forme dispose d'un unique agent AM_D. La connexion/déconnexion d'une plate-forme se traduit donc par la création ou la suppression d'un agent AM_D. Le seul intérêt d'un environnement relatif aux plates-formes de simulation semble résider dans la représentation de la capacité de simulation de chacune des plates-formes (performances de l'ordinateur hébergeant la plate-forme). Or la performance de l'ordinateur n'est pas une valeur directement accessible : elle ne peut être qu'évaluée. Cette évaluation revenant à la charge de l'AM_D, nous ramenons cette mesure comme une valeur inhérente au corps de l'agent dans l'environnement de simulation.

L'environnement de simulation représente donc la simulation thématique dans sa globalité et le corps de chaque agent est considéré comme un moteur d'exécution de la simulation grâce

auquel un AM_D pourra faire exécuter une partie de la simulation. Ce moteur d'exécution/corps de l'agent possède sa propre puissance de simulation que nous notons *power*.

Considérant uniquement l'exécution des agents de simulation dans la simulation thématique, il n'est pas nécessaire de modéliser les environnements de la simulation thématique dans notre environnement de simulation. Notons que si notre algorithme d'exécution parallèle s'appuie sur la connaissance des environnements perçus et influencés, il ne nécessite pas de connaître des informations sur ces environnements autres que leur nom ; il n'est donc pas nécessaire de leur donner une représentation autre qu'un identifiant.

Devant représenter les agents de simulation de la simulation thématique, nous avons opté pour une vision des plus synthétiques afin de faciliter la représentation et donc l'initialisation de l'environnement de simulation dans notre simulation. La modélisation de la simulation thématique s'appuie sur des *familles d'agent de simulation* définies par le tuple suivant : $F_{SimAgent} = \{id, cost, nbAgents, agentSize, repartition\}$ avec :

- id : l'identifiant de la famille d'agents de simulation.
- $cost$: le coût d'exécution d'un agent de cette famille. Il s'agit du temps nécessaire à simuler un agent de cette famille sur un ordinateur de référence.
- $nbAgents$: le nombre d'agents de simulation de cette famille.
- $agentSize$: il s'agit de la taille en Ko des structures de données associées à un agent. Ce sont ces structures qui sont déplacées d'une plate-forme à une autre dans le cadre d'un transfert d'agent (le code associé au comportement est un élément statique dupliqué sur toutes les plates-formes de simulation).
- $repartition$: ce tableau donne la répartition en nombre d'agents de cette famille sur les plates-formes de simulation.

Notre algorithme d'exécution parallèle de la simulation s'appuie très largement sur la description des temporalités du modèle de simulation thématique. Nous devons donc modéliser un des environnements de la simulation thématique dans notre environnement de simulation : l'environnement temporel.

L'environnement de simulation contient donc un objet représentant l'environnement temporel de la simulation. Conformément au modèle à temporalité, cet objet « environnement temporel » contient des tempos construits à partir des temporalités des agents de la simulation thématique. Comme il aurait été fastidieux de configurer toutes les temporalités de tous les agents, nous avons opté pour la description suivante des temporalités : $t = \{id, d, f, p, F_{SimAgent}id, E_R, E_W\}$ avec

- id : l'identifiant de la temporalité.
- d : l'instant de début de la temporalité.
- f : l'instant de fin de la temporalité.
- p : la période de la temporalité.
- $F_{simAgent}id$: l'identifiant de la famille d'agent à laquelle est associée la temporalité.

- E_R : l'ensemble des environnements potentiellement perçus au cours de l'exécution de la temporalité.
- E_W : l'ensemble des environnements potentiellement influencés au cours de l'exécution de la temporalité.

Dans ce modèle, nous considérons que tous les agents d'une même famille exécutent les mêmes temporalités. Après l'initialisation, et après chaque modification de l'objet représentant l'environnement temporel de la simulation thématique, les mécanismes internes à l'environnement de simulation se chargent de déterminer les tempos.

5.3.4 La question du temps réel simulé

Nous relevons ici le fait que chaque agent possède une unique temporalité et s'exécutent en même temps : la simulation définie une seule temporalité au cours de laquelle tous les agents sont activés. Ceci constitue un cas particulier où le modèle à Temporalité se comporte de manière similaire au modèle à pas de temps constant. Nous exploitons cette similitude afin de mieux clarifier nos propos. Nous utiliserons donc le terme *pas de temps* pour désigner l'activation des agents AM_D.

Lorsque l'on procède à une simulation orientée agent d'un phénomène, on plonge celui-ci dans un temps virtuel. Lorsque ce phénomène est lui-même une simulation orientée agent, il suffit de virtualiser une fois de plus tous les éléments, le temps y compris. Dans notre prototype nous avons donc les temps suivants :

- le *temps réel* : c'est le temps (continue) que nous percevons et dans lequel s'exécute notre simulation.
- le *temps simulé de notre simulation* : le moteur d'exécution fait avancer virtuellement le temps dans notre simulation. Il s'agirait alors d'un « *temps réel simulé* » dans la mesure où ce temps est le temps réel pour la simulation thématique.
- le *temps simulé* dans notre modèle simulé thématique : c'est le temps simulé de la simulation thématique dont nous simulons l'exécution. Il s'agit donc d'un « *temps simulé simulé* ».

Les agents AM_D sont ici dédiés à l'exécution de la simulation thématique, c'est-à-dire qu'ils font évoluer le temps simulé dans celle-ci. Pour cela, ils utilisent les algorithmes d'ordonnancement parallèle de la simulation et d'équilibrage de charges que nous avons proposés. Pour observer l'exécution parallèle de la simulation thématique, il est nécessaire de pouvoir représenter la durée d'exécution de chaque tempo de la simulation thématique sur chaque plate-forme. Nous nous intéressons donc à la représentation du temps réel simulé et plus particulièrement à la représentation de sa durée. Deux options s'offrent à nous :

- un agent AM_D a la possibilité de traiter virtuellement un tempo de la simulation thématique (que nous désignerons par *tempo thématique*) à chaque pas de temps.
- l'exécution associée à une temporalité correspond à une durée de traitement de la simula-

tion. Auquel cas l'exécution d'un tempo thématique peut nécessiter plusieurs pas de temps ou, au contraire, plusieurs tempos pourraient être exécutés au cours d'un pas de temps en fonction de la puissance de la plate-forme hébergeant un agent AM_D.

Nous montrons que la première option ne convient pas au travers d'un exemple. Considérons deux agents AM_D1 et AM_D2 au temps réel simulé $t_{rs} = 0$ (ce temps correspond au lancement de l'exécution de la simulation thématique). Au premier pas de notre simulation, AM_D1 et AM_D2 exécutent tous deux un tempo thématique, c'est-à-dire qu'ils déterminent le temps réel simulé mis à exécuter ce tempo. Après le premier pas de temps, le temps réel simulé sur la plate-forme hébergeant AM_D1 est $t_{rs} = 12$ tandis que la plate-forme hébergeant AM_D2 affiche $t_{rs} = 20$. Cela signifie qu'au prochain pas de temps simulé, AM_D1 gérera l'exécution du temps réel simulé à $t_{rs} = 12$ alors que AM_D2 est censé simuler le comportement de notre système à $t_{rs} = 20$. Le choix du tempo thématique à exécuter par AM_D2 dépend néanmoins des tempos qui auront été exécutés avant $t_{rs} = 20$. Cette information ne sera disponible que lorsque AM_D1 affichera $t_rs \geq 20$. Ainsi, le traitement d'un tempo thématique à chaque pas de simulation contraint fortement l'évolution parallèle de l'exécution de la simulation thématique.

La deuxième méthode semble plus adaptée puisqu'elle propose de représenter la durée du temps réel simulé par une durée réelle (le pas de simulation). Mais se pose alors toutes les problématiques liées à la discrétisation du temps. En effet, si un pas de temps simulé représente une durée importante de temps réel simulé, cela signifie qu'un agent AM_D peut exécuter plusieurs tempos thématiques en un pas de temps. Or, pour pouvoir exécuter un tempo thématique il est nécessaire de vérifier ses dépendances. La mise à jour des dépendances ne peut avoir lieu qu'à la réception des messages de synchronisation des autres AM_D qui a lieu une seule fois par pas de temps. S'inspirant des règles d'échantillonnage, et notamment à partir du théorème de Shannon [Shannon, 2001 1949], il faut alors que la durée de temps réel simulé écoulée lors de l'exécution d'un pas de temps de notre simulation soit deux fois plus petite que le plus grand commun diviseur de la durée d'exécution virtuelle de tous les tempos thématiques de la simulation. La durée d'exécution virtuelle d'un pas dépendant à la fois des temporalités, du coût d'exécution des agents des familles considérées dans le tempo, du nombre d'agents simulés hébergés par la plate-forme de l'AM_D et de la puissance de traitement de cette dernière, une telle évaluation est impossible à faire.

Nous proposons une troisième voie constituant une solution intermédiaire entre ces deux approches. Nous attribuons une durée fixe de temps réel simulé à chaque pas de temps. Cependant au maximum un seul tempo thématique peut être virtuellement exécuté par AM_D par pas de temps : un AM_D nécessitant moins d'un pas de temps pour simuler l'exécution d'un tempo thématique ne peut procéder à l'exécution virtuelle d'un autre tempo thématique qu'au pas de temps suivant. Néanmoins les durées d'exécution des tempos à considérer pour l'équilibrage de charges ne seront pas les durées réelles simulées associées aux pas de temps mais la durée de simulation théorique calculée à partir des tempos, des familles d'agents et de la puissance de

traitement de la plate-forme. De cette manière, l'équilibrage de charges, qui se base sur la durée d'exécution des tempos pour redistribuer la simulation, n'est pas perturbé par la discrétisation du temps réel simulé et l'utilisateur peut constater en temps réel l'exécution (virtuelle) parallèle de la simulation thématique sur l'infrastructure d'exécution qu'il aura configuré.

5.3.5 Le comportement des agents

Nous avons opté pour une double représentation du temps réel simulé : il existe sous la forme d'une durée dans le temps réel pour pouvoir observer l'exécution parallèle de la simulation et le temps d'exécution exact de chaque tempo est calculé et employé pour conserver la précision de l'algorithme d'équilibrage de charges. Ayant fait ce choix de faire durer dans le temps réel le temps d'exécution virtuelle d'un tempo thématique, les agents de flexibilité (intégré dans les AM_D du point de vue du moteur de GEAMAS-NG) alternent trois phases : sélection d'un tempo thématique à exécuter, exécution de ce dernier et synchronisation des plates-formes/équilibrage de charges.

Sélection d'un tempo thématique à exécuter

Le but premier des agents de notre prototype est d'exécuter la simulation thématique, c'est-à-dire faire progresser le temps simulé simulé associé à cette dernière. Celle-ci étant basée sur le modèle à Temporalité, nos agents macro de distribution s'appuient sur l'ordonnancement parallèle que nous avons décrit au chapitre 3. Les agents disposent donc d'une table de dépendances sur la base de laquelle ils vont exécuter la simulation.

Par rapport au système réel, nous avons choisit de centraliser la table de dépendances. Dans la solution réelle, celle-ci est déterminée sur toutes les plates-formes participant à la simulation. Cette redondance est nécessaire pour la mise en œuvre d'une infrastructure non-centralisée mais, dans le cadre de notre prototype, il n'était pas nécessaire de faire exécuter le même algorithme de calcul de la table par chacun de nos agents. La table de dépendances est donc un élément de l'objet « environnement temporel thématique » de l'environnement de simulation accessible aux agents par requêtes de perception. A un instant donné de notre simulation, elle est représentée de la manière suivante :

```
TC | TB<Distributor0: 6, Distributor: 4>
TB |
TA | TC<Distributor0: 8, Distributor: 8> TB<Distributor0: 6, Distributor: 4>
```

Il existe une ligne de dépendance par tempo. Par exemple, ici T_A dépend de T_C et de T_B. Les tempos peuvent être exécutés dans des ordres et à des vitesses différentes sur les plates-formes. Une occurrence d'un tempo peut être partiellement exécutée : elle n'a été exécutée que par certains agents. Dans notre exemple, nous constatons que l'agent *Distributor*0 exécutera la

prochaine fois le tempo T_B à son occurrence $t_{ss} = 6$[17] alors que *Distributor* l'exécutera à $t_{ss} = 4$. En l'état, la table de dépendances ne pourrait autoriser l'exécution de T_C qu'à l'occurrence $t_{ss} = 4$. Quand un agent souhaite exécuter un tempo, il obtient de la table de dépendances l'instant d'activation le plus avancé de chaque tempo qu'il est autorisé à exécuter.

Il est possible que l'agent ait déjà exécuté certains tempos à l'instant d'activation indiqué dans la table. Pour ne pas exécuter une deuxième fois la même occurrence d'un tempo, chaque agent dispose d'une *table d'activation* où il note pour chaque tempo quelle est la prochaine occurrence de ce dernier qu'il exécutera.

Un agent détermine donc la liste de tous les tempos qu'il peut exécuter à un moment réel de la simulation à partir de la table de dépendances et de sa table d'activation. Une fois cette liste dressée, il sélectionne les tempos partageant le temps d'activation le plus tôt. Deux types de tempos sont à considérer ici :

- ceux pour lesquels il n'aura aucun traitement à faire. Il s'agit des tempos pour lesquels la plate-forme hôte ne dispose pas d'agents de simulation thématique à exécuter (ce qui est symptomatique d'une mauvaise répartition). Du point de vue de l'agent, il s'agit de tempo *vides*.
- ceux pour lesquels il dispose d'une charge d'agents de simulation thématique à exécuter.

L'agent exécutera tant que possible un tempo non-vide. S'il n'y a pas de tempo non-vide exécutable, il choisira le tempo devant s'exécuter le plus tôt sur l'axe temporel. A partir de la charge d'agents de simulation thématique associés à ce tempo et hébergés par la plate-forme, il détermine le temps nécessaire à exécuter le tempo :

$$simTime(a, T_I) = \frac{\sum_{t_k \in T_I}(repartition_{F_{SimAgent}(t_k)}(a) \times cost_{F_{SimAgent}(t_k)})}{power(a)}$$

avec a l'agent macro de distribution, T_I le tempo concerné, t_k les temporalités associées à ce tempo et $F_{SimAgent}(t_k)$ la famille d'agents activés au tempo t_k.

La plate-forme associée à l'agent a mettra $simTime(a, T_I)$ à exécuter le tempo. Ce temps représente une durée en nombre de pas de simulation : nous obtenons la durée dans le temps réel simulé en multipliant cette valeur par la durée d'un pas de temps $simTime_{rs}(a, T_I) = stepTime \times simTime(a, T_I)$. Le nombre de pas nécessaires au traitement du tempo est obtenu en prenant l'entier majorant la valeur $simTime(a, T_I)$: $nbPas = ceil(simTime(a, T_I))$.

