Les Abstractions de Communication dans les Jeux en Réseaux Mobiles

Rapport du stage de fin d'études

Réalisé par

LE Van Tuan

Promotion 8, IFI

Sous la direction de

M. Antoine Beugnard

Département Informatique

Ecole National Supérieur des Télécommunications de Bretagne
France, 2004

Table des matières

1 État de l'Art **2**
1.1 Les types des jeux en ligne . . . 2
1.2 Architecture . . . 4
1.2.1 Topologie de réseau . . . 4
1.2.1.1 Client-Serveur . . . 4
1.2.1.2 Pair-à-Pair . . . 5
1.2.1.3 Hybride . . . 6
1.2.2 Architecture logiciel . . . 6
1.2.2.1 Intergiciel . . . 6
1.2.2.2 Médium . . . 7
1.2.2.3 Modèle Publish/Subscribe . . . 7
1.3 Verrous techniques . . . 8
1.3.1 La latence dans les jeux en ligne . . . 8
1.4 Synchronisation et consistance . . . 10
1.4.1 Dead reckoning - DR . . . 10
1.4.2 Bucket Synchronization avec DR (BS) et Stop-and-wait Synchronization (SWS) . . . 11
1.4.3 Time Warp Synchronization (TWS) et Trailing State Synchronization (TSS) . . . 12
1.5 Les plates-formes existantes . . . 13
1.5.1 Continuum . . . 13
1.5.1.1 Architecture de plate-forme . . . 13
1.5.2 TerraPlay . . . 14
1.5.3 ButterFly . . . 16
1.6 conclusion . . . 17

Bibliographie **19**

Chapitre 1

État de l'Art

Les jeux vidéo tiennent une place très importante dans l'informatique industrielle, de l'ordre de millions de dollars par année industrielle au japon, et double aux Etats-Unix. Dans ces prochaines années avec l'explosion de l'Internet, le jeu en ligne sera certainement un segment croissant rapidement de l'industrie de jeu vidéo. Mais avant que les futurs jeux en ligne puissent simuler la réalité avec plus de précision, ils doivent être capables de supporter les interactions entre les compétiteurs en temps réels. Dans ce chapitre, nous décrivons tout d'abord la situation actuelle des jeux en ligne, ainsi que les obstacles que les développeurs de ce type d'application doivent surmonter. Puis quelques points sur les architectures sur lesquelles les jeux sont construits. Finalement, comme le but du projet MEGA est la réalisation d'une plate-forme pour les jeux massivement multijoueurs, les plates-formes telles que TerraPlay, Continuum, ButterFly.net, etc. sont abordées. La sécurité de communication est actuellement hors de discussion de ce chapitre.

1.1 Les types des jeux en ligne

Avec des processeurs plus puissants, de nouveaux accélérateurs 3D, les jeux sont de plus en plus sophistiqués, plus réalistes et donc plus attrayants. Ils essaient d'attirer les joueurs non seulement par l'aspect multimédia mais aussi en leur donnant l'impression de se plonger dans un monde virtuel, faire face aux adversaires humains se situant au bout du monde. Cela modifie considérablement les habitudes des joueurs. Ils ont actuellement tendance à se présenter dans un très vaste espace - un champ de bataille virtuel avec nombre participants partout dans le monde par exemple. Malheureusement, les actuels jeux en ligne sont limité en nombre de joueurs, tous au plus quelques centaines. Le problème apparaît quand le nombre de participants du jeu augmente, le passage à l'échelle reste donc le défi majeur pour ce type d'application.

En fait, le degré de difficulté d'adaptation à l'augmentation du nombre de participants dépend de la nature du jeu. Par exemple, un jeu de type actions en temps réels et ayant l'architecture pure client-serveur est impossible d'avoir plus de centaines de joueurs dans le contexte actuel d'Internet. Selon l'encyclopédie WIKIPEDIA [23], il y a quatre classes

principales de jeux en ligne : *First-person shooters* (FPS), *Massive Multiplayer Online Role-Playing Games* (MMORPG), *Turn-based*, et *Real-time strategy* (RTS).

FPS est un genre de jeux vidéo combinant les actions de jeu vidéo avec la vision du monde sur l'écran qui simule celle du personnage. Trois jeux très connus, ***DOOM***, ***Quake*** et ***Half-Life*** font partie de cette classe. Ce type de jeu en ligne fournit un monde virtuel en temps réel. Les actions des joueurs doivent instantanément être reflétées dans le monde virtuel, aux vues de tous les joueurs concurrés par cette action, quelque soit leur position géographique. Les jeux de ce type restent donc limités à quelques dizaines de joueurs actuellement.

MMORPGs sont les mondes persistants virtuels situés sur l'Internet. Ils sont un sous-ensemble spécifique de jeux en ligne massivement multijoueurs dans lesquels les joueurs agissent l'un sur l'autre par l'avatar, c.-à-d., une représentation graphique du personnage qu'ils jouent. Ces jeux ne demandent pas d'exigences strictes d'interactivité comme les précédents. Néanmoins, le nombre de joueurs supporté n'atteint qu'au plus quelques centaines. Sans exhaustivité, voici quelques exemples des jeux MMORPG : ***Dragon Empires***, ***EverQuest, EverQuest II, Warring Factions, World of Warcraft, World War II Online***.

Turn-based : ***Sid Meier's Civilization, Heroes of Might and Magic, Chess, Go, Othello, Risk, Diplomacy*** sont des jeux de type Turn-based, également connus sous le nom de jeu de TBS (pour Turn-Based Strategy), dans lesquels chaque joueur doit attendre son tour pour prendre une décision, ou bien pour contrôler le personnage. Une fois que chaque joueur a pris son tour, le tour se termine. Pour ce type de jeux en ligne, le nombre de joueurs n'est pas limité par la communication réseau, car un retard de la diffusion de nouvel état est tolérable pour les compétiteurs.

RTS est un type de jeu vidéo qui n'a pas des « tours » comme les jeux turn-based conventionnels, mais le temps de jeu progresse en « temps réel » : c.-à-d., il est continu plutôt que discret. Deux jeux de MicroSoft : ***Age of Empires*** et ***Empires : Rise of the Modern World*** font partie de ce type, mais le premier jeu RTS était ***The Ancient Art of War*** de Evryware. ***Warcraft, Empire Earth, Command and Conquer, Total Annihilation***, et ***StarCraft*** sont aussi les jeux RST populaires. En général, les joueurs d'un jeu RTS suivent les trois pas (1) construire un infrastructure de base : villageois, fermes, casernes, etc. puis (2) collecter des ressources, et ensuite, (3) attaquer les ennemis, essayer de détruire leur patrie, de s'approprier leurs ressources. Vers la fin du jeu, la charge de la communication réseau augmente rapidement comme dans le cas de jeu FPS, mais ici, un joueur pourrait, à la fois, attaquer plus d'un ennemis, et être attaqué par les autres. C'est donc beaucoup plus difficile d'étendre le nombre de participants. Le jeu *Massively Multiplayer Online Real-Time Strategy* (MMORTS) est une combinaison des jeux massivement

multijoueur avec le jeu Real-time Strategy (par exemple : ***Shattered Galaxy***, et ***Mankind***).

