Conception et développement d'une infrastructure de communication collaborative

Septimia-Cristina Pop

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

THÈSE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : « Informatique : Systèmes et Communications »

préparée au laboratoire LSR – IMAG
dans le cadre de l'École Doctorale
« Mathématiques, Sciences et Technologies de l'Information »

présentée et soutenue publiquement

par

Septimia-Cristina POP

Le 19 Décembre 2005

Titre :

Conception et développement d'une infrastructure de communication collaborative

---

Directeur de thèse : M. Andrzej DUDA

JURY

M. Jacques MOSSIERE,	Président
M. Guy BERNARD,	Rapporteur
M. Michel RIVEILL,	Rapporteur
M. Andrzej DUDA,	Directeur de thèse

À ma mère. À la mémoire de mon père.

Influencée peut-être par la beauté de la région grenobloise, j'ai toujours imaginé la préparation de la thèse comme une randonnée à la montagne. J'ai commencé l'ascension en 2001... et pendant quelques années je suis montée... et ...

Comme vous le savez déjà, quand on fait une première sortie à la montagne (surtout s'il s'agit de hautes montagnes) c'est impératif d'avoir un guide, un maître. J'ai eu la chance d'avoir un guide remarquable : par son enthousiasme il m'a aidée à découvrir des parties fascinantes de la montagne, à sentir le goût de la recherche. Avec son style personnel, il m'a incité à apprendre à m'orienter et à découvrir de possibles sentiers pour la montée. J'aimerais, donc, tout d'abord remercier M. Andrzej Duda pour l'encadrement de ma thèse, pour ses qualités professionnelles, pour ses conseils et le soutien qu'il a bien voulu m'accorder pendant toutes ces années de thèse.

... en fin, maintenant je suis arrivée au sommet de la montagne !

M. Guy Bernard et M. Michel Riveill, rapporteurs de ma thèse, ont accepté de m'accompagner sur les derniers mètres de cette montée, l'évaluant et confirmant les efforts par un titre de docteur. Je leur adresse mes remerciements pour avoir accepté d'évaluer ma thèse. J'adresse mes remerciements également à M. Jaques Mossière qui m'a fait l'honneur de présider le jury.

Mais je devrais commencer avec le début... Alors, en printemps 2001 j'ai commencé l'ascension pleine d'enthousiasme, envoûtée par le paysage, par la perspective de la découverte de nouveaux horizons... Si au début le paysage paraissait mystérieux et attractif, au fur et à mesure que j'avançais le paysage changeait, il devenait parfois effrayant et le chemin était de temps en temps très abrupt, et même il disparaissait quelquefois... Il ne sert à rien d'insister ni sur mon enthousiasme quand je découvrais des clairières, ni sur les moments de désespoir quant je ne voyais plus le ciel... Ces sont des moments "uniques", spécifiques à de telles ascensions (tous les thésards savent de quoi je parle...).

En plus de ces moments spécifiques vécus sur cette montagne, j'ai eu la chance de découvrir des inestimables trésors : des relations d'amitié et/ou de camaraderie avec des personnes remarquables de cultures très différentes.

C'est le bon moment de remercier mes collègues d'équipe pour leur disponibilité et les bons moments passés ensemble. Merci à tous les chercheurs ou enseignants chercheurs : Dominique, Franck, Gilles (spécialement pour sa patience pour mon français), Jacques, Martin, Pascal. Merci également à mes "collègues de randonnée", notamment à Pawel (je lui accorde la médaille "super collègue de bureau"), Sorin (la médaille "super collègue d'équipe"), Sonia (une médaille spéciale), Vincent (une médaille d'or pour ses qualités humaines remarquables), Yan (une médaille pour sa bonne humeur et... humour), Paul (médaille spéciale pour nos discussions ... interminables et très intéressantes), Mohammad (médaille pour sa gentillesse), Hoang, Justinian... Merci aussi à

Binh pour sa collaboration. Muchas gracias, Polo, por tu valiosa amistad ! Je remercie aussi les autres collègues et personnes très sympathiques du laboratoire avec qui j'ai eu la chance de collaborer ou simplement d'avoir passé des bons moments (je ne peux pas continuer sans mentionner Pascale, Martine, Christiane, François...). Je profite de l'occasion pour remercier également Luciano et Severine, Khalid, Trinh, Quynh, Pawel (oui, encore lui...) et Monika, Gennaro, Maria, "the last but not the least" Giang, pour les moments réconfortants passés ensemble.

Maintenant, je suis arrivée au but. Mais, je n'y serais pas arrivée, si je n'avais pas découvert cette montagne.

Je tiens à remercier, donc, toutes les personnes qui m'ont déterminé à trouver le chemin jusqu'à la cime. Parmi ces personnes, j'aimerais mentionner Mme Maria Necsulescu : elle m'a fait découvrir l'univers poétique (et... sans la poésie, la vie est sèche) ; M. Ioan Purdea qui m'a initiée au rationnement et à la rigueur des mathématiques ; M. Sever Groze qui a vu en moi un bon informaticien (a-t-il eu raison ? !...). Je mentionne également M. Adalbert Orban et M. Florian Boian. Il y a encore de nombreuses personnes extraordinaires, des professeurs que j'ai eu la chance de rencontrer pendant mon cursus scolaire et universitaire : je leur porte une vive reconnaissance à tous.

De la cime de cette montagne, je regarde en arrière avec sérénité. Longue, longue randonnée...

Je partage ces moments de bonheur tranquille avec ceux avec qui j'ai partagé des bons et mauvais moments pendant cette ascension, avec mon "groupe" : Ioana, Lidia, Daniela et Guy (qui m'a beaucoup aidé à améliorer mon français), Valentina et Guillaume. Un grand, grand merci à vous tous. Un grand merci également à Tudor, un dévoué ami qui m'a soutenu pendant tout ce temps. Je tiens à mentionner une personne (qui malheureusement nous a quittés) avec qui j'ai partagé aussi une sincère amitié : Marta Kerekes. Un gînd bun pentru tine, Marti scumpã.

Mes remerciements les plus chaleureux à mes cousins Beatrice et Marius pour leur sincère et profonde amitié.

Comment pourrais-je exprimer la profonde gratitude que je porte à mes parents ? !... Au moins, par un simple geste de coeur... Je dédie cette thèse à ma mère et à la mémoire de mon père.

Eh bien, je suis arrivée au sommet... Je regarde autour de moi pour découvrir d'autres vallées et sommets à conquérir...

Cristina

Table des matières

Chapitre 1

Introduction

Le domaine de réseaux de communication évolue rapidement depuis l'apparition de la technologie sans fil de type 802.11 (*Wi-Fi*). Les points d'accès qui offrent la connectivité avec l'Internet global sont actuellement largement déployés, on les trouve par exemple dans des hôtels et des bars, sur des campus universitaires, dans les salles d'attente des aéroports ou des gares. Ceci favorise le nomadisme des utilisateurs disposant d'ordinateurs portables ou d'équipements de type PDA, et facilite l'avènement des *environnements spontanés de communication*. Une communication facile et omniprésente ouvre une nouvelle voie à la collaboration personnelle accrue. Notre thèse s'intègre dans ce contexte-là.

1.1 Objectif de la thèse

Plus particulièrement, notre thèse se concentre sur l'étude des infrastructures pour le partage d'affichage d'applications dans le contexte d'un environnement spontané de communication. L'idée est de pouvoir fournir à des utilisateurs des fonctionnalités de partage d'écran afin d'intégrer facilement des applications existantes dans l'environnement spontané de communication. Nous avons choisi comme base de notre travail le système de fenêtrage X (X Window System) [57] et le système d'affichage à distance VNC (Virtual Network Computing) [67]. Notre travail suit l'idée d'ajouter à ces systèmes le support nécessaire pour les transformer en systèmes coopératifs adaptables à une variété de contextes offerts par un environnement spontané de communication.

1.2 Contributions

Les principales contributions apportées au système X peuvent se résumer ainsi :

Multiplexeur X comme service actif. Notre système coopératif fournit un multiplexeur de flots X conçu comme un service actif qui peut être chargé dynamiquement sur une passerelle située dans le réseau. Ce service offre la possibilité de multiplier des flots X, ce qui transforme le système X en un système coopératif de partage d'affichage d'application.

Système de partage d'applications X plus flexible. Nous avons étudié les propriétés du système X et nous avons intégré d'une façon transparente une interface graphique de contrôle d'entrée (souris, clavier) qui le rend plus flexible. Cette interface est fournie par un client X qui ajoute sa fenêtre à la fenêtre principale de l'application partagée, ce qui constitue la base d'implémentation pour d'autres fonctionnalités adaptées au contexte de travail dans un environnement spontané de communication.

La deuxième partie de travail concerne le système VNC. Il manque de flexibilité en ce qui concerne le support coopératif et l'adaptation à la variété des contextes de collaboration spécifiques à un environnement spontané de communication. Nous l'avons amélioré principalement par un support coopératif au niveau de flots de données. Nous identifions trois caractéristiques principales de notre contribution :

Un *proxy VNC* contrôlé dynamiquement à distance. Le support coopératif est construit autour d'un *proxy* nommé *Newkey*) intégré de façon transparente dans le système VNC et contrôlé dynamiquement à distance. Le contrôle est possible grâce au *protocole de contrôle NewkeyCP* qui permet d'organiser et de gérer des sessions de collaboration basées sur le partage de l'écran et de fournir le contrôle d'entrée adéquat.

Traitement des flots RFB. *Newkey* peut être programmé pour gérer différentes sessions de collaboration en traitant de manière spécifique un flot RFB (*Remote FrameBuffer*), par exemple sa multiplication. L'initialisation de la communication peut se faire via un serveur, mais également via des clients VNC.

L'ouverture vers des politiques de contrôle d'entrée. Grâce à deux primitives du protocole de contrôle *NewkeyCP*, il est possible de fournir au *Newkey* le protocole de contrôle d'entrée adéquat au contexte de collaboration. Dans notre travail, nous proposons également une interface graphique qui donne l'accès et fournit des informations nécessaires à un utilisateur sur le mécanisme de contrôle d'entrée. Cette interface est fournie par un serveur VNC dédié qui s'affiche sur l'écran de l'utilisateur et communique avec le service qui gère le contrôle d'entrée.

1.3 Organisation du mémoire de thèse.

La suite du mémoire est organisée en deux grandes parties qui présentent nos contributions.

La première partie est dédiée à la présentation du système coopératif de partage d'applications X. Elle commence par le chapitre 2 avec une présentation du système X, suivie de l'état de l'art sur des solutions existantes de partage d'applications X. Le chapitre 3 inclut la description de notre proposition du système de partage d'applications X. Nous présentons un scénario qui illustre le système coopératif, validé ensuite par un premier prototype. Cette partie inclut également la présentation de la passerelle active choisie pour implémenter le multiplexeur X en tant que service actif. La description du prototype inclut également la présentation du client X spécialisé qui permet la synchronisation du droit à la parole (*floor control*).

La deuxième partie du mémoire est consacrée au support coopératif pour le système VNC. Le chapitre 5 présente l'anatomie du système d'affichage à distance VNC et analyse ses caractéristiques coopératives. Dans le chapitre suivant, nous décrivons notre support coopératif ajouté au système VNC. La description du support coopératif commence par la présentation de ses éléments. Celle-ci inclut des nombreux scénarios d'utilisation qui illustrent les fonctionnalités du support. Les prototypes, qui mettent en pratique ces éléments, sont présentés par la suite.

Finalement nous concluons sur ce travail dans le chapitre 7. Un bilan de travail est présenté et des perspectives d'études futures sont exposées. Une annexe donne des informations complémentaires utiles pour la meilleure compréhension de notre exposé.

Chapitre 2

Le partage d'applications X

2.1 Le système de fenêtrage X

Le principe d'un système de fenêtrage est de gérer l'écran physique en plusieurs zones logiques appelées fenêtres. Chaque fenêtre agit comme un dispositif d'entrée-sortie séparé sous le contrôle de différentes applications.

Le Système de Fenêtrage X[1] est un standard de-facto du système d'exploitation Unix. Son idée de base est de permettre à des applications qui se exécutent sur différentes machines de gérer une fenêtre à distance. Ainsi, le système X a été conçu comme un système distribué.

Dans cette section nous présentons l'architecture de X, le protocole d'échange entre un serveur et un client X et nous expliquons quelques éléments de base du système X.

2.1.1 Architecture

Avant d'introduire l'architecture du système X, nous clarifions quelques éléments qui apparaissent dans sa présentation :

- un dispositif d'affichage (*display*) de type bitmap est un dispositif de sortie (*output device*) où chaque pixel affiché sur l'écran du moniteur (*monitor*) correspond directement à plusieurs bits dans la mémoire vidéo de la machine ;
- un écran (*screen*) est la portion d'un moniteur qui affiche des informations sous forme de texte ou graphique.

L'architecture générale du système est représentée dans la Figure 2.1.1. Il s'agit d'une architecture client-serveur :

[1] Il s'agit du Système de Fenêtrage X, la version 11. Dans le cadre de ce document, nous l'appelons aussi (le Système) *X*.

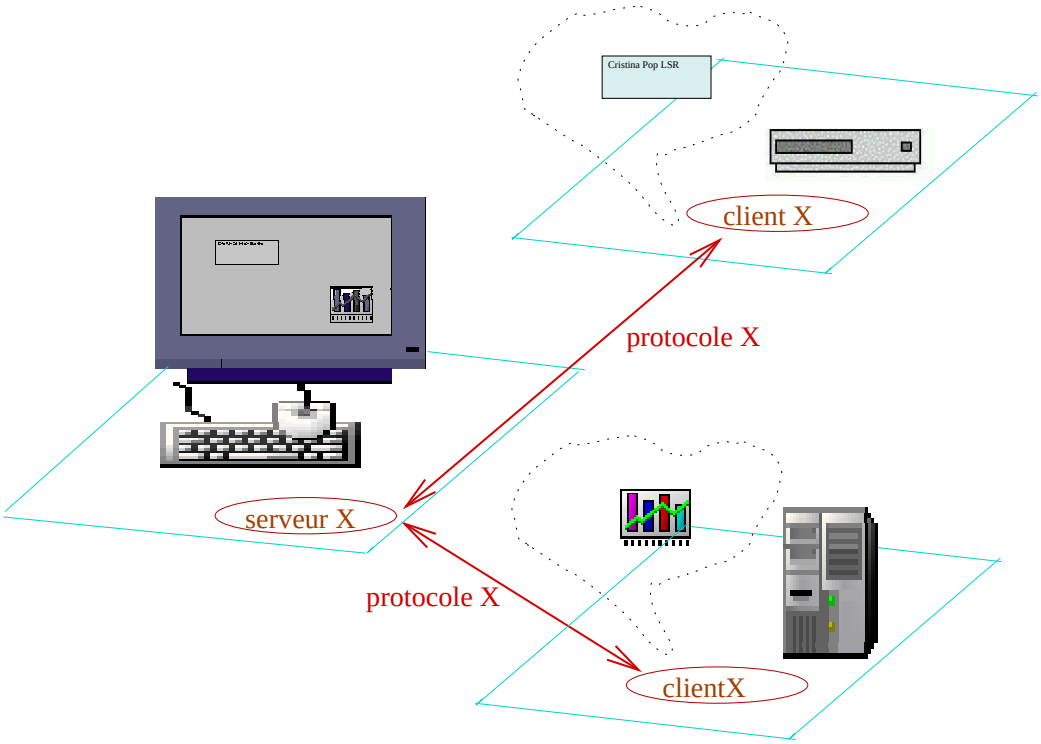

FIG. 2.1 – Architecture du Système X. Les clients X, qui peuvent se trouver sur des machines différentes, ont besoin du serveur pour afficher leurs interfaces graphiques et gérer les dispositifs d'entrée.

– *le client* est une application qui a besoin de gérer des fenêtres à distance et d'avoir accès à des dispositifs d'entrée. Ainsi, dans le cadre du système le client prend en charge la partie concernant les opérations graphiques et les événements d'entrée. Les opérations graphiques peuvent être très variées[2], en conséquence le client peut avoir un degré de complexité très grand. Quelques exemples de clients X classiques sont : xterm, xemacs, xclock.
– *le serveur* est un programme qui gère le dispositif d'affichage (*display*) de type bitmap, le clavier et la souris d'une machine. Un display peut avoir plusieurs écrans (*screens*)[3]. Le serveur offre des services graphiques aux clients pour n'importe quel nombre d'écrans ainsi que l'accès au clavier et à la souris. Il peut gérer plusieurs clients dans le même temps. Il gère les caractéristiques de système d'exploitation et du matériel informatique.

2.1.2 Le protocole de communication X

La communication entre un client et un serveur X s'effectue selon le protocole de communication X [55]. Il est complexe : il comporte plus de 150 types de messages qui ont une sémantique élaborée.

Dans la phase initiale d'une session de communication X, le client contacte le serveur et lui envoie des informations concernant ses caractéristiques[4]. Si le serveur accepte la communication, sa réponse donne les caractéristiques de l'écran (taille, profondeur, couleurs), les intervalles d'identificateurs de ressources (*Id*) allouées à ce client et d'autres informations de protocole.

Ensuite, la communication fait appel à quatre types de messages : des Requêtes, des Réponses, des Évènements et des Erreurs. Selon le protocole, le client transmet au serveur des Requêtes. Quand le serveur reçoit une Requête, il vérifie sa validité. Il répond par un message d'Erreur s'il ne peut pas l'accomplir ou il l'exécute et pour certaines Requêtes renvoie une Réponse (et/ou parfois un Évènement). L'entrée du clavier et de la souris se traduit par des Évènements qui peuvent être transmis au client. Dans la Figure 2.7 nous esquissons un échange des messages entre le client et le serveur.

Du point de vue de la structure, un message X est un bloc des données composé d'un code d'opération (*opcode*) et des paramètres de longueur fixe ou variable. Nous décrivons ici chaque type de message :
– *Requêtes*. Dans le protocole X on peut distinguer deux grands types de Requêtes : graphiques et de gestion. En principe, celles de gestion servent à l'administration des

[2]par exemple, à cause du besoin d'utilisation d'un grand nombre de fenêtres.
[3]Dans la terminologie du système X, le *display* signifie le serveur avec des écrans et des dispositifs d'entrées [55].
[4]Par exemple, la version du protocole.

ressources et des atomes[5]. Les opérations graphiques sont de niveau très bas : il y a des fonctions pour dessiner des points, des lignes, des segments, des rectangles, des arcs, toutes appliquées à un ensemble d'objets. Il y a aussi des fonctions de base pour manipuler des images et du texte, et pour réaliser des opérations sur des périmètres d'écran ou des opérations de remplissage de figures élémentaires. Le protocole spécifie avec précision des points graphiques qui sont impliqués dans chaque opération graphique. La structure générale d'un message de Requête X est présentée dans la Figure 2.2.

FIG. 2.2 – Structure générale d'un message de Requête dans le Protocole X.

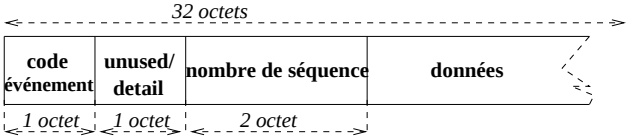

FIG. 2.3 – Structure générale d'un message d'évènement X.

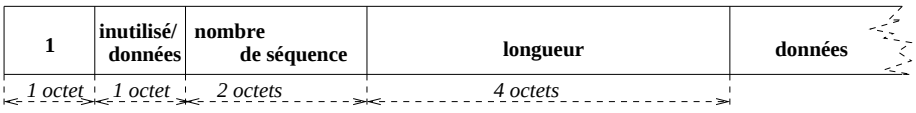

FIG. 2.4 – Structure d'une réponse X.

- *Évènements.* Comme nous avons précisé, les Évènements peuvent être générés par des actions de l'utilisateur qui utilise des dispositifs standard d'entrée. Ce type de message peut être aussi un effet secondaire d'exécution d'une Requête. Pour chaque fenêtre, le client précise quels sont les Évènements pertinents. Le serveur envoie seulement des Évènements demandés d'une façon asynchrone. D'autre part, un client peut envoyer des Évènements à d'autres clients. Dans la Figure 2.3 on peut voir la structure générale d'un message Évènement du protocole X.
- *Réponses.* Certaines Requêtes demandent une Réponse, mais en général le client n'arrête pas d'envoyer des Requêtes pour attendre une Réponse. Le format général de ce type de message est présenté dans la Figure 2.4.
- *Erreurs.* En général, une Erreur indique une tentative d'utiliser une ressource qui n'est pas identifiée ou si des règles du protocole ne sont pas respectées. Sa structure générale est présentée dans la Figure 2.5.

[5]Un atome est un identificateur d'une chaîne d'octets. Nous revenons sur les atomes dans la Section 2.1.3.