Si un unique pas de temps suffit à simuler l'exécution du tempo thématique considéré, l'agent clôt l'exécution de son pas de temps en diffusant le message de fin d'exécution du pas de temps (le temps de diffusion de ce message est bien évidemment déterminé en s'appuyant sur l'environnement réseau). Comme nous l'avons dit précédemment, l'information de durée du pas de temps communiquée ne sera pas le nombre de pas de simulation nécessaire, mais la valeur

[17] t_{ss} est le temps simulé simulé, c'est-à-dire la valeur du temps simulé dans la simulation thématique.

$simTime(a, T_I)$ de manière à ce que l'équilibrage de charges éventuel garde un maximum de précision.

Exécution du tempo sélectionné

Cette phase consiste tout simplement à simuler la durée d'exécution d'un tempo thématique, cette exécution pouvant nécessiter plusieurs pas de temps réel. Le nombre de pas de temps nécessaires à l'exécution virtuelle du tempo thématique est calculé à la fin de la phase de sélection de tempo à exécuter. En phase d'exécution, un agent n'a donc aucun traitement à effectuer si ce n'est de mettre à jour le compteur du nombre de pas et vérifier la fin d'exécution du tempo thématique.

A la fin de cette exécution, l'agent macro de distribution diffuse un message de fin d'exécution de tempo à tous les autres agents macro de distribution et détermine le temps mis par le réseau à diffuser ce message. Comme nous l'avons vu dans les précédents chapitres, ce message de fin d'exécution sert aussi bien à la synchronisation des plates-formes (notamment la mise à jour des environnements) qu'à la diffusion d'informations utiles à l'équilibrage de charges ainsi qu'à la mise à jour des temps d'activation indiqués dans la table de dépendances. Dans le cadre de notre prototype, où la simulation thématique est modélisée très simplement, les messages de fin d'exécution servent uniquement l'équilibrage de charges.

synchronisation des plates-formes/équilibrage de charges

Dans notre système réel, la fin d'exécution d'un tempo entraîne la diffusion d'informations utiles à la synchronisation des plates-formes (les influences produites notamment) et à l'équilibrage de charges (phase de diffusion d'informations décrite au paragraphe 4.4.1). Dans notre prototype, la simulation thématique est représentée de manière synthétique et, nous focalisant uniquement sur son exécution, nous n'avons pas à prendre en compte de réelles synchronisations pour mise à jour des environnements. Durant cette phase, les agents se focalisent principalement sur l'équilibrage de charges et suivent les phases de notre algorithme telles que décrites au paragraphe 4.4.

Au début de chaque pas de simulation, un AM_D relève ses messages. Tous les messages reçus sont des messages de fin d'exécution de tempos. Pour chacun d'entre eux, il vérifie si le tempo complet a été exécuté, c'est-à-dire si le nombre de plates-formes ayant déclaré l'intention d'exécuter le tempo est égal au nombre de messages de fin d'exécution du tempo considéré. Si tel est le cas, l'agent détermine son état de satisfaction vis-à-vis de l'exécution du tempo et procède si besoin à un équilibrage de charges. Toutefois, dans ce prototype, les plates-formes ne négocient pas l'équilibrage de charges : il est directement produit sous la forme d'une influence dans l'environnement de simulation et sans accord de la plate-forme destinatrice.

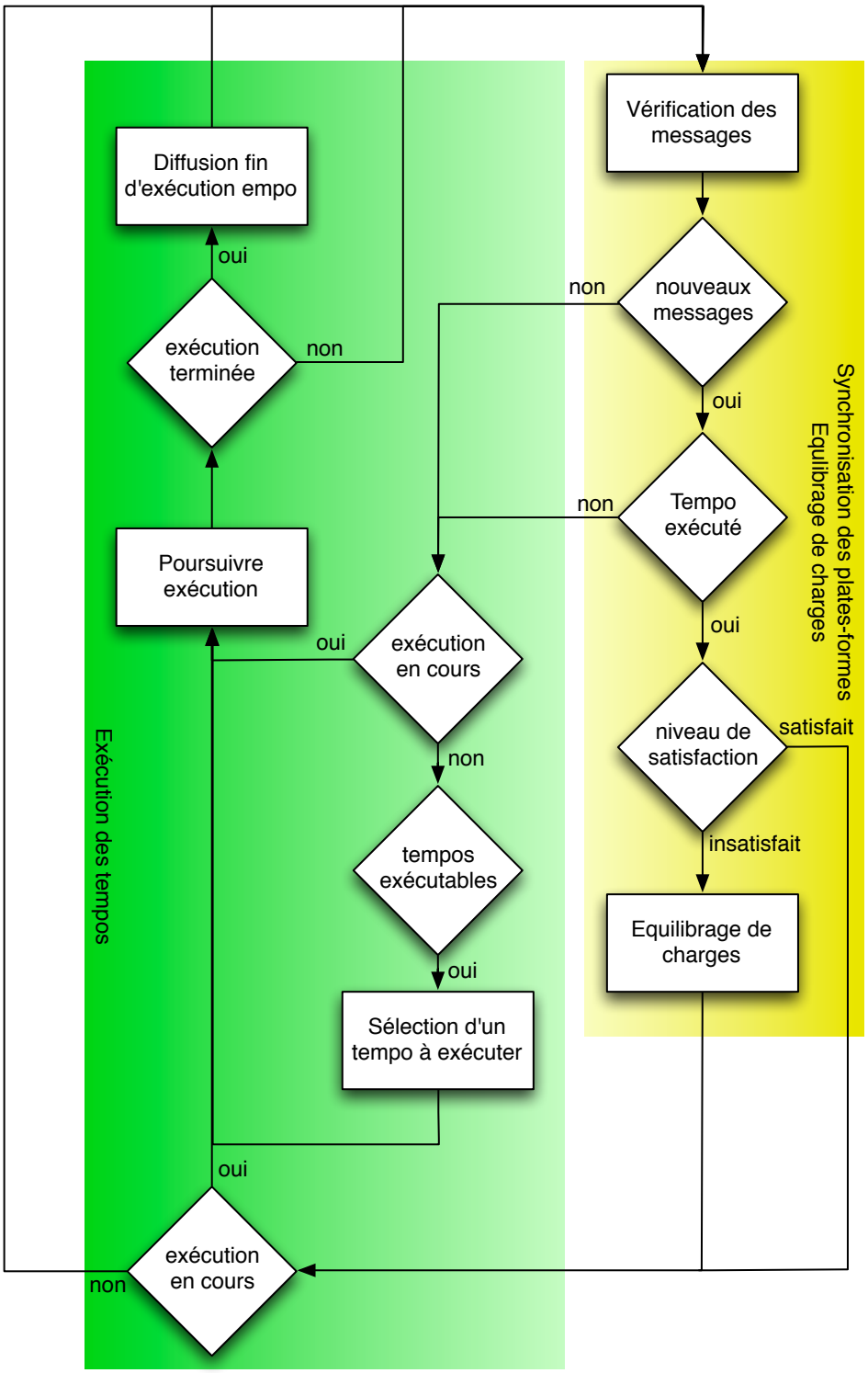


FIG. 5.3 – Comportement global d'un agent macro de distribution dans notre prototype.

Comportement global

Le comportement des agents macro de distribution est basé sur les trois étapes décrites précédemment. La logique d'enchaînement de ces phases au cours du comportement d'un agent est présentée en figure 5.3. La première étape de leur comportement est la relève des messages pour procéder à un éventuel équilibrage de charges. En effet, dans l'éventualité d'un équilibrage de charges, l'agent macro de distribution ne doit pas démarrer l'exécution d'un nouveau tempo puisque sa charge serait amenée à évoluer au pas de temps suivant. Les agents macro de distribution étant satisfaits de l'exécution d'un tempo ne savent pas s'ils vont accueillir de nouveaux agents de la simulation thématique : ils ne peuvent pas non plus lancer l'exécution d'un tempo. A partir du moment où un agent constate l'exécution complète d'un tempo thématique, il considère que les plates-formes sont en synchronisation et ne commencera l'exécution d'un tempo qu'au pas de temps suivant.

Un agent peut néanmoins poursuivre l'exécution d'un tempo ayant été entamée à un pas de temps précédent. Considérons un tempo T_B en cours d'exécution lorsque l'agent macro de distribution *Distributor*0 valide la fin de l'exécution du tempo T_A (c'est-à-dire qu'il a reçu tous les messages relatifs à l'exécution de T_A). Si l'exécution de T_B a pu avoir lieu avant la synchronisation des plates-formes relative à la fin de T_A, c'est que le tempo T_B est indépendant de T_A : aucun agent activé durant T_A est activé durant T_B et les environnements perçus au cours de l'exécution de T_B ne peuvent être influencés au cours de T_A.

Si un agent n'a effectué aucune synchronisation au cours de son pas de temps et qu'il n'a pas non plus un tempo en cours d'exécution, il lance l'exécution d'un nouveau tempo. Les agents s'appuie sur leur table d'activation et la table de dépendances pour obtenir la liste des tempos exécutables et choisissent de préférence le tempo non-vide possédant l'occurrence la plus tôt. Une fois le tempo choisi, il détermine le nombre de pas de simulation nécessaires à simuler l'exécution du tempo ainsi que le temps mis par le réseau à envoyer l'ensemble des requêtes de perception que sont censés émettre les agents de la simulation thématique au cours de l'exécution du tempo.

5.4 Expérimentations

Nos expérimentations se scindent en deux phases :

- une phases didactique au cours de laquelle nous vérifions le fonctionnement de notre prototype et exploitons celui-ci pour affiner notre algorithme d'équilibrage de charges. Nous serons alors amenés à décrire des cas ne nécessitant pas systématiquement un cadre d'exécution distribué. Ces exemples sont avant tout présentés pour mettre en exergue différents phénomènes.
- une phase de montée en charge où nous montrons l'intérêt de notre prototype.

5.4.1 Vérification du fonctionnement général

Nous considérons ici une simulation thématique faisant intervenir deux familles d'agents et exécutée sur deux plates-formes de même puissance appelées *Distributor* et *Distributor*0. Les agents de la simulation thématique se situent tous sur *Distributor* en début de simulation, ceux de la famille *simAgentB* sont deux fois moins nombreux que ceux de *simAgentA* mais nécessitent deux fois plus de temps pour leur exécution. Les agents de la première famille, *simAgentA* possèdent deux tempos T_A et T_C tandis que les agents *simAgentB* en possèdent un. Ces tempos se répartissent sur l'axe temporel comme le montre la figure 5.4. La table de dépendances est la suivante :

```
TC | TC<Distributor0: 4, Distributor: 4> TA<Distributor0: 5, Distributor: 5>
TB | TB<Distributor0: 4, Distributor: 4>
TA | TC<Distributor0: 4, Distributor: 4> TB<Distributor0: 4, Distributor: 4>,
          TA<Distributor0: 5, Distributor: 5>
```

Dans ce premier cas nous avons configuré la simulation thématique de manière à ce que T_B ne dépende ni de T_A ni de T_C. Cela signifie que les agents de *simAgentB* n'utilisent aucune information produite par les agents de *simAgentA*. Ils influencent par contre au moins un environnement perçus par les agents de *simAgentA* au cours de leur activation au tempo T_A. Le réseau a un débit de $100mb/s$ et possède un temps de réaction de $7ms$.

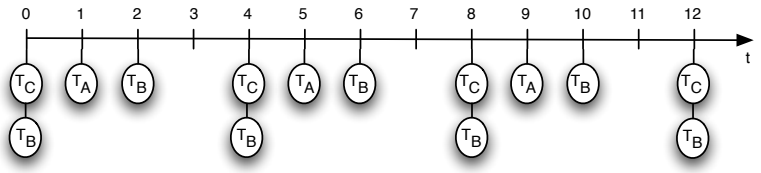


FIG. 5.4 – Axe temporel du cas n°1.

Ne disposant pas encore d'outil de visualisation poussée permettant de représenter toutes les informations sur l'exécution virtuelle de la simulation, nous utilisons une sortie texte synthétique à chaque pas de simulation. Ainsi, au premier pas de l'exécution de la simulation thématique, nous obtenons les informations suivantes :

A $t = 0$,

```
[Distributor] executing tempo: executing TC(0): 1/2.0
[Distributor0] executing tempo: execution finished TC(0) 0.0 + 0.0 -> sendTime: 14.625
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:0): 4, Distributor (simAgentA:100): 0>,
          TA<Distributor0 (simAgentA:0): 1, Distributor (simAgentA:100): 1>
TB (t2) | TB<Distributor0 (simAgentB:0): 0, Distributor (simAgentB:50): 0>
TA (t3) | TC<Distributor0 (simAgentA:0): 4, Distributor (simAgentA:100): 0>,
          TB<Distributor0 (simAgentB:0): 0, Distributor (simAgentB:50): 0>,
          TA<Distributor0 (simAgentA:0): 1, Distributor (simAgentA:100): 1>
```

Les deux plates-formes ont la possibilité d'exécuter T_C et T_B tous deux avec comme instant d'activation $t_{ss} = 0$. *Distributor* possédant 100 agents *simAgentA* et ayant une capacité de

traitement de 50 agents de cette famille par pas de temps commence donc la simulation de T_C. *Distributor*0 n'ayant aucun tempo non-vide à exécuter, il déclare avoir fini d'exécuter T_C. Cette action est diffusée à tous les agents et conduit à la mise à jour de l'exécution partielle du tempo T_C dans la table de dépendances : dans la ligne de dépendance de T_A, le prochain temps d'activation de T_C pour la plate-forme *Distributor*0 sera $t_{ss} = 4$.

A $t = 1$,

```
[Distributor] executing tempo: execution finished TC(0) 1000.0 + 76.5 -> sendTime: 14.625
[Distributor0] executing tempo: execution finished TB(0) 0.0 + 0.0 -> sendTime: 14.625
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:0): 4, Distributor (simAgentA:100): 4>,
          TA<Distributor0 (simAgentA:0): 1, Distributor (simAgentA:100): 1>
TB (t2) | TB<Distributor0 (simAgentB:0): 2, Distributor (simAgentB:50): 0>
TA (t3) | TC<Distributor0 (simAgentA:0): 4, Distributor (simAgentA:100): 4>,
          TB<Distributor0 (simAgentB:0): 2, Distributor (simAgentB:50): 0>,
          TA<Distributor0 (simAgentA:0): 1, Distributor (simAgentA:100): 1>
```

La plate-forme *Distributor* a reçu et traité les informations relatives à la fin d'exécution de $T_C(0)$[18] sur *Distributor*0. Etant précédemment en train d'exécuter $T_C(0)$, elle termine cette exécution et diffuse les informations relatives à la fin de ce tempo : elle a mis $1000ms$ à exécuter $T_C(0)$, la diffusion des messages réseau relatif à son exécution distribuée a demandé $76,5ms$ et la diffusion du message de fin d'exécution de $T_C(0)$ a nécessité $14,625ms$. En parallèle, *Distributor*0 n'étant pas en train d'exécuter un tempo, il cherche, à partir de sa table d'activation et de la table de dépendances, un tempo à exécuter. Le seul tempo éligible est T_B, ce dernier étant vide, il déclare avoir terminé de l'exécuter.