La difficulté du passage à l'échelle provient également du fait que l'Internet - un environnement typiquement hétérogène - n'est pas capable de fournir une garantie d'une basse latence, ce qui cause l'incohérence du jeu. Par exemple, si deux joueurs sont tête à tête, on ne peut pas avoir une situation où le joueur A pense qu'il a tué le joueur B tandis que le joueur B pense qu'il a tué le joueur A. Si on a des contradictions en cours du jeu, c'est parce que le jeu n'est pas bien synchronisé.

Bref, la latence élevée, et l'incohérence qui en découle sont des obstacles primordiaux à surmonter avant que les prochaines générations de jeux massivement multijoueurs en ligne puissent accomplir la transition aux mondes virtuels réalistes. La plupart des études récentes se concentrent sur la communication réseau et la synchronisation comme des points cruciaux. D'une part, on cherche un mécanisme efficace qui doit être assez intelligent pour éviter de « bombarder » le réseau, en n'envoyant que les informations nécessaires, et qu'à ceux qui s'y intéressent. D'autre part, des études sur les techniques de synchronisation tels que « Bucket synchronization », « Stop-and-wait », et « dead reckoning » sont réalisées parallèlement afin d'éviter l'incohérence du jeu lors de la perte, ou bien le retard de messages transmis à travers Internet. Dans la suite de ce chapitre, nous présentons les techniques actuelles pour mettre en œuvre un jeu multijoueurs (et penser à ce que on essaie de faire d'un niveau d'abstraction).

1.2 Architecture

1.2.1 Topologie de réseau

Les deux topologies de réseaux les plus communes pour construire les jeux sont le *client-serveur*, et le *pair-à-pair*. Le client-serveur est conceptuellement plus simples et plus faciles à mettre en œuvre que le pair-à-pair.

1.2.1.1 Client-Serveur

Dans les jeux de type client-serveur, tous les joueurs - les clients - se connectent à une machine centrale, le serveur. C'est celui-ci qui est chargé de maintenir l'état global du jeu. Il collecte périodiquement les informations locales sur chaque client, calcule le nouvel état global puis distribue les mises à jour aux clients. Il faut donc une machine spécialisée pour le serveur, dont le prix est vraiment un problème. Les clients peuvent être considérés tous simplement comme des terminaux finaux. La notion de client-serveur est logiquement découplée ; parfois, le serveur réside sur la machine du joueur, on appelle ce cas, le « *serveur local* » ou le « *serveur d'écoute* ».

Les avantages de cette architecture pour les jeux en ligne sont que, en concentrant les calculs du nouvel état du jeu sur le serveur, la « triche » peut donc être empêchée

efficacement. La cohérence du jeu est totalement assurée, la seule chose dont les clients s'occupent est l'affichage du graphique des mondes simulés fournis par le serveur. Le serveur devient rapidement un point faible du système lorsque le nombre de participants augmente, pour deux raisons :

Passage à l'échelle : toutes les données sont convergentes sur le serveur. De plus, comme les données doivent passer par le serveur pour aller aux autres, la latence est donc augmentée.

Fragile aux fautes : dans tel système centralisé, le serveur est un point sensible aux fautes. Si tout à coup, le serveur tombe en panne, le système est inutilisable.

1.2.1.2 Pair-à-Pair

Comme l'indique le nom, dans ce type de jeu, les clients communiquent en direct chacun avec les autres, collectent des informations puis eux-mêmes calculent l'état présent du jeu. Les machines des clients sont déléguées pour calculer le jeu, prendre des décisions, et accéder aux bases des données. De ce fait, les tricheurs ont une chance d'intervenir dans le jeu, de modifier le résultat, etc.

Comme les messages sont transmis directement entre les clients, il n'y a pas de passage par serveur. La latence est donc réduite considérablement. Mais on rencontre la même situation qu'avec les jeux ayant une architecture client-serveur : bien que les calculs du jeu soient répartis auprès de chaque client, le passage à l'échelle reste un problème, car chaque client doit également envoyer lui-même les informations locales aux autres, et collecter les états des autres pour calculer les mises à jours. Le nombre des informations transmises à travers le réseau est donc le même dans les deux cas. De plus, l'inconsistance du jeu est plus difficile à régler. L'administration n'est maintenant plus concentrée comme dans le cas du client-serveur.

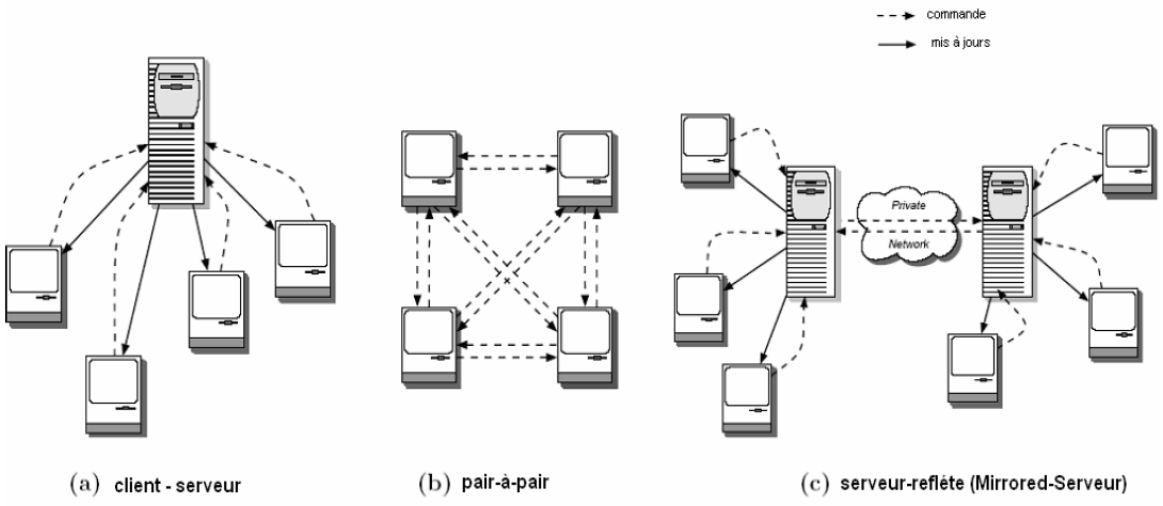

FIG. 1.1 – Topologies de réseaux

1.2.1.3 Hybride

En réalité, un système pur client-serveur, ou pair-à-pair est rarement implémenté. La solution adoptée est un système hybride, celui-ci ayant l'architecture client-serveur comme topologie de réseau, mais la charge est répartie auprès des clients, et ne converge pas sur le serveur. Dans un tel système, les clients sont autorisés à contrôler leur propre mouvement, et puis à leur tour, rapportent leur position au serveur qui modérera probablement le mouvement rapporté, dans les cas nécessaires. Autrement dit, le serveur joue le rôle d'un arbitrer dont le travail est de juger l'état global du jeu. S'il trouve une contradiction, alors il intervient au jeu en appliquant un algorithme de synchronisation (discuté ultérieurement). Cela réduit considérablement le travail du serveur ; malheureusement, on rencontre encore le problème de la sécurité.