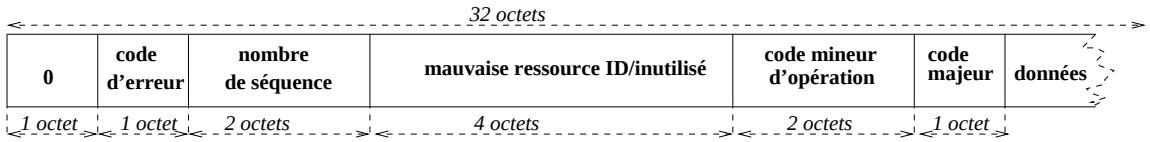

FIG. 2.5 – Structure d'un message d'erreur.

Le protocole X est un protocole asynchrone (propriété illustrée dans la Figure 2.7, Section 2.1.6.). Les messages envoyés peuvent être en transition dans les deux directions simultanément. Le client n'est pas obligé d'attendre pour que le serveur exécute une Requête avant d'en générer une autre : les messages peuvent être envoyés en cascade. Le serveur ne confirme ni la réception, ni l'exécution de Requêtes. De toutes façons, chaque message envoyé par le serveur inclut un numéro de séquence (*sequence number*) qui désigne la Requête du client à qui correspond ce message ou qui a été la dernière à être traitée. Ces caractéristiques du protocole demandent que la communication s'effectue sur une couche de transport qui fournit une livraison fiable et en ordre des données dans les deux sens (*duplex byte stream*). Pour cette raison, X utilise le protocole de transport TCP. Quand le client et le serveur se trouvent sur la même machine, pour gagner en performance,la communication entre eux est habituellement implémentée en utilisant une mémoire partagée.

2.1.3 Ressources

Les ressources sont des objets abstraits créés et administrés par le serveur sous contrôle d'un client. Quand c'est possible, le serveur permet leur partage entre plusieurs clients. Dans les messages X, les ressources sont identifiées par des identificateurs *(ID)*, des nombres représentés sur 32 bits générés par le client à la création des ressources. Un identificateur d'une ressource est unique dans le domaine de ressources gérées par le serveur. L'unicité est assurée par le serveur qui, au début de la communication, envoie au client la plage qui lui est réservée pour générer des identificateurs. En plus, un identificateur est public : n'importe quel client qui a l'accès au serveur peut l'utiliser. La durée de vie d'une ressource dépend de la durée de vie de la connexion sur laquelle elle a été créée.

Dans le système X on considère les ressources suivantes : la carte de pixels (*pixmap*), le *curseur* de la souris, la police (*font*), le contexte graphique (*gcontext*), la carte de couleurs (*colormap*), la fenêtre. Nous allons décrire les plus intéressantes d'entre elles pour notre travail :

– *La carte de couleurs.* L'écran est vu de manière bidimensionnelle avec une valeur de pixel de N bits stockée à chaque coordonnée[6]. Le nombre de bits dans un pixel et la

[6] *N* représente le nombre de plans d'un bit de la structure du mémoire vidéo, nommé aussi la *profondeur* de la mémoire.

façon dont une valeur est associée à une couleur dépendent du matériel informatique. Le système X cache ce type de dépendances physiques sous la forme d'une abstraction qui s'appelle "visuel". X supporte six types de visuels (*visual*)[7]. Une carte de couleurs (*colormap*) est un ensemble d'entrées qui définissent des valeurs de couleurs. Elle est utilisé pour associer une valeur de pixel à une couleur. La Figure 2.6 présente le mécanisme de représentation d'un pixel de la mémoire vidéo sur l'écran qui implique l'utilisation de *colormap*.

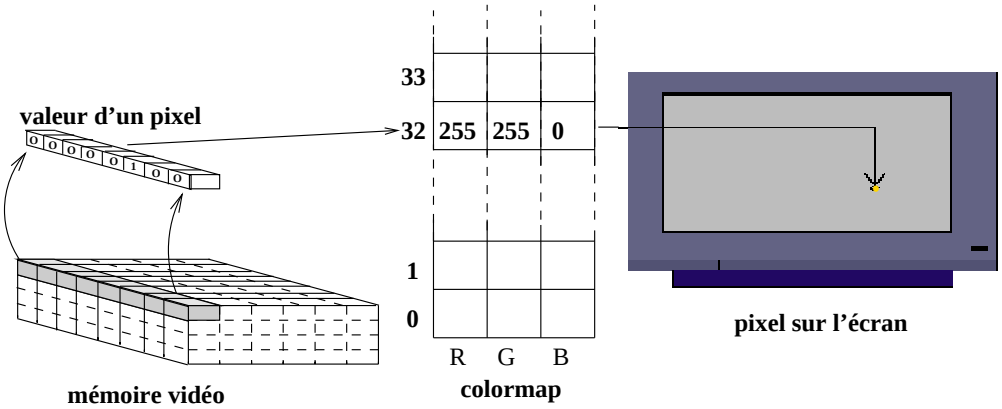

FIG. 2.6 – Représentation d'un pixel sur l'écran. La figure illustre la classe visuelle PseudoColor où la valeur d'un pixel est un index dans la *colormap*. Sur l'écran, le point graphique a la couleur qui se trouve à cette position dans la *colormap*. Pour un autre type de visuel, DirectColor, la valeur d'un pixel est décomposée dans des champs Rouge-Vert-Bleu (*RGB*) et chaque valeur de champ représente séparément un index dans la *colormap* correspondante.

- *Pixmaps*. Une *pixmap* est une région de mémoire vidéo hors écran sur laquelle le client peut faire des opérations graphiques. Pour cette raison, les *pixmaps* et les fenêtres, en général, sont nommées des planches (*drawables*). Le contenu d'une *pixmap* n'est pas directement visible. Cette ressource est utilisée pour remplir, par exemple, une bordure ou le contenu d'une fenêtre. Une *pixmap* de profondeur 1 s'appelle une *bitmap*.
- *Fenêtres*. Dans le système X, l'interface graphique d'une application est conçue comme une hiérarchie arborescente de fenêtres classiquement rectangulaires. Même un bouton dans l'interface graphique d'une application est une fenêtre. La fenêtre-racine couvre tout l'écran. Ainsi la fenêtre est la ressource la plus utilisée par des applications. À chaque fenêtre sont attachés des attributs (*attributes*), comme par exemple, le pixel ou la *pixmap* qui est utilisé pour le fond ou pour la bordure de la fenêtre. À une fenêtre sont attachées aussi des structures de données nommées propriétés (*properties*). Elles sont utilisées dans la communication avec le gestionnaire de fenêtres et avec d'autres clients. Le client doit retenir toutes les informations sur les fenêtres. Une portion de contenu d'une fenêtre est redessinée par le client à la réception d'un évènement *Expose*.
- *Polices*. X supporte des polices de type bitmap (*bitmap font*).[8] Le système permet la

[7]Pour plus de détails, voir [57] [27].

[8] La police de type bitmap est une police où chaque caractère est mémorisée comme un tableau de pixels.

représentation des polices spécifiques du serveur. Quand le client veut créer ce type de ressource, il demande au serveur la police désirée en utilisant son nom. Le serveur peut la charger à partir d'un répertoire bien précis ou d'un serveur de polices.
– *Le contexte graphique.* Le contexte graphique (*gcontext*) est un ensemble de paramètres factorisant tous les modes des fonctions graphiques. Il inclue des informations comme par exemple, le pixel de premier plan (*foreground*), le pixel de fond (*background*), la *pixmap* utilisée comme pochoir (*stipple*). Un contexte graphique est utilisé par plusieurs requêtes graphiques pour transmettre au serveur des paramètres d'une façon compacte.

La Section A de l'Annexe présente l'échange de messages entre le client et le serveur dans le cas d'une application simple de type "Hello World"[9] et la figure 2.7 (Section 2.1.6) illustre l'utilisation des ressources X pour la première partie de cette communication X.

2.1.4 Atomes

Un atome (*atom*) est un identificateur unique correspondant à un nom. Des atomes sont utilisés pour identifier, par exemple, des propriétés et des types. Un client crée des atomes en utilisant la Requête "InternAtom" qui contient la chaîne d'octets[10] qui doit être enregistrée sur le serveur. Il reçoit comme réponse l'identificateur. Cette chaîne en suite est référée par son atome pendant toute la durée de vie du serveur. Pour réduire l'utilisation excessive de cette requête, il y a une liste des atomes prédéfinis [27]. Les atomes sont utilisés principalement dans la communication inter-clients. Les règles de cette communication sont précisées dans le Manuel des Conventions de Communication Inter-Client (*Inter-Client Communication Conventions Manual*)(ICCCM) [54].

2.1.5 Symboles de clavier

Quand l'utilisateur appuie ou relâche une touche (*key*) de clavier, un évènement est envoyé au client. Cet évènement contient le code de clavier correspondant à cette touche et l'état de touches-modificateur (Shift, Lock, Control ou Meta). Il appartient au client de traduire le code de clavier en caractères correspondants. Dans ce but, le serveur détient et fournit au client la liste de symboles (*keysyms*) qui correspondent à chaque touche d'un type de clavier.

2.1.6 Extensions du protocole X

Un des critères de conception du système X a été la possibilité d'extension du protocole X [57]. Une extension est un ensemble de requêtes, d'évènements et d'erreurs possibles. Les

[9] Il s'agit d'une application qui affiche dans une fenêtre le message "Hello World !".
[10] En général, cette chaîne d'octets est interprétée comme une chaîne de caractères (*string*).

clients peuvent en bénéficier si le serveur les supporte. L'extension la plus utilisée est MIT-SHM qui permet l'utilisation de la mémoire partagée, si le client et le serveur se trouvent sur la même machine. D'autres extensions enrichissent la fonctionnalité du protocole X, par exemple, l'acceptation des fenêtres non-rectangulaires ou plus de dispositifs d'entrée.

La Figure 2.7 illustre un échange de messages X, parmi lesquels une demande d'une extension.

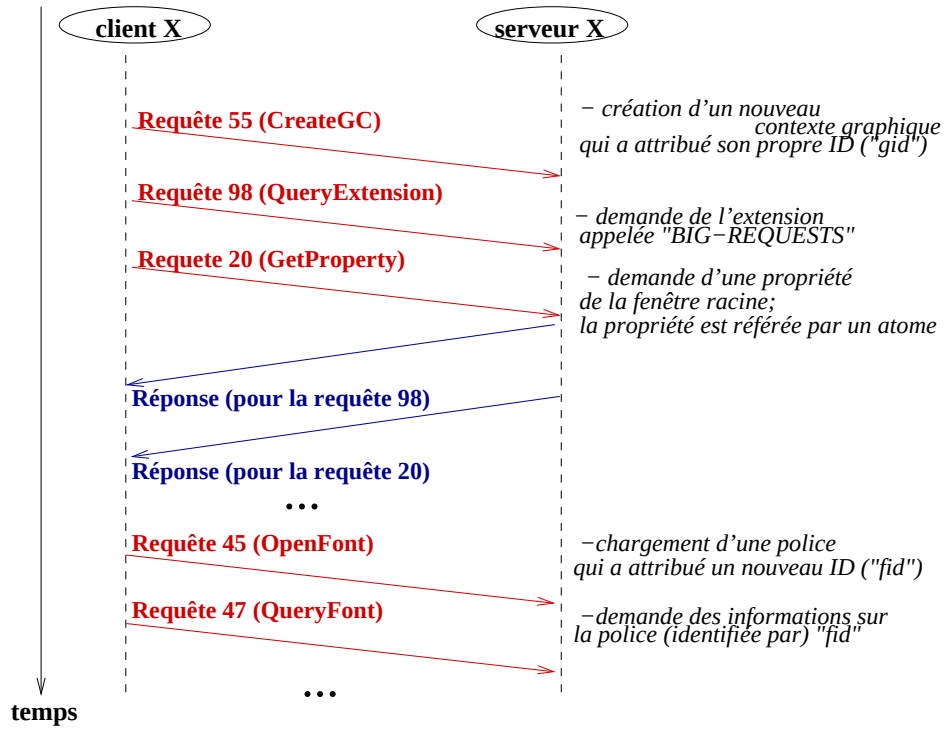

FIG. 2.7 – Exemple d'un échange selon le protocole X, pris du début de la communication entre un client de type "Hello World" et un serveur X (voir la Section A de l'Annexe). On peut remarquer le fonctionnement asynchrone du protocole X.

2.1.7 Gestionnaire de fenêtres

X sépare la politique de gestion de fenêtres des mécanismes de base du système de fenêtrage [27]. Cette séparation repose sur un client X ayant un rôle particulier : le gestionnaire de fenêtres (*window manager*). Celui-ci :

- gère l'aspect des fenêtres : il décore les fenêtres principales par divers éléments graphiques, comme par exemple une bordure, une barre de titre ;
- fournit des moyens par lesquels l'utilisateur peut les manipuler, par exemple, les déplacer, les superposer sur des fenêtres des autres applications, les redimensionner.

Son fonctionnement se base principalement sur la possibilité pour un client d'avoir l'accès aux ressources des autres clients. L'application peut préciser, en utilisant des propriétés, ses conseils (*hints*) concernant l'affichage d'une fenêtre, mais le gestionnaire peut les ignorer. Le gestionnaire de fenêtre peut avoir des fonctionnalités très complexes et former la base d'un Environnement Graphique de Bureau (*Graphical Desktop Environment*)[11]. Comme exemple des gestionnaires de fenêtres, nous pouvons mentionner twm, gwm, olwm, fvwm.

2.1.8 Sécurité

Une fois connecté, un client a l'accès à toutes les ressources du serveur, donc un contrôle d'accès au serveur s'impose. Le Système X permet l'utilisation de divers protocoles d'authentification : au début de la communication, le client peut préciser quel protocole il préfère, mais il faut encore que le serveur l'implémente. Actuellement, il y a deux mécanismes de contrôle d'accès qui sont disponibles sur tous les serveurs :

- *authentification d'hôte* (*host authentication*) : l'acceptation de la connexion d'un client se fait sur la base de l'identification de l'hôte origine de la communication. La gestion de la liste des hôtes qui ont accès au serveur peut se faire en utilisant le programme *xhost*. L'inconvénient de ce mécanisme est que souvent l'hôte du client supporte plusieurs utilisateurs en même temps.
- *authentication par jeton* (connu comme *MIT-MAGIC-COOKIE*) : la vérification de chaque client se fait par le jeton que celui-ci offre au serveur. En utilisant le programme *xauth* le client obtient un *magic-cookie* (le jeton) qui lui donne le droit d'accès au serveur X. La première forme d'authentification par l'hôte est prioritaire à celle effectuée par le client.

La sécurité de la transmission des données n'est pas spécifiée dans le protocole. Une solution à ce problème est offerte par ssh [9]. La connexion à distance par ssh permet l'acheminement sécurisée des sessions X. La Figure 2.8 illustre le mécanisme. En plus de la sécurité de transport à travers un tunnel chiffré, ssh offre un niveau d'authentification meilleur que l'authentification standard de X.

2.1.9 Interface de programmation X

Pour faciliter la programmation d'applications en X, le système offre une interface de programmation (API) sous forme d'une bibliothèque (Xlib). Des boites à outils (*toolkits*), comme par exemple Xt, Motif, Qt, Gtk+ fournissent des abstractions de plus haut niveau permettant une programmation plus facile en X.

[11] Un Environment Graphique de Bureau dans le Système X consiste en un gestionnaire de fenêtre, un ensemble de thèmes de style graphique, des programmes et bibliothèques pour la gestion du bureau (*desktop*). Les environnements graphiques de bureau les plus connus sont KDE, Gnome, CDE, OpenLook.

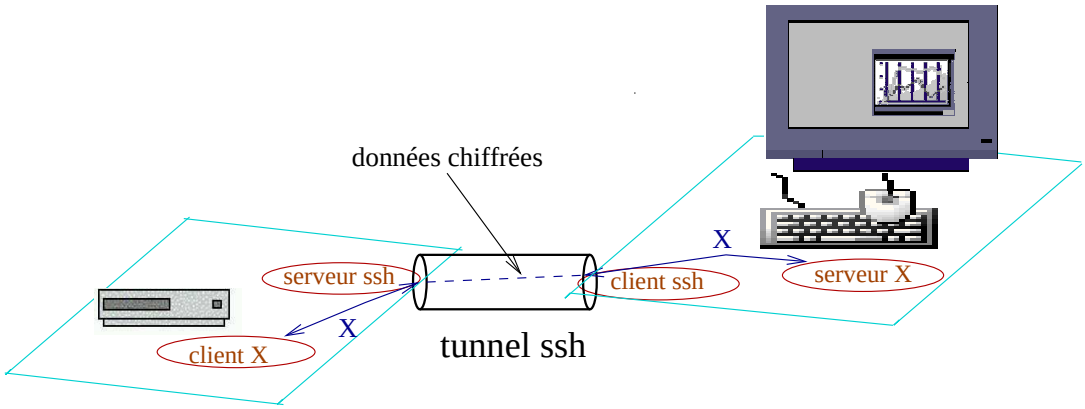

FIG. 2.8 – Acheminement d'une session X par ssh.

Étant donnés tous les éléments de X, la Figure 2.9 présente une vision globale de la structure du système.

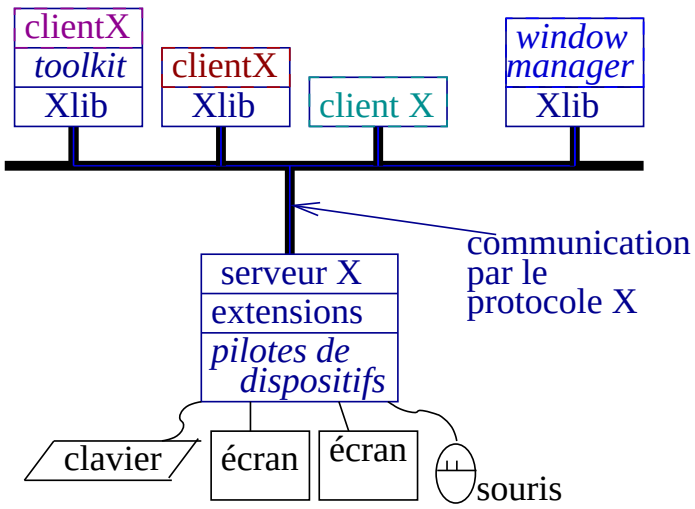

FIG. 2.9 – Structure du système X (dessin adapté du [57]). Les applications et le serveur X peuvent se trouver sur la même machines ou sur des machines différentes.

2.1.10 Comportement du système en réseau

Le système X a été conçu pour fonctionner sur un réseau local offrant un débit de 10 Mb/s, comme Ethernet apparu dans les années 80. Plus tard, il s'est posé la question de l'utilisation du système X sur des réseaux à grande distance (WAN, *Wide Area Network*) caractérisés par une latence importante. La latence a une grande influence sur le temps d'initialisation de l'affichage d'une application. En plus, la plupart des applications X étant interactives, le

délai a une importance majeure : l'utilisateur s'attend à voir immédiatement le résultat de ses actions sur l'écran. En conséquence, le protocole dans la version actuelle[12] est conçu pour éviter l'effet d'une latence importante sur les performances des applications. Le protocole réduit le volume de messages en utilisant des éléments comme le contexte graphique ou des atomes pour identifier des chaînes d'octets. Ainsi, la Requête la plus courte a seulement quatre octets. D'autres fonctionnalités du protocole minimisent le trafic sur le réseau : l'allocation des IDs par le client, l'existence des atomes prédéfinis, l'"abonnement" du client aux événements pertinents. En plus, son caractère asynchrone évite les délais imposés par la nécessité d'attente des réponses du serveur. Malheureusement, préoccupés probablement par le compromis entre la bande passante disponible et la puissance de calcul des ordinateurs [44], ou peut-être pour éviter d'imposer un algorithme de compression qui pourrait devenir dépassé plus tard, les créateurs de X ont laissé le transport des images non compressées. C'est pour cette raison que les Requêtes qui impliquent des images (par exemple *GetImage* et *PutImage*) sont des grands consommateurs de la bande passante [45].

Malgré sa conception soignée, le système X génère un trafic important[13]. Ceci est causé par l'évolution du système[14] : les interfaces graphiques des utilisateurs et les applications X sont devenues de plus en plus complexes. Beaucoup de concepteurs de *toolkits* et d'applications X ont négligé à prendre en compte l'aspect communication. Par exemple, des applications peuvent avoir un comportement anormal en demandant des informations rédondantes[15]. Même la bibliothèque Xlib inclue des fonctions synchrones qui introduisent un délai de plus dans la communication par l'attente de la réponse du serveur [45].

Les points faibles du système ont été analysés par des spécialistes qui ont proposé des améliorations[16] dans la perspective d'évolution des réseaux et de la technologie informatique en général [25]. Pourtant, une meilleure conception des applications X s'impose, principalement par l'introduction d'un système de cache à ce niveau-là pour garder des informations nécessaires ultérieurement et par l'évitement des fonctions synchrones inclues dans l'outil de développement utilisé [45].