Suite à l'exécution des agents (après l'exécution du pas de temps de notre simulation), la table de vérité montre bien l'exécution complète de $T_C(0)$ ainsi qu'une exécution partielle de $T_B(0)$.

A $t = 2$,

```
[Distributor] waiting for synchro: transfers 50.0 simAgentA to Distributor0 -> sendTime: 46.0625
[Distributor0] waiting for synchro: TC(0) is accomplished! -> sendtime: 14.625
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TA<Distributor0 (simAgentA:50): 1, Distributor (simAgentA:50): 1>
TB (t2) | TB<Distributor0 (simAgentB:0): 2, Distributor (simAgentB:50): 0>
TA (t3) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TB<Distributor0 (simAgentB:0): 2, Distributor (simAgentB:50): 0>,
          TA<Distributor0 (simAgentA:50): 1, Distributor (simAgentA:50): 1>
```

A ce stade de notre simulation, nos deux agents ont détecté la fin d'exécution du tempo T_C pour $t_{ss} = 0$ et peuvent donc procéder à la mise à jour de l'environnement à l'aide des influences produites au cours de l'exécution de $T_C(0)$. Suite à cette mise à jour, nos agents déterminent leur état de satisfaction. L'agent *Distributor* assurant l'exécution de tous les agents de simulation *simAgentA* depuis le lancement de l'exécution, il effectue un transfert de charges. Dans notre

[18] Nous utilisons cette notation pour désigner l'activation de T_C à l'instant $t_{ss} = 0$.

cas, les deux plates-formes disposent de la même puissance de simulation : les informations communiquées à *Distributor* lui permettent de déterminer un transfert optimal (50 agents de *simAgentA*). La table de dépendances qui en résulte traduit cette nouvelle répartition.

A $t = 3$,

```
[Distributor] executing tempo: executing TB(0): 1/2.0
[Distributor0] waiting for synchro: cannot execute any tempo
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TA<Distributor0 (simAgentA:50): 1, Distributor (simAgentA:50): 1>
TB (t2) | TB<Distributor0 (simAgentB:0): 2, Distributor (simAgentB:50): 0>
TA (t3) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TB<Distributor0 (simAgentB:0): 2, Distributor (simAgentB:50): 0>,
          TA<Distributor0 (simAgentA:50): 1, Distributor (simAgentA:50): 1>
```

$T_C(0)$ ayant été exécuté, *Distributor* entame l'exécution du deuxième tempo se trouvant sur le slot temporel $t_{ss} = 0 : T_B(0)$. La répartition des agents de simulation *simAgentB* n'ayant pas été modifiée depuis le début de la simulation, *Distributor* aura besoin de deux pas de simulation pour exécuter $T_B(0)$. *Distributor*0 ayant terminé l'exécution de $T_C(0)$ et $T_B(0)$ ne peut exécuter $T_A(0)$ car ce tempo dépend de $T_B(0)$ qui est encore en cours d'exécution. $T_B(2)$ s'appuie bien évidemment sur $T_B(0)$ et $T_C(4)$ dépend de $T_A(1)$. *Distributor*0 ne peut donc exécuter aucun tempo à ce stade de la simulation.

A $t = 4$,

```
[Distributor] executing tempo: execution finished TB(0) 1000.0 + 45.25 -> sendTime: 14.625
[Distributor0] waiting for synchro: cannot execute any tempo
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TA<Distributor0 (simAgentA:50): 1, Distributor (simAgentA:50): 1>
TB (t2) | TB<Distributor0 (simAgentB:0): 2, Distributor (simAgentB:50): 2>
TA (t3) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TB<Distributor0 (simAgentB:0): 2, Distributor (simAgentB:50): 2>,
          TA<Distributor0 (simAgentA:50): 1, Distributor (simAgentA:50): 1>
```

Distributor termine l'exécution de $T_B(0)$ à ce pas de temps et diffuse donc les informations de fin d'exécution de ce tempo. *Distributor*0 ne peut toujours rien exécuter à ce pas de simulation puisque la table de dépendances n'a pas évolué (*Distributor* n'avait pas terminé l'exécution de $T_B(0)$).

A $t = 5$,

```
[Distributor] waiting for synchro: transfers 25.0 simAgentB to Distributor0 -> sendTime: 26.53125
[Distributor0] waiting for synchro: TB(0) is accomplished! -> sendtime: 14.625
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TA<Distributor0 (simAgentA:50): 1, Distributor (simAgentA:50): 1>
TB (t2) | TB<Distributor0 (simAgentB:25): 2, Distributor (simAgentB:25): 2>
TA (t3) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TB<Distributor0 (simAgentB:25): 2, Distributor (simAgentB:25): 2>,
          TA<Distributor0 (simAgentA:50): 1, Distributor (simAgentA:50): 1>
```

$T_B(0)$ ayant été exécuté sur les deux plates-formes, il y a équilibrage de charges des agents *simAgentB* : *Distributor* transfert 25 agents de la famille *simAgentB* à *Distributor*0. Les agents de la simulation thématique sont maintenant répartis de manière optimale sur les plates-formes qui ne prendront plus qu'un seul pas de temps simulé pour exécuter virtuellement les tempos thématiques. Ainsi, les prochains pas de temps sont :

```
t=6-----------------------
[Distributor] executing tempo: execution finished TA(1) 500.0 + 45.25 -> sendTime: 14.625
[Distributor0] executing tempo: execution finished TA(1) 500.0 + 45.25 -> sendTime: 14.625
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TA<Distributor0 (simAgentA:50): 5, Distributor (simAgentA:50): 5>
TB (t2) | TB<Distributor0 (simAgentB:25): 2, Distributor (simAgentB:25): 2>
TA (t3) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TB<Distributor0 (simAgentB:25): 2, Distributor (simAgentB:25): 2>,
          TA<Distributor0 (simAgentA:50): 5, Distributor (simAgentA:50): 5>

t=7-----------------------
[Distributor] waiting for synchro: TA(1) is accomplished! -> sendtime: 14.625
[Distributor0] waiting for synchro: TA(1) is accomplished! -> sendtime: 14.625
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TA<Distributor0 (simAgentA:50): 5, Distributor (simAgentA:50): 5>
TB (t2) | TB<Distributor0 (simAgentB:25): 2, Distributor (simAgentB:25): 2>
TA (t3) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TB<Distributor0 (simAgentB:25): 2, Distributor (simAgentB:25): 2>,
          TA<Distributor0 (simAgentA:50): 5, Distributor (simAgentA:50): 5>

t=8-----------------------
[Distributor] executing tempo: execution finished TB(2) 500.0 + 29.625 -> sendTime: 14.625
[Distributor0] executing tempo: execution finished TB(2) 500.0 + 29.625 -> sendTime: 14.625
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TA<Distributor0 (simAgentA:50): 5, Distributor (simAgentA:50): 5>
TB (t2) | TB<Distributor0 (simAgentB:25): 4, Distributor (simAgentB:25): 4>
TA (t3) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TB<Distributor0 (simAgentB:25): 4, Distributor (simAgentB:25): 4>,
          TA<Distributor0 (simAgentA:50): 5, Distributor (simAgentA:50): 5>

t=9-----------------------
[Distributor] waiting for synchro: TB(2) is accomplished! -> sendtime: 14.625
[Distributor0] waiting for synchro: TB(2) is accomplished! -> sendtime: 14.625
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TA<Distributor0 (simAgentA:50): 5, Distributor (simAgentA:50): 5>
TB (t2) | TB<Distributor0 (simAgentB:25): 4, Distributor (simAgentB:25): 4>
TA (t3) | TC<Distributor0 (simAgentA:50): 4, Distributor (simAgentA:50): 4>,
          TB<Distributor0 (simAgentB:25): 4, Distributor (simAgentB:25): 4>,
          TA<Distributor0 (simAgentA:50): 5, Distributor (simAgentA:50): 5>

t=10----------------------
[Distributor] executing tempo: execution finished TC(4) 500.0 + 45.25 -> sendTime: 14.625
[Distributor0] executing tempo: execution finished TC(4) 500.0 + 45.25 -> sendTime: 14.625
dependencyTable updated:
TC (t1) | TC<Distributor0 (simAgentA:50): 8, Distributor (simAgentA:50): 8>,
```

```
          TA<Distributor0 (simAgentA:50): 5, Distributor (simAgentA:50): 5>
TB (t2) | TB<Distributor0 (simAgentB:25): 4, Distributor (simAgentB:25): 4>
TA (t3) | TC<Distributor0 (simAgentA:50): 8, Distributor (simAgentA:50): 8>,
          TB<Distributor0 (simAgentB:25): 4, Distributor (simAgentB:25): 4>,
          TA<Distributor0 (simAgentA:50): 5, Distributor (simAgentA:50): 5>
```

Dans cette configuration, les deux plates-formes exécutent maintenant la simulation à la même vitesse en respectant l'ordre des tempos.

5.4.2 Etude de l'équilibrage de charges

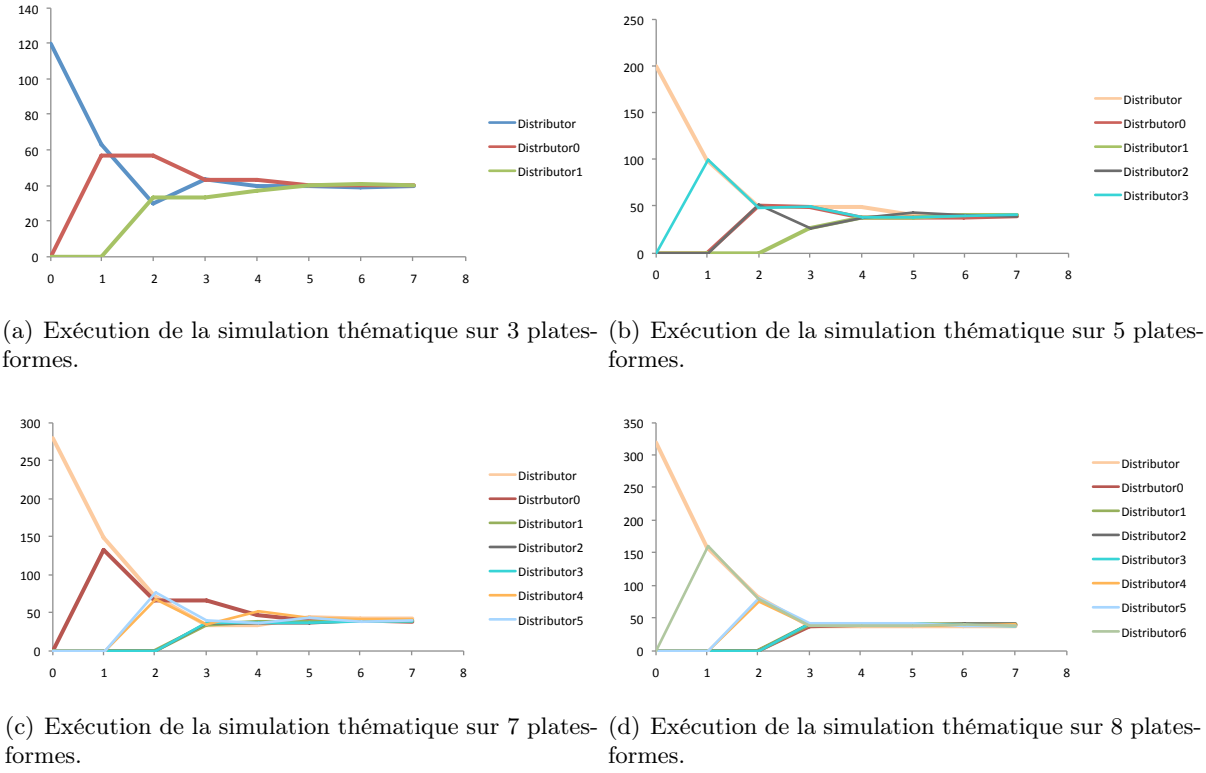
Les divers calculs nécessaires à l'équilibrage de charges s'appuient, dans le système réel, sur des valeurs mesurées et donc à la précision variable. Dans l'exemple précédent, nous avons utilisé directement les valeurs fournies en paramètres de la simulation, ce qui nous a conduit à l'obtention de l'équilibre optimal des charges en un seul transfert. Comme une telle situation n'est pas réalisable, nous introduisons une variation aléatoire de la durée calculée des tempos thématiques et de la durée des transferts réseau de plus ou moins 10%. Ceci nous permet de nous rapprocher du cas réel pour lequel a été défini notre algorithme d'équilibrage de charges : il y a très peu de chances que deux plates-formes partagent exactement la même durée d'exécution d'un même tempo et viennent alors perturber un éventuel équilibrage de charges.

Observation de la stratégie de transfert de charges

La stratégie d'équilibrage de charges que nous proposons s'appuie sur des transferts d'agents de simulation établis entre deux plates-formes : une plate-forme souhaitant alléger sa charge de simulation n'adresse d'agents de simulation qu'à une seule autre plate-forme (voir paragraphe 4.4.3). Un tel choix peut s'avérer peu performant dans certains cas. Nous observons ici la réaction de notre algorithme vis-à-vis de situations de profond déséquilibre de charges entre les plates-formes composant l'infrastructure d'exécution de la simulation thématique.

Dans la série de tests qui suit nous considérons que la simulation thématique se compose d'une famille d'agents $simAgentA$ possédant un unique tempo T_A débutant à $t_{ss} = 0$ et ayant une période de 1 (le fonctionnement de la simulation est alors le même que si la simulation thématique était basée sur le modèle temporel à pas de temps constant). L'infrastructure d'exécution est composée d'un nombre variable de plates-formes de puissances égales. Ces plates-formes peuvent traiter 40 agents $simAgentA$ par pas de simulation. Pour chacune des expériences, le nombre d'agents de $simAgentA$ est égal à $40 \times nbPlatform$: l'équilibre optimal des charges est atteint lorsque chaque plate-forme assure l'exécution de 40 agents.

Nous nous intéressons au temps simulé simulé (ici égal au nombre d'occurrences de T_A) nécessaire à atteindre une situation proche de la répartition optimale en fonction du nombre de plates-formes présentes. Nous soulignons que l'augmentation du nombre d'agents de $simAgent$



(a) Exécution de la simulation thématique sur 3 plates-formes.

(b) Exécution de la simulation thématique sur 5 plates-formes.

(c) Exécution de la simulation thématique sur 7 plates-formes.

(d) Exécution de la simulation thématique sur 8 plates-formes.

FIG. 5.5 – Variation de la durée d'équilibrage de charges en fonction du nombre de plates-formes.

en fonction du nombre de plates-formes ne perturbe pas notre mesure puisque les transferts se font avec la même facilité quelque soit le nombre d'agents de simulation thématique à déplacer (ce qui n'est pas le cas dans le système réel) : nous faisons varier ce nombre pour identifier plus simplement la situation d'équilibre des charges. Nous relevons à la figure 5.5 l'évolution de la répartition de charges dans les cas où l'infrastructure d'exécution est composée de 3, 5, 7 et 8 machines (l'abscisse représente le temps simulé simulé et l'ordonnée le nombre d'agents de simulation thématique).