Une autre variance est appelée *Serveur-Reflété* (Mirrored-Server [11]) : plusieurs serveurs sont ajoutés au système. Les machines spécialisées sont les serveurs du système qui s'étendent dans le monde entier, et qui peuvent être vus comme un serveur unique à l'extérieur du système. Puisque il n'y a pas un seul serveur, le point unique d'échec du système est éliminé. Les clients prennent contact l'un avec l'autre au moyen des serveurs comme s'il n'y en avait qu'un. Le mécanisme est presque transparent comme pour un système client-serveur traditionnel. A l'intérieur, les serveurs communiquent directement de manière pair-à-pair. Cette solution donne une meilleure latence de réseau en gardant l'administration chez l'exploitant du système. Par contre, l'implémentation d'un tel système est plus complexe. La communication entre les serveurs, entre les serveurs et les clients demande un protocole sophistiqué pour la consistance du jeu.

1.2.2 Architecture logiciel

1.2.2.1 Intergiciel

Intergiciel (*middleware* en anglais) est un terme informatique qui est employé pour décrire un logiciel agissant en tant qu'intermédiaire, ou en tant que membre d'un groupe d'intermédiaires, entre différents composants qui ne fonctionnent pas tous sur les mêmes système d'exploitation et ne sont pas écrit dans les mêmes langages. Dans l'industrie du jeux, un intergiciel a pour but de faciliter les travaux des développeurs de jeux en fournissant une suite des robustes logiciels (les modèles graphiques, les textures, l'animation, la communication réseau, etc.). Les intergiciels aident les développeurs de jeux à ne pas s'écarter de leur métier de base. Une bonne explication convenant bien au cas des jeux en ligne provient du Middleware FAQ [17] : *"Middleware is the intersection of the stuff that network engineers don't want to do with the stuff that applications developers don't want to do"*. TerraPlay, Continuum, et Quazal sont des exemples de plates-formes existantes, qui supportent les jeux massivement multijoueurs en lignes.

1.2.2.2 Médium

Dans un système distribué, les composants échangent, ou partagent les informations nécessaires à l'aide de composants de communication. Afin de différencier des autres composants du système (les composants métiers, ou les composants fonctionnels, etc.), Eric Cariou dans sa thèse doctorat [20] appelle ces composants de communication des médiums, dont la définition est : « *Un médium est une composant logiciel de communication ou d'interaction qui réifie*[1] *une abstraction de communication* ». Un médium peut implémenter une abstraction de communication, un protocole consensus par exemple.

Le premier pas de construction d'une composant de communication est de déterminer les services que ce composant mettra à disposition des autres composants ; et aussi les services requis par le médium pour fonctionner. C'est ainsi que la façon dont les entités logicielles communiquent est abstraite de sorte que « les détails sont cachés » et non pas « la notion est floue et mal définie »[20]. Dans ce pas, le médium est vu comme un unique entité qui fait partie du système. Pour la transformation complète d'un médium à partir de sa spécification abstraite vers son implémentation, E. Cariou a proposé un processus de transformation à tous les niveaux du cycle de vie du logiciel, avec trois étapes :

- *Introduction des gestionnaires de rôle* : la première étape du processus de raffinement a pour but de faire apparaître les types de gestionnaires de rôle qui forment le médium lors de déploiement.
- *Choix de conception* : le résultat de cette étape est une spécification d'implémentation.
- *Choix de déploiement* : il s'agit de définir l'architecture de déploiement des gestionnaires du médium.

A chaque étape de ce processus, des diagrammes de collaboration UML sont utilisés pour décrire l'aspect structurel du médium, mettre en évidence la sémantique et le comportement dynamique de chacun des services du médium. Les diagrammes de collaboration ont pour objectif de décrire le médium au niveau spécification. Dans la dernière étape, le diagramme de déploiement est employé pour chaque type de déploiement défini. Outre, des contraintes OCL, des diagrammes d'états sont utilisés en plus pour une spécification plus claire du médium.

1.2.2.3 Modèle Publish/Subscribe

Dans le modèle *Publish/Subscribe*, l'idée de base est que les destinataires ne reçoivent que les messages auxquels ils s'intéressent, les expéditeurs envoient les messages sans avoir besoins de savoir à qui ceux-ci s'adressent. Pour le faire, les expéditeurs, appelés également les *publishers* diffusent les messages au composant de communication, auquel les destinataires - les *subscribers* - s'inscrivent pour pouvoir recevoir les messages. L'envoi et la réception des messages peuvent être effectué selon le mode (1) push-based et

[1] La réification est l'opération de transformation en quelque chose de concret.

(2) pull-based. En mode push-based, les messages sont automatiquement diffusés à tous les subscribers. Tandis qu'en mode pull-based les messages doivent être retirés par les subscribers.

Les implémentations d'un modèle publisher/subscriber dans les systèmes réels peuvent différer les uns des autres, mais en général, on peut les diviser en deux catégories principales [3, 22] (1) *basé sur sujet* (suject-based) et (2) *basé sur contenu* (content-based).

La première catégorie organise les canaux de communication, appelés également réseau de communication, par lesquels un ensemble de messages sont transmis selon des conditions prédéfinies. Les publishers diffusent les messages en choisissant les canaux. On remarque que seuls les messages qui s'accordent avec le sujet d'un canal sont placés dans celui-ci. Les subscribers, de leur côté, s'inscrivent aux canaux dont le sujet les intéresse. Ensuite, ils reçoivent tous les messages transmis par ces canaux.

Dans le mode basé sur contenu, il n'existe plus la notion de canaux. Les subscribers expriment leur désir sur le contenu de messages qu'ils veulent recevoir, ce qui est réalisé en général par une requête. Ce mode de communication est plus flexible [22], car il n'est pas nécessaire que les subscribers sélectionnent les canaux parmi ceux qui existent a priori. Mais l'implémentation pour ce modèle est plus compliquée. Pour la diffusion correcte des messages, on doit construire une fonction de correspondance (matching function) qui a pour but d'associer correctement les messages en provenance des publishers aux subscribers. L'algorithme basé sur sujet est plus simple à implémenter que celui basé sur contenu.