Simultanément, il y a des efforts pour faire fonctionner le système X dans des réseaux caractérisés par la bande passante limitée et une grande latence. LBX (Low Bandwidth X) est une tentative allant dans cette direction [23]. Fondamentalement, LBX est un schéma de compression adapté à X et un système de cache conçu pour minimiser le trafic de X. Cette solution est basée sur une extension du protocole X. LBX peut être utilisé avec n'importe quelle application, car son architecture inclu un *proxy* du côté de client, qui transforme le

[12] Il s'agit de la version 11.

[13] Par exemple, une session X normale qui implique la navigation sur Internet génère des centaines de megaoctets de données du protocole ("A normal X session, browsing the Internet or accessing common desktop applications, generates hundreds of megabytes of protocol data ") [42].

[14] Dernièrement le système X est utilisé plutôt comme un système d'affichage local. Par exemple, *gdm* (*GNOME display manager*), est configuré implicitement pour exclure la communication X en réseau : le serveur X n'ouvre pas le port TCP pour attendre des connexions.

[15] L'exemple le plus représentatif est la demande répétitive du même atome.

[16] Par exemple, la gestion de police du côté du client. Des tests ont montré que la gestion de police du côté de serveur demande 40 messages aller-retour en plus pour lister les polices et retrouver leurs métriques [45].

protocole X régulier en codage LBX.

Dans le contexte de réseaux caractérisés par une latence élevée, la solution proposée par LBX ne s'avère pas plus efficace que l'utilisation de ssh avec l'option de compression. Étant donné que ssh offre en plus des solutions en ce qui concerne l'authentification et la sécurité de communication, celui-ci est préféré à LBX [45].

NoMachine NX [41] a comme but la réduction de la bande passante nécessaire au protocole X. Essentiellement, cette solution opère à trois niveaux [41] :
- elle compresse le trafic réseau par plusieurs moyens qui incluent des algorithmes différentiels par message, des méthodes avancées de cache, une compression des images avec ou sans des pertes ;
- elle réduit les échanges de message en maximisant le débit ;
- elle adapte la bande passante en temps réel en fonction du contexte réseau.

Son architecture comporte principalement deux *proxy*, un à côté de chaque composant de X qui communiquent à travers ssh. Ainsi, il bénéficie du double avantage de la sécurité de transport et d'authentification de ssh.

À la différence de LBX, cette solution est adaptée à tous les types de réseau, sans pénaliser la communication. En plus de ses avantages, NX ouvre les portes de X aux autres systèmes, comme RDP (Remote Desktop Protocol) [48] et VNC (Virtual Network Computing) [51] : un *proxy* NX peut jouer le rôle de "l'interprète" entre X et des clients/serveurs des autres protocoles.

Dans la perspective d'un réseau ubiquitaire et pervasif, quelques chercheurs ont fait des expérience avec le système X au dessus des réseaux locaux sans fil en utilisant des petits dispositifs à stylo [33]. Ces dispositifs communiquent entre eux par des ondes radio sur un rayon de 3 à 4 mètres. L'expérience a conduit à la conclusion que dans sa forme présente, le système X n'est pas adapté à ce type d'environnement sans fils. D'abord à cause des événements d'entrée générés par le mouvement du stylo ou par des commandes. Ces événements sont très nombreux et petits, ce qui engendre un trafic de paquets IP "inondant" le réseau de communication. En plus, des erreurs de communication peuvent diminuer la bande passante et augmenter la latence qui rend les dispositifs pratiquement inutilisables.

2.1.11 Conclusions

Nous avons présenté dans cette section le Système de Fenêtrage X. Les principales caractéristiques du système sont les suivantes :

- *l'indépendance du client et du dispositif d'affichage* : l'architecture et la plate-forme du dispositif sont complètement transparentes à l'application. Comme nous l'avons précisé, le client n'est informé de quelques caractéristiques de l'écran qu'au début de la com-

munication. L'application X adapte ses opérations graphiques à ces caractéristiques.
- *la transparence du réseau* : le client et le serveur sont deux processus qui peuvent s'exécuter sur la même machine ou sur des machines distantes et sur des plate-formes différentes. Ils communiquent en utilisant un protocole indépendant du matériel informatique ou du système d'exploitation.
- *l'ouverture vers des extensions* : X réserve 128 codes d'opération pour des extensions. Une extension utilisée par un client doit être supportée par le serveur X impliqué.
- *l'indépendance de l'aspect graphique et de la gestion des fenêtres* : le système fournit un mécanisme, mais la politique est donnée par un client X particulier, le gestionnaire de fenêtres.
- *le protocole X est asynchrone.*

Ces caractéristiques donnent à X une très grande flexibilité qui a conduit à son expansion sur diverses plate-formes. Actuellement, il y a diverses versions du serveur pour les systèmes Mac OS X, Windows, OS/2 ou Amiga. Un serveur spécial est offert par un "fournisseur d'applications" (*ASP, Application Service Provider*) WiredX.net [58] sous forme d'applet Java qui permet l'accès à des applications X, d'un navigateur web qui supporte Java. Les propriétés du serveur X l'ont même mené vers de petits dispositifs de type PDA (*Personal Digital Assistant*). En plus, NX le rend utilisable sur tous les types de réseaux. Grâce à son expansion, on peut dire que le système X est devenu pratiquement "ubiquitaire".

2.2 Le partage d'applications X

En tant qu'un système distribué qui supporte l'hétérogénéité de plate-formes, X est très attractif comme support de collaboration à base d'applications existantes auxquelles on ajoute la possibilité de partage de l'affichage. Pour le transformer en un système de fenêtrage partagé[17], on peut suivre deux chemins possibles : soit éviter de modifier le système, soit apporter des modifications à la partie client ou au serveur. Nous discutons ci-dessous les solutions existantes.

2.2.1 Les solutions de type "proxy"

La facilité de capter et transformer un flot de communication X a conduit naturellement à l'introduction d'un niveau d'indirection entre le client et le serveur (cf. Figure 2.10). Le proxy introduit dans le système joue le rôle d'un *multiplexeur de X* (*Mux X*) : il filtre les communications X, multiplie les Requêtes du client vers les serveurs impliqués et gère les messages-réponses de ceux-ci pour donner au client l'impression d'une session X habituelle. Tous les messages doivent être adaptés aux différences physiques de dispositifs d'affichage. De cette façon, la sortie graphique de l'application X est dupliquée sur plusieurs dispositifs

[17]Dans ce mémoire, nous nous y rapportons aussi par *le système de partage X.*

d'affichage. Toutes ces opérations se déroulent d'une façon transparente pour les composants du système X concernés.

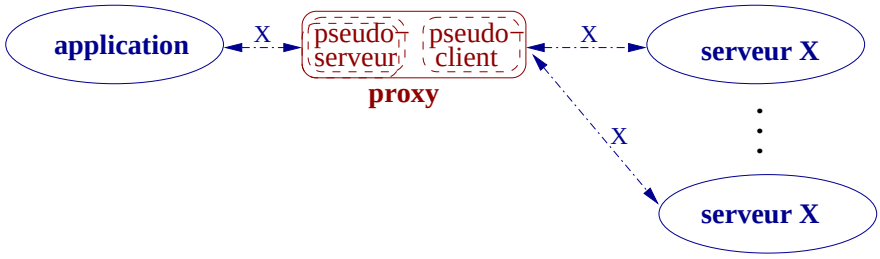

FIG. 2.10 – Architecture centralisée. Un proxy est interposé entre le client et le serveur X. Il est vu comme un serveur habituel par l'application et comme un client X par les serveurs.

Cette architecture est très flexible. Le proxy peut se trouver sur n'importe quelle machine, mais si une application X ne communique pas avec le serveur via le proxy dès le début, son partage est impossible. Ainsi, l'utilisateur qui lance une application potentiellement partagée doit toujours accepter ce surcoût lié à l'indirection. En plus, le proxy peut-être vulnérable à une surcharge causée par un nombre trop grand de participants. Une éventuelle défaillance conduit à l'interruption de la session de collaboration.

XMX dans sa première version [8] est un exemple de multiplexeur du système X. Pour traiter le problème des capacités différentes de dispositifs d'affichage, un serveur-maître fournit au proxy les caractéristiques qui sont présentées au client. Une application partagée à l'aide de XMX est affichée dans une fenêtre-racine virtuelle créée sur chaque serveur X participant. Le contrôle d'entrée se fait par la fenêtre virtuelle.

En plus des opérations de base d'un multiplexeur, le proxy peut jouer un rôle plus complexe, par exemple, le rôle d'un pseudo-serveur ayant la capacité de générer des messages et de répondre au client. En combinaison avec des techniques de mémorisation des ressources créées pendant le déroulement de la session X, cette caractéristique du proxy est une solution au problème des retardataires (*latecomers*) [16] dans un système de fenêtrage partagé basé sur X. Dans ce cas, le proxy est capable de "rejouer" la session X jusqu'à un moment donné, pour refaire l'affichage sur un nouveau dispositif d'affichage.

Un autre exemple, XTV [29] permet de joindre des sessions en cours et offre une interface graphique évoluée sur chaque serveur intégrée dans un bureau virtuel (*virtual desktop*). L'utilisateur a une vision claire des sessions auxquelles il participe et peut prendre le contrôle sur des applications partagées.

La dépendance de la session de collaboration du serveur-maître peut être éliminée si le proxy prend aussi en charge la gestion de ressources. De cette façon, il présente au client des caractéristiques physiques d'un dispositif d'affichage et il est capable de répondre aux demandes d'allocation des couleurs des clients. Ensuite, dans la communication X avec d'autres serveurs, il essaie d'adapter ses caractéristiques à celles de ces serveurs. Le projet Vir-

tualX [60] et XMX dans la version courante [10], présentent cette fonctionnalité. En général, ce type de proxy est très utile pour résoudre le problème de migration (téléportation) [50] [62].

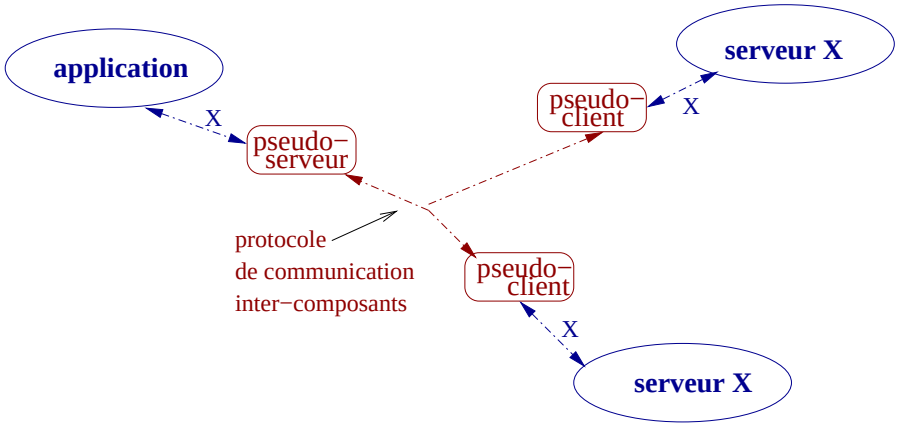

FIG. 2.11 – Architecture distribuée. Ce type d'architecture implique l'existence d'un composant pseudo-client du proxy du côté de chaque serveur X et d'un pseudo-serveur du côte du client qui doit être partagé. Cette configuration permet la communication entre les composants du système collaboratif basé sur la communication multipoint.

On retrouve ce comportement complexe du serveur X dans des projets de systèmes de fenêtrage partagé basés sur une architecture distribuée (Figure 2.11). Cette architecture répartit les tâches du multiplexeur sur chaque machine impliquée dans la collaboration. Cette solution évite la concentration des fonctionnalités de partage dans un point unique du système et ainsi supporte un grand nombre de participants à la collaboration.

Le système présenté en [28] propose une architecture distribuée où un pseudo-client communique avec les pseudo-serveurs par une extension du protocole X. Ce projet offre une infrastructure qui sépare les mécanismes de la politique en ce qui concerne le système de fenêtrage partagé et le contrôle d'entrée et d'accès. Il est aussi ouvert vers l'utilisation de différents mécanismes de communication entre les composants du système. Pour faciliter la collaboration, on peut modifier la fenêtre principale de l'application partagée pour offrir des informations utiles sur la session de partage et un outil de contrôle d'entrées. Dans d'autres projets [35] [12], la communication entre des composants du système de partage pour transmettre des requêtes graphiques est basée sur un mécanisme de *multipoint*.

2.2.2 Des modifications du système X

La première idée consiste à modifier la bibliothèque Xlib pour que les applications deviennent sensibles à la collaboration. Ainsi, elles sont capables de supporter plusieurs serveurs X en même temps (Figure 2.12). Cette solution ne nécessite pas la modification du client. Malheureusement, l'application doit être dynamiquement liée à la nouvelle bibliothèque Xlib

et pour pouvoir utiliser une applications X en mode partagé, la version modifiée de Xlib doit être disponible sur cette machine-là. ShX (cité en [62]) est un exemple de système de fenêtrage partagé X basé sur cette idée.

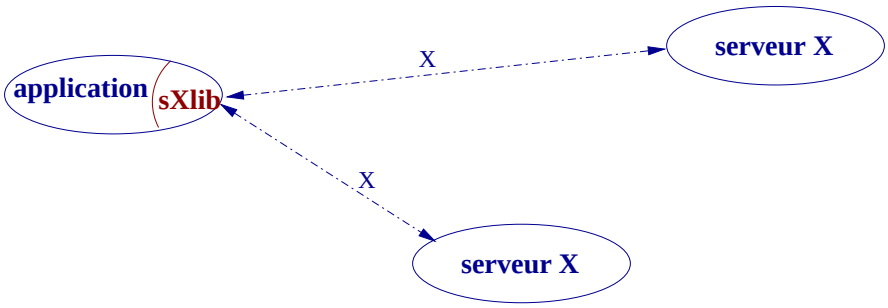

FIG. 2.12 – Système de fenêtrage partagé basé sur la modification de la bibliothèque Xlib.

Pour réaliser le partage d'applications, SharedX de HP [19] apporte une extension au serveur X (Figure 2.13). Ainsi, les fonctionnalités du multiplexeur sont ajoutées au serveur. Une application spéciale X offre différents niveaux de dialogue, des informations et un service de souris partagé. L'inconvénient de cette solution est la dépendance du serveur du matériel informatique, en conséquence, cette approche ne semble pas être disponible sur toutes les plates-formes. Comme les opérations de partage sont concentrées sur le serveur, il se pose aussi le problème de passage à l'échells (*scalability*) dans ce cas.

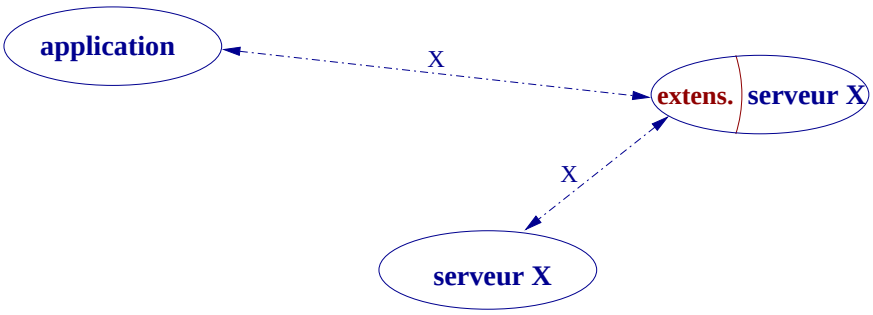

FIG. 2.13 – Système basé sur la modification du serveur X.

2.2.3 Conclusions

Nous pouvons observer que la plupart des travaux existants pour transformer X en un système de fenêtrage partagé sont axés sur l'idée du multiplexeur. Les solutions basées sur la modification du système X ont l'avantage en plus de permettre le partage de n'importe quelle application X, même après son lancement. Par contre, elles présentent un inconvénient lié à la disponibilité du système X modifié.

Le multiplexeur est conçu comme un ensemble d'applications interposées de façon transparente entre le client et le serveur. Le nouveau système construit autour du multiplexeur peut offrir un environnement riche de collaboration. Nous avons vu des projets qui incluent le contrôle d'entrées [28] [19] [35] [10] [6] [60], le contrôle d'accès et la souris partagée [19] [28], et d'autres applications de communication synchrone, comme le *chat* [6] et *whiteboard* [19] [6]. Par ses fonctionnalités complexes, un proxy pourrait offrir même des services de collaboration asynchrone, si on lui ajoute la fonctionnalité d'enregistrer et de rejouer une session X entière[18]. Ainsi, X devient un système à caractère coopératif.

[18] À notre meilleure connaissance, les multiplexeurs existants supportent actuellement partiellement cette capacité dans le but de recevoir des nouveaux participants (le problème *latecomers*). Mais, il y a déjà des applications spéciales conçues dans ce but pour rejouer une session X entière, comme par exemple Xnee [73]. Alors, on peut considérer que l'ajout de cette fonctionnalité est faisable.

Chapitre 3

Le système de partage d'applications X

Dans ce chapitre nous présentons un système de fenêtrage X enrichi de nouvelles caractéristiques coopératives fournies par un multiplexeur X. Pour sa simplicité, nous avons choisi d'implémenter l'architecture centralisée dans laquelle la multiplication des flots X est effectuée par une entité active du réseau.

3.1 Implémentation

Comme nous l'avons déjà mentionné (dans la Section 2.2.1), le rôle principal d'un multiplexeur X est de répliquer l'interface graphique d'une application sur plusieurs dispositifs d'affichage d'une manière transparente pour le système X. Comme X n'est pas conçu pour supporter le partage de fenêtres, l'implémentation d'un multiplexeur implique de nombreux problèmes techniques dûs à la différence des caractéristiques physique des dispositifs d'affichage impliqués dans le partage.

Dans la suite, nous présentons le prototype et les aspects les plus importants de son fonctionnement.

L'interception du trafic et la mise en oeuvre de la connexion

Pour pouvoir dupliquer l'affichage graphique d'une application, le multiplexeur (MUX) doit intercepter le flot X entre le client et le serveur.

En général, un serveur X écoute sur le port 6000 pour gérer l'écran 0. Ainsi, pour pouvoir s'afficher sur un écran géré par un tel serveur, un client X se connecte au port 6000 de la

machine à laquelle est attaché le dispositif d'affichage. Simultanément, une application X offre habituellement l'option *-display* pour préciser dans la ligne de commande le *display* (dispositif d'affichage) et l'écran où celle-ci doit affichée[1]. En conséquence, pour utiliser un serveur X qui gère l'écran 0, on doit préciser l'option : *-display dispositif_d'affichage :0*.

Pour pouvoir intercepter le flot X, le multiplexeur apparaît à un client comme un serveur qui gère l'écran 9 du dispositif d'affichage. Ainsi, il écoute et accepte des connexions de la part de clients sur le port 6009. En conséquence, pour dupliquer l'affichage graphique d'une application, on doit utiliser les moyens mis à disposition par l'application ou par le système d'exploitation pour lui préciser d'utiliser le serveur qui gère l'écran 9. Dans la ligne de commande, cela revient à l'option : *-display dispositif_d'affichage :9*.

Une fois la connexion réalisée, les messages du client peuvent être distribués aux serveurs impliqués. La première Requête de connexion, qui détermine le comportement du client pendant toute la session X, reçoit normalement une réponse de la part de chaque serveur, mais l'application doit connaître la présentation d'un seul serveur. La solution la plus simple est de choisir un serveur qui va jouer le rôle de serveur-maître pendant toute la session X et qui est présenté à l'application. Dans notre cas, le premier de la liste de la ligne de commande est choisi comme serveur principal. La Figure 3.1 présente l'interception et la distribution de trafic.

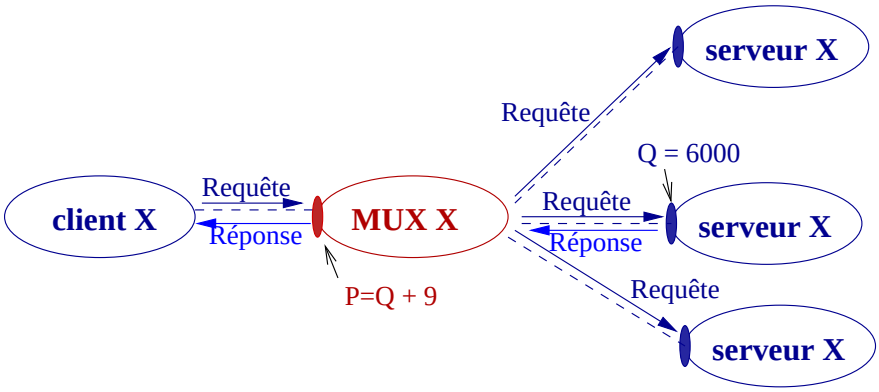

FIG. 3.1 – Interception et distribution de trafic.