Nous constatons que, pour un nombre de plates-formes compris entre 3 et 8, une répartition des charges « acceptable » entre les plates-formes est atteinte aux alentours de 6 *sessions* de transfert d'agents de simulation thématique (nos agents ont 6 fois l'occasion d'effectuer un transfert d'agents.). En fait, considérant la configuration initiale, l'étape la plus critique est le démarrage de la simulation : tous les agents de simulation thématique se situant sur *Distributor*, elle est la seule à pouvoir effectuer un transfert de charges. Par la suite, les transferts de charges se font en parallèle les uns des autres.

La figure 5.5(d) affiche la plus grande vitesse d'équilibrage de charges avec un équilibre atteint au bout de 3 sessions de transfert d'agents alors que c'est la situation où nous avons le plus de plates-formes. Ce résultat s'explique très simplement. Ayant supposé toutes les plates-formes de puissance égale, un agent macro de distribution souhaitant transférer des agents de simulation

thématique à une plate-forme n'en possédant pas transfère la moitié de sa charge (à l'erreur de mesure près). Ainsi, après la première exécution de T_A, deux plates-formes possèdent chacune une moitié des agents de *simAgentA*. Après le deuxième pas de temps, ces deux plates-formes transfèrent toutes deux la moitié de leur charges à deux autres plates-formes ne possédant aucun agent de simulation thématique. Pour un nombre de plates-formes pair, l'équilibre est atteint dès que toutes les plates-formes ont reçu une part de simulation. L'algorithme permet de doubler le nombre de plates-formes possédant une charge de simulation à chaque pas de temps. Pour 8 plates-formes, l'équilibre est théoriquement atteint au bout de 3 pas de temps ($2^3 = 8$). Lorsque le nombre de machines est impair, le mécanisme est le même tant que le nombre de plates-formes sans charge est supérieur au nombre de plates-formes ayant reçu des agents de simulation thématiques.

Dans cet exemple, notre algorithme d'équilibrage se révèle moins efficace qu'un algorithme qui scinde la charge initiale de *Distributor* et transfère la charge idéale d'agents de simulation thématique aux autres plates-formes. Ce mécanisme pose toutefois problème lorsque plusieurs agents macro de distribution souhaitent effectuer des transferts de charges en même temps. En effet, un agent macro de distribution dimensionne la charge de manière à ce que sa plate-forme et la plate-forme destinatrice partagent le même temps d'exécution du tempo concerné. Les agents macro de distribution faisant tous ce calcul, une plate-forme recevant des agents de simulation thématique provenant de sources différentes verra son temps d'exécution du tempo considéré dépasser les valeurs attendues par les agents macro de distribution des plates-formes émettrices. Notre algorithme reste donc plus efficace dans cette situation : même s'il peut nécessiter plusieurs transferts d'agents de simulation thématique pour équilibrer de grands écarts de répartition de charges, il permet de minimiser le nombre de transferts d'agents nécessaires.

Impact de l'évaluation du niveau de satisfaction sur l'équilibrage de charges

Pour un agent macro de distribution (et à une échelle plus fine, l'agent de flexibilité de la plate-forme), la condition de satisfaction est discriminante pour un transfert d'agents de simulation. Cette condition est donnée par $d_{ptf}(id_{uls}) - \overline{d}(id_{uls}) \leq tolerance$ avec $d_{ptf}(id_{uls})$ le temps mis par la plate-forme ptf à simuler l'*unité logique de simulation* dont l'identifiant est id_{uls} et $\overline{d}(id_{uls})$ la moyenne des durées d'exécution de id_{uls} sur l'ensemble des plates-formes. Nous nous intéressons ici à l'influence du terme *tolerance* sur l'équilibrage de charges. Pour cela nous considérons une simulation thématique basée sur le modèle à Temporalité ; l'*unité logique de simulation* est le tempo.

Dans l'idéal, la répartition optimale des agents de simulation doit être telle que toutes les plates-formes mettent rigoureusement le même temps à exécuter un tempo considéré. Le terme *tolerance* est alors nul. Sur le système réel, où les temps d'exécution peuvent fluctuer légèrement, il se passe alors un phénomène d'oscillation. Pour mieux appréhender le phénomène, nous reprenons la simulation de l'expérience précédente avec une infrastructure composée de 3 plates-

formes. La courbe d'évolution des charges est celle de la figure 5.5(a). Au lieu de nous intéresser aux transferts d'agents de la simulation thématique avant l'atteinte de l'équilibre, nous observons en figure 5.6 l'évolution de la charge de chaque plate-forme après avoir atteint cet équilibre.

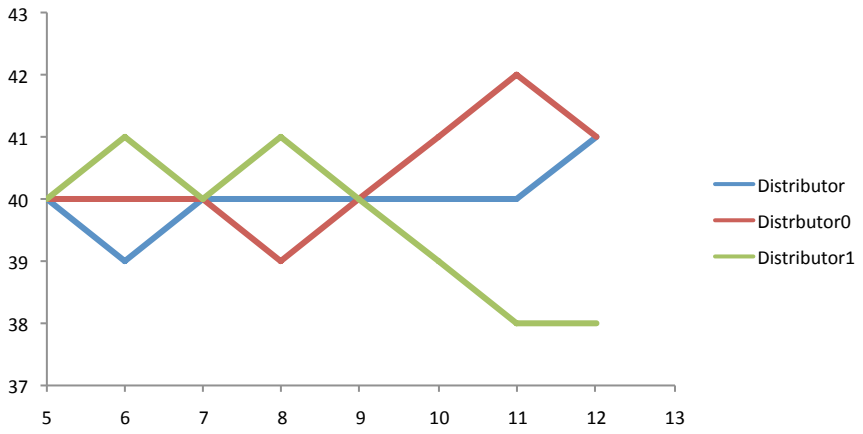


FIG. 5.6 – Evolution de la répartition des agents de $simAgentA$ avec $tolerance = 0$.

Nous constatons que les erreurs de mesure et écarts de performances se traduisent par de petits transferts d'agents qui font évoluer la répartition autour de la répartition optimale. Dans le système réel, le transfert d'agents de simulation a un coût non-négligeable : l'oscillations de charges allonge la durée globale d'exécution de la simulation. Il est donc préférable de se prémunir de ce phénomène même si la contrepartie est de ne plus garantir l'obtention de la répartition optimale des agents de simulation.

Partant du principe que les erreurs s'annulent en moyenne, $\overline{d}(T_K(i))$, la durée moyenne d'exécution de l'occurrence i du tempo T_K, est une valeur fiable. Dans la détermination du niveau de satisfaction, l'erreur provient alors de $d_{ptfa}(T_K(i))$. Pour estimer la valeur de cette erreur, une solution consiste à comparer cette valeur à $d_{ptfa}(T_K)$, la durée moyenne d'exécution du tempo T_K sur ptf_a à l'équilibre des charges. L'erreur de mesure $err_{T_K(i)}(ptf_a)$ est donnée par $err_{ptfa}(T_K(i)) \approx d_{ptfa}(T_K(i)) - d_{ptfa}(T_K)$. La condition de satisfaction de l'agent de flexibilité de la plate-forme ptf_a est donnée par $d_{ptfa}(T_K(i)) - \overline{d}(T_K(i)) \leq |err_{ptfa}(T_K(i))|$, c'est à dire

$$d_{ptfa}(T_K(i)) - \overline{d}(T_K(i)) \leq |d_{ptfa}(T_K(i)) - d_{ptfa}(T_K)|$$

Cette valeur n'est utilisable que lorsque la répartition des agents de simulation est proche de l'équilibre. Qui plus est, la valeur de la durée moyenne d'exécution de T_K sur ptf_a, $d_{T_K}(ptf_a)$, ne doit pas prendre en compte les durées d'exécution de T_K alors que la répartition des agents de simulation associés à T_K n'est pas proche de la répartition optimale. Les conditions de validité de l'expression précédente étant assez contraignantes, il est préférable d'estimer la valeur de l'erreur err_{T_K} et de l'utiliser dans le critère de satisfaction.

Nous reprenons la simulation précédente (exécution de 120 agents de *simAgentA* sur 3 plates-formes aux performances égales) et nous prenons pour condition de satisfaction $d_{ptfa}(T_K(i)) - \overline{d}(T_K(i)) \leq \frac{3}{40}$. Le terme $\frac{3}{40}$ correspond à notre estimation de l'erreur de mesure. Nous nous intéressons à la répartition des agents de simulation thématique lorsque cette répartition est déjà proche de la répartition optimale.

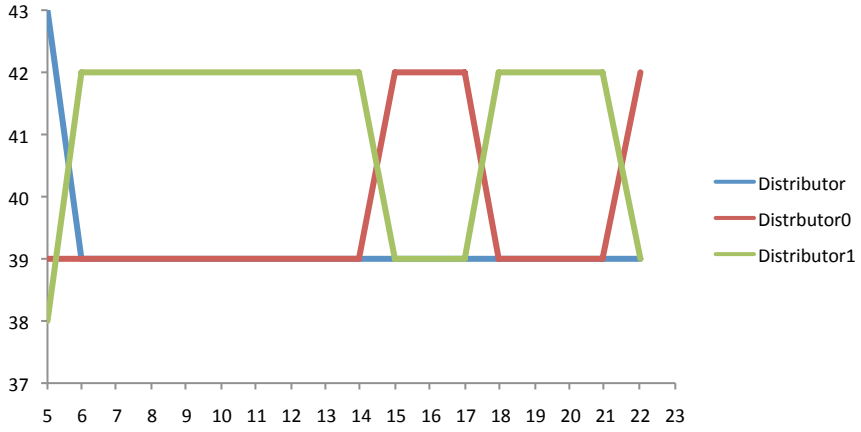


FIG. 5.7 – Evolution de la répartition des agents de *simAgentA* avec un critère d'équilibrage de charges prenant en compte les perturbations de mesures.

Nous observons à la figure 5.7 que les oscillations sont beaucoup moins nombreuses mais toujours présentes. Ces oscillations se font par transferts minimaux de 3 agents de simulation thématique (le terme $\frac{3}{40}$ représente le temps mis par une plate-forme à exécuter 3 agents de *simAgentA*). Les oscillations sont présentes quelle que soit l'estimation de l'erreur que nous utilisons pour la condition de satisfaction car la répartition des agents de simulation thématique n'est pas optimale.

Le fait d'alléger la condition de satisfaction permet d'arrêter l'équilibrage de charges alors que la situation optimale n'est pas atteinte : dans une situation « quasi-optimale », une faible variation de la durée d'exécution d'un tempo peut relancer l'équilibrage de charges. Nous illustrons ce phénomène à la figure 5.8. Considérant le nombre d'agents de simulation thématique sur *Distributor*1, une petite augmentation de la durée de simulation (qui se traduit sur le graphe par une augmentation virtuelle du nombre d'agents hébergés sur *Distributor*1) suffit à ce que la charge de *Distributor*1 dépasse la tolérance introduite dans la condition de satisfaction et déclenche donc un transfert de 3 agents de *simAgentA* à une autre plate-forme. Une telle situation ne poserait pas de problème si *Distributor*1 avait atteint la répartition optimale : une légère variation de sa durée de simulation la fait rester dans les limites tolérées par la condition de satisfaction.

*Distributor*1 dépassant « accidentellement » le seuil d'équilibrage de charges, il transfert 3 agents de *simAgentA* à une autre plate-forme qui risque de se trouver dans une situation si-

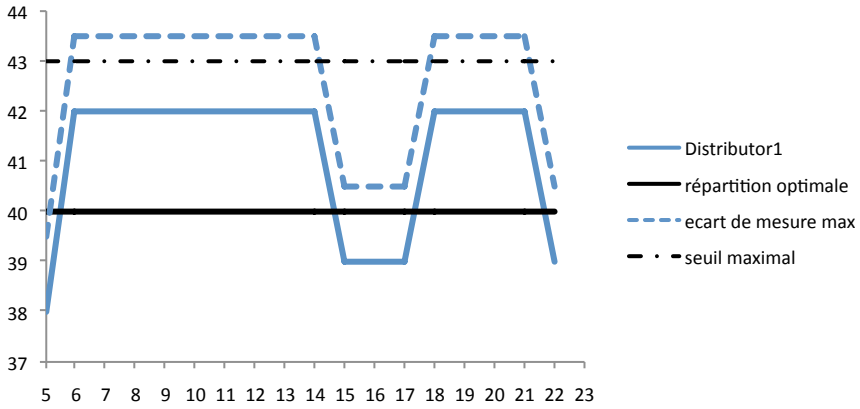


FIG. 5.8 – Persistance des oscillations de charge sur *distributor*1 due à une répartition non-optimale de la charge.

milaire. Pour supprimer ce phénomène, il faudrait que la répartition des agents de simulation thématique soit suffisamment proche de la répartition optimale de manière à ce que les faibles variations de durée d'exécution de tempos ne dépassent pas le seuil d'équilibrage de charges.

La solution que nous proposons consiste à modifier la quantité d'agents de simulation à transférer lorsque la différence de temps entre les deux plates-formes impliquées dans le transfert d'agents est compris entre *tolerance* et $tolerance \times 2$. Au lieu de procéder à l'évaluation usuelle de la charge d'agent à déplacer, l'algorithme impose alors une charge plus petite de manière à faire converger la répartition vers la répartition optimale.

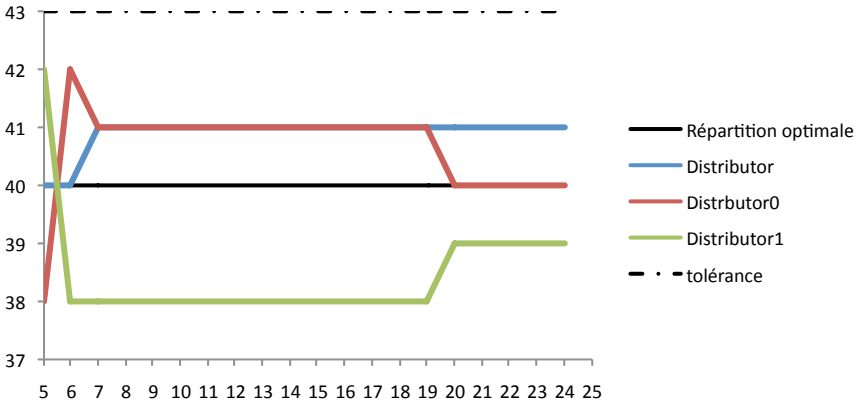


FIG. 5.9 – Evolution de la charge des plates-formes avec les règles de prévention d'oscillations.