1.3 Verrous techniques

1.3.1 La latence dans les jeux en ligne

Afin de pouvoir répartir la charge des serveurs vers les clients dans un système client-serveur, les machines de joueur sont autorisées à simuler le monde du jeu lui-même mais de manière synchronisée avec les autres. Idéalement, pour un instant déterminé et au niveau global, toutes les simulations sur les machines chez les clients devraient être uniques (ou presque) du point de vue des joueurs. Mais parfois, due à la latence, la représentation d'une action d'un joueur (appelée événement) n'est pas totalement représentée à temps sur les machines autres que celle qui l'a générée. En conséquence, l'inter-cohérence est brisée. Par exemple, dans un jeu multijoueur : Au moment t1, le joueur A commence à s'avancer. Le message est envoyé par le réseau au serveur, il arrive au moment t2 un peu plus tard. En cet instant (t2), le serveur copie l'état du joueur A (s'avancer). Malheureusement, c'était déjà derrière la position réelle de A (parce qu'il s'était avancé en moment t1). Maintenant le serveur annonce à tous les clients que A avait commencé à s'avancer. Chaque participant, y compris A, reçoit le message à l'instant t3. Et à cause à de la latence tous les clients sont maintenant encore derrière le serveur.

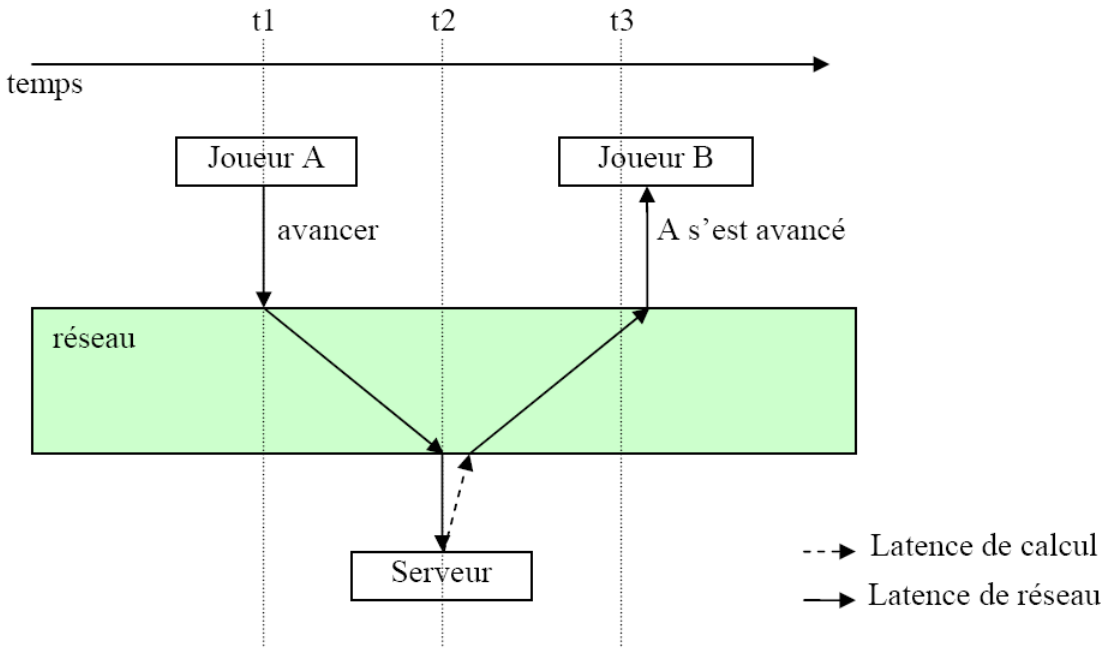

FIG. 1.2 – L'effet de la latence

Le dictionnaire Webopédia [10] donne une définition de la latence :

- *En général, c'est la période qu'un composant dans un système attendant un autre composant. Donc, la latence est le temps gaspillé. Par exemple, pour l'accès des données sur un disque, la latence est définie par le temps que cela prend à positionner le secteur approprié sous la tête de lecture/écriture.*
- *En terme de réseau, la latence est la quantité de temps que prend un paquet pour aller de la source à la destination. Ensemble, la latence et la bande passants définissent la vitesse et la capacité d'un réseau.*

Dans les jeux multijoueurs, la latence est l'intervalle de temps entre l'action d'un joueur et la représentation de cette action sur le terminal de l'autre joueur (temps pour l'envoi, et la réception des données), abaisser la latence a pour but d'assurer mieux la cohérence du jeu. Au niveau réseau et application, il y a trois sources possibles de latence dans les jeux en ligne en provenance d'infrastructure réseau [12] :

Latence de protocole réseau : le temps d'opération que le système met pour placer les données dans réseau physique, pour les en retirer (y compris éventuellement le temps pour les encrypter et les décrypter, bien entendu, cela augmente également la latence !).

La latence de transmission : le temps passé pour envoyer des données à travers le réseau jusqu'au récepteur. Comme dit dans la section précédente, les jeux qui emploient la topologie client-serveur donnent les latences plus élevées que ceux ayant l'architecture pair-à-paire. En effet, les messages doivent passer par le serveur au lieu d'être envoyés directement des expéditeurs aux destinataires.

Latence de traitement : c'est le temps que l'application met elle-même à manipuler les données lors de l'envoi et lors de la réception.

La *gigue* (jitter - en anglais), une variante de la latence, cause le clignotement sur l'écran. La gigue peut être provoquée par un certain nombre de problèmes de matériel, tels qu'un taux de rafraîchissement faible de l'écran, un défaut de fonctionnement de matériel dans le moniteur, ou une synchronisation faible dans les signaux étant envoyés. La question qui se pose est donc : que peut-on faire pour réduire la latence, s'il est impossible de faire circuler les informations à travers le réseaux à la vitesse égale ou supérieure à celle de la lumière ?

1.4 Synchronisation et consistance

L'exemple précédent montre que, à cause de la latence, ce que le joueur A voit sur son écran est différent de ce que voient les autres. La synchronisation est donc extrêmement importante dans un contexte distribué. Cela a pour but d'assurer la cohérence du jeu. Toujours sur le même l'exemple, après avoir reçu le message du serveur indiquant sa nouvelle position, avec un effort de synchronisation du client A, il essayera de compenser la différence entre le message du serveur et son état actuel en passant le personnage du client A à l'endroit où le serveur a indiqué qu'il devraient être. Ainsi, le joueur A voit ses mouvements et ceux des autres sont saccadés. Cela est la conséquence de l'application d'une technique de synchronisation nommée « *Local presentation delay* », un des mécanismes de synchronisation le plus simple. Cet effet peut être éliminé en le combinant avec une autre technique qui retarde l'affichage sur l'écran du joueur. Cependant, les techniques ci-dessus ne conviennent pas aux applications plus sophistiquées qui demandent des techniques de synchronisation plus complexes. Dans la suite de cette section, nous présentons quelques mécanismes qui sont largement implémentés dans les jeux (MiMaze, Quake, par exemple) [13, 15, 16].