Le contrôle d'entrée

Une application X suppose un seul utilisateur. La réception de tous les messages qui viennent de la part de tous les serveurs peut générer un comportement anormal et l'arrêt prématuré du client (par exemple, la réception de la Réponse de tous les serveurs). Dans cette situation, pour une Requête, le client recevrait une Réponse en plusieurs exemplaires, ce qui n'est pas attendu. De même pour les Erreurs et les Évènements qui sont des effets

[1]Sous Unix on peut préciser aussi le serveur par la variable d'environnement *DISPLAY*.

secondaires de l'exécution des Requêtes. Dans la phase actuelle d'implémentation, nous acheminons au client touts ces messages-réponses du serveur principal. Il y a quand même des Évènements qui doivent être envoyés depuis tous les serveurs. Par exemple, nous réexpédions tous les Évènements *Expose*[2] de tous les serveurs. Il faut aussi contrôler les Évènements qui proviennent des dispositifs d'entrée des utilisateurs. Par exemple, si deux utilisateurs appuient sur un bouton de souris au même moment, ces opérations génèrent une séquence d' Évènements qui est impossible en provenance d'un seul serveur[3].

La méthode la plus simple pour contrôler l'entrée est celle d'avoir un seul participant qui peut interagir avec l'application. Cette méthode est trop stricte en général pour la collaboration. Une autre méthode est celle du "jeton" : seuls les Évènements d'entrée de celui qui détient le "jeton" sont transmis au client. Ainsi, le contrôle d'entrée se situe dans le contexte du droit au contrôle (*control floor*)[4].

Le problème central du droit au contrôle dans le cas du partage de fenêtres X concerne le choix du moment où ce droit passe à un autre participant. Si le droit est demandé par un utilisateur au moment où le serveur qui détient ce droit attend une réponse à une requête, le droit doit être passé après l'arrivée de la réponse. Le choix du moment est compliqué aussi par la spécificité de chaque application : certaines opérations (par exemple des opérations graphiques) ne peuvent pas être interrompues[5] (voir aussi [28]). Pour le moment, notre multiplexeur accepte seulement des messages du serveur-maître. Dans la Figure 3.1, nous avons mis en évidence l'idée que le multiplexeur n'accepte que des messages venant d'un seul serveur. Dans le dessin, le deuxième serveur est le serveur principal.

Un autre problème dans ce contexte est posé par le sort des entrées générées par un participant qui n'a pas le droit au contrôle. Une solution pourrait être de les retenir dans un tampon et de les fournir au client quand le participant obtient de nouveau le droit. Jusqu'à l'implémentation complète des mécanismes du contrôle de ce droit, nous ne retenons pas les messages des serveurs secondaires.

Nous sommes partisans de la séparation du mécanisme qui met en place le droit au contrôle, de la politique qui donne les règles de contrôle. Alors, nous pensons que le multiplexeur peut offrir d'un côté une politique implicite de contrôle et de l'autre côté, une interface de contrôle qui donne la possibilité à des services extérieurs de gérer la politique. Pour le moment, le multiplexeur offre seulement au serveur-maître la possibilité d'avoir le

[2]L'événement Expose est envoyé quand il y a des régions d'une fenêtre pour laquelle le contenu n'est plus disponible.

[3]Un Évènement généré par l'enclenchement d'un bouton est suivi toujours par celui généré par sa libération, avant de la génération d'un autre évènement identique.

[4]*Control floor* se réfère au contrôle d'accès temporaire à une ressource [20]). En général, la notion de *floor* signifie le *droit d'accès* à la ressource. Dans ce document, *floor* a la signification de "droit de générer des événements d'entrée".

[5]Par exemple, prenons le cas d'un programme de dessin. Pour dessiner un arc de cercle, l'utilisateur choisit d'abord un point de départ en appuyant sur un bouton de la souris. Celui-ci est suivi des autres points qui donnent le contour de l'arc. L'envoie d'un événement qui signifie le début d'une autre opération avant la fin de la précédente peut être interprété de manière confuse par le programme.

contrôle d'entrée.

Client de contrôle

Nous avons décrit le mécanisme du droit au contrôle au niveau de protocole X. Le problème qui apparait alors logiquement est celui de l'interface entre les utilisateurs et ce mécanisme. Ci-dessous, nous présentons une proposition de solution à ce problème.

De la même manière que le système X inclue le *window manager* (Section 2.1.7) pour gérer des fenêtres, nous introduisons un client X spécial qui offre l'interface graphique pour gérer le droit au contrôle. Cette interface graphique est présentée sous forme d'un bouton attaché à la fenêtre principale de l'application partagée.

Comme tous les participants doivent avoir accès au mécanisme du droit au contrôle, toutes les copies de l'interface graphique de l'application doivent avoir ce bouton attaché. Cet élément graphique offre la possibilité de demander et de céder le droit d'entrée. L'admission et l'accomplissement de chaque opération dépendent de la politique du droit au contrôle appliquée à chaque session de collaboration. Naturellement, le bouton devrait fournir aussi des informations sur l'état du droit et même de la collaboration. Nous le voyons comme un outil qui peut proposer des fonctionnalités utiles pour la collaboration[6].

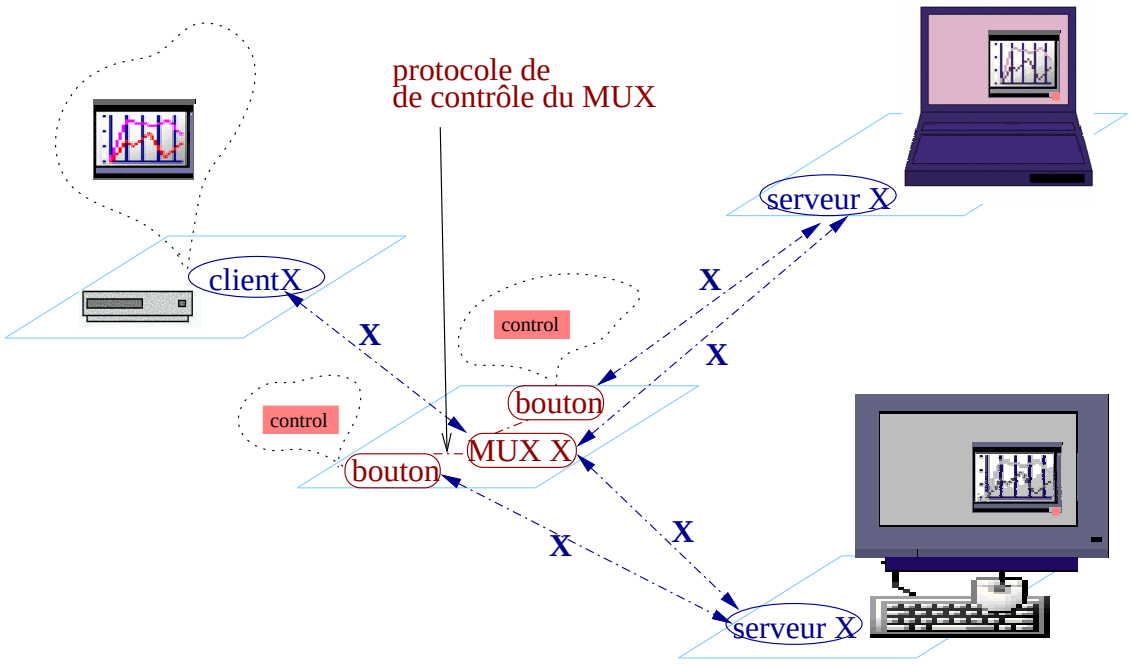

FIG. 3.2 – Architecture de système de fenêtrage partagé qui inclu la gestion du droit au contrôle (noté "bouton" dans la figure).

[6]Par exemple, même la possibilité d'envoyer des messages à des autres participants à la collaboration.

Nous avons conçu un client X qui est lancé en exécution par multiplexeur pour chaque serveur impliqué dans la session de collaboration[7] (Figure 3.2.). Il colle sa fenêtre dans le coin inférieur droit de la fenêtre principale de l'application partagée (voir la Figure 3.3). Dans la phase actuelle de l'implémentation, il offre une seule fonctionnalité : l'appuie sur ce bouton agrandit la fenêtre principale de l'application sur tout l'écran. Grâce à cette opération les participants peuvent se concentrer exclusivement sur la collaboration. Ultérieurement notre client devrait translater les évènements d'entrée introduits par l'utilisateur dans des commandes de contrôle transmises au multiplexeur[8].

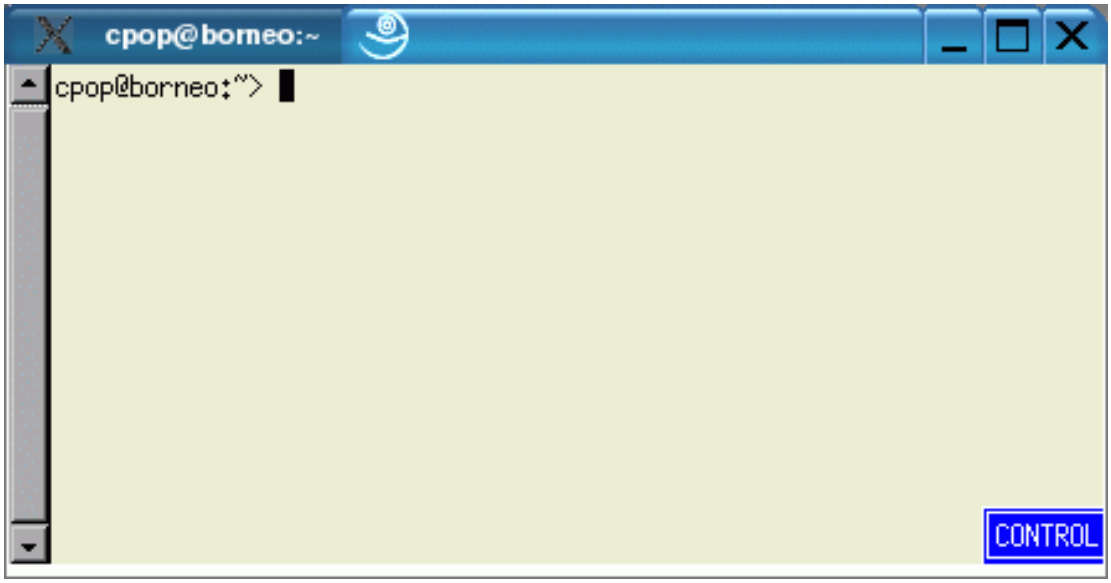

FIG. 3.3 – Fenêtre principale de *xterm* avec le client de contrôle attaché.

L'attachement du bouton à la fenêtre principale se base sur la même propriété que celle sur laquelle se base le fonctionnement du *window manager* : notre client a accès à toutes les ressources du serveur impliqué. Pendant la communication entre le client et le serveur X, le multiplexeur cherche à trouver la fenêtre principale de l'application. Comme X donne une grande liberté à la conception d'applications, leurs arbres de fenêtres diffèrent beaucoup. Notre méthode est basée sur l'attribut "WM_STATE" qui lui est attachée par le gestionnaire de fenêtre [56]. multiplexeur démarre le client de contrôle avec l'ID de la fenêtre appropriée de l'application comme paramètre. La fenêtre principale de notre client devient la fille de celle de l'application.

Un inconvénient de cette méthode vient des applications qui détiennent plusieurs fenêtres principales, comme par exemple *xv*. Comme elles ne sont pas très nombreuses, on pourrait les identifier (selon le "nom" de la fenêtre principale de l'application, qui est gardé dans la

[7]Une variante plus intéressante du client pourrait afficher elle-même son interface graphique sur les serveurs impliqués

[8]Il s'agit des commandes spécifiques de l'interface du contrôle du multiplexeur. Nous ne sommes pas arrivés à cette phase de l'implémentation.

propriété "WM_NAME" de cette fenêtre) et appliquer à chacune la méthode convenable pour déterminer la bonne fenêtre pour l'affichage du bouton. Un autre inconvénient concerne la dimension de la fenêtre du client. Celle ci devrait être minimale pour ne pas gêner l'utilisation habituelle de l'application partagée. Des autres fonctionnalités pourraient être assurées en utilisant une sous-fenêtre *pop-up*.

Par rapport aux travaux similaires, nous avons la vision d'un client X spécial dont l'interface graphique s'intègre dans celle de l'application partagée. Cette intégration se réalise d'une façon transparente pour le fonctionnement de l'application. D'autres solutions sont basées sur l'idée d'un client X de contrôle, mais qui s'affiche normalement sur les écrans des participants [6]. Une autre approche crée une fenêtre qui devient la racine pour l'application partagée [28]. Dans cette fenêtre, l'interface de contrôle est insérée à côté de la fenêtre principale de l'application.

La translation des ressources

Il y a deux types de ressources sur un serveur. D'un côté, il y a celles qui sont générales pour tous les clients (par exemple, la fenêtre-racine et la *colormap* implicite) identifiées par des IDs spécifiques sur chaque serveur.

De l'autre côté, comme nous avons précisé dans la description des ressources X (Section 2.1.3), il y a celles créées par le client. Chaque serveur fournit à chaque client un domaine pour générer des identificateurs pour ses ressources. Ces domaines différent d'un serveur à un autre. Ainsi, pour chaque Requête de création d'une ressource, le multiplexeur doit jouer son rôle de client et générer un identificateur adéquat pour chaque serveur secondaire. La Figure 3.4 montre cette opération. Le domaine pour générer des identificateurs est précisé par deux ensembles de bits (*resource-id-base* et *resource-id-mask*). La méthode de génération d'un nouvel identificateur est donnée dans la spécification du protocole [55]. Il s'agit de prendre à chaque fois un sous-ensemble de bits de *resource-id-mask* différent et de lui appliquer l'opération OU avec *resource-id-base*.

À l'arrivée d'une Requête qui contient un identificateur (qui correspond soit à une ressource générale, soit à une ressource spécifique d'un client), la Requête est multipliée et la valeur d'identificateur est remplacée par la valeur qui correspond à chaque serveur secondaire (Figure 3.5).

La façon de mémoriser et de retrouver les identificateurs peut avoir une influence sur la performance du multiplexeur à cause du grand nombre des messages envoyés. Nous avons conçu une structure de données sous forme d'un "tableau" n x m, où n est le nombre de serveurs impliqués et m est le nombre des IDs. Dans chaque ligne du tableau, à chaque colonne i se trouve l'identificateur qui correspond au serveur i[9]. Comme au cours de la session X, il y a des identificateurs qui se rajoutent, la dimension du tableau n'est pas fixe.

[9]Il s'agit de l'ordre dans la liste donnée dans la ligne de commande.

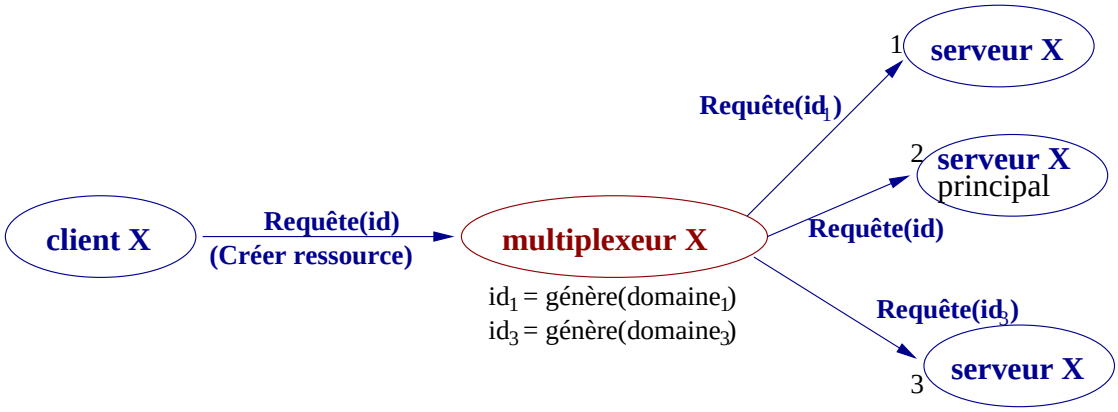

FIG. 3.4 – Génération d'un nouvel identificateur pour chaque serveur secondaire. Chaque Requête de création d'une ressource contient l'ID de celle-ci. En utilisant le domaine accordé par chaque serveur secondaire ($domaine_i$), le multiplexeur génère un nouvel ID pour chacun d'entre eux. Il multiplie la Requête et y remplace l'ID adéquat.

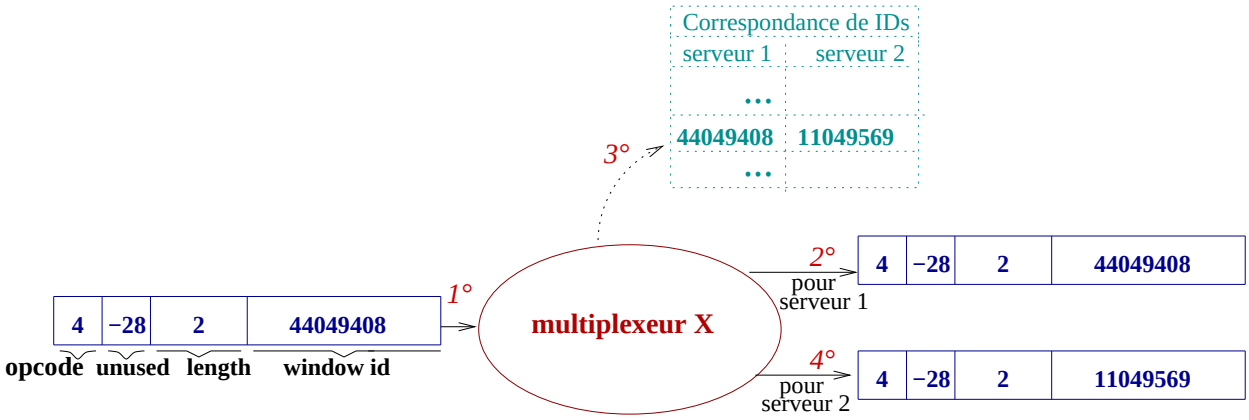

FIG. 3.5 – Translation des identificateurs pour les Requêtes. La figure illustre le cas concret d'un message simple qui demande la destruction d'une fenêtre. Le premier serveur est principal.

Après une étude sur la performance des structures de données en Java, nous avons choisi la structure ArrayList [64] pour l'implémentation. Les tests ont montré son efficacité pour les opération de rajout d'éléments en fin de liste et d'accès. Ainsi, nous avons retenu les identificateurs en utilisant cette structure en combinaison avec HashMap [31]. À chaque identificateur du client, le tableau *hash* lui associe la liste des identificateurs générés par multiplexeur pour chaque serveur secondaire.

Le problème de translation des IDs se pose aussi pour les messages-réponse qui contiennent des identificateurs. Cette fois l'opération est inverse : leurs valeurs sont remplacées, si nécessaire, par des valeurs qui correspondent au serveur principal connues par le client.

La translation des atomes

Il se pose le même type de problème pour des atomes (décrits dans la Section 2.1.4). À la demande du client, chaque serveur retourne l'atome associé à un chaînes de caractères donnée. Ces atomes peuvent être différents sur chaque serveur. Le multiplexeur retient ces valeurs des messages de Réponse. Pendant la session, il doit remplacer les atomes avec des valeurs adéquates dans chaque Requête concernée à destination de serveurs secondaires.

De point de vue pratique, pour l'enregistrement des atomes des serveurs secondaires dans le tableau de translation associé, on doit prendre en compte un retard possible dans la réponse du serveur principal. En principe, dans ce cas, les atomes des serveurs secondaires doivent être enregistrés sans avoir l'atome du serveur principal comme repère. Cela rend cette opération encore plus difficile.

Sans avoir la prétention d'avoir choisi la meilleure solution, nous présentons ensuite l'algorithme utilisé dans notre implémentation. Nous nous somme servis encore une fois d'un tableau n x m, où n est le nombre de serveurs et m est le nombre d'atomes. L'algorithme est basé sur l'idée que pour chaque valeur d'atome reçu, il y a une ligne dans le tableau où les valeurs des atomes qui manquent encore sont 0.

Quand arrive une Réponse qui inclue un atome de serveur j :
- on cherche la première ligne de tableau où sur la position j il y a la valeur 0. Ainsi,
  - si cette ligne n'existe pas (cela vaut dire que celui-ci est le premier serveur qui donne la Réponse) ⇒ il crée une nouvelle ligne dans le tableau, avec la valeur d'atome sur sa position j et 0 en reste ;
  - si cette ligne existe (cela vaut dire que il y a un serveur qui a envoyé déjà la Réponse) ⇒ on remplace 0 de sa position j avec la valeur d'atome.

D'autres atomes qui doivent être retenus se trouvent dans la Réponse de la Requête qui demande des informations sur une police (*Requête 47*). Ces informations incluent des propriétés qui sont référées par des atomes. La liste des atomes diffère la police utilisé. En plus, nous avons aussi retenu l'atome inclu dans l'Événement qui annonce le changement

d'une propriété d'une fenêtre (*Événement 28*).

Une autre difficulté dans la stratégie de mémorisation des atomes est provoquée par le comportement curieu de certaines applications qui demandent le même atome plusieurs fois. Cela peut entraîner la mémorisation d'informations redondantes.