La figure 5.9 montre ce fonctionnement. Ayant fixé une tolérance sur la condition de satisfaction équivalente à un écart de 3 agents de *simAgentA*, un transfert de 3 agents de simulation thématique sera ramené à un transfert d'un seul agent : c'est le cas pour les transferts ayant lieu à $t_{ss} = 7$ et $t_{ss} = 20$. Ce mécanisme permet bien aux agents macro de distribution de faire

évoluer lentement la répartition des agents de simulation thématique vers la répartition optimale.

5.4.3 Exemple d'utilisation

Nous appuyant sur un prototype sous forme de simulation, nous pouvons aisément faire varier la configuration de la simulation thématique à exécuter ou l'infrastructure d'exécution. Nous décrivons ici un cas d'utilisation de ce prototype. Nous disposons d'une simulation thématique que nous souhaitons exécuter : la représentation de cette simulation dans notre prototype permet d'évaluer les performances de simulation par rapport au nombre de plates-formes composant l'infrastructure d'exécution.

Nous considérons l'exécution d'une simulation thématique proche de celle décrite au paragraphe 3.5.1. Les agents éleveurs et agriculteurs possèdent chacun plusieurs tempos comme le montre l'axe temporel de la figure 5.10.

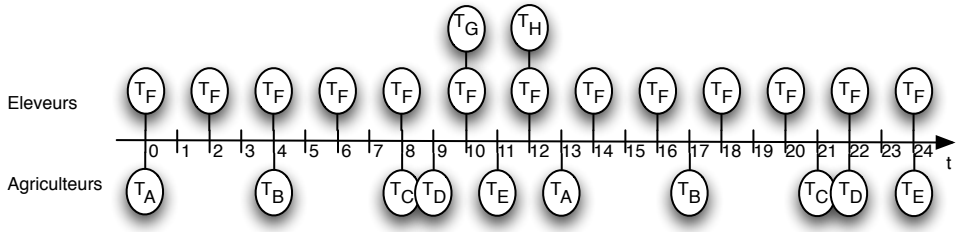


FIG. 5.10 – Axe temporel de la simulation thématique.

Les agents agriculteurs disposent de leur propre dynamique liée à leur activité agricole et représentée par les tempos T_A, T_B et T_C. Les agents éleveurs doivent gérer régulièrement leur élevage : cette dynamique est représentée par le tempo T_F possédant une période de 2 (tous les autres tempos ont une période de 13). A la période de plantation, les agriculteurs négocient avec les éleveurs du lisiers pour fertiliser leurs terres : cette interaction a lieu au cours des tempos T_D, T_G, T_E et T_H, considérant ainsi que les négociations aboutissent au bout de quelques échanges de messages.

Le modèle simulé met en jeu 60 éleveurs et 200 agriculteurs possédant le même coût de simulation et la même taille de structures de données. Le réseau a un temps de réaction de $7ms$ et un débit de $66Mb/s$ (ce qui constitue un débit « réaliste » pour un réseau $100Mb/s$). Toutes les plates-formes ont la même capacité de traitement et prennent $12,5ms$ pour exécuter un agent de la simulation thématique. Lorsque l'infrastructure d'exécution est constituée de plus d'une plate-forme, les agents éleveurs sont situés sur une plate-forme et les agriculteurs sur une autre. Nous considérons l'exécution de la simulation thématique de son lancement à l'exécution du slot temporel situé au temps simulé simulé $t_{ss} = 130$ (ce qui correspond à 10 périodes globales de simulation).

Nous simulons l'exécution de ce modèle simulé en faisant varier le nombre de plates-formes

composant l'infrastructure d'exécution. Nous considérons une infrastructure composée au mieux de 8 plates-formes ; il s'agit d'une configuration que nous considérons comme réalisable dans les projets de simulation actuels. Pour chacune des configurations, nous mesurons à chaque fois deux valeurs : le temps moyen de simulation sur chaque plate-forme,$simTime$, et le temps moyen mis par chaque plate-forme à atteindre $t_{ss} = 130$, $totalTime$ (cette valeur comprend la première valeur mesurée ainsi que le temps réel simulé nécessaire aux échanges réseau). Nous obtenons alors les courbes de la figure 5.11. L'abscisse représente le nombre de plates-formes et l'ordonnée le temps réel simulé en ms.

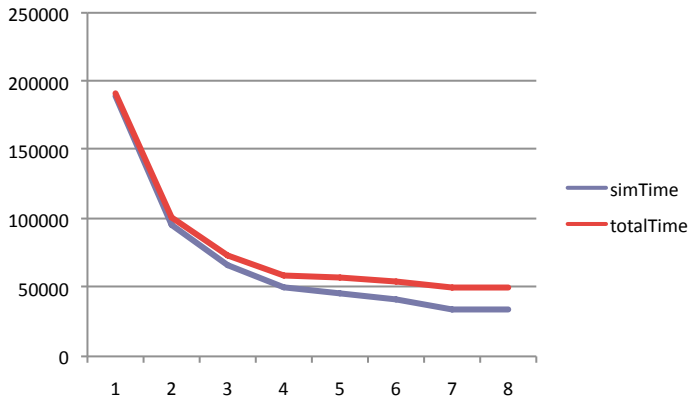


FIG. 5.11 – Evolution des temps moyens de simulation avec et sans comptabilisation des temps réseaux en fonction du nombre de plates-formes.

Le temps de communication réseau, représenté par l'écart entre $simTime$ et $totalTime$, augmente avec le nombre de plates-formes. Cette augmentation s'explique par le modèle de communication que nous avons choisi. Dans les réseaux actuels, un protocole de communication *multicast* adresse un message par destinataire : plus le nombre de plates-formes est élevé, plus le réseau est sollicité. Ainsi, pour un nombre élevé de plate-forme, nous pourrions arriver au cas extrême où les communications réseaux nécessitent plus de temps que l'exécution même de la simulation : une augmentation du nombre de plates-formes ne se traduit pas systématiquement par une diminution du temps global de simulation.

Nous constatons que plus le nombre de plates-formes augmente plus le gain de performance est faible et ce, indépendamment du temps induit par le réseau. Pour mieux comprendre ce phénomène, nous nous intéressons au temps global de simulation, c'est-à-dire à la somme des temps de simulation $simTime$ de chaque plate-forme. La figure 5.12 représente ce temps en fonction du nombre de plates-formes.

Alors que la charge de simulation est fixe, le temps global de simulation augmente. Cette augmentation est induite par la performance variable de chaque plate-forme au cours de l'exécution de la simulation. En effet, dans la mesure où les plates-formes doivent régulièrement être synchronisées, les plates-formes ayant montré les meilleures performances sur l'exécution du tempo

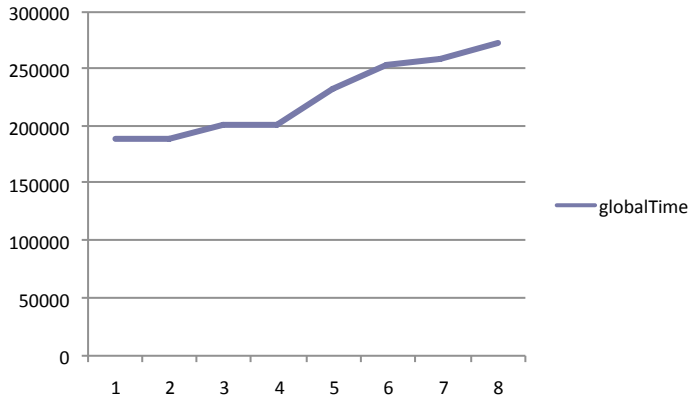


FIG. 5.12 – Evolution du temps global de simulation en fonction du nombre de plates-formes.

courant doivent attendre celles ayant été les plus lentes : la simulation s'exécute globalement à la vitesse des machines les plus lentes, même si l'algorithme d'équilibrage de charges tend à réduire les écarts. Plus le nombre de plates-formes augmente, plus il y a de chances que les écarts de temps réel simulé soient grands entre la plate-forme la plus rapide et la plus lente à l'exécution d'un tempo. Notons toutefois que l'algorithme d'ordonnancement parallèle permet de réduire le nombre de points de synchronisation nécessaires et minimise donc légèrement ce phénomène.

En permettant d'évaluer le temps nécessaire à exécuter une simulation sur une infrastructure distribuée, notre prototype peut être utilisé avant le déploiement de la simulation. Il offre la possibilité d'évaluer le gain de performance en fonction du nombre et du type de machines connectées. Il faut toutefois noter qu'il ne se focalise que sur les temps d'exécution et de communication liés aux agents de la simulation. Pour avoir une vision plus précise des performances réelles, il est encore nécessaire de considérer l'impact de l'exécution distribuée des environnements ainsi que celui des services des plates-formes (notamment l'observation de la simulation).

5.5 Discussions

5.5.1 Limites du prototype actuel

Prise en compte du réseau dans l'algorithme d'équilibrage de charges

Nos travaux s'appuient sur le réseau pour supporter l'exécution parallèle et distribuée de la SOA. Si nous nous sommes avant tout focalisés sur les aspects logiciels de la distribution, le réseau n'en reste pas moins un sujet d'étude important. Notre prototype fournit une première évaluation des durées de communication induites par le réseau. Ces valeurs ne restent toutefois qu'indicatives car nous pouvons difficilement prendre en compte la simultanéité des communications réseau.

En effet, sur un vrai réseau un engorgement peut avoir lieu si de nombreuses communications

ont lieu en même temps. Or, la représentation du temps réel simulé que nous avons adoptée nous permet difficilement d'identifier les instants précis de ce temps où les plates-formes échangent des messages sur le réseau : ne pouvant identifier les accès concurrents au réseau, nous ne gérons pas les éventuelles surcharges sur ce dernier. Il est certes possible d'induire un surcoût dans les communications pour quantifier globalement l'impact du réseau mais la modélisation précise du comportement de ce médium nécessite une approche de modélisation différente.

Prise en compte de la mémoire disponible sur chaque plate-forme

Notre modèle de simulation associe à chaque « plate-forme », c'est-à-dire au corps d'un agent macro de distribution dans l'environnement de simulation thématique, une capacité de traitement des agents de simulation thématique par pas de temps. Cette capacité représente la puissance de traitement de la plate-forme, et donc par extension le processeur de l'ordinateur. La mémoire n'est quant à elle pas représentée. Ne nous plaçant pas dans les cas où celle-ci peut entraîner un blocage de l'ordinateur et donc une déconnexion accidentelle de la plate-forme, elle peut se modéliser assez simplement dans notre prototype. Chaque agent de la simulation thématique ayant un coup mémoire et les plates-formes une capacité mémoire fixée, si la somme des coûts mémoires des agents de la simulation thématique hébergés sur la plate-forme dépasse le coût mémoire nous introduisons un coefficient multiplicateur sur le calcul de la durée d'exécution virtuelle des tempos thématiques sur la plate-forme. L'exécution d'un tempo affiche alors de mauvaises performances et l'agent macro de distribution effectuera des transferts de charge pour retrouver une durée de simulation acceptable.

Modélisation des agents de la simulation thématique

Notre prototype permet d'étudier l'algorithme d'équilibrage de charges sous l'angle des masses transférées et non des agents de la simulation thématique transférée. Nous avons fait ce choix pour plusieurs raisons. La première d'entre elle est la difficulté à représenter et configurer chaque agent de la simulation thématique. Vouloir les représenter pour modéliser une simulation large-échelle revient à créer une simulation où la large-échelle existe en terme de nombre d'objets dans un environnement. La deuxième raison tient dans le fait que le mécanisme d'identification des agents de simulation à transférer, dans notre algorithme d'équilibrage de charges, doit encore être travaillé avant d'être implémenté (voir paragraphe4.4.5). Enfin, un des intérêts de la représentation physique des agents de la simulation thématique dans notre prototype est d'étudier plus finement l'impact de la répartition des agents sur le réseau. Ne disposant pas encore d'un modèle précis de ce dernier, l'intérêt de la représentation des agents de la simulation thématique est réduit.

5.5.2 Discussions

Vers une meilleure analyse temporelle des simulations

Notre prototype simule l'exécution distribuée et parallèle d'une SOA sur un réseau de plates-formes de simulation. L'algorithme permettant l'exécution parallèle de la simulation se base sur le modèle à Temporalité. Même si nous avons simplifié au possible le modèle de simulation thématique, nous avons gardé la description temporelle qu'impose le modèle à Temporalité. Toutefois, au lieu de construire l'axe temporel de la simulation thématique, nous avons construit uniquement la table de dépendances. Chaque agent/plate-forme dispose de sa table d'activation, simple liste associant chaque tempo à sa prochaine date d'activation, et exploite la table de dépendances pour exécuter les tempos de la simulation thématique. Notre prototype nous montre donc qu'il est possible d'exécuter une simulation sans avoir à considérer un axe temporel.

Au travers de l'emploi du modèle à Temporalité, nous fournissons un cadre où l'identification des liens de causalité est aisée. En effet, le modèle à Temporalité permet d'aborder le temps dans la simulation sous la forme d'un axe temporel des plus classiques et ce n'est que par la suite que la table de dépendances est déterminée : notre approche aide le thématicien a définir sa table de dépendances puisqu'il n'a pas à considérer directement les liens de causalité entre les tempos qu'il définit.

Ainsi, avant même l'exécution de la simulation, la table de dépendances peut fournir des informations concernant les dynamiques du système au thématicien. Il pourrait identifier une dynamique qu'il ne soupçonnait pas ou repérer les interactions entre dynamique. En simulation, l'analyse de la table des dépendances permet d'identifier très simplement un enchaînement d'événements ayant conduit à un phénomène.

Nous pensons que de nombreuses réflexions sont à mener dans cet axe : le temps dans les simulations ne doit pas être perçu uniquement comme une étiquette associée à chaque événement mais peut devenir une véritable connaissance du modèle qui aide le thématicien à mieux appréhender son système. Mieux l'utilisateur pourra insérer et représenter ses connaissances du système réel dans le modèle, plus la simulation orientée agent sera à même de lui fournir les éléments pour analyser au mieux la simulation.

Optimiser le parallélisme pour améliorer les descriptions comportementales

La construction de la table de dépendances s'appuie sur le modèle à Temporalité mais aussi sur la connaissance des environnements qui sont potentiellement perçus ou influencés au cours de l'exécution des agents associés au tempo. Une telle information n'est pas habituellement intégrée au modèle. Cependant il ne faut pas voir ici une contrainte simplement liée à l'utilisation de notre algorithme d'exécution parallèle de la simulation. Cet enrichissement de la description des temporalités est plus à considérer comme une nouvelle approche de modélisation.

Les modélisateurs tendent à identifier l'instant auquel se déroule une action ; le comportement d'un agent évolue alors au cours du temps. Cependant, une telle approche dénote le fait que le comportement est contraint par le temps. Une des causes de ce constat est probablement liée aux modèles temporels usuels et notamment au modèle à pas de temps constant dont la description des artefacts temporels est extrêmement pauvre. Le modèle à Temporalité permet d'établir une liaison entre un comportement et son inscription dans le temps : une temporalité possède son propre identifiant auquel l'agent peut associer un comportement spécifique. L'aspect temporel est alors complètement intégré à la notion de comportement et non l'inverse.