1.4.1 Dead reckoning - DR

Le « Dead reckoning » est une technique d'extrapolation utilisée dans les systèmes aériens pour calculer une évaluation de la position courante d'un avion en se basant sur la connaissance de sa position passée et sur sa trajectoire. DR est appliqué dans les jeux multijoueurs, utilisé dans le cas où on voudrait diminuer la latence pour les joueurs, ou contre la perte de messages envoyés. L'idée de base est de choisir à l'avance un ensemble d'algorithmes qui peuvent être employés par tous les nuds des joueurs pour extrapoler le comportement des entités dans le jeu, et choisir à quel moment on doit permettre à l'application de réaliser ces algorithmes. Dans le domaine du jeu, il y a deux catégories de DR souvent utilisées : (1) prédiction de la position : prophétise la nouvelle position en se basant sur la position courante et la trajectoire de l'objet. Et (2) la prédiction

d'événements entrés par les joueurs : l'entrée attendue est utilisée pour calculer la nouvel position de l'objet.

Afin de pouvoir utiliser DR dans des environnements virtuels distribués (EVD), on les partitionne en des entités isolés, une personne, une voiture, un avion ou une balle [14]. Chacune de ces entités est contrôlée par exactement une application participant dans EVD. Les applications qui s'intéressent à une entité sont capables de prédire comment l'état de l'entité changera au cours du temps.

On suppose qu'on applique le DR comme le mécanisme de synchronisation dans l'exemple de la section précédente. Le contrôleur de l'application du joueur A prévoit sa nouvelle position au moment t2. Il prévient au serveur la nouvelle position prévue. Donc quand le serveur informe les autres de nouvelle position du joueur A, l'effet de la latence est diminué. Les mouvements des joueurs sont alors plus fluides. Mais en réalité, l'action des joueurs n'est pas toujours prévisible. Que se passera-t-il si le joueur change brusquement de direction, ou de vitesse ? Martin Mauve donne deux exemples de contradiction en appliquant le DR [14] : *A Dead Man that Shoots* (un homme mort qui tire) et *A Flying Tank* (un tank volant).

Cependant, le DR est largement implémenté dans les systèmes distribués [14] pour deux raisons (1) l'auto-préparation : une transmission perdue sera compensée par une autre transmission d'état pour la même entité. (2) il n'a pas besoin d'une gestion centralisée des états. Chaque application est capable de gérer localement les états au moyen de prédiction et de réception des mises à jour d'état.

1.4.2 Bucket Synchronization avec DR (BS) et Stop-and-wait Synchronization (SWS)

L'idée de base de ces techniques est de distinguer le temps réel de celui du jeu, autrement dit, c'est une façon de discrétiser le temps. Le temps dans le jeu est vu comme des séries d'intervalles. Le nouvel état n'est calculé qu'à la fin d'un intervalle. Ainsi, plus cet intervalle est court, plus les mouvements dans le jeu sont fluides.

BS divise le temps du jeu en séries d'intervalles constants. Pendant un intervalle - qui pourrait être assez long - un client collecte tous les mis à jour des autres, les messages (événements) collectés sont placés dans un seau de synchronisation. Tous les clients doivent attendre la fin de l'intervalle pour calculer le nouvel état du jeu, les algorithmes d'extrapolation sont appliqués pour les messages perdus ou retardés (technique DR). Ce mécanisme est implémenté dans le jeu MiMaze [9]. MiMaze n'essaie pas de détecter les contradictions ou de récupérer. Il se peut que dans un seau, il y ait plus d'un événement. Mais seul celui qui est le plus récent est exécuté. Dans le cas où il n'y a aucun événement, l'événement dans le seau précédent est interpolé avec le DR. Pour la deuxième technique - SWS, les intervalles de temps peuvent être de taille variable selon le temps d'attente du client le plus lent.

Dans de tels systèmes, il est impossible d'avoir des inconsistances, car l'ordre des événements générés est conservé efficacement. Par contre, ces techniques de synchronisation ne résolvent pas la latence pour les jeux étant sensibles à la latence (par exemple les jeux de type FPS). Il n'y a aucun moyen pour garantir que le jeu avancera à un rythme régulier et constant, ou assez vite pour les jeux interactifs. De ce fait, ces mécanismes ne conviennent pas aux jeux qui demandent des mises à jour très fréquentes, une forte consistance, sont très sensibles à la latence.

1.4.3 Time Warp Synchronization (TWS) et Trailing State Synchronization (TSS)

Ces mécanismes exécutent les événements au fur et à mesure de leur apparition [13, 16], même si un événement antérieur aurait pu arriver. Ils effectuent des retours en arrière pour réparer les inconsistances nécessaires.

Dans TWS, l'état de tous les objets est sauvegardé à chaque intervalle fixé de temps avant d'exécution d'un événement. En cas de retour en arrière (une contradiction détectée), tous les événements entre la sauvegarde et le moment d'exécution sont re-exécutés. Cependant, dans cette technique de synchronisation, comme un point de contrôle (checkpoint [16]) est nécessaire pour chaque message, une mémoire énorme est exigée. Par conséquent, cette technique n'est pas pratique.

L'algorithme TSS est implémenté dans Quake [4, 13]. L'état global du jeu est parallèlement exécuté par les remorquages d'état (trailing state) avec les retards différents. Par exemple, à l'instant t, autre que l'état actuel c'est celui qui est sur le point d'exécuter ; il y a un remorquage d'état qui exécute la simulation à l'instant t - d_1, et le deuxième remorquage à l'instant t - d_2, etc. Dans le cas d'inconsistance détectée, l'algorithme copie alors un des remorquages pour re-exécuter.

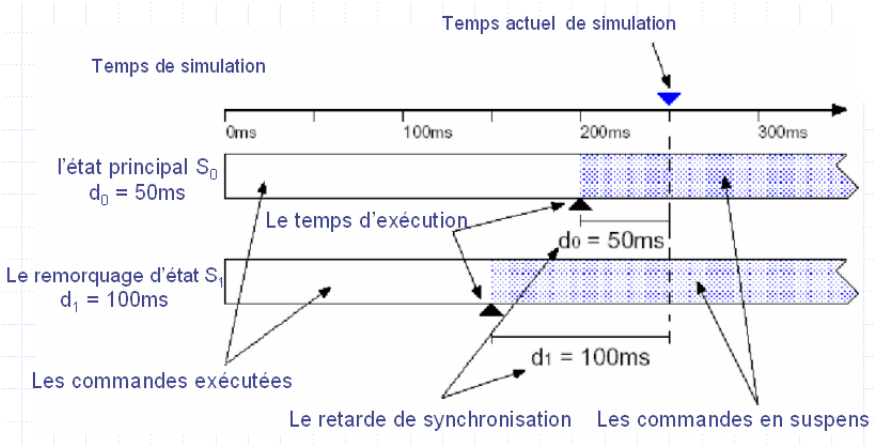

FIG. 1.3 – L'exécution du remorquage d'état de synchronisation dans Quake

1.5 Les plates-formes existantes

Cette section s'attache donc à rapporter quelques intergiciels existants pour les jeux massivement multijoueurs.

1.5.1 Continuum

Continuum [5, 6] est une pièce essentielle du projet européen baptisé Ping, dont le but est de fournir une infrastructure dédiée aux jeux vidéo en réseau, capable de faire interagir en temps réel des milliers de joueurs dispersés à travers le monde.