Le retard de Réponses de serveurs secondaires génère un autre type de problème. En général, une Réponse contenante un atome est suivie par une Requête qui l'utilise. Dans la situation où un des serveurs secondaires n'a pas répondu en donnant l'atome, le multiplexeur ne peut faire la translation d'atome pour ce serveur. Alors, il faut attendre toutes les Réponses, ce qui diminue les performances du multiplexeur.

Les couleurs

À la requête du client concernant l'allocation d'une couleur, le serveur peut allouer une cellule (*cell*) dans une carte de couleurs (*colormap*). Cette couleur est désignée dans les Requêtes par la valeur du pixel associé à sa cellule dans la carte de couleurs (voir la Figure 2.6). Comme chaque serveur peut attribuer des cellules avec des indexes différents, cette valeur doit être remplacée par celle correspondant à chaque serveur secondaire. Le problème semble être très similaire à celui de la translation des identificateurs. Mais, l'utilisation pratique de la structure de données *ArrayList* a prouvé son inefficacité à cause d'un grand nombre de pixels qui sont impliqués dans certaines Requêtes[10]. Le retard d'opération de recherche de pixels dans la structure de données suivi par le remplacement dans les messages ont conduit à trop diminuer les performances du multiplexeur. De l'autre côté, nous avons constaté de manière pratique que pour les dispositifs d'affichage (*displays*) similaires, on attribue en général les mêmes valeurs de pixels [11]. Alors, nous avons laissé ouvert ce problème, parce que notre objectif principal n'était pas une implémentation optimisée de multiplexeur, mais la validation des concepts.

Un problème apparaît aussi quand un des serveurs ne peut pas allouer une couleur. Pour maintenir la similarité des interfaces graphiques sur tous les dispositifs d'affichage, le multiplexeur pourrait avoir un mécanisme permettant de rechercher une couleur similaire et de la demander. Cette opération devient encore plus difficile quand les serveurs gèrent des dispositifs d'affichage ayant des visuels différents (Section 2.1.3)[12]. [19]

La manipulation des extensions du X

Le problème soulevé par les extensions X est lié au fait qu'elles ne sont pas implémentées sur tous les serveurs. En plus, si elles sont implémentées, le multiplexeur doit connaître

[10]Par exemple les Requêtes qui manipulent des images : *PutImage* et *GetImage*.

[11]Cela arrive principalement dans le cas où les dispositifs d'affichage utilisent des visuels "TrueColor".

[12]Heureusement, ce type de problèmes est plus rare grâce à l'utilisation de plus en plus de dispositifs d'affichage ayant des visuels "TrueColor" [26].

leur spécifications, ce qui est pratiquement impossible étant donné l'ouverture de X pour les extensions. Dans ce cas-là, on propose deux solutions : accepter les extensions et les acheminer vers les serveurs intéressés, ou nier leur existence sur tous les serveurs. Dans le premier cas, l'interface graphique du client pourrait être différente sur les serveurs qui travaillent avec des extensions. Dans le deuxième cas, il y a le risque que l'extension soit essentielle pour le déroulement de l'application, ce qui va empêcher son déroulement corrrecte.

Dans la version actuelle, multiplexeur supporte seulement le noyau du protocole X : il nie au client l'existence des extensions.

Relayage des Requêtes X

Il n'est pas nécessaire d'envoyer toutes les Requêtes à tous les serveurs. Évidemment, une requête qui modifie l'état d'une ressource doit être envoyée à tous les serveurs, mais il y en a d'autres qu'on doit envoyer seulement à celui qui détient le droit au contrôle (*floor*, par exemple la Requête "QueryPointer" qui demande la position du pointeur). Si une Requête demande des informations au serveur, par exemple sur des polices disponibles (la Requête "ListFont"), le multiplexeur pourrait collecter toutes les Réponses envoyées par tous serveurs et envoyer au client une liste des polices communes (solution suggérée par ailleurs [29]).

De manière similaire, il y a des Évènements concernant la modification de l'état d'une ressource. Ils peuvent être envoyés au client de n'importe quel serveur, bien sûr en un seul exemplaire. Normalement, les évènements générés par les dispositifs d'entrée sont envoyés par le serveur qui détient le droit au contrôle (*floor*). D'autres évènements, comme par exemple "Expose", sont envoyés au client par tous les serveurs.

Ces aspects importants doivent être pris en compte dans le développement du multiplexeur dans notre contexte du travail.

Le démarrage de multiplexeur

Le multiplexeur peut être démarré par une ligne de commande ayant comme arguments la liste des serveurs impliqués au début dans la collaboration. Il faut lui également donner le droit d'accès par une des méthodes que nous avons vues précedemment (Section 2.1.8).

Autres problèmes posés par un multiplexeur X

D'autres aspects généraux doivent être pris en compte dans le développement d'un multiplexeur, par exemple la différence physique des claviers (Section 2.1.5) gérés par des serveurs impliqués [19]. L'existence de différentes polices sur des serveurs pose des difficultés en ce qui concerne la maintenance de la cohérence d'interfaces graphiques sur tous les dispositifs d'affichage. D'un autre côté, il y a aussi des applications qui posent des problèmes par leur

comportement. Par exemple, des applications qui ouvrent plusieurs connexions avec un client (*ghostview* et *framemaker* sont des exemples classiques). D'autres aspects qui doivent être encore considérés par multiplexeur X sont mentionnés dans un Rapport Technique [29] suite aux expériences d'implémentation d'un multiplexeur [6].

Un problème ouvert dans ce domaine est posé par l'apparition des dispositifs d'affichage très variés avec des écrans de différentes dimensions. Pour le moment, le multiplexeur reste une solution viable pour des serveurs avec des écrans des dimensions similaires [26].

En plus des problèmes relatif à l'implémentation du multiplexeur X, un autre sujet qui doit être pris en compte dans ce contexte est le contrôle d'accès. Nous voyons deux solutions à ce problème : soit l'implémentation d'un mécanisme propre de contrôle, soit de garder implicitement celui du système X. Nous pensons qu'en général, il est plus convenable d'avoir plusieurs méthodes de contrôle d'accès.

3.1.1 Conclusions concernant l'implémentation du multiplexeur X

Dans la version actuelle, le multiplexeur multiplie l'interface graphique de la plupart des applications X testées[13] sur un deuxième serveur. Nous ne sommes pas arrivés au stade de l'implémentation du contrôle de droit d'entrée car le deuxième serveur répond parfois encore par des Erreurs. La plupart de ces Erreurs sont provoquées par des atomes ou des identificateurs de ressources qui ne se trouvent pas dans la table de translation. La résolution de ces Erreurs impose une analyse encore plus approfondie du protocole, du trafic des communications X et du comportement des certaines applications[14].

À notre avis, la conception et l'implémentation d'un multiplexeur X qui répond à tous les besoins reste un problème ouvert. Des études de performances réalisées dans le cadre d'autres travaux dans le domaine soutiennent cette affirmation [8].

3.2 Multiplexeur X - service actif

L'idée de base d'un *réseau actif* est d'ouvrir des fonctionnalités des routeurs en permettant l'exécution du code associée à des paquets (*PDUs*) qui traversent le réseau. Cette exécution peut intervenir grâce au code contenu dans les paquets eux-mêmes ou à l'aide du code placé

[13] Nous avons choisi des applications qui ne sont pas spécifiques à un environnement bureautique.

[14] À un certain niveau du travail, la compréhension du comportement de l'application a un rôle important dans le développement du multiplexeur. La plupart des applications qui ne peuvent pas être multipliées par notre multiplexeur s'arrêtent soudainement sans aucune Erreur envoyée par le serveur principal. Apparemment, c'est à cause du manque de certaines extensions X. Par exemple, après plus de neuf milles Requêtes envoyées, le traitement de texte de OpenOffice [61] s'arrête et affiche une fenêtre avec le message "An unrecoverable error has occured".

sur certains noeuds du réseau. À cause de la taille limitée de paquets, un paquet actif peut contenir une référence à un programme qui sera téléchargé à partir des routeurs voisins, ou de serveurs de code. Dans l'approche basée sur des *noeuds programmables*, on charge des programmes spécifiques appelés *services actifs* sur des passerelles pour effectuer certains traitements sur des paquets : la compression, le découpage, le filtrage, la combinaison de paquets. Les paquets sont interceptés grâce à un module intégré au noyau pour être ensuite classifiés selon certains critères et fournis aux services actifs[15].

Dans cette section, nous présentons un prototype d'une passerelle programmable utilisée pour implémenter notre multiplexeur.

3.2.1 Passerelle programmable utilisée

Dans son travail, Hoa-Binh Nguyen [40] a suivi l'idée d'un noeud actif ayant un environnement d'exécution générique qui permet d'activer et désactiver des *services actifs* dynamiquement. La passerelle active générique appelée *ProAN* supporte plusieurs environnements d'exécution. Son architecture générique pour les services actifs permet d'opérer sur des unités de protocole à différents niveaux : réseau, transport ou application. Cet environnement génère automatiquement l'analyseur et le générateur de PDU (Protocol Data Unit) d'un protocole donné à partir de sa description. ProAN est également adapté aux services sensibles au contexte qui peuvent réagir aux changements d'état de l'environnement sans l'intervention de l'utilisateur.

L'idée du réseau actif correspond bien à notre vision de la collaboration. D'un côté, un noeud actif peut présenter sur le réseau des fonctionnalités intéressantes comme le support pour la collaboration entre des personnes. Ces fonctionnalités peuvent être activées à la demande. D'un autre côté, l'utilisation des noeuds actifs en bordure du réseau facilite l'utilisation des fonctionnalités mises à disposition par des services actifs dans un environnement spontané de communication. L'utilisation d'un service actif pour réaliser le multiplexeur X devrait faciliter l'adaptation au contexte, ce qui implique la réactivité vis à vis des messages de contrôle et/ou à des événements extérieurs.

Nous présentons ci-dessous le fonctionnement et la structure d'un service actif dans la plate-forme de ProAN.

Fonctionnement et structure d'un service actif

La version de la passerelle programmable que nous avons utilisée dans notre travail supporte des services actifs conçus comme un ensemble de composants ayant chacun un rôle bien défini. Un composant se présente sous forme d'un module de code indépendant appelé *plugin*. Nous présentons ces composants et leurs fonctionnalités :

[15] Un état d'art complète sur les réseaux actifs peut être trouvé ailleurs [40].

– *in Plugin* : récupère les unités de protocole d'un niveau donné,
– *PDU Contrôle* : contrôle le traitement des unités de protocole en fonction des instructions spécifiées et en utilisant des fonctions spécifiques au protocole utilisé,
– *PDU Plugin* : traite des unités de protocole conforme au contrôle reçu,
– *out Plugin* génère des unités de protocole à la sortie de la passerelle.

La Figure 3.6 présente l'architecture générale d'un service actif et met en évidence les composants.

Une fois capturé par le service, un paquet est analysé, et les valeurs des champs de PDU sont extraites. Certaines conditions concernant les champs de PDU génèrent des actions spécifiques. Ces actions peuvent engager la modification des valeurs de champs. La dernière étape de ce processus consiste à recomposer une unité de protocole avec certaines valeurs modifiées des champs, et à l'expédier vers la destination.

Pour qu'un service suive ce processus de fonctionnement, on peut identifier alors ses composants fonctionnels principaux :

– le *parseur* qui analyse et détecte les valeurs des champs des unités de protocole. Il implique que la structure des paquets d'un protocole à considérer est définie dans un langage de spécification.

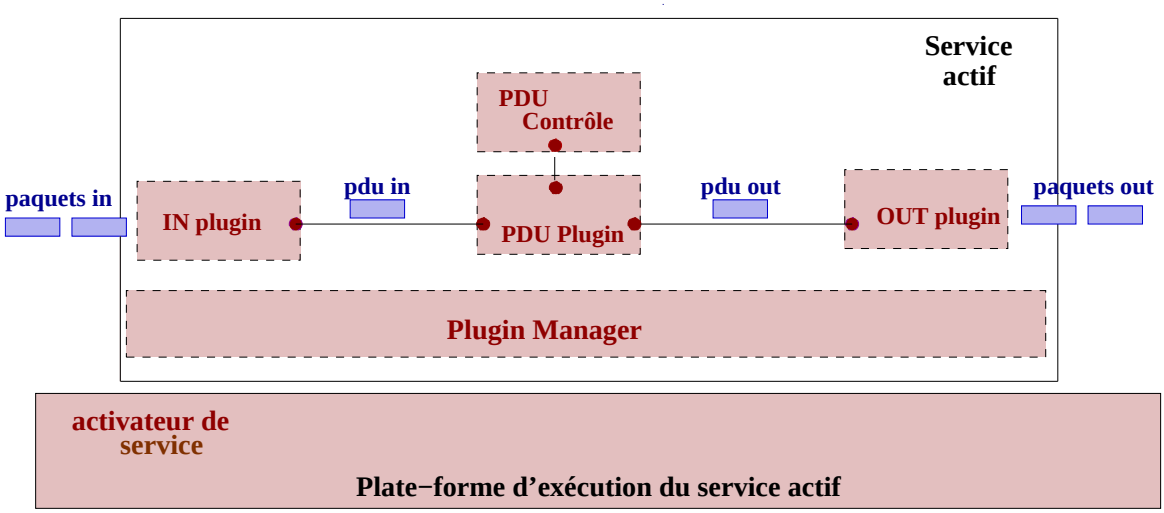

FIG. 3.6 – L'architecture générale d'une passerelle programmable. Les fonctionnalités du *parseur* et du *formatteur* sont réunies dans le composant "PDU Plugin". Le "PDU Contrôle" représente le composant *moteur*. En plus des composants décrits, deux autres *plugins* sont présentés dans la figure : "In Plugin" extrait le paquet du niveau de protocole désiré et "Out Plugin" génère le PDU sortant.

– le *moteur* qui applique des actions que l'on veut effectuer sur les unités de protocole. Elles actions sont spécifiées à l'aide d'un language de script de haut niveau. La programmation de services implique l'utilisation des champs spécifiques au protocole et des fonctions predéfinies fournies dans une bibliothèques attachées au service.

– le *formatteur* qui compose les unités de protocole en sortie. Celui-ci utilise également la spécification de la structure des paquets.

3.2.2 Multiplexeur X - service actif

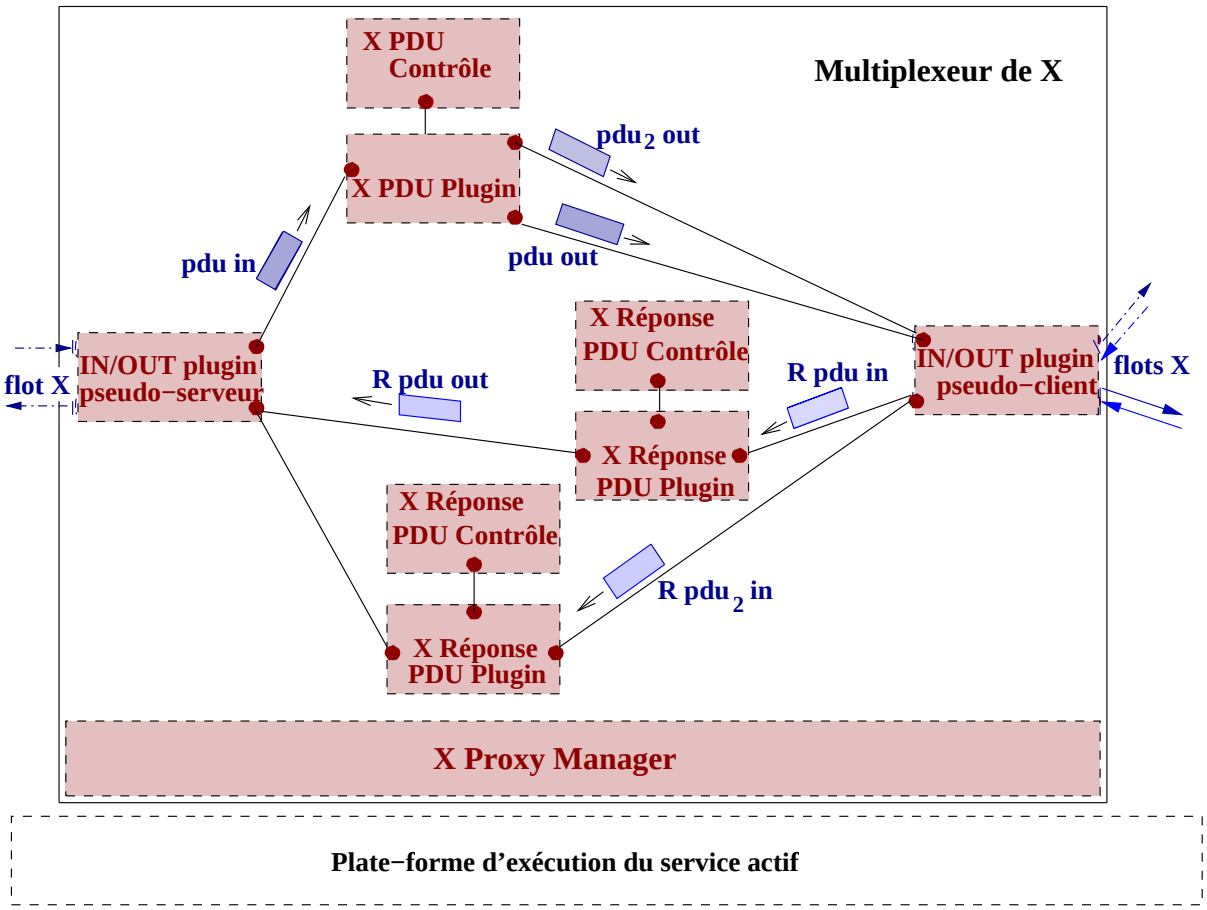

FIG. 3.7 – La structure du multiplexeur X - service actif pour le fonctionnement avec deux serveurs.

La Figure 3.7 montre la structure du multiplexeur implémenté sur la plate-forme de services actifs.

Dans notre cas, le service doit gérer des Requêtes et des Réponses de serveurs, par conséquent, sa structure est relativement complexe. Nous l'avons implémenté en plusieurs fils d'exécution (*threads*) qui se déroulent en parallèle (un pour la gestion de service et un pour le traitement de paquets.)

Dans le cas du multiplexeur, le traitement des unités de protocole consiste en la modifica-

tion des valeurs de champs des messages (de type requête ou réponse) et en la multiplication des unités de protocole des Requêtes. La figure 3.8 esquisse la fonctionnalité du service à l'arrivée d'une Requête.

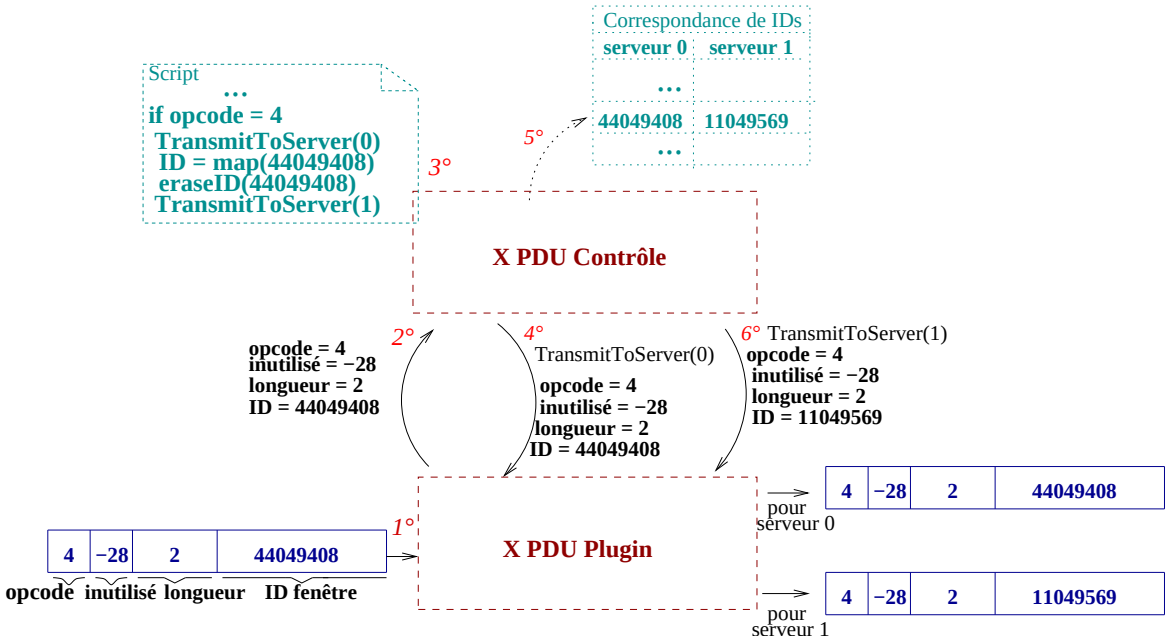

FIG. 3.8 – Détail du mécanisme de multiplication d'une Requête. Nous avons choisi une Requête avec une structure simple qui demande la destruction d'une fenêtre (Requête 4 dans la spécification du protocole X).