A un comportement spécifique d'un agent donné, il est réaliste de pouvoir identifier les environnements qui y seront potentiellement perçus et ceux potentiellement influencés. Ces ensembles peuvent alors être rattachés à la temporalité associée à ce comportement spécifique de l'agent considéré.

Ainsi l'approche que nous prônons pour modéliser un système consiste à identifier chaque comportement que chaque agent peut avoir et y associer une temporalité : plus le comportement d'un agent pourra être segmenté en comportements spécifiques, plus cet agent possédera de temporalités. Notre algorithme d'ordonnancement parallèle de la simulation s'appuie sur la considération des temporalités et des environnements potentiellement perçus et influencés qui y sont rattachés.

L'emploi de plusieurs temporalités pour représenter les différents comportements d'un agent confère plusieurs avantages au modèle de simulation et notamment faciliter l'analyse de la simulation en percevant mieux quel comportement de l'agent a engendré le phénomène observé, faciliter la réutilisation des agents, supporter le développement incrémental des agents. Faisant l'hypothèse que plus le comportement d'un agent sera segmenté en temporalités, moins il y aura d'environnements potentiellement perçus et influencés durant l'exécution d'une temporalité, l'emploi massif de temporalités pour la description du comportement d'un agent favorise l'ordonnancement parallèle tel que nous le proposons. Ainsi lorsqu'un modélisateur cherche à optimiser le parallélisme de sa simulation à l'aide de notre prototype, il travaille à la façon de mieux différencier les comportements de ses agents : optimiser un modèle de simulation pour son exécution parallèle revient alors à améliorer les descriptions comportementales de ses agents.

5.6 Synthèse

La réalisation technique de l'infrastructure d'exécution constitue une barrière pour le prototypage de nos propositions. Elle nécessite d'une part un important travail d'ingénierie (sur la base duquel dépendent notamment les performances globales) et limite d'autre part les possibilités d'évaluation aux matériels dont nous disposons.

Considérant alors l'approche par simulation, nous y avons trouvé plusieurs avantages. Le

premier d'entre eux est de pouvoir tester nos algorithmes sans que ceux-ci soient perturbés par des problématiques que nous n'avons pas encore traitées. Par exemple, les environnements ne sont pas encore distribués et constituent des goulots d'étranglement pour l'exécution de la simulation distribuée. Un deuxième avantage est de pouvoir faire varier facilement les conditions de test : nous pouvons aisément configurer l'infrastructure d'exécution ainsi que le modèle de simulation thématique devant être exécuté sur cette infrastructure.

Notre prototype mettant à disposition un modèle d'exécution parallèle et distribué d'une simulation, nous avons proposé un cadre d'utilisation de celui-ci proche des filtres de Kalman utilisés en automatique. Notre prototype peut être connecté au système réel et permet d'estimer des valeurs utiles à l'optimisation de l'exécution réelle de la simulation thématique.

Notre prototype est implémenté sous la forme d'une simulation basée sur la plate-forme GEAMAS-NG. Nous avons défini un unique type d'agent au niveau de GEAMAS-NG, l'agent macro de distribution, intégrant les agents micro de distribution qui interagissent entre eux par échanges de messages synchrones. Cet agent est en interaction avec trois environnements, un dédié à la communication (échange de messages par diffusion), le second à la prise en compte du réseau dans la durée d'exécution de la simulation thématique et le dernier modélisant tous les éléments se rapportant à la simulation thématique. L'environnement de simulation thématique contient :

- les corps des agents qui sont apparentés aux moteurs d'exécution des plates-formes. Le corps d'un agent définit une puissance de traitement de la plate-forme : cette valeur représente le nombre d'agents de simulation thématique qui peuvent être traités par pas de temps.
- les familles d'agents de la simulation thématique. Ne pouvant modéliser un à un tous les agents de la simulation thématiques, nous avons considéré des familles d'agents. Chaque famille est répartie sur les différentes plates-formes et possède ses propres temporalités.
- l'environnement temporel. Nos propositions s'appuient fortement sur l'emploi du modèle à Temporalité, nous l'avons donc modélisé en représentant les différents concepts qu'il définit. Les autres environnements n'ayant pas de sémantique particulière dans nos propositions, nous les avons considérés uniquement sous forme d'identifiant permettant d'établir les liens de dépendances entre tempos.

Notre prototype effectue des sorties textes présentant toutes les informations nécessaires à l'évaluation de nos algorithmes. Ces informations nous ont notamment permis d'analyser la performance de l'équilibrage de charges selon le nombre de plates-formes dans des conditions de fort déséquilibre dans la répartition des agents de simulation thématique. Nous avons aussi étudié le phénomène d'oscillation de la charge des plates-formes autour de la répartition optimale : les agents macro de distribution s'échangent continuellement un faible nombre d'agents de simulation thématique, ce qui peut ralentir l'exécution de la simulation. Pour éviter ce phénomène, nous avons étudié l'intérêt de la tolérance dans la condition de satisfaction des agents macro de distribution. Nous introduisons également une règle de dimensionnement des transferts d'agents

pour faire converger la répartition vers la répartition optimale et supprimer les oscillations.

Considérant un cas réaliste, nous utilisons notre prototype pour étudier les performances d'exécution de la simulation en fonction du nombre de plates-formes connectées. Nous soulignons qu'une augmentation du nombre de plates-formes ne se traduit pas nécessairement par une diminution de la durée de simulation dans la mesure où le trafic réseau se trouve lui aussi augmenté.

Ayant du faire un choix en terme de représentation du temps réel simulé dans notre prototype, nous identifions les limites de ce dernier. Notre modèle reste limité en terme de modélisation du réseau et ne nous permet donc pas d'aborder les problèmes d'engorgement du réseau comme de l'identification des agents de simulation thématique à transférer.

Au regard de notre prototype, nous abordons plusieurs pistes de réflexion ayant trait à la représentation du temps dans les simulations orientées agent. Notre prototype actuel permet de simuler l'exécution parallèle et distribuée d'une SOA sans pour autant intégrer la moindre notion d'axe temporel. Cette représentation n'étant pas nécessaire pour permettre l'exécution de la simulation, nous remettons en question son utilisation pour le thématicien. La table de dépendances nous semble à l'inverse constituer un outil à présenter à ce dernier : elle permet d'identifier très simplement l'enchaînement de causes et de conséquences ayant conduit à un phénomène. Nous estimons ainsi qu'un travail pourrait être entamé sur la considération du temps comme une connaissance du modèle.

Nous abordons une piste de réflexion sur ce sujet en présentant l'approche de modélisation sur laquelle est basée notre algorithme d'ordonnancement parallèle de la simulation. Le modèle à Temporalité permet d'associer un temps à un comportement plutôt que l'inverse. Cela signifie que l'information temporelle est rattachée à un comportement. A une temporalité d'un agent correspond alors un comportement spécifique de l'agent. Dans ce cadre où l'agent associe une temporalité à chacun de ces comportements spécifiques, il est réaliste de déterminer les environnements potentiellement perçus et potentiellement influencés à chaque temporalité.

Conclusion

Sommaire

1 Synthèse des propositions 145

2 Perspectives 147

1 Synthèse des propositions

La simulation orientée agent est employée par des thématiciens pour apporter des réponses à des questions que ceux-ci se posent sur une partie du monde réel. La réalisation de simulations de systèmes complexes naturels ou sociaux fait interagir entre eux de nombreux agents aux comportements complexes. Les ressources informatiques d'un seul ordinateur ne permettent pas toujours d'assurer la représentation et l'exécution de telles simulations. Nos travaux se focalisent sur la problématique de l'exécution de SOA large échelle. Nous proposons d'y apporter une réponse par la distribution et l'exécution parallèle de la SOA sur une infrastructure d'exécution composée d'ordinateurs connectés en réseau.

Notre état de l'art a fait ressortir l'intérêt porté à la distribution dans les SOA : de plus en plus de plates-formes intègrent des fonctionnalités distribuées. Ces solutions restent bien souvent *ad hoc* et n'abordent pas la distribution dans sa vision la plus globale. Nous avons par la suite considéré différentes infrastructures d'exécution distribuée sous l'angle de l'exécution distribuée de SOA pour conclure qu'une infrastructure d'exécution dédiée à la SOA distribuée restait à définir.

Ainsi, nous avons proposé, au chapitre 2, une architecture pour la réalisation d'une telle infrastructure d'exécution. Celle-ci est composée de quatre couches : Dorsale de communication, SMA de distribution, Plate-forme virtuelle et SOA distribuée. Le recours à un SMA de distribution est une spécificité de notre approche. Son emploi permet de définir des stratégies complexes d'exécution distribuée de la SOA tout en tenant compte à la fois de la dynamique de simulation et de la dynamique de l'infrastructure d'exécution. A l'échelle de l'infrastructure d'exécution, chaque nœud de simulation possède un Agent Macro de Distribution. A l'échelle de la plate-forme, cet agent est composé d'Agents Micro de Distribution. Nos travaux visant spécifiquement

l'exécution des agents de la simulation, nous avons défini trois agents micro de distribution : l'Agent de Cohérence, l'Agent de Flexibilité et l'Agent Réseau. Leur nombre et les interactions qu'ils développent peuvent être étendus pour piloter d'autres aspects de la SOA distribuée.

Pour exploiter les ressources informatiques mises à disposition par l'infrastructure d'exécution, nous avons présenté au chapitre 3 un algorithme d'ordonnancement parallèle de type conservatif. L'originalité de cet algorithme repose sur son exploitation du modèle de simulation. En effet, notre approche se base sur les descriptions temporelles associées aux comportements des agents pour relever les liens de causalité entre ces comportements. Cette analyse produit une table de dépendances qui est exploitée en simulation par l'ordonnanceur pour permettre l'exécution parallèle ; le comportement d'un agent peut être déclenché dès que les comportements dont il dépend auront été exécutés. Notre ordonnanceur présente une caractéristique peu courante pour un ordonnanceur conservatif : il offre la possibilité de rompre la croissance monotone du temps dans l'exécution de la simulation sur un nœud de simulation, propriété qui augmente le potentiel d'exécution parallèle.

Conjointement à notre algorithme d'ordonnancement parallèle de la simulation, nous avons proposé au chapitre 4 un algorithme d'équilibrage dynamique de charges. Cet algorithme est mis en œuvre par les agents de flexibilité A_F de chaque nœud de simulation et gère la répartition des agents de la SOA sur l'infrastructure d'exécution. S'appuyant sur un mode de communication en diffusion, nous avons défini un algorithme de type SDM (*Simple Diffusion Method*) s'appuyant sur trois phases : Diffusion d'informations, Détection de surcharge et Transfert d'agents. Chaque agent A_F alterne les deux premières phases et, s'il détecte une surcharge sur sa plate-forme, procède à un transfert d'agent. Ce transfert est établi entre la plate-forme initiatrice et la plate-forme qu'elle estime la plus apte à accepter une charge de simulation supplémentaire. Si nous avons traité la question de la dimension du transfert, nous soulignons la problématique de l'identification des agents de simulation à transférer. Considérant le modèle temporel employé au chapitre 3, nous avons étudié le couplage théorique de nos algorithmes d'ordonnancement parallèle et d'équilibrage de charges.

L'important travail d'ingénierie nécessaire à la mise en œuvre réelle de notre architecture étant difficilement compatible avec le cadre de ce travail de recherche, nous avons considéré la validation de nos algorithmes par simulation. De l'architecture distribuée que nous avons proposée, notre prototype reprend principalement le SMA de distribution réparti. Le système complexe applicatif, désigné par le terme *simulation thématique*, est ramené à un environnement avec lequel les agents macro de distribution interagissent pour simuler l'exécution parallèle et distribuée de la simulation thématique. Nous avons employé notre prototype pour analyser nos algorithmes et pour étudier l'impact de la composition de l'infrastructure d'exécution sur la durée de simulation. Nous avons enfin identifié les limites de ce prototype, notamment au niveau de la prise en compte du réseau.

Ce travail de recherche nous a donné l'opportunité de nous intéresser à de nombreuses pro-

blématiques touchant à des domaines très variés. Ne pouvant explorer toutes ces problématiques, nous nous sommes centrés sur celles se rapprochant le plus du monde agent et visant au support et à l'optimisation de l'exécution parallèle et distribuée de simulations orientées agent. Dans cette démarche, nos efforts ont permis d'identifier de nouvelles problématiques mais aussi de nouvelles idées qui dépassent le cadre de la distribution de SOA : la recherche dans la distribution de SOA est source d'innovation pour le cadre général des systèmes multi-agents.

2 Perspectives

La première perspective à considérer est bien évidemment l'implémentation de notre architecture à l'aide de la plate-forme GEAMAS-NG, non plus dans un cadre simulé mais dans le cadre réel d'une infrastructure distribuée. Outre l'intérêt certain que revêt cette infrastructure dans nos projets de simulation, elle constitue le fil rouge autour duquel les problématiques de recherche liées à la distribution pourront être abordées.

Le réseau constitue une de ces problématiques. Nous avons considéré celui-ci comme un simple médium de communication pouvant imposer des contraintes (temps de réaction, débit). Nous sommes conscients de cette limite mais estimons qu'elle était nécessaire dans cette première approche de distribution de la SOA. En effet, cette minimisation de l'importance du réseau nous a permis de nous focaliser avant tout sur les concepts logiciels permettant de supporter la simulation distribuée. Etudier les aspects réseau de notre architecture devient encore plus pertinent maintenant que nous disposons de ces concepts logiciels puisque nous pouvons axer cette étude sur l'usage spécifique qu'en fait notre architecture.

Nous estimons que conjointement à ce travail sur ces couches du réseau, un travail est à mener sur les interactions entre l'agent réseau de chaque plate-forme et le réseau. L'objectif serait alors de montrer que l'agent réseau peut piloter la configuration du réseau au travers de ses couches basses (réseau et transport) et non plus seulement au niveau de la couche applicative que nous avons considéré dans ce travail. En s'appuyant sur les critères de qualité de service propres à ce niveau de communication, il peut affiner ses stratégies de communication.

Une autre étape fondamentale vers la mise en œuvre de notre architecture réside dans la distribution de l'environnement. La distribution de l'environnement nécessite en effet autant d'attention que la distribution des agents de la simulation. Le choix d'une solution multi-environnement constitue une première approche puisqu'elle permet de répartir les environnements sur les différentes plates-formes. Mais la localisation d'un environnement sur une seule plate-forme peut former un goulot d'étranglement puisque les environnements sont les sièges de toutes les interactions entre agents. Certaines approches de génie logiciel, par exemple dans le domaine des composants, peuvent apporter des solutions ad hoc. Nous menons notamment une réflexion sur l'utilisation de composants ubiquitaires [Hoareau, 2007]. Plus proche du monde agent, nous avons proposé la notion d'*agents proxy* comme solution pour le couplage de modèles simulés dans un

contexte de simulation distribuée [Sébastien *et al.*, 2006]. Le concept d'agents proxy pourrait être généralisé pour définir la notion d'environnements proxy. Ce qui constiturait un travail de recherche à part entière. En effet, les environnements constituent encore un vaste domaine de recherche [Weyns *et al.*, 2004] où les considérations distribuées doivent être prises en compte dans les nouvelles propositions.