Continuum vise à la construction d'un monde virtuel sur Internet. Un monde virtuel contient un ensemble d'objets virtuels, appelés *entités*, se situant dans le monde. Un *réplica* est conceptuellement une reproduction d'une entité unique. Eventuellement, la simulation de ce monde virtuel est partiellement (globalement si nécessaire) représentée dans la machine d'un client, appelé *espace de simulation*, par les réplicas des entités participant à ce monde. Une entité sera reproduite dans un espace de simulation si elle a influence sur d'autres objets dans cet espace. Parmi les réplicas d'une entité, on distingue le *réplicas maître* (master replica), ce qui peut être compris comme l'objet lui-même, les autres sont les copies de l'objet. Le réplica maître communique avec les autre en observant une politique prédéterminée de consistance (mécanisme de communication).

Les espaces de simulation du système se connectent les uns aux autres au moyen des canaux de communication et de manière synchronisée. Chacun d'entre eux est donc capable de calculer les interactions entre des entités.

1.5.1.1 Architecture de plate-forme

Application (comportement des objets, affichage graphique)	comprend des objets simulés qui contiennent des informations spécifiques tels que modèle graphique, comportements, règle d'interaction entre des entités.
Gestion d'objets	gère la correction des manipulations des entités réplicatives (book-keeping, cycle de vie, et ramasse miettes)
Gestion d'événements	le service de synchronisation (causalité en ordre, buffering, timestamp)
Gestion d'Aura (Aura management)	responsable de filtrage des événements selon le modèle de réplication. C'est cette couche qui route les événements qui circulent par le réseau.
Groupe de communication	fournit divers services de communication
Supporte de réseau	intégration des protocoles réseau

FIG. 1.4 – L'architecture en couche de Continuum.

Continuum est organisé en couche, la frontière entre ces couches est bien claire. Chaque couche se distingue par ses politiques et mécanismes et peut être étendue pour s'adapter

aux besoins des applications. Le noyau décrit un ensemble de composants abstraits, dont chacun est visible des autres seulement par ses interfaces.

Les communications entre les réplicas représentant un objet sont effectuées par les événements. Une fois qu'un événement est engendré, il est daté (pourrait ultérieurement être utilisé par des mécanismes de synchronisation implémentés dans l'application). Les algorithmes de synchronisation peuvent être intégrés à plate-forme facilement, en fonction des caractéristiques de l'application.

1.5.2 TerraPlay

TerraPlay [18] est une plate-forme qui a pour but de fournir une solution de haute qualité pour les développeurs de jeux en temps réels, en réseaux fixe ainsi qu'en réseaux mobiles. Le système se compose d'un faisceau de serveurs, qui sont chargés de gérer et synchroniser les jeux. Ils ne sont pas un centre pour distribuer les données temps réels du jeu. Le système TerraPlay s'occupe de toutes les copies et les multicast des données du jeu. Les développeurs, qui voudraient construire leurs jeux sur cette plate-forme pour bénéficier des services de communications, devraient développer deux types d'applications : « *lobby serveur* » et « *application d'utilisateur* ».

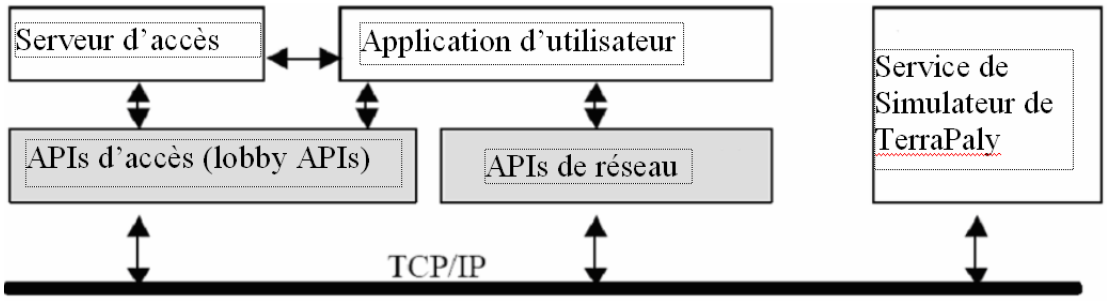

FIG. 1.5 – Les composants dans un jeu construit sur TerraPlay

Lobby serveur est utilisé par les joueurs pour définir les paramètres d'une session du jeu, pour authentification, la recherche des compétiteurs en ligne, et le démarrage du jeu.

Application d'utilisateur une instance de jeu sur la machine de joueur.

Les applications de jeu interagissent avec les APIs réseau de TerraPlay, qui font partie du système TerraPlay mais résident sur les machines des clients.

Une session du jeu est typiquement commencée depuis le lobby serveur. Les joueurs accèdent au lobby serveur pour recevoir les informations nécessaires pour qu'ils puissent rejoindre une session du jeu. Ce sont :

- l'adresse de réseau de GAS (*Game Access Server*)
- le numéro de port sur le serveur GAS
- le mot de passe.

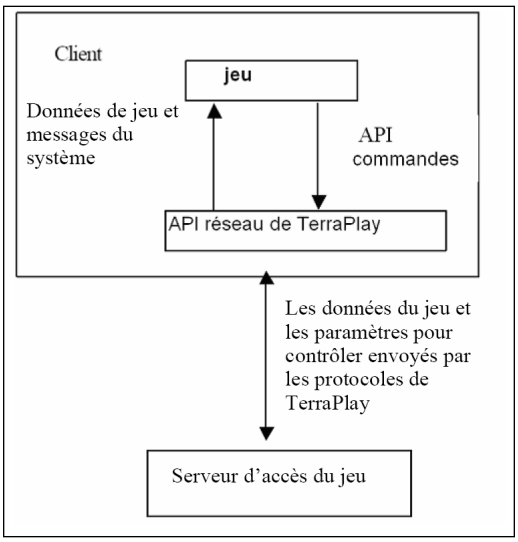

FIG. 1.6 – Une architecture typique d'un jeu sur TerraPlay

Pour la communication entre les clients, TerraPlay fournit trois mécanismes différents : **stream object**, **basic object**, et **message** pour l'envoi des mises à jours. Les deux premiers mécanismes, stream object et basic object, sont très similaire du modèle publisher/subscriber (respectivement basé sur sujet, et basé sur contenu).

Les mises à jour ne sont envoyées qu'aux clients qui voudraient les recevoir, et pour recevoir ces informations, ils doivent s'y inscrire explicitement. Les messages sont utilisés pour l'envoi des informations à un joueur ou bien un groupe de joueurs déterminés (avec des adresses de destinataires bien indiquése). Les informations envoyées sont privées et secrètes. Les mises à jours envoyées par stream object peuvent être datés en utilisant le service de temps du système. Cependant, dans les deux modes stream ou basic object, seulement le dernier mises à jour est prise en compte.