Nous expliquons le mécanisme illustré par la figure. À l'arrivée de la Requête (1), le *parseur* obtient les valeurs des champs appropriés. Celles-ci sont transmises au module de *contrôle* (2) qui applique le traitement associé à la Requête (3). Dans le "script" (la partie de programmation), on retrouve des fonctions spécifiques et des références aux champs de la Requête. En suivant les indications de la programmation, le module de contrôle envoie les valeurs actuelles de champs au *PDU Plugin* (4). Le formatteur (inclus dans le module *PDU Plugin*) compose la Requête en format non modifié à destination du serveur principal. Simultanément, le module de contrôle, selon l'indication du "script", remplace la valeur de l'identificateur de la fenêtre par celle correspondante au serveur secondaire (5). Les valeurs courantes des champs sont envoyées au *formatteur* à destination du deuxième serveur (6).

La programmation du service et la description du protocole incluses dans le module de contrôle sont réalisées en Java, en bénéficiant de tous les avantages de ce langage de programmation. Dans l'annexe, Section B nous fournissons un fragment du code de contrôle de Requêtes.

3.3 Comportement du prototype

Selon nos tests, les applications X s'exécutent via notre multiplexeur à une vitesse satisfaisante. On remarque un retard habituel d'initialisation des applications. Le retard varie en fonction de l'application et il est parfois notablement plus grand en cas d'utilisation du multiplexeur. On remarque aussi un retard de chargement d'une image de très grand taille (par exemple, JPEG 10752 x 2048) par *xv*[16]. En ce qui concerne l'interaction de l'utilisateur avec des applications, les réponses d'affichage sont rapides. Il semble que la seule exception sont le jeu *xboing*[17], qui par sa lenteur d'animation, paraît idéal pour les débutants. Au sujet de l'affichage, il y a parfois des problèmes de couleurs, suite à la translation des pixels. Ce problème est laissé ouvert.

Pour juger le comportement de notre prototype, nous présentons quelques mesures réalisées. Notre intérêt se focalise sur le délai introduit par le multiplexeur dans la communication au sein du système X. Il est intéressant de voir la performance du multiplexeur quand il multiplie l'affichage, par rapport à son comportement comme simple filtre du protocole X (quand il s'interpose simplement dans la communication client-serveur X).

La réalisation des mesures n'est pas évidente à cause de la nature distribuée du système. Les applications existantes de *benchmark* de X (*x11perf* [71], *xbench* [72]) sont conçues pour tester des serveurs X et la rapidité des opérations graphiques. Pratiquement, ils innondent le serveur avec des messages X. Ce comportement n'est pas spécifique qu'à une partie des applications X. Donc, ils ne donnent pas une idée précise du comportement du multiplexeur dans une session X habituelle.

Pour estimer le temps de déroulement d'une session X, nous nous orientons vers un niveau plus bas : la mesure du temps écoulé pendant l'échange des paquets dans une session X. Dans ce but, l'utilisation des outils comme *tcpdump* [63] et *(t)ethereal* [22] semble la plus appropriée[19].

Les performances d'un système de partage X basé sur un multiplexeur dépend de plusieurs facteurs qui incluent le comportement d'application [62], la performance de la machine sur laquelle s'exécute le multiplexeur, la charge du réseau. Nous avons réalisé nos mesures en utilisant trois machines. Le tableau 3.1 donne une description de chacune et son rôle dans nos tests. Les tests ont été réalisés sur un réseau local, les trois machines étant isolées par un commutateur.

Le premier pas à faire est le choix de la session X à mesurer. En général, il faut choisir des parties de sessions identiques, où la communication X suit le même modèle d'échange des messages. Les applications X étant interactives, l'intervention de l'utilisateur implique un

[16] xv est un programme interactif de manipulation d'image pour le système de fenêtrage X [13].

[17] *xboing* est un jeu de type *blockout* inclus dans la distribution du système X.

[19] Ces outils capturent des paquets du réseau et attachent à chacun une estampille de temps. *tcpdump* a été déjà utilisé pour l'analyse d'influence du modèle de flux de données (synchrone ou asynchrone) sur la performance du *xmove* [62].

Machine	Caractéristiques de la machine	Caractéristiques du système X	Rôle dans les tests
sicile	– AMD Athlon, 860MHz, 512 Mb mémoire ; – système opérationnel : Suse Linux 8.1 ;	– serveurX : XFree86 4.20 – display : 1280 X 960, de profondeur 24 planes, supporte 8 visuels, TrueColor[18] et DirectColor ;	exécution des applications X et du serveur secondaire
zanzibar	– AMD Athlon, 860MHz, 256 Mb mémoire ; – système opérationnel : Fedora Core 1	non significatif pour les testes	machine intermédiaire sur laquelle roule le *proxy* TCP/ multiplexeur
borneo	– Intel Pentium III, 1066Mhz, 256 Mb mémoire ; – système opérationnel : Suse Linux 8.2	– serveur X : XFree86 4.30 ; – display : 1024 X 768, de profondeur 24 planes, supporte 4 visuels, TrueColor ; supporte les mêmes profondeurs que sicile ;	la machine sur laquelle on fait les testes et exécute le serveur principal

Tab. 3.1 – Machines utilisées pour les testes.

degré de variabilité d'une session à l'autre. Nous avons soigneusement préparé des expériences pour trois applications en éliminant le plus possible cette variabilité. Nous avons évité pendant les tests toutes les opérations accidentelles qui pourraient générer des Événements X vers l'application impliquée.

Test 1 : application de type "Hello world !"

Nous avons considéré d'abord une application simple qui crée une fenêtre dans laquelle on affiche un texte (voir Annexe, Section A). Nous nous sommes proposés de mesurer la durée de la session X à partir du premier message d'initialisation de la session jusqu'à l'affichage du texte. Dans ce but, nous avons utilisé *tcpdump* pour filtrer la communication TCP sur le serveur. Le Tableau 3.3 présente les résultats obtenus.

Le tableau montre les résultats obtenus pour plusieurs configurations. D'abord, nous avons mesuré le temps pour le système X classique : le client communique directement avec le serveur X. Les mesures en utilisant un *proxy* TCP simple (*rinetd*[20]), directement interposé dans la communication entre le client et le serveur, peuvent donner le délai introduit par la

[20] *rinetd* est un pseudo-serveur qui redirige des connexions TCP d'une adresse IP à une autre. [65].

Configuration de test	Résultat
client → serveur X (sur borneo)	0.0687s
client → *proxy* TCP simple → serveur X (sur borneo)	0.1235s
client → MUX → serveur X (sur borneo)	0.2903s
client → MUX → serveur X principale → serveur X secondaire	0.2964s

TAB. 3.2 – Les résultats obtenus pour l'application de type "Hello World !".

simple opération de capture et de redirection des paquets. La différence entre ces mesures et celles faites en utilisant le multiplexeur pour un seul serveur (la 3ème ligne) pourrait fournir le délai généré par l'analyse de messages X. L'introduction du deuxième serveur conduit aux pénalités de temps causées par le contrôle du multiplexeur sur les messages X (la 4ème ligne).

Selon ces résultats pour une application très simple, on peut remarquer qu'environ la moitié de délai imposé par le multiplexeur est du au filtrage des paquets du réseau. Bien que par rapport à la session X habituelle, celle dans laquelle est impliquée notre multiplexeur se déroule sans l'utilisation des extensions X, ce fait n'influence pas beaucoup les résultats de mesures étant donnée la simplicité de l'application.

Test 2 : extrait d'une session de Xterm

Le première application n'implique pas l'intervention de l'utilisateur. Pourtant il est intéressant de voir le délai dans le cas d'interaction avec l'utilisateur. Pour éviter la variabilité dont nous avons déjà parlée, nous avons choisi le test dans lequel l'entrée est indépendante de l'utilisateur. Celui-ci lance l'application *xterm*[21], appuie sur une touche de clavier et la touche reste appuyée jusqu'au remplissage de deux lignes entières de la fenêtre d'édition. Les mesures sont réalisées en utilisant *tethereal* pour filtrer la communication TCP sur borneo. Le Tableau 3.3 montre les résultats obtenus.

Configuration de test	Résultat
client → serveur X (sur borneo)	4.3454s
client → *proxy* TCP simple → serveur X (sur borneo)	4.3464s
client → MUX → serveur X (sur borneo)	4.3811s
client → MUX → serveur X principale → serveur X secondaire	4.3825s

TAB. 3.3 – Résultats des mesures réalisées pour xterm. Nous avons utilisé la notation *MUX* pour le multiplexeur.

Nous avons effectué les mêmes types de mesures comme dans le premier cas. Cette fois-ci, la section de communication X est indépendante des extensions.

[21] *xterm* est un émulateur de terminale en système X [74]

Dans le cas de *xterm*, les résultats des mesures montrent que l'application souffre des délais assez petits à cause du multiplexeur.

Test 3 : extrait d'une session xv

Nous avons ensuite choisi *xv*[22] pour mieux observer le comportement du multiplexeur dans le cas d'une session plus riche en messages.

Configuration de teste	Résultat
client → serveur X (sur borneo)	0.6620s
client → *proxy* TCP simple → serveur X (sur borneo)	0.7275s
client → MUX → serveur X (sur borneo)	1.5951s
client → MUX → serveur X principale → serveur X secondaire	1.7975s

TAB. 3.4 – Mesures réalisées sur xv pour le chargement d'une image 1024 X 768.

Concrètement, nous avons mesuré seulement le temps de chargement d'une image. Nous avons lancé *ethereal* juste avant d'appuyer sur le bouton de chargement ("*Load*") et l'avons arrêté une fois l'opération finie. Au niveau de la communication X, cette séquence est encadrée entre l'Événement "*ButtonPress*"[23] et le marquage du changement du curseur (ce qui montre la fin d'opération de chargement d'image), par la Requête "*ChangeWindowAttribute*"[24]. Cette fois, la section de communication X captée fait encore usage des extensions X.

Les premiers résultats (voir le Tableau 3.3) montrent le temps obtenu pour le chargement d'une image GIF 1024 X 768. Pratiquement, le délai d'une seconde n'est pas perceptible par l'utilisateur. Dans ce cas, on peut remarquer une différence de temps plus grande introduite par l'analyse des paquets X. Ceci s'explique par la quantité des informations qui doit être analysées par le multiplexeur X.

Ce test est aussi une bonne occasion pour mesurer le délai observé pendant le chargement d'images de grand format. Nous avons effectué les mêmes mesures pour charger la même image en format JPEG, 10752 X 2048. Le tableau 3.3 illustre les résultats obtenus.

Cette fois les différences de temps sont très grandes. Une explication de ce délai pourrait être l'utilisation de l'extension "*BIG-REQUESTS*"[25] par l'application, qui est déniée par notre multiplexeur. La grande différence entre le nombre des messages échangés pendant une session X sans et avec le multiplexeur (5366/9590 de messages) soutient cette hypothèse.

[22]Cet application échange plus de 6000 messages avec le serveur.

[23]Il s'agit de l'événement généré quand une touche de clavier est appuyée.

[24]Il s'agit de la Requête qui demande le changement d'un attribut d'une fenêtre.

[25]L'extension "*BIG-REQUESTS*" permet l'utilisation de Requêtes X qui dépassent la longueur de 262140 octets. Plus de détails peuvent être trouvées ailleurs [11].

Configuration de test	Résultat
client → serveur X (sur borneo)	1.1683s
client → *proxy* TCP simple→ serveur X (sur borneo)	2.5434s
client → MUX → serveur X (sur borneo)	32.4334s
client → MUX → serveur X principale → serveur X secondaire	32.8496s

TAB. 3.5 – Mesures réalisés sur xv pendant le chargement d'une image 10753 X 2048.

3.4 Travail futur

Le travail futur sur ce prototype pourrait se dérouler en trois directions. D'abord, une étude approfondie des comportements d'applications X est nécessaire pour implémenter la fonctionnalité complète du multiplexeur X. Sur le deuxième plan, il faut exploiter les fonctionnalités que le client de contrôle peut offrir. En plus, dans la version actuelle, le multiplexeur lance en exécution un client pour chaque participant en collaboration : cela pourrait alourdir le fonctionnement de la machine concernée. La continuation du travail sur ce client pourrait inclure l'étude de la possibilité de s'afficher simultanément sur plusieurs écrans. Troisièment, on doit prendre en compte la nouvelle version de la plate-forme de noeud actif et, également, mettre à profit la propriété d'adaptation au contexte qu'elle offre.

3.5 Conclusions

Nous nous sommes proposés d'étudier le partage d'affichage d'applications dans le contexte d'un environnement spontané de communication. Pour transformer le système de fenêtrage X en un système de partage de fenêtres, cette étude nous a conduit à l'idée d'intégration transparente d'un *proxy*. Sa fonctionnalité de multiplier des flots X, qui implique des opérations au niveau de paquets de réseau, se prête bien à l'emploi d'un service actif. Ainsi, cela devient une nouvelle fonctionnalité du réseau.

Pour valider cette idée, nous avons implémenté le multiplexeur X en tant qu'un service actif. La plate-forme générique de noeud actif présentée dans la section 3.2.1 s'y prête bien parce qu'elle permet le chargement dynamique des services. Ainsi, le multiplexeur est construit sur l'architecture de services actif proposée par cette plate-forme.

Bien qu'il reste encore du travail à faire, son comportement actuel est la preuve de la pertinence de nos idées. Il multiplie sur un serveur secondaire l'affichage de la plupart des applications testées sans pénalité de temps excessive pour utilisateur.

Simultanément, pour améliorer la flexibilité du système coopératif, nous pensons que la souplesse de la gestion du droit au contrôle joue un rôle important. Pour cette raison, nous avons conçu un client X spécial qui s'attache à l'interface graphique de chaque copie de l'affi-

chage de l'application et peut offrir un moyen de contrôle à son utilisateur. Sa fonctionnalité peut être étendue par d'autres fonctions utiles dans la collaboration dans un environnement spontané de communication, comme par exemple la communication entre les participants par des messages textuels.

Chapitre 4

Le partage d'écrans par le système VNC

4.1 Le système VNC

VNC (Virtual Network Computing) est un système d'affichage à distance (*remote display system*) [67]. Il permet d'avoir l'accès à l'Environnement Graphique de Bureau d'un ordinateur à distance. Il a été conçu en tant que logiciel libre pour fonctionner sur des plates-formes très diverses [47] [51]. Il est basé sur le protocole RFB (*Remote FrameBuffer*) [52] qui permet d'accéder à distance à la mémoire vidéo (*framebuffer*) d'un ordinateur.

Dans cette section, nous détaillons les caractéristiques du système VNC et du protocole RFB.

4.1.1 Architecture

L'architecture client-serveur du système VNC est présentée dans la Figure 4.1 : client-serveur :

– *le client* (ou *viewer*) est une partie du système qui gère le dispositif d'affichage de type bitmap (voir la Section 2.1.1) et ceux des entrées. Il a comme rôle principal l'affichage d'un tampon mémoire vidéo (*framebuffer*) d'une machine distante. Dans la conception du système VNC l'accent a été mis sur l'idée d'un client léger (*thin client*) : le protocole RFB prévoit peu de demandes concernant le client[1]. Une autre caractéristique du client VNC est l'absence de mémorisation des états intermédiaires de l'affichage : une fois déconnecté, on retrouve le même état de l'interface graphique à la reconnexion au

[1]Cela peut être vu même dans le nombre de messages réservés à destination du client (4 messages) par rapport à 7 messages réservés à destination du serveur.

même serveur.
– *le serveur* est un programme qui fournit le tampon mémoire vidéo et ses mises à jour. Sa principale caractéristique est le support qu'il offre pour l'accès concurrant de plusieurs clients.

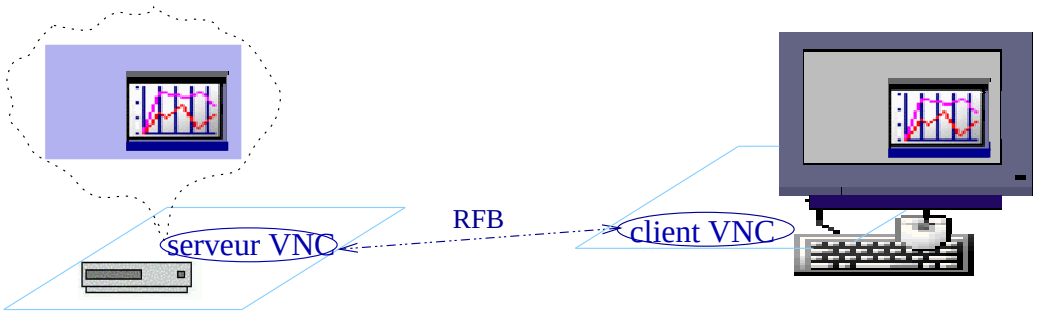

FIG. 4.1 – Architecture du système VNC. Le système permet d'afficher à distance un tampon mémoire vidéo (*framebuffer*) à l'aide du protocole RFB.

L'architecture du système VNC présente une particularité : contrairement à un système client-serveur classique où le client initie toujours la communication, il est possible pour un client VNC de fonctionner en mode d'écoute. Dans ce cas-là, c'est au serveur VNC d'initier la communication.

4.1.2 Le protocole de communication RFB

La communication entre un client et un serveur VNC se réalise conformément au protocole RFB. Il est relatievement simple au point de vue de ses messages et de sa sémantique.

La phase initiale de la communication entre un client et un serveur RFB inclut trois étapes. D'abord, il y a une négociation de la version du protocole entre le serveur et le client. Pour le moment, il existe trois versions du protocole qui sont valides. Le client doit répondre avec le numéro de la version qui est inférieure ou égale à celle fournie par le serveur. La deuxième étape concerne la sécurité : le serveur décide du type de la sécurité demandée. En plus des politiques de sécurité basées sur l'interdiction/permission totale d'accès et sur le mot de passe (nommé "authentification VNC"), la dernière version enrichit la sécurité et fournit même la possibilité de transmission chiffrée de données. En cas d'échec, le serveur fournit une explication. La dernière partie de la phase initiale est celle de l'initialisation du client et du serveur. Le client demande au serveur l'accès exclusif ou partagé à son tampon mémoire vidéo[2]. En ce qui concerne le serveur, celui-ci fournit des informations sur son tampon mémoire vidéo : sa dimension, le format de pixel, etc..

[2]Il y a des implémentations de serveurs VNC qui offrent des options de configuration concernant l'accès exclusif demandé par des clients : ainsi, il peut être configuré pour ignorer ce type de demandes. Un tel exemple de serveurs est Xvnc conçu pour le système de fenêtrage X.

FIG. 4.2 – Le format général d'un message RFB. Il commence par un octet qui précise le type de message suivi par des données qui diffèrent d'un type de message à l'autre.

Après la phase initiale de communication, les messages échangés ont un format général illustré dans la Figure 4.2. L'échange des messages entre le client et le serveur suit en général le modèle d'opérations indiqué dans la Figure 4.3. Le protocole RFB inclue deux grandes parties que nous présentons ci-dessous :

Le protocole d'affichage

La partie affichage du protocole est construite sur la base d'une seule primitive graphique : "afficher un rectangle de pixels à une position donnée (x,y)". La puissance du protocole consiste en la variété des codages des pixels qui peuvent utilisés à destination du client. Le protocole propose plusieurs codages qui offrent différents degrés de compression, par exemple, la transmission des données de pixels sous forme d'un tableau (le codage *raw encoding*), la transmission d'une aire de pixels sous forme de références à des rectangles contenants la même valeur de pixels (le codage *RRE encoding*) ou un codage qui permet que chaque rectangle soit considéré comme un ensemble de tuiles (*tiles*), chaque tuile ayant son propre codage (le codage *hextile*). La dernière version du protocole propose également un codage qui inclut la compression de données en utilisant zlib[3] (le codage ZRLE).

Une séquence de rectangles est la base d'une mise à jour de l'affichage du tampon mémoire vidéo sur le client qui représente les modifications survenues dans le tampon. La mise à jour du tampon mémoire vidéo est toujours commandée par le client : ses modifications ou le tampon entier sont envoyées comme réponse à la demande explicite du client. Le serveur peut envoyer une telle mise à jour comme une réponse à plusieurs demandes différentes du client.

Un autre aspect du protocole qui contribue à sa flexibilité est le format de pixels : les deux possibilités sont présentées dans la Figure 2.6 du chapitre Chapitre 2.

Le protocole qui gère des entrées

La gestion d'entrée est basée sur le modèle standard d'une station de travail équippée d'un clavier et d'une souris à plusieurs boutons. Les événements d'entrée sont envoyés quand

[3]http ://www.gzip.org/zlib.

l'utilisateur appuie/relâche un bouton de la souris ou une touche du clavier ou quand il déplace la souris.