Si les algorithmes de distribution des environnements restent à concevoir, nous proposons un cadre pour leur intégration dans l'architecture de distribution au travers des agents micro de distribution. Nous avons défini trois agents micro de distribution pour supporter l'exécution des agents de la simulation mais ce nombre peut être étendu. L'emploi des agents micro de distribution révèlera d'ailleurs pleinement son potentiel lorsque ceux-ci seront plus nombreux et que des interactions complexes seront nécessaire à la supervision de la plate-forme relativement à son intégration dans l'infrastructure d'exécution.

Poursuivre les efforts vers la réalisation d'une architecture distribuée pour la simulation orientée agent ne se limite pas à la réalisation technique de cette architecture. Il faut dès maintenant définir ce qu'est l'activité de simulation collaborative. Même si notre SMA de distribution réparti permet d'assurer le portage des services existant sur le réseau, il faut explorer ce nouveau milieu que constitue la simulation distribuée et créer des outils adaptés. Nous sommes convaincus que certains outils amèneront une réflexion plus générale qui sera valable quelque soit le support d'exécution de la SOA. Nous en avons nous-même fait l'expérience puisque nos travaux sur l'ordonnancement parallèle de la SOA nous ont conduit à une réflexion plus large sur la considération des connaissances temporelles dans le modèle de simulation. Le domaine distribué fournit une occasion d'éprouver nos outils actuels et de les améliorer autant que d'en développer de nouveaux.

Architectures distribuées, *Grille*, travail collaboratif, télé-travail, jeux massivement multi-joueurs, réseaux sociaux, *etc.* A l'heure où l'informatique se conçoit de manière distribuée, nous voyons un enjeu stratégique pour le développement des simulations orientées agent. Contribuons à un système collaboratif de création d'agents. Chacun libre de créer ses propres agents, d'utiliser ou modifier ceux des autres et de réaliser ses propres modèles de simulation, ce serait renforcer les applications scientifiques actuelles mais également ouvrir les systèmes multi-agents à des applications ludiques, artistiques ou créatives. Au travers de la distribution, nous devons saisir l'opportunité d'ouvrir les simulations orientées agent à un large spectre d'applications nouvelles.

Bibliographie

[Akyildiz *et al.*, 1993] Ian F. Akyildiz, Liang Chen, Samir Ranjan Das, Richard M. Fujimoto, and Richard F. Serfozo. The effect of memory capacity on Time Warp performance. *Journal of Parallel and Distributed Computing*, 18(4) :411–422, 1993.

[Anderson *et al.*, 2002] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@ home : an experiment in public-resource computing. *Communications of the ACM*, 45(11) :56–61, 2002.

[Anelli *et al.*, 2008] Pascal Anelli, Fanilo Harivelo, and Emmanuel Lochin. Détection des événements de congestion de TCP. In *Colloque francophone sur l'ingénierie des protocoles CFIP2008*, Les Arcs, France, 25-28 March 2008.

[Azaiez *et al.*, 2007] Selma Azaiez, Georges Habchi, Marc-Philippe Huget, Magali Pralus, and Jihene Tounsi. Multiagent oriented modeling and simulation for manufacturing systems control. In *Industrial Informatics, 2007 5th IEEE International Conference on*, volume 2, 2007.

[Bagrodia *et al.*, 1998] Rajive Bagrodia, Richard Meyer, Mineo Takai, Yu-An Chen, Xian Zeng, Jay Martin, and Ha-Yoon Song. Parsec : a parallel simulation environment for complex systems. 1998.

[Balbo *et al.*, 2004] Flavien Balbo, Nicolas Maudet, and Julien Saunier. Interactions opportunistes par l'écoute flottante. In *Actes des Journées Francophones sur les Systèmes Multi-Agents 2004*, pages 265–270, 2004.

[Beer *et al.*, 1990] Randall D. Beer, Hillel J. Chiel, and Leon S. Sterling. A biological perspective on autonomous agent design. *Robotics and Autonomous Systems*, 6(1-2) :169–186, 1990.

[Boehm, 1986] Barry W. Boehm. A spiral model of software development and enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4) :14–24, 1986.

[Boer *et al.*, 2009] Csaba A. Boer, Arie de Bruin, and Alexander Verbraeck. A survey on distributed simulation in industry. *Journal of Simulation*, 3 :3–16, 2009.

[Bousquet *et al.*, 1998] François Bousquet, Innocent Bakam, Hubert Proton, and Christophe Le Page. Cormas : Common-pool resources and multi-agents systems. *Lecture Notes in Computer Science*, pages 826–837, 1998.

[Buyya and Venugopal, 2004] R. Buyya and S. Venugopal. The gridbus toolkit for service oriented grid and utility computing : An overview and status report. In *1st IEEE International Workshop on Grid Economics and Business Models, 2004. GECON 2004*, pages 19–36, 2004.

[Buyya and Venugopal, 2005] Rajkumar Buyya and Srikumar Venugopal. A gentle introduction to grid computing and technologies. *CSI Communications*, 29(1) :9–19, 2005.

[Collier, 2003] Nicholson Collier. Repast : An extensible framework for agent simulation. *The University of Chicago's Social Science Research*, 2003.

[Corradi *et al.*, 1999] Antonio Corradi, Letizia Leonardi, and Franco Zambonelli. Diffusive load-balancing policies for dynamic applications. *IEEE concurrency*, 7(1) :22–31, 1999.

[Courdier *et al.*, 1998] Rémy Courdier, Pierre Marcenac, and Sylvain Giroux. Un processus de développement en spirale pour la simulation multi-agents. *L'Objet*, 4(1) :73–86, mars 1998.

[Courdier *et al.*, 2002] Rémy Courdier, François Guerrin, Fenintsoa Andriamasinoro, and Jean-Marie Paillat. Simulation agent appliquée à la gestion collective d'effluents d'élevage. *Journal of Artificial Societies and Social Simulation*, 5(3), 2002.

[David and Courdier, 2009] Daniel David and Rémy Courdier. See emergence as a metaknowledge, a way to reify emergent phenomena in multiagent simulations ? In *International Conference on Agents and Artificial Intelligence (ICAART'09)*, pages 564–569, Porto, Portugal, January 2009.

[David *et al.*, 2007] Daniel David, Denis Payet, Aurélie Botta, Gilles Lajoie, Sébastien Manglou, and Rémy Courdier. Un couplage de dynamiques comportementales : Le modèle DS pour l'aménagement du territoire. In *Journées Francophones Systèmes Multi-agents (JFSMA07)*, pages 129–138, Carcassone, France, 17-19 October 2007.

[David *et al.*, 2009] Daniel David, Denis Payet, Rémy Courdier, and Yassine Gangat. Xeloc : un support générique pour la configuration et l'initialisation de systèmes multi-agents. In Cepadues, editor, *Journées Francophones Systèmes Multi-agents (JFSMA'09)*, Lyon, France, 19-21 October 2009.

[Demazeau, 1997] Yves Demazeau. Steps towards multi-agent oriented programming. In *1st International Workshop on Multi Agent Systems*, 1997.

[Drogoul *et al.*, 1995] Alexis Drogoul, Bruno Corbara, and Steffen Lalande. MANTA : New experimental results on the emergence of artificial ant societies. *Artificial Societies : the computer simulation of social life*, pages 190–211, 1995.

[Drogoul *et al.*, 2003] Alexis Drogoul, Diane Vanbergue, and Thomas Meurisse. Simulation Orientée Agent : où sont les agents ? *Actes des Journées de Rochebrune, Rencontres interdisciplinaires sur les systèmes complexes naturels et artificiels*, 2003.

[Eugster *et al.*, 2003] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM computing Surveys*, 35(2) :114–131, 2003.

[Ferber *et al.*, 2004a] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations : an organizational view of multi-agent systems. *Lecture Notes in Computer Science*, pages 214–230, 2004.

[Ferber *et al.*, 2004b] Jacques Ferber, Fabien Michel, and José A. Baez-Barranco. AGRE : Integrating environments with organizations. *E4MAS*, pages 48–56, 2004.

[Ferber, 1995] Jacques Ferber. *Les systèmes multi-agents : vers une intelligence collective.* InterEditions Paris, 1995.

[Ferscha and Tripathi, 1994] Alois Ferscha and Satish K. Tripathi. Parallel and distributed simulation of discrete event systems. *Computer Science Technical Report Series ; Vol. CS-TR-3336*, page 51, 1994.

[Foster and Kesselman, 1997] Ian Foster and Carl Kesselman. Globus : A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2) :115–128, 1997.

[Foster *et al.*, 2004] Ian Foster, Nicholas Jennings, and Carl Kesselman. Brain Meets Brawn : Why Grid and Agents Need Each Other. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 8–15. IEEE Computer Society Washington, DC, USA, 2004.

[Foster, 2002] Ian Foster. What is the grid ? A three point checklist. *GRID today*, 1(6) :22–25, 2002.

[Fujimoto, 1990] Richard M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10) :30–53, 1990.

[Gaillard *et al.*, 2008] François Gaillard, Yoann Kubera, Philippe Mathieu, and Sébastien Picault. A reverse engineering form for multi agent systems. In Alexander Artikis, Gauthier Picard, and Laurent Vercouter, editors, *Proceedings of the 9th International Workshop Engineering Societies in the Agents World (ESAW'2008)*. Springer, 2008.

[Gentzsch, 2002] Wolfgang Gentzsch. Response to Ian Fosters "what is the grid ?". *Grid Today*, 1(8) :5, 2002.

[Grimshaw and Wolf, 1996] Andrew S. Grimshaw and William A. Wolf. Legion – a view from 50,000 feet. In *Proc. 5th IEEE Symp. on High Performance Distributed Computing*, 1996.

[Gupta and Nau, 1992] Naresh Gupta and Dana S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3) :223–254, 1992.

[Gutknecht and Ferber, 2000] Olivier Gutknecht and Jacques Ferber. Madkit : a generic multi-agent platform. In *Proceedings of the fourth international conference on Autonomous agents*, pages 78–79. ACM New York, NY, USA, 2000.

[Gutknecht, 2001] Olivier Gutknecht. Proposition d'un modèle organisationnel générique de systèmes multi-agents. Thèse de doctorat. *Universite de Montpellier II, Montpellier, France*, 2001.

[Guyot and Honiden, 2006] Paul Guyot and Shinichi Honiden. Agent-based participatory simulations : Merging multi-agent systems and role-playing games. *Journal of Artificial Societies and Social Simulation*, 9(4), 2006.

[Hayes, 2008] Brian Hayes. Cloud computing. *Communication ACM*, 51(7) :9–11, 2008.

[Hey and Trefethen, 2002] Tony Hey and Anne E. Trefethen. The UK e-science core programme and the grid. *Future Generation Computer Systems*, 18(8) :1017–1031, 2002.

[Hoareau, 2007] Didier Hoareau. *Composants ubiquitaires pour réseaux dynamiques.* PhD thesis, Université de Bretagne-Sud, Décembre 2007.

[Holcombe *et al.*, 2006] Mike Holcombe, Simon Cakley, and Rod Smallwood. A general framework for agent-based modelling of complex systems. Technical report, Department of Computer Sciences, University of Sheffield, 2006.

[Jansen, 2000] Wayne A. Jansen. Countermeasures for mobile agent security. *Computer Communications*, 23(17) :1667–1676, 2000.

[Jefferson and Reiher, 1991] David R. Jefferson and Peter Reiher. Supercritical speedup [discrete event simulation]. In *Simulation Symposium, 1991. Proceedings of the 24th Annual*, pages 159–168, 1991.

[Jefferson, 1985] David R. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3) :404–425, 1985.

[Jonquet *et al.*, 2006] Clément Jonquet, Pascal Dugénie, and Stefano A. Cerri. Intégration orientée service des modèles Grid et Multi-Agents. 2006.

[Kalman and Bucy, 1973] Rudolf E. Kalman and Richard S. Bucy. New results in linear filtering and prediction theory. *Random processes*, 15 :181, 1973.

[Katabi *et al.*, 2002] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 SIGCOMM conference*, volume 32, pages 89–102. ACM New York, NY, USA, 2002.

[Klein, 2004] Etienne Klein. *Les tactiques de Chronos.* 2004.

[Kubera *et al.*, 2008] Yoann Kubera, Philippe Mathieu, and Sébastien Picault. Interaction-oriented agent simulations : From theory to implementation. In *ECAI 2008 : Proceedings, 18th European Conference on Artificial Intelligence*, pages 383–387. IOS Press, July 21-25 2008.

[Kuhl *et al.*, 1999] Frederick Kuhl, Richard Weatherly, and Judith Dahmann. *Creating computer simulation systems : an introduction to the high level architecture.* Prentice Hall PTR Upper Saddle River, NJ, USA, 1999.

[Lamport, 1978] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7) :558–565, 1978.

[Lan *et al.*, 2001] Zhiling Lan, Valérie E. Taylor, and Greg Bryan. Dynamic load balancing for structured adaptive mesh refinement applications. In *International Conference on Parallel Processing*, pages 571–579, 2001.

[Larson *et al.*, 2009] Stefan M. Larson, Christopher D. Snow, Michael Shirts, and Vijay S. Pande. Folding@ Home and Genome@ Home : Using distributed computing to tackle previously intractable problems in computational biology. *eprint arXiv : 0901.0866*, 2009.

[Lees *et al.*, 2004] Michael Lees, Brian Logan, Rob Minson, Ton Oguara, and Georgios Theodoropoulos. Distributed Simulation of MAS. *Proceedings of the Joint Workshop on Multi-Agent and Multi-Agent-Based Simulation*, pages 21–30, 2004.

[Lubachevsky, 1988] Boris D. Lubachevsky. Bounded lag distributed discrete event simulation. *SCS Multiconference on Distributed Simulation, 1988*, 1988.

[Luenberger, 1964] David G. Luenberger. Observing the state of a linear system. *IEEE Transactions on Military Electronics*, 8(2) :74–80, 1964.

[Luke *et al.*, 2005] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. MASON : A Multiagent Simulation Environment. *Simulation*, 81(7) :517–527, 2005.

[Luque *et al.*, 1995] Emilio Luque, Ana Ripoll, Ana Cortes, and Tomas Margalef. A distributed diffusion method for dynamic load balancing on parallel computers. In *Proceedings of 3rd Euromicro Workshop on Parallel and Distributed Processing*, 1995.

[Mamei and Zambonelli, 2004] Marco Mamei and Franco Zambonelli. Motion coordination in the quake 3 arena environment : a field-based approach. In *Environments for Multi-agent Systems E4MAS*, volume 19. Springer, 2004.