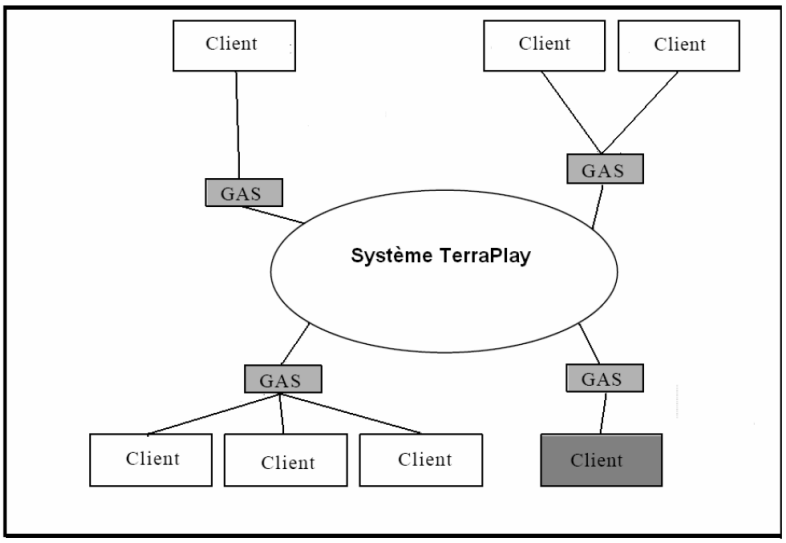

FIG. 1.7 – La communication dans système TerraPlay

1.5.3 ButterFly

ButterFly.net [19] est un environnement virtuel réseau de type client-serveur avec multiple serveurs qui emploie la technologie de grille[2] (grid technology). Cette plateforme supporte les jeux massivement multijoueurs et temps réels. Elle fournit également une suite complète de logiciels pour aider les développeurs des jeux à développer, tester, opérer, et gérer.

Dans ButterFly.net, on divise l'espace virtuel en multiples régions. Chaque région est assignée à un serveur. Les serveurs sont totalement « *maillés* », c'est-à-dire que chacun est connecté à tous les autres dans le système par des fibres optiques à haute vitesse. Plus d'un million de participants peuvent jouer simultanément.

Il y a deux faisceau de serveurs dans ButterFly.net composés de 50 eServer xSeries de IBM placés à IBM Sterling, VA et Los Angeles, avec quatre catégories (basé sur leur rôle à l'intérieur de la grille)

- *Serveur de jeu* (Game Server) - la plupart des serveurs dans chaque faisceau sont serveurs de jeu. Ils sont généralement responsables d'exécuter des jeux dans la grille. Leurs fonctions principales incluent la médiation du jeu, imposant les règles de jeu stockées sur des serveurs de base de données (ci-dessous) et commandant des interactions avec d'autres serveurs de jeu. Ces serveurs fonctionnent sur le matériel Linux-basé xSeries x330.
- *Serveur d'accès* (Gateway servers) - fonctionnant aussi sur xSeries x330, ils sont responsables de relier des joueurs (c.-à-d., clients) aux serveurs de jeu. Les serveurs de passage effectuent également des traductions de protocole et conduisent des raccordements de joueur aux serveurs de jeu.
- *Les contrôleurs de démon* (Daemon controllers) : sont des serveurs d'intelligence artificielle (fonctionnant sur le matériel de xSeries x330) dont la fonction est de commander des personnages de non-joueur et d'autres éléments du jeu, pas directement commandés par des actions des joueurs.
- *Les serveurs de bases de données* (Database servers) - sont responsables de stocker toute l'information (la physique, la géométrie, les règles du jeu, etc.) nécessaire pour maintenir la persistance dans le monde. Ces bases de données, logées sur les serveurs montés sur les xSeries x232, maintiennent également la comptabilité et les données d'authentification et fournissent l'interface aux fonctions de soutien de facturation.

[2]Une grille est un système (un infrastructure des matériels et des logiciels) de type parallèle et réparti qui permet le partage, le choix, et l'agrégation des ressources « autonomes » de manière dynamique et géographiquement distribuées au temps d'exécution selon leur disponibilité, possibilités, performances, coût, et les demandes du qualité-de-service des utilisateurs.