Le protocole prévoit également l'utilisation d'autres dispositifs d'entrée non standard, dont les événements peuvent être traduits par l'entrée au clavier ou à la souris.

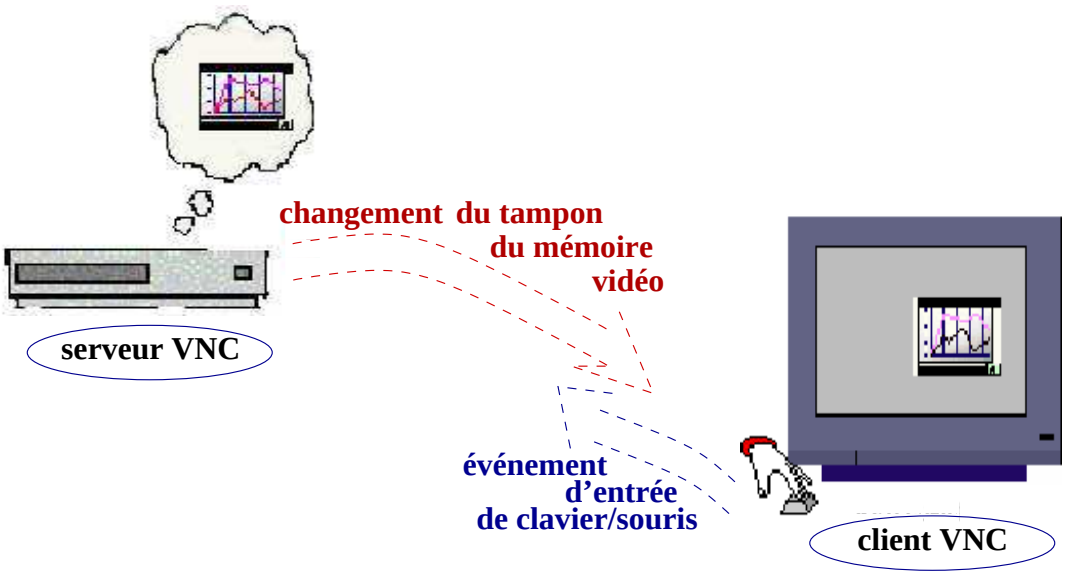

FIG. 4.3 – Vue d'ensemble sur des opérations qui se déroulent au cours d'une session de communication VNC. On peut observer qu'en général le serveur envoie des changements du tampon mémoire vidéo, tandis que le client envoie des événements d'entrée.

4.1.3 Extensions du protocole RFB

L'extension du protocole RFB peut être réalisée par l'ajout des nouveaux types de codage. En plus des messages compris dans le noyau du protocole, la dernière version du protocole permet aux clients de demander certains pseudo-codages (*pseudo-encoding*), qui peuvent être pris ou non en compte par le serveur. Les pseudo-codages concernent pour le moment la capacité d'un client à dessiner localement un curseur et celle de faire face au changement dynamique de la dimension du tampon mémoire vidéo. Le serveur peut demander la réalisation de ces opérations en utilisant le message de mise à jour du tampon, qui inclut des informations nécessaires dans le format du pseudo-codage spécifique.

4.1.4 Sécurité

Comme nous avons déjà mentionné, dans sa première version le protocole offre trois types de sécurité. La sécurité basée sur le mot de passe (le type de sécurité nommé "l'authentifi-

cation VNC") consiste en un défi sous forme d'un tableau d'octets aléatoire envoyé par le serveur, qui doit être chiffré avec l'algorithme DES par le client en utilisant le mot de passe fourni par l'utilisateur et envoyé comme réponse. Ainsi, le mot de passe n'est pas envoyé en clair au serveur. Par contre, les données concernant la mise à jour d'écran sont envoyées en clair. La connexion par SSH peut résoudre ce problème par l'acheminement sécurisé de la communication RFB. Le mécanisme est illustré dans la Figure 2.8 pour une session X (Section 2.1.8). Il y a des distributions du système, comme par exemple, TightVnc [3] ou RealVNC (la version 3.3) qui permettent l'acheminement automatique de la communication RFB par SSH. La nouvelle version du protocole RFB mentionne d'autres types de sécurité, mais aucun d'entre eux n'est encore défini.

Une nouvelle distribution du système VNC (*VNC Enterprise Edition*) inclut dans l'étape d'authentification du serveur l'échange des clés publiques/privés comme phase de démarrage d'une session RFB chiffrée[4] [47]. La même distribution accepte le système d'authentification de la plate-forme hôte.

Concernant le contrôle d'accès, les nouvelles versions des serveurs VNC[5] incluent une option qui permet à ses utilisateurs d'accepter ou de rejeter les demandes de connexions des clients. Ces versions sous certaines plates-formes, comme par exemple sous Windows, offrent également le mécanisme de contrôle d'accès à base de l'adresse source de la connexion. Il y a également des *patches* qui complètent ce mécanisme dans le serveur Xvnc sous Unix [67].

4.1.5 Comportement du système VNC en réseau

Le protocole RFB a été conçu pour fonctionner sur n'importe quel couche de transport qui offre une communication fiable. En pratique, le système VNC utilise le protocole de transport TCP. L'accès au serveur VNC peut se faire par une connexion à un port TCP/IP donné en utilisant un client autonome, ou par une connexion à un pseudo-serveur Web en utilisant un navigateur qui supporte Java. Les dernières versions du système VNC acceptent les deux types de connexions sur le même port TCP/IP.

Le principal avantage du système VNC est le support de l'adaptation au contexte du réseau grâce aux caractéristiques du protocole RFB (voir la Section 4.1.2) :

- la variété des codages pour l'envoi de la mise à jour du tampon mémoire vidéo et la possibilité que chaque rectangle de pixels soit transmis en utilisant un schéma de codage différent. Ceci favorise le choix du meilleur type de codage pour un contenu de mémoire vidéo donné ou pour la bande passante réseau disponible ;
- l'opération de la mise à jour de l'écran est conduite par le client : plus le client ou le réseau sont lents, moins la mise à jour est fréquente. On ignore ainsi des états de transition dans le tampon mémoire vidéo en réduisant le trafic réseau généré par la session VNC et les tâches d'affichage pour le client.

[4] http ://www.realvnc.com/pipermail/vnc-list/2005-March/050019.html.

[5] Nous nous référons aux versions 4 de la distribution libre RealVnc.

Les dernières versions du système VNC (par exemple la VNC Enterprise Edition) incluent des mécanismes de détection de la bande passante réseau disponible.

4.1.6 Conclusions

Dans ce chapitre nous avons présenté le système d'affichage à distance VNC. Nous résumons ses principales caractéristiques :

- *le système d'affichage à distance qui opère au niveau de la mémoire vidéo* : cet aspect lui confère l'indépendance de la plate-forme hôte ; le système peut être déployé sur n'importe quelle entité réseau équipée d'un processeur et d'un tampon mémoire vidéo ;
- *le système basé sur un client léger* : ceci facilite le déploiement des clients VNC pour une grande diversité de plates-formes des systèmes d'exploitation ou des systèmes de fenêtrage ;
- *l'adaptabilité aux conditions réseau* : ceci permet de l'utiliser au-dessus de divers réseaux ;
- *l'adaptabilité à la capacité de l'hôte client* : ainsi, on peut développer des clients pour des dispositifs ayant des capacités réduites.
- *l'acceptation d'accès concurrent de plusieurs clients au serveur* : cette caractéristique permet le partage du même environnement graphique par plusieurs clients.

À l'heure actuelle, le système VNC opère sur des plates-formes diverses, à partir d'Unix[6] et de Windows, jusqu'aux dispositifs de type Palm et des téléphones cellulaires possédant Java [59]. La figure 4.4 illustre la diversité des plate-formes.

4.2 Les caractéristiques coopératives du système VNC

L'acceptation d'accès concurrent à un serveur donne au système VNC un caractère coopératif : il est possible d'exporter l'affichage d'un écran vers plusieurs clients. Dans cette situation, VNC devient un système de partage d'écran qui permet également le partage d'applications. Dans cette section, nous présentons le contexte coopératif offert par le système VNC et les contributions qui ont essayé de l'enrichir.

4.2.1 Le partage de l'environnement graphique

Le contrôle d'accès se fait par la configuration du côté client ou serveur. Le client accepte deux options de partage de l'environnement graphique :

- *partagé* (*shared*) : on demande au serveur le partage de l'environnement graphique avec d'autres clients ;

[6]Le système VNC est même inclu dans des distributions libres de Linux, comme Suse et Fedora.

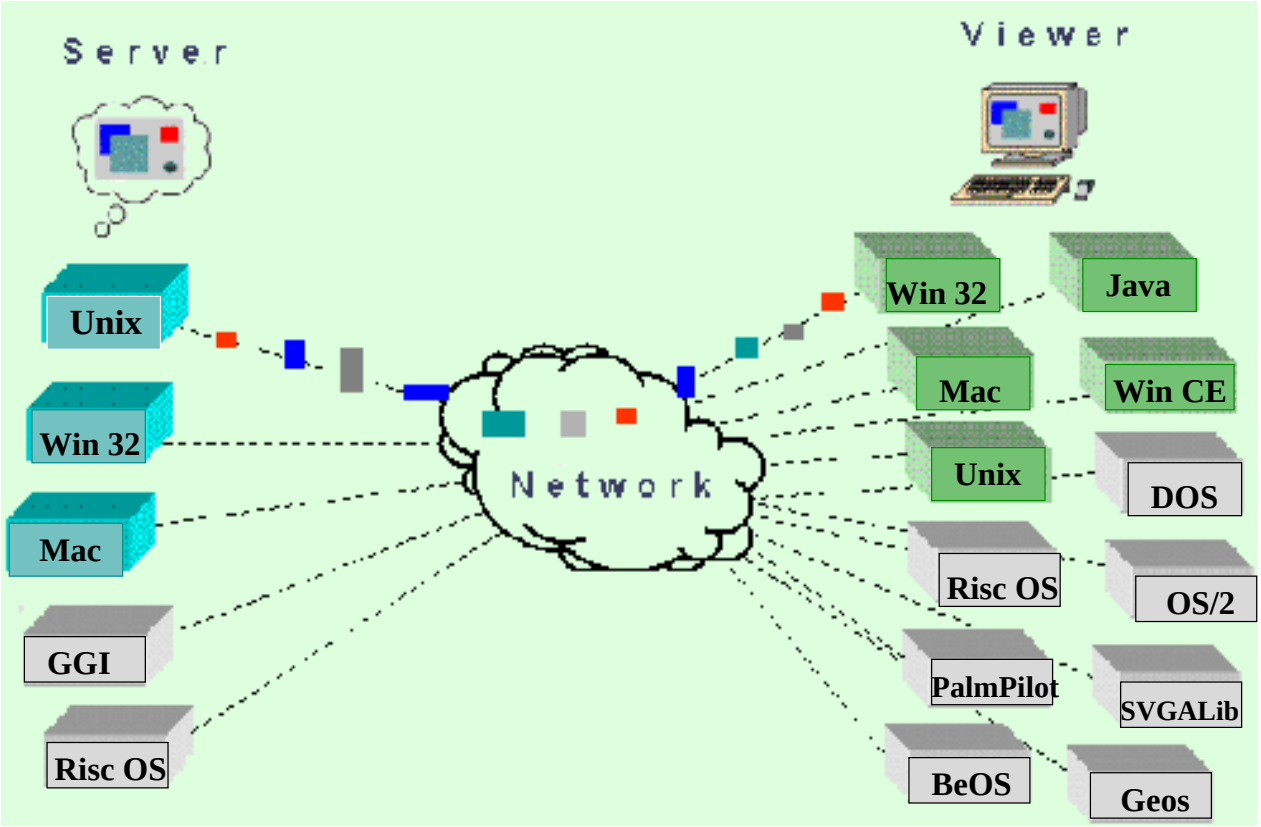

FIG. 4.4 – La diversité des plate-formes qui supportent VNC (dessin repris du RealVnc [47]).

– *seulement la visualisation* : l'accès à l'environnement graphique se fait en mode visualisation, c'est-à-dire, sans envoyer d'événements d'entrée au serveur.

Une autre option spéciale est "la mise en attente" d'un client (*listen*) : le client attend la connexion d'un serveur. Dans ce cas, c'est le serveur qui initie la connexion à un ou plusieurs clients.

Le paramétrage du client concernant l'accès au serveur peut être inhibé par la configuration du serveur : il y a des implémentations de serveur VNC (par exemple, le serveur Xvnc, la version 4.1) qui acceptent en option la possibilité d'admettre ou non le partage de l'environnement graphique, et ceci en dépit des demandes des clients.

En fait, les versions 4 de la distribution libre RealVNC du système VNC incluent des serveurs plus riches en options. Elles introduisent également un petit programme "vncconfig" utile pour configurer dynamiquement une instance de serveur : il l'aide à se connecter à un client en mode "à l'écoute" et il change également des valeurs de paramètres. Parmi ces paramètres, les plus intéressants dans le contexte coopératif sont ceux qui permettent d'accepter ou non les événements d'entrée envoyés par des clients.

Sur la plupart des plates-formes, le système VNC exporte l'environnement graphique de la machine sur laquelle se exécute le serveur. Sous Unix on peut trouver en plus un type de serveur particulier : pour chaque connexion d'un client, il crée un nouvel environnement graphique. Donc, nous avons un serveur sous Unix que nous appellerons "de type xOrfbserver"[7] qui exporte l'environnement graphique et un serveur "de type Xvnc" qui crée un nouvel environnement pour chaque connexion RFB reçue[8].

4.2.2 Contributions aux aspects coopératifs de VNC

D'abord, l'aspect coopératif de VNC a été enrichi par son apparition dans des environnements coopératifs, comme par exemple VRVS [66], un système de vidéo-conférence qui l'inclut comme outil. L'utilité de VNC comme outil de collaboration a conduit à son introduction dans des environnements graphiques tels que Gnome ou KDE [34].

Simultanément, son statut de logiciel libre a encouragé le développement des nombreuses contributions [67]. Certaines d'entre elles sont axées sur l'enrichissement des aspects coopératifs. Une modification du serveur WinVNC permet le partage d'une seule fenêtre d'une application. SharedAppVNC [2] permet également le partage d'une seule fenêtre au choix sous Unix. Cette solution implique aussi des modifications du système VNC. Rfbproxy et rfbplaymacro sont deux programmes qui permettent d'enregistrer et de jouer des sessions

[7] Il y a plusieurs versions de ce type de serveurs : x0rfbserver, x0vncserver, X11VNC ou des serveurs Xfree86 qui ont ajouté une bibliothèque dynamique le transformant en un serveur VNC.

[8] Le serveur Xvnc peut être considéré comme un serveur virtuel X qui présente un faux écran physique aux clients VNC. Dans le fichier $HOME/.vnc/xstartup on peut préciser quelles sont les applications X à lancer dans ce nouvel environnement graphique.

VNC et donc d'apporter au système des caractéristiques de collaboration asynchrone. VNC-Proxy [70] permet d'interposer un proxy d'une façon transparente entre des clients et un serveur VNC. Le proxy accepte des connexions des clients et les achemine vers un serveur selon des indications précisées dans un fichier de configuration. VNCReflector [17] est un proxy qui peut commuter dynamiquement la session de communication VNC entre plusieurs clients en préservant la connexion aux clients. Il supporte également le raccordement renversé serveur-client.

Une des contributions au système VNC est sa distribution libre nommé TightVNC [3]. Parmi les améliorations apportées au système de base, il y quelques aspects coopératifs qui permettent d'avoir l'accès au serveur en mode partagé ou seulement en visualisation selon le mot de passe utilisé. La version conçue pour Windows supporte le transfert de fichiers entre des machines à distance. La version sous Unix inclue la possibilité de contrôler l'accès au serveur VNC sur la base de l'adresse IP.

TightVNC a été complètement transformé en un système coopératif par un *patch* nommé *Collaborative VNC*[9] [1]. Les deux fonctionnalités importantes que le *patch* apporte sont :

- des flèches de curseur de différentes couleurs pour les participants à un partage de l'environnement graphique ;
- un système de contrôle de droits d'entrée *floor* (voir la Section 3.1) qui permet la prise de contrôle de l'environnement graphique à distance par un seul participant.

Nous décrivons ici ce système de contrôle du droits d'entrée pertinent par rapport à notre travail. Un participant à une session coopérative Collaborate Vnc peut avoir deux rôles : un simple collaborateur, ou maître. Une session doit avoir un seul maître. En tant qu'simple collaborateur, si un participant a le droit d'entrée *floor*, il peut le céder à quelqu'un d'autre ou le relâcher. Un participant qui n'a pas de contrôle peut le demander et l'obtenir si le possesseur du droit d'entrée *floor* est inactif pendent une durée (réglable par la configuration). Le maître peut obtenir le contrôle et le céder à qui il veut à tout le moment. Nous remarquons qu'il n'y a pas de possibilité de changer la politique de contrôle de droit d'entrée *floor* pendant la session. Le contrôle de droit d'entrée *floor* se fait en cliquant sur le bouton de souris ou par un menu de type *pop-up* qui apparaît en appuyant sur la touche "F8". Cette solution offre un support coopératif riche, mais seulement pour la distribution TightVNC du système qui a installé ce *patch*.

Mitre Desktop Collaborative Workspace [38] introduit des nouvelles caractéristiques coopératives concernant la gestion de la session en VNC par une extension du protocole RFB consistant en l'introduction transparente de deux *proxy* dans l'architecture de VNC. Un autre travail [36] introduit des aspects de collaboration asynchrone dans le système. Cette proposition se base sur une architecture client-proxy-serveur et s'appuie sur la communication multipoint. Elle offre également deux politiques de contrôle de droit d'entrée *floor*.

[9] Cette contribution a été faite au cours de l'année 2004 après l'implémentation de notre prototype.

4.2.3 Conclusions

Dans cette section, nous avons analysé les caractéristiques coopératives du système VNC qui apportent le partage de l'environnement graphique à distance et un support minimal du contrôle d'accès. Nous avons également analysé des contributions apportées au système du point de vue de l'amélioration de son aspect coopératif.

Chapitre 5

Support coopératif pour le système VNC

Nous avons vu dans la description du système VNC (Section 4.2.1), qu'il offre un cadre coopératif relativement limité. Certains travaux complémentaires ont apportés des améliorations sur l'aspect coopératif (Section 4.2.2), mais il manque toujours de la flexibilité d'adaptation à des contextes différents de collaboration, en particulier ceux liés aux environnements de communication spontanés.

Dans ce chapitre, nous utilisons des propriétés du système VNC pour l'enrichir avec un support qui rend la collaboration plus flexible. Notre solution est basée sur un proxy dynamiquement contrôlable, intégré d'une façon transparente dans le système VNC. Ainsi, la nouvelle architecture du système de partage d'écran VNC inclue trois éléments principaux :

- le proxy nommé *Newkey*,
- le contrôleur à distance,
- le système VNC (le serveur et les clients VNC).

Les trois éléments sont mis en évidence dans la Figure 5.1. Nous présentons dans la suite les nouveaux composants introduits dans l'architecture du VNC. Nous décrirons également une proposition de la gestion du contrôle de droit d'entrée *floor* pour le système VNC enrichi. La présentation introduira aussi des scénarios d'utilisation de ce système. Dans notre travail, nous utiliserons le terme de "session de coopération VNC" (ou "session", "session coopérative du partage d'écran") pour nous référer à l'ensemble des connexions TCP/IP des clients à un serveur VNC, ainsi qu'à des informations de collaboration. Chaque session est identifiée d'une façon unique et attachée à un propriétaire et à un groupe de participants caractérisés par des propriétés différentes.

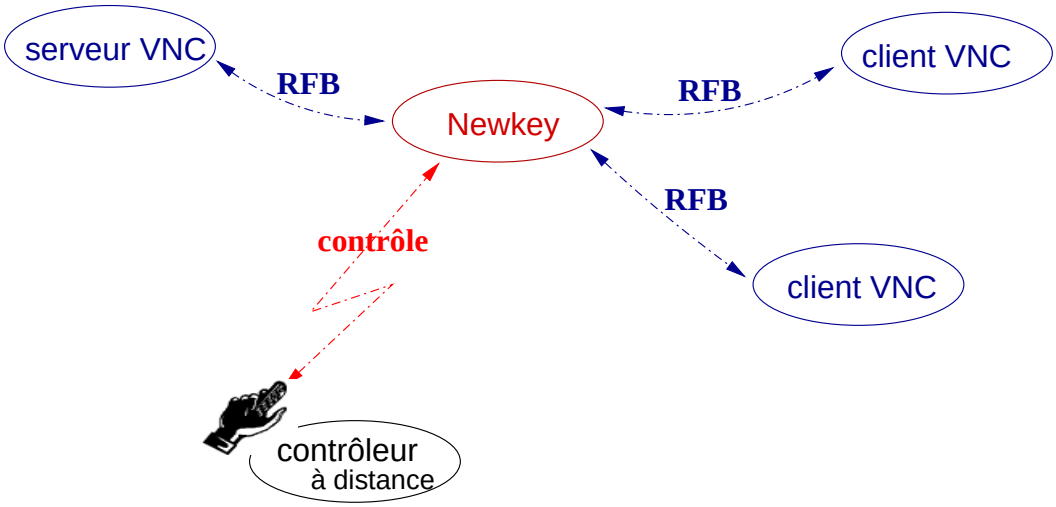

FIG. 5.1 – Architecture du système coopératif de partage d'écran VNC enrichi.