[Marcenac and Giroux, 1998] Pierre Marcenac and Sylvain Giroux. Geamas : A generic architecture for agent-oriented simulations of complex processes. *Applied Intelligence*, 8(3) :247–267, 1998.

[Mattern, 1989] Friedemann Mattern. Virtual time and global states of distributed systems. *Parallel and Distributed Algorithms*, pages 215–226, 1989.

[McCormick *et al.*, 2000] John McCormick, Daria Chacón, Susan Mcgrath, and Craig Stoneking. A Distributed Event Messaging System for Mobile Agent Communication. *Second International Conference on Agent Systems and Applications and Fourth International Conference on Mobile Agents (ASA/MA 2000)*, 2000.

[McCown, 2002] Robert L. McCown. Changing systems for supporting farmers' decisions : problems, paradigms, and prospects. *Agricultural Systems*, 74(1) :179–220, 2002.

[Mertens *et al.*, 2004] Koenraad Mertens, Tom Holvoet, and Yolande Berbers. Adaptation in a distributed environment. In *First International Workshop on Environments for Multiagent Systems, New York*, 2004.

[Michel *et al.*, 2002] Fabien Michel, Pierre Bommel, and Jacques Ferber. Simulation distribuée interactive sous MadKit. *JFIADSMA 2002*, 2 :175–178, 2002.

[Minar *et al.*, 1996] Nilson Minar, Roger Burkhart, Chris Langton, and Manor Askenazi. The Swarm Simulation System : A Toolkit for Building Multi-Agent Simulations. Santa Fe Institute, 1996.

[Minsky, 1965] Marvin L. Minsky. Matter, mind and models. In *Proceedings-International Conference on Information Processing*, pages 45–49. UNESCO, 1965.

[Moigne, 1990] Jean-Louis Le Moigne. La modélisation des systèmes complexes. *Dunod*, 1990.

[Montresor *et al.*, 2002] Alberto Montresor, Hein Meling, and Özalp Babaoğlu. Messor : Load-balancing through a swarm of autonomous agents. In *Agents and Peer-To-Peer Computing : First International Workshop*, pages 125–135. Springer, 2002.

[Müller, 2004] Jean-Pierre Müller. The MIMOSA Generic Modelling and Simulation Platform : the Case of Multi-Agent Systems. In *Proceedings of the 5 th Workshop on Agent-based Simulation, Lisbon*, 2004.

[Nilsson, 1982] Nils J. Nilsson. *Principles of artificial intelligence*. Springer, 1982.

[North *et al.*, 2006] Michael J. North, Nicholson T. Collier, and Jerry R. Vos. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 16(1) :1–25, 2006.

[Osman and Ammar, 2002] Abdalla Osman and Hany H. Ammar. Dynamic load balancing strategies for parallel computers. *Sci. Ann. Cuza Univ.*, 11 :110–120, 2002.

[Padovitz *et al.*, 2003] Amir Padovitz, Seng Wai Loke, and Arkady Zaslavsky. Using the publish-subscribe communication genre for mobile agents. *Lecture notes in computer science*, pages 180–191, 2003.

[Payet *et al.*, 2006a] Denis Payet, Rémy Courdier, Tiana Ralambondrainy, and Nicolas Sébastien. Le modèle à Temporalité : pour un équilibre entre adéquation et optimisation du temps dans les simulations agents. In *Journées Francophones Systèmes Multi-agents (JFSMA06)*, pages 23–26, Annecy, France, 17-20 October 2006.

[Payet *et al.*, 2006b] Denis Payet, Rémy Courdier, Nicolas Sebastien, and Tiana Ralambondrainy. Environment as support for simplification, reuse and integration of processes in spatial MAS. In *2006 IEEE International Conference on Information Reuse and Integration*, pages 127–131, 2006.

[Peirce *et al.*, 1966] Charles Sanders Peirce, Charles Hartshorne Paul Weiss Arthur W., and Burks. *Collected papers*. Harvard University Press Cambridge, MA, 1966.

[Piunti *et al.*, 2007] Michele Piunti, Cristiano Castelfranchi, and Rino Falcone. Anticipatory coordination through action observation and behavior adaptation. 2007.

[Pollack and Ringuette, 1990] Martha E. Pollack and Marc Ringuette. Introducing the tileworld : Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189, 1990.

[Quaglia and Baldoni, 1999] Francesco Quaglia and Roberto Baldoni. Exploiting intra-object dependencies in parallel simulation. *Information Processing Letters*, 70(3) :119–125, 1999.

[Quesnel *et al.*, 2003] Gauthier Quesnel, Raphael Duboz, and Eric Ramat. Comparaison d'approches de simulations distribuées à événements discrets d'entités spatialisées. In *MAJECSTIC 2003*, 2003.

[Ralambondrainy, 2009] Tiana Ralambondrainy. *Observation de simulations multi-agents à grande échelle*. PhD thesis, Université de la Réunion, 2009.

[Ramat, 2006] Eric Ramat. Introduction à la modélisation et à la simulation à événements discrets. pages 49–74, 2006.

[Rao *et al.*, 1998] Dhananjai Madhava Rao, Narayanan V. Thondugulam, Radharamanan Radhakrishnan, and Philip A. Wilsey. Unsynchronized parallel discrete event simulation. In *Proceedings of the 30th conference on Winter simulation*, pages 1563–1570. IEEE Computer Society Press Los Alamitos, CA, USA, 1998.

[Roy *et al.*, 2005] Sarbani Roy, Saikat Halder, and Nandini Mukherjee. A multi-agent framework for performance tuning in distributed environment. In *HiPC*, 2005.

[Schaerf *et al.*, 1995] Andrea Schaerf, Yoav Shoham, and Moshe Tennenholtz. Adaptive Load Balancing : A Study in Multi-Agent Learning. *Journal of Artificial Intelligence Research*, 2 :475–500, 1995.

[Scheutz and Schermerhorn, 2006] Matthias Scheutz and Paul Schermerhorn. Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. *Journal of Parallel and Distributed Computing*, 66(8) :1037–1051, 2006.

[Schrödinger, 1935] Erwin Schrödinger. Die gegenwärtige Situation in der Quantenmechanik. *Naturwissenschaften*, 23(49) :823–828, 1935.

[Shannon, 2001 1949] CE Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1) :3–55, 2001 (1949).

[Simon, 1962] Herbert A. Simon. The architecture of complexity. *Proceedings of the American Philosophical Society*, pages 467–482, 1962.

[Soulié *et al.*, 1998] Jean-Christophe Soulié, Pierre Marcenac, Stéphane Calderoni, and Rémy Courdier. GEAMAS V2.0 : an Object Oriented Platform for Complex Systems Simulations. In *Proceedings of the 26th International Conference on Technology of Oriented-Object Languages and Systems (TOOLS USA'98)*, pages 230–242, 1998.

[Soulié, 2001] Jean-Christophe Soulié. *Vers une approche multi-environnement pour les agents*. PhD thesis, Université de la Réunion, 2001.

[Sycara, 1998] Katia P. Sycara. Multiagent systems. *AI magazine*, 19(2) :79–92, 1998.

[Sébastien *et al.*, 2006] Nicolas Sébastien, Rémy Courdier, and Marc-Philippe Huget. Toile sma pour la distribution de simulations larges échelles. In *Journées Francophones Systèmes Multi-agents (JFSMA06)*, pages 23–26, Annecy, France, 17-20 October 2006.

[Sébastien *et al.*, 2008a] Nicolas Sébastien, Rémy Courdier, Didier Hoareau, and Marc-Philippe Huget. Analyse des dépendances temporelles des influences et perceptions pour l'exécution distribuée de simulations orientées agent. In Cepadues, editor, *Journées Francophones Systèmes Multi-agents (JFSMA'08)*, volume ISBN 978.2.85428.861.2, pages 33–42, Brest, France, 15-17 October 2008. René Mandiau and Pierre Chevallier.

[Sébastien *et al.*, 2008b] Nicolas Sébastien, Rémy Courdier, Didier Hoareau, and Marc-Philippe Huget. Analysis of temporal dependencies of perceptions and influences for the distributed execution of agent-oriented simulations. In Eurosis, editor, *European Simulation and Modelling Conference (ESM2008)*, volume ISBN 978-90-77381-44-1, pages 343–347, Le Havre, France, 27-29 October 2008. Cyrille Bertelle and Aladdin Ayesh.

[Sébastien *et al.*, 2009] Nicolas Sébastien, Rémy Courdier, Didier Hoareau, Marc-Philippe Huget, and Denis Payet. Ordonnancement parallèle de simulations orientées agent : Une approche basée sur l'analyse des dépendances temporelles des influences et perceptions. *Revue d'Intelligence Artificielle*, 23(5-6) :675–697, November 2009.

[Taillandier and Drogoul, 2005] Patrick Taillandier and Alexis Drogoul. Simulations multi-agents participatives, Conception d'un environnement distribué de simulation participative. Master's thesis, INSA Lyon, 2005.

[Tand *et al.*, 2005] Yarong Tand, Kalyan Perumalla, Richard Fujimoto, Homa Karimabadi, Jonathan Driscoll, and Yuri Omelchenko. Optimistic parallel discrete event simulations of physical systems using reverse computation. In *Principles of Advanced and Distributed Simulation, 2005. PADS 2005. Workshop on*, pages 26–35, 2005.

[Tatara *et al.*, 2006] Eric Tatara, Michael J. North, Thomas R. Howe, Nicholson T. Collier, and Jerry R. Vos. An Introduction to Repast Simphony Modelling Using a Simple Predator-Prey Example. In *Proceedings of the Agent 2006 Conference on Social Agents : Results and Prospects*, 2006.

[Thiry, 1998] Philippe Thiry. *Notions de logique*. De Boeck Université, 1998.

[Tisue and Wilensky, 2004] Seth Tisue and Uri Wilensky. NetLogo : A simple environment for modeling complexity. In *International Conference on Complex Systems*, pages 16–21, 2004.

[Vally and Courdier, 1999] Jean-Dany Vally and Rémy Courdier. A Conceptual "Role-Centered" Model for Design of Multi-Agent Systems. *Lecture notes in computer science*, pages 33–46, 1999.

[Vigna, 1998] Giovanni Vigna. *Mobile agents and security*. Springer-Verlag London, UK, 1998.

[Wang *et al.*, 2004] Fang Wang, Stephen John Turner, and Lihua Wang. Agent communication in distributed simulations. In *Workshop on Multiagent Based Simulation (MABS)*, pages 11–24. Springer, 2004.

[Weyns *et al.*, 2004] Danny Weyns, H. Van Dyke Parunak, Fabien Michel, Tom Holvoet, and Jacques Ferber. Environments for multiagent systems state-of-the-art and research challenges.

Environments for Multi-Agent Systems, First International Workshop, E4MAS 2004, 3374, 2004.

[Willebeek-LeMair and Reeves, 1993] Marc Willebeek-LeMair and Anthony P. Reeves. Strategies for dynamic load balancing on highly parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(9) :979–993, 1993.

[Williams, 1991] Roy D. Williams. Performance of dynamic load balancing algorithms for unstructured mesh calculations. *Concurrency : Practice and experience*, 3(5) :457–481, 1991.

[Wooldridge and Jennings, 1995] Michael Wooldridge and Nicholas Robert Jennings. Intelligent agents : Theory and practice. *Knowledge engineering review*, 10(2) :115–152, 1995.

[Zimmermann, 1980] Hubert Zimmermann. OSI reference model–The ISO model of architecture for open systems interconnection. *Communications, IEEE Transactions on [legacy, pre-1988]*, 28(4) :425–432, 1980.

Table des figures

1.1 Le processus de simulation en quatre étapes. 12

1.2 Architecture d'une simulation CORMAS avec Simulación, extrait de [Taillandier and Drogoul, 2005]. 15

1.3 Architecture de MASON, issue de la documentation en ligne. 16

1.4 Architecture de SWARM, extrait de [Minar *et al.*, 1996]. 18

1.5 Matrice d'Interactions d'un modèle proies/prédateurs à 4 entités, extrait de [Kubera *et al.*, 2008]. 21

1.6 Architecture de MadKit, extrait de [Gutknecht and Ferber, 2000]. 22

1.7 Architecture en couches de la *Grille*, extrait de [Buyya and Venugopal, 2005]. . . 26

2.1 Structuration en couches de l'architecture de distribution. 41

2.2 Vue générale du SMA de distribution réparti. 43

2.3 Les agents systèmes d'un space et les environnements qu'ils perçoivent. 48

2.4 Architecture de distribution détaillée. 50

3.1 Exemple d'une progression du temps simulé par rapport au temps réel. 56

3.2 Structures de données associées au modèle à Temporalité. 58

3.3 Axe temporel simulé du modèle à Temporalité 59

3.4 Exemple d'exécution asynchrone 66

3.5 Axes temporels du couplage de Biomas sur Grand-Ilet et le Petit Tampon 72

3.6 Modèle à pas de temps constant, synchronisation pas par pas. 73

3.7 Modèle à Temporalité, synchronisation minimale. 73

3.8 Modèle à Temporalité, fonctionnement parallèle. 73

3.9 Evolution du temps simulé par rapport au temps réel selon le type d'ordonnancement global 75

4.1 Exécution parallèle d'une application de type *phase parallel*, extrait de [Osman and Ammar, 2002]. 85

4.2 Objectif de l'équilibrage de charges : minimiser les pauses sur chaque plate-forme. 89

4.3 Comportement d'un agent de flexibilité pour l'équilibrage de charges. 95

4.4 Illustration des différentes séquences de l'équilibrage de charges. 96

4.5 Des axes temporels simulés différents pour la même simulation distribuée. 99

4.6 Equilibrage de charges basés sur les tempos. 101

4.7 Exemple d'axe temporel où l'on suppose que T_D peut être exécuté juste après T_A. 104

4.8 Les différents modes d'exécution de la simulation. 105

5.1 Représentation sous forme de *schéma-bloc* du mécanisme d'optimisation de la simulation thématique. 114

5.2 Emploi de la simulation d'exécution distribuée de simulation thématique pour l'optimisation de l'exécution distribuée réelle. 114

5.3 Comportement global d'un agent macro de distribution dans notre prototype. . . 125

5.4 Axe temporel du cas n°1. 127

5.5 Variation de la durée d'équilibrage de charges en fonction du nombre de plates-formes. 132

5.6 Evolution de la répartition des agents de $simAgentA$ avec $tolerance = 0$. 134

5.7 Evolution de la répartition des agents de $simAgentA$ avec un critère d'équilibrage de charges prenant en compte les perturbations de mesures. 135

5.8 Persistance des oscillations de charge sur $distributor1$ due à une répartition non-optimale de la charge. 136

5.9 Evolution de la charge des plates-formes avec les règles de prévention d'oscillations. 136

5.10 Axe temporel de la simulation thématique. 137

5.11 Evolution des temps moyens de simulation avec et sans comptabilisation des temps réseaux en fonction du nombre de plates-formes. 138

5.12 Evolution du temps global de simulation en fonction du nombre de plates-formes. 139