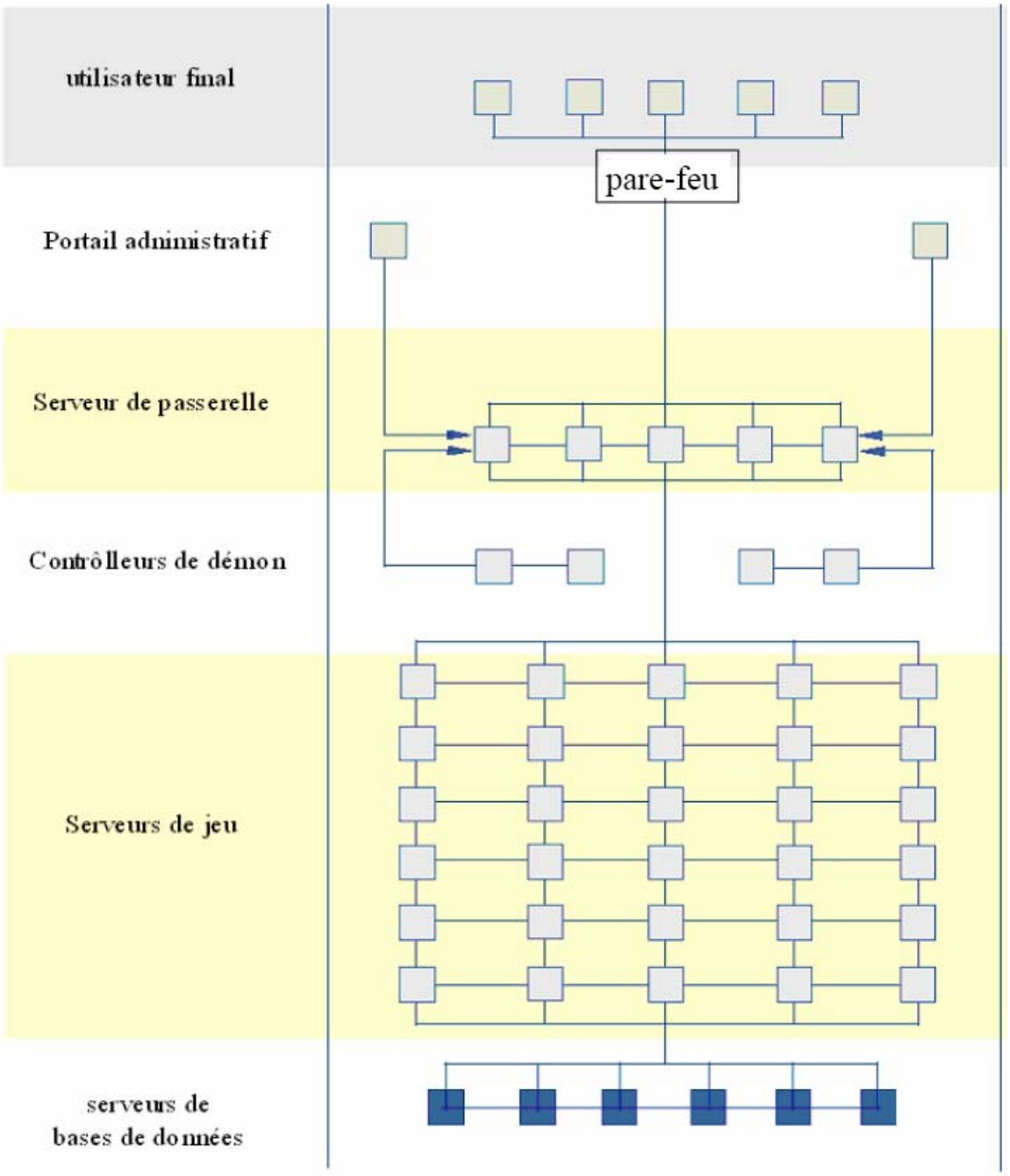

FIG. 1.8 – Architecture de ButterFly

1.6 conclusion

Avant qu'un jeu soit publié, il y de nombreux de travaux à accomplir, surtout pour les jeux massivement multijoueurs en ligne. Par contre, pour qu'un jeu puisse se rendre maître du marché, le temps de développement doit être le plus court possible. Pour cette raison il apparaît sur le marché des plates-formes visant à aider les développements de ce type d'application. En utilisant les suites des logiciels robustes sur lesquels ses jeux sont construits, les temps de développement sont abrégés, car les développeurs se concentrent sur les jeux, et non pas sur la communication réseau ou bien les modèles physiques par

exemple. Ce stage se déroule dans le cadre de projet Méga, qui a pour but de construire une plate-forme pour les jeux massivement multijoueurs et en réseau mobile.

Les travaux de ce stage sur « *Les abstractions de communication dans les jeux en réseau mobile* », ont commencé il y a deux mois. Nous nous concentrons sur l'aspect communication réseau, dont les aspects importants sont l'efficacité, la robustesse, la latence faible, et le passage à l'échelle du mécanisme de communication du système dans le contexte des réseaux mobiles. Ce chapitre vise à donner une vue générale sur les obstacles qui empêchent les jeux en ligne d'atteindre les succès comme les jeux sur une machine, ou un réseau local. Un des défis majeurs de ce type d'application a bien été reconnu ; c'est la latence de communication. Les effets issus de la latence peuvent être efficacement réduit d'une part au moyen de mécanismes de synchronisation implémentés dans système présentés ci-dessus, et d'autre part, par une stratégie intelligente de communication. C'est la deuxième approche dont le travail du stage fait partie.

Un mécanisme de communication est intelligent et efficace s'il n'est pas superflu en terme de montant d'information voyageant sur le réseau. Par exemple, il est clairement inutile d'envoyer un événement à un objet si ce dernier ne subit aucune influence[3] de cet événement. La bande passante du réseau peut être sauvegardée en communiquant des données seulement aux objets qu'il influence. Il faut noter que le choix de l'information à envoyer est le travail des concepteurs du jeu. Ce que fournit l'intergiciel sont des mécanismes de synchronisation et de communication réseau efficace (latence faible, fault-tolérance, etc). Le modèle de communication publish/subscribe est abondamment étudié dans ces récentes années, et a été implémenté dans quelques plates-formes. Dans la suite de ce stage nous allons approfondir ce modèle de communication qui nous semble celui qui est le plus convenable pour ces types d'application.

[3]On appelle la région d'influence d'un objet un cercle dont le centre est la position de l'objet, et le largeur du rayon est la distance que l'objet peut avoir des influences sur d'autres

Bibliographie

[1] S. Caltagirone, M. Keys, B. Schlief, M. Jane Willshire. *Architecture for a Massively Multiplayer Online Rôle Playing Game Engine*. Consortium for Computing Sciences in Colleges 2002

[2] D. Saha, S. Sahu, A. Shaikh. *A Service Platform for On-Line Games*. NetGames 2003, USA.

[3] S. Fiedler, M. Wallner, M.Weber. *A Communication Architecture for Massive Multiplayer Games*. NetGames 2002, Germany.

[4] E. Cronin, B. Filstrup, A. Kurc. *A Distributed Multiplayer Game Server System*. IEEE 2001

[5] A. Gérodolle, F. Dang Tran, L. Garcia-Banuelos. *A Middleware Approach to building Large-Scale Open Shared Virtual Worlds*. IEEE 2000.

[6] F. Dang Tran, M. Deslaugiers, A. Gérodolle, L. Hazard, N. Rivierre. *An Open Middleware for Large-Scale Networked Virtual Environments*. IEEE 2002.

[7] F. Fitzek, Gerrit Schulte, M. Reisslein. *System Architecture for Billing of Multi-Player Games in a Wireless Environment using GSM/UMTS and WLAN Service*. NetGames 2002, Germany.

[8] A.E. Saddik, A.Dufour. *Peer-to-Pear Suitability for Collaborative Multiplayer Games*. IEEE 2003.

[9] C. Diot, L. Gautier. *A distributed Architecture for Multiplayer Interactive Applications on the Internet*. IEEE 1999

[10] *Webopedia - The online dictionary and search engine for computer and Internet technology definitions* : http://www.webopedia.com/

[11] E. Cronin, B. Filstrup, A. Kurc. *Quake's technical report*. University of Michigan, 2001. http://warriors.eecs.umich.edu/games/papers/quakefinal.pdf

[12] *Quazal Multiplayer Connectivity* : http://www.quazal.com

[13] E. Cronin, B. Filstrup, A. R. Kurc, S. Jamin. *An Efficient Synchronization Mechanism for Mirrored Game Architectures*. NetGames 2002, Germany.

[14] M. Mauve. *How to Keep a Dead Man from Shooting*.

[15] L. Gautier et C. Diot. *Design and Evaluation of MiMaze, a Multi-Player Game on the Internet*. Proc. IEEE Multimedia (ICMCS'98), Austin, TX, USA, 1998, pp. 233-236.

[16] K. Fujikaway, A. Itohyy, S. Shimojoyyy, et H. Miyahara. *Synchronization Issues on Networked Virtual Environments of Large-Scale Multiparty Applications*. Technical report of IEICE. The Institute of Electronics, Information and Communication Engineers. Japan 2003.

[17] *Middleware FAQ* : http://middleware.internet2.edu/overview/middleware-faq.html

[18] *TerraPlay system* : http://www.terraplay.com/

[19] *ButterFly.net* : http://www.butterfly.net

[20] E. Cariou. *Contribution à un processus de reification d'abstractions de communication*. thèse, Université de Rennes 1, France. 2003.

[21] Y. Liu, B. Plate. *Survey of Publish Subscribe Event Systems*. Computer Science Dept, Indian University.

[22] AR. Bharambe, S. Rao, S. Seshan. *Mercury : A Scalable Publish-Subscribe System for Internet Games*. NetGames2002, Germany 2002.

[23] *Wikipedia, the free encyclopedia* : http://en.wikipedia.org/wiki/Main_Page