5.1 Newkey

Newkey est un proxy introduit d'une façon transparente dans le système VNC. D'un côté, il attend des connexions venant des clients VNC et, d'un autre côté, des commandes de contrôle.

La caractéristique du proxy de pouvoir être contrôlé dynamiquement permet sa programmation pour la mise en place des sessions de collaboration. Dans ce sens, il a deux grandes fonctionnalités. Derrière chaque fonctionnalité, Newkey peut offrir deux mécanismes différents de traitement d'un flot RFB. Ci-dessous, nous présentons ces caractéristiques du prototype. Des scénarios mettent en évidence leur utilité pratique[1].

5.1.1 Fonctionnalités de Newkey

Newkey exploite l'existence de l'état "à l'écoute" (*listen*) des clients VNC. Ainsi, le proxy peut être programmé pour mettre en place des sessions de travail coopératif initiées, d'un côté, vers des serveurs VNC, d'un autre côté, vers des clients VNC en mode "à l'écoute"[2].

Ainsi, d'un côté, le proxy se place dans la communication entre des composants du système et réalise la connexion des clients à un serveur conformément au contrôle reçu. Cette fonctionnalité peut-être utile dans le cadre d'une réunion par exemple : à une école d'été,

[1]Dans nos scénarios nous supposons que les dispositifs de participants aux sessions de coopération communiquent entre eux spontanément en mode ad-hoc ou par un point d'accès.

[2]Nous nommons ces types de sessions "connectToServer" (*connexion à un serveur*) et respectivement "connectToClient" (*connexion à un client*).

un conférencier peut contrôler Newkey pour configurer une session de partage de son écran – dans ce but, il associe un certain port TCP[3] sur sa machine de sorte que les participants aient la possibilité de s'y connecter en utilisant leurs clients VNC pour suivre la présentation sur les écrans de leurs ordinateurs.

D'un autre côté, le proxy joue le rôle d'un faux serveur : il aide à initier des connexions de la part d'un serveur donné vers des clients qui sont en mode "à l'écoute". L'opération se passe d'une façon transparente pour les composants du système VNC. Un scénario d'utilisation de cette fonctionnalité pourrait se placer dans le contexte d'enseignement : un professeur fait une démonstration de l'utilisation d'un logiciel à ses étudiants. Il utilise Newkey et le configure pour envoyer l'affichage de l'écran de son ordinateur aux clients VNC en mode "à l'écoute" de ses étudiants.

Les deux fonctionnalités peuvent être combinées dans des situations de collaboration plus complexes. Le scénario suivant montre une telle situation. Supposons qu'on utilise le partage d'un écran pour des travaux pratiques d'étudiants qui nécessitent l'utilisation d'ordinateurs. Chaque étudiant, qui le souhaite, peut configurer Newkey pour diriger la connexion du client du professeur vers son serveur : les demandes d'étudiants sont traitées ainsi dans l'ordre chronologique. À chaque fois que le professeur se connecte au proxy en utilisant son client, celui-ci peut apporter son aide à l'étudiant qui la demande par le partage de l'écran de l'étudiant[4]. Si l'aide et les observations faites à un étudiant sont intéressantes pour les autres, le professeur peut envoyer l'affichage de l'écran aux clients VNC "à l'écoute" des autres étudiants en utilisant Newkey[5].

5.1.2 Mécanismes de traitement des flots RFB

Derrière ces deux fonctionnalités de Newkey, il y a deux mécanismes de traitement des flots RFB :

Newkey - "routeur" de flot RFB

Newkey peut jouer le rôle de "routeur" de flot RFB : il achemine des connexions reçues vers les destinations précisées d'avance. Donc, le proxy est programmé pour acheminer les connexions venant de certaines machines vers des destinations précises. Les connexions venant des machines qui n'ont pas été programmées, sont refusées. Dans le cas d'une configuration d'une session "connectToServer", les connexions venant de clients sont dirigées vers

[3]Dans le contexte de ce mémoire, nous y référons aussi simplement par "port".

[4]À cette occasion, nous remarquons une possible utilisation du partage d'affichage d'un écran : l'utilisation du partage de l'interface graphique d'un éditeur texte comme moyen de communication entre deux personnes par des messages textuels.

[5]Il est possible d'utiliser plusieurs clients VNC sur une machine au même moment et en différents modes, c'est à dire "à l'écoute" ou par lancement d'exécution.

un serveur précisé, et dans l'autre cas (de session "connectToClient"), les connexions sont orientées du serveur vers des clients spécifiés. La Figure 5.2 illustre ce mécanisme.

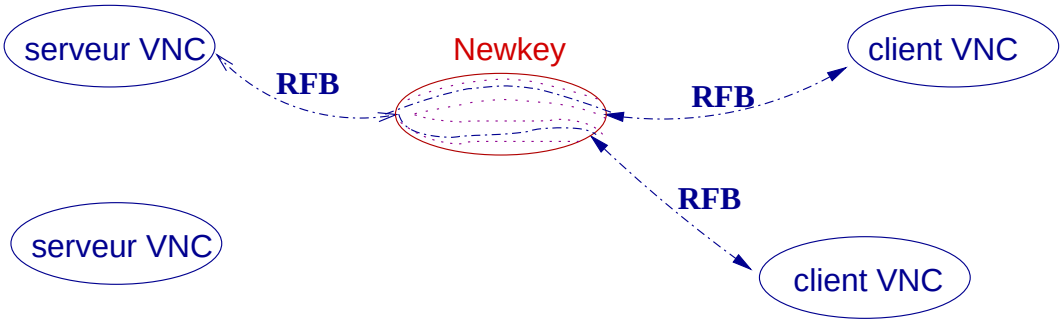

FIG. 5.2 – Le mécanisme de l'acheminement de flot RFB.

Pour s'assurer de la possibilité de partage de l'accès au serveur VNC du point de vue du client, Newkey met toujours une valeur positive à l' indicateur de partage (*shared flag*) du message d'initialisation du client (voir Section 4.1.2).

Comme exemple de scénario, nous considérons la présentation d'une leçon à l'école illustrée par un diaporama. Le professeur peut programmer son Newkey pour acheminer toutes les connexions venant des clients VNC de ses élèves vers son serveur. De cette manière, les élèves peuvent suivre la présentation sur leurs écrans d'ordinateur.

Newkey - multiplexeur RFB

Nous avons vu dans le paragraphe précédent que Newkey peut offrir un cadre organisé pour le partage d'affichage d'écran. Simultanément il met à profit la possibilité d'utiliser des clients en mode "à l'écoute". Cependant, le mécanisme proposé n'offre pas de support coopératif aux serveurs Unix de type "Xvnc"[6] (voir Section 4.2.1). En même temps, nous pensons que dans certains cas, une délimitation plus claire des sessions de collaboration concernant la connexion à Newkey est nécessaire. Dans ce contexte, nous proposons un deuxième mécanisme de traitement de flots RFB. Ainsi, Newkey peut jouer le rôle de multiplexeur de flots RFB : il peut multiplier un flot RFB vers plusieurs clients VNC. La Figure 5.3 illustre ce mécanisme.

Si le mécanisme de multiplication de flot RFB constitue la base de la fonctionnalité qui permet l'initialisation de la connexion à un serveur, Newkey associe un port de communication TCP à chaque session. Ainsi, la connexion d'un client VNC sur le port associé peut lui donner accès à une certaine session suite à la duplication du flot RFB du serveur. Dans le cas de la fonctionnalité qui permet l'initialisation des connexions aux clients qui sont en mode "à l'écoute", la simple configuration de la session par la programmation de Newkey

[6]Nous rappelons que ce type de serveur est typique sous Unix. Il crée un nouvel environnement graphique de bureau à chaque connexion d'un client.

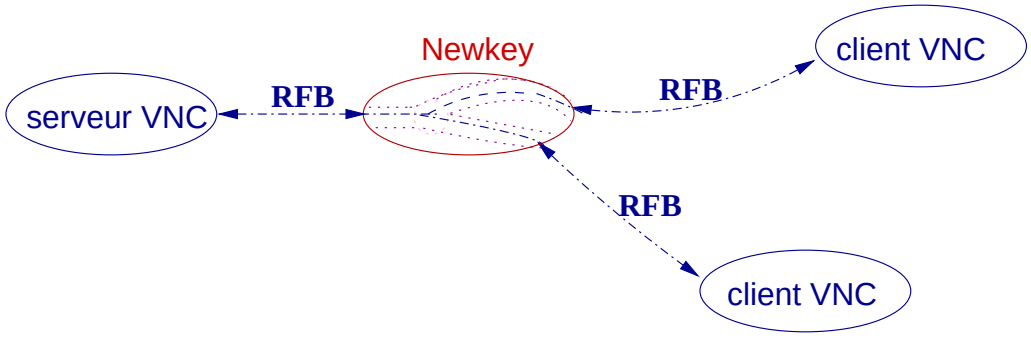

FIG. 5.3 – Le mécanisme du duplication de flot RFB.

implique sa mise en place. Dans cette circonstance, le proxy initie la communication avec les clients, puis duplique le flot RFB du serveur vers les clients.

Nous remarquons que par la duplication du flot RFB, plusieurs utilisateurs peuvent avoir l'accès partagé au même moment au même environnement graphique créé par une nouvelle connexion à un serveur de type Xvnc. Le scénario suivant met en évidence cette fonctionnalité. Nous prenons comme exemple le contexte de l'environnement scolaire concernant les travaux pratiques d'informatique. Le professeur peut distribuer des projets à plusieurs groupes de travail. On dispose d'un serveur Unix sur lequel il utilise des serveurs VNC de type "Xvnc" (voir la Section 4.2.1) qui écoutent sur des ports différents. Pour chaque serveur VNC, il associe un port correspondant à un projet sur Newkey. De cette façon, par l'utilisation de Newkey, les membres d'une équipe peuvent travailler ensemble sur leur projet en bénéficiant de leur propre environnement graphique, tout en ayant accès aux mêmes ressources sur le serveur. Le professeur pourrait suivre et intervenir dans leurs travaux en utilisant des clients VNC connectés aux ports adéquats.

5.2 Contrôleur à distance

Dans notre vision, l'adaptation de Newkey aux différentes situations de collaboration peut se réaliser par son contrôle dynamique à distance. Dans ce but, notre proxy offre une interface de contrôle. Ainsi, le contrôle peut être réalisé par n'importe quelle entité en réseau qui connaît le protocole accepté par Newkey. Dans cette section, nous présentons ce protocole de contrôle *NewkeyCP*.

5.2.1 Protocole de contrôle *NewkeyCP*

Dans la conception du protocole de contrôle de Newkey, nous nous sommes orientés vers quatre grandes directions. Dans l'Annexe, Section C, nous décrivons le protocole *NewkeyCP*

en détail. Nous présentons ici les quatre principaux aspects du protocole.

Connexions de type "connectToServer"

À la réception d'un tel type de contrôle, Newkey prépare une session de partage d'environnement graphique pour le serveur VNC demandé. Le comportement du proxy dépend du mécanisme demandé pour la session. Le protocole supporte les deux mécanismes décrits précédemment.

Dans le cas d'une session basée sur la duplicaiton d'un flot RFB, Newkey lui associe un port TCP. Ainsi, tous les clients VNC qui veulent participer à cette session de partage d'affichage d'écran doivent se connecter au *proxy* à ce port. Une fois une connexion réalisée, la communication est dirigée automatiquement vers le serveur demandé.

Dans l'autre cas, celui d'une session basée sur l'acheminement des flots X, Newkey reçoit une liste de clients et l'indication d'un serveur VNC. Puis, il achemine automatiquement toutes les connexions de ces clients vers le serveur indiqué. Ce mécanisme convient aux serveurs qui exportent des environnements graphiques. Une telle session pour un serveur Unix de type Xvnc n'offre pas un cadre coopératif.

Le protocole prend en compte également la configuration d'une seule connexion d'un client à un serveur donné.

Connexions de type "connectToClient"

Ce contrôle concerne les clients en mode "à l'écoute". Nous avons nommé ce type de contrôle "initialisation" parce que par celui-ci Newkey offre un support aux serveurs pour initier des connexions vers des clients.

Dans cette situation, le proxy joue un rôle de pseudo-serveur pour les clients et il apparaît comme un client pour le serveur. Ainsi, ce type de session convient pour tous les types de serveurs indépendamment du mécanisme qui se trouve à sa base. Quand le proxy reçoit un message de ce type de contrôle, il initie les connexions demandées par le serveur aux clients VNC spécifiés.

Le protocole prend en compte également l'initialisation d'une seule connexion d'un serveur à un client donné.

Les primitives de contrôle du droit d'entrée (*control floor*)

Dans ce contexte, chaque participant à une session gérée par Newkey attache le droit d'entrée ou pas à l'environnement graphique à distance. Pour la gestion de cette caractéristique, le protocole inclue deux fonctions :

– *setFloor* qui a la signification de donner le droit d'entrée à un client,
– *freeFloor* avec la signification de retirer le droit d'entrée à un client.

Tout trafic RFB de clients qui passe par Newkey est filtré : en fonction de l'état attaché à un client, ses messages sont, ou ne sont pas envoyés au serveur.

Administration de Newkey

Ce type de contrôle est nécessaire pour administrer intérieurement les sessions et la configuration du proxy. Nous pensons à des modifications concernant des participants à une session, leurs caractéristiques ou le contrôle d'accès. Pour le moment, le protocole inclue un seul message qui fournit l'affichage des informations sur des sessions qui sont gérées par le proxy (voir l'Annexe, Section C).

5.3 Mécanisme de contrôle d'événements d'entrée

À partir des scénarios présentés dans la description de Newkey (Section 5.1), nous pouvons remarquer que le contrôle d'événements d'entrée fournit par le système VNC n'est pas toujours suffisant. Par exemple, dans le cas d'une présentation PowerPoint qui est dupliquée sur d'autres écrans, la configuration de certains paramètres (dans les dernières versions de la distribution libre RealVNC du système) permet de bloquer les événements d'entrée du côté de serveur (voir Section 4.2.1). Mais, dans d'autres contextes, par exemple quand un professeur fait une démonstration d'un logiciel à ses étudiants, il devrait pouvoir donner sélectivement le droit d'entrée à l'étudiant de son choix. En plus, dans le cas du travail en mode coopératif en partageant l'affichage de l'écran, chaque participant pourrait avoir le droit "à la souris" à la demande : mais, si plusieurs étudiants prennent le contrôle au même moment, cela pourrait mener à la confusion. Dans certains cas, le groupe de travail pourrait avoir un chef qui gère ce contrôle. Dans le scénario présenté, le professeur devrait avoir également la possibilité d'intervenir à sa volonté dans les travaux des étudiants et de prendre le contrôle exclusif d'entrée. Donc, à chaque type de travail coopératif qui utilise VNC correspond pratiquement sa propre politique de gestion des événements d'entrée.

Ce problème de la politique de gestion d'événements d'entrée appartient au domaine du contrôle de droit d'entrée (*floor*) expliqué déjà plus haut (Section 3.1) dans le contexte du système de fenêtrage X. Dans ce contexte, il y a des travaux qui essaient d'enrichir le système VNC avec le contrôle de gestion des événements d'entrée, par exemple, Mitre Desktop Collaborative Workspace [38], [36] ou Collaborative VNC [1], mais tous fournissent un nombre limité de politiques de droit d'entrée *control floor*. Notre travail fournit le mécanisme pour ce contrôle et laisse la liberté d'implémenter n'importe quelle politique sur ce support[7].

[7]L'idée de la séparation de mécanisme de la politique de droit d'entrée *control floor* est déjà connue dans

Le mécanisme s'appuie sur deux fonctions "setFloor" et "freeFloor". Une entité quelconque du réseau peut implémenter des politiques de contrôle et les mettre en pratique en utilisant ces deux fonctions (voir la Figure 5.6). L'existence de cette entité n'est pas essentielle pour le fonctionnement du système de partage, car Newkey accepte implicitement la politique de liberté totale pour le droit d'entrée, qui est implicite dans le système VNC. Ainsi, à chaque instant le trafic RFB venant de clients est filtré conformement au protocole de contrôle qui est valide.

5.4 Implémentation

Dans cette section, nous présentons l'implémentation des composants présentés dans la section antérieure et d'un service de contrôle qui fournit des protocoles de droit d'entrée *control floor* selon le mécanisme décrit. Nous avons choisi le langage Java pour l'implémentation.

5.4.1 Newkey

Nous avons construit Newkey à partir de VNCProxy[8] [70], un proxy interposé d'une façon transparente entre un client et serveur VNC. Le programme est implémenté en Java. Sa fonctionnalité principale est d'acheminer des connexions de clients vers un serveur conformément à une configuration précisée dans un fichier. Le fichier fournit également le support pour limiter l'accès au proxy par l'adresse IP ou par le domaine de noms.

Nous avons transformé ce proxy pour le rendre contrôlable dynamiquement à distance. Pour le moment, notre prototype accepte seulement la version 3.3 du protocole RFB. Une fois lancé en exécution, il attend des connexions de clients sur le port TCP 5903 et des commandes de contrôle sur le port 5800. Le protocole de contrôle permet de lui associer également d'autres ports pour recevoir des connexions RFB. Newkey fonctionne indépendamment de la machine du client ou du serveur VNC.

D'un point de vue technique, toutes les informations concernant les sessions enregistrées sur Newkey sont gardées dans des structures Java de type "ArrayList". Newkey fonctionne de manière multi-fils (*multi-threading*) : à chaque session, le programme assigne un fil d'exécution séparé. En plus de VNCProxy, nous avons également fait appel à l'algorithme de chiffrement mis à disposition par la version libre de VNC.

Si la connexion d'un client échoue, Newkey utilise le message-réponse de la phase initiale de communication du protocole RFB, qui informe le client du motif de l'échec de sa connexion.

---

le domaine de TCAO (Travail Coopératif Assisté par Ordinateur) (à voir, par exemple, [28], [53]).

[8]VNCProxy est un logiciel libre sous licence de type BSD.

5.4.2 Contrôleur à distance

Au début, nous avons implémenté un contrôleur à distance qui offre une interface en ligne de commande. La Figure E.1 de l'Annexe, illustre une capture de l'affichage des options de la ligne de commande dans une fenêtre d'un terminal X. En utilisant cette interface, un utilisateur est capable de contrôler un proxy local ou à distance par le protocole *NewkeyCP*. Le prototype implémente ce protocole complet dans la version présentée en Annexe (Section C).

Dans la ligne de commande (voir la Figure E.1), toutes les options incluent le proxy qui doit être contrôlé. Le proxy à contrôler et les clients sont identifiés par leurs machines. Le reste, les serveurs et les clients en mode "à l'écoute" sont identifiés par la machine et le port utilisé. Une machine est précisée par son adresse IP ou par son nom DNS.

Une fois lancé en exécution avec une option valide, notre contrôleur à distance essaie de se connecter à la machine indiquée sur le port 5800. Si la connexion réussi, il traduit l'option reçue en message du protocole *NewkeyCP* qui lui correspond. Le résultat de l'envoi du message est affiché à l'utilisateur.

Les messages du protocole de contrôle sont assez complexes, générant une interface en ligne de commande du contrôleur assez riche. Pour rendre plus facile l'utilisation du contrôleur, nous avons commencé la réalisation d'une interface graphique à distance. L'affichage graphique offre la possibilité de donner plus d'informations et un support d'aide général qui rend l'accès au protocole du contrôle facile. Dans l'Annexe, Section E, nous présentons quelques fenêtres de l'interface graphique. Pour le moment, le programme a implémenté seulement la partie de connexion au serveur ("connectToServer") et celle de l'affichage sur les sessions du protocole *NewkeyCP*.

5.4.3 Service de droit d'entrée *control floor*

Pour valider nos idées concernant la possibilité d'implémentation d'un protocole quelconque de contrôle de droit d'entrée *floor* sur le support offert par *Newkey*, nous avons implémenté un autre contrôleur à distance spécialisé dans la gestion du contrôle d'événements d'entrée[9]. Son rôle principal est de fournir des politiques de contrôle de droit d'entrée *floor* à Newkey. Ayant le mécanisme de contrôle de droit d'entrée *floor* et la mise en place de la politique de contrôle, un autre problème qui s'avance logiquement est l'interface entre l'utilisateur et le mécanisme.

Notre prototype répond à ces deux questions par ses fonctionnalités suivantes :
– la génération des politiques de gestion du contrôle d'événements d'entrée pour une session coopérative du partage d'écran gérée par Newkey ;

[9]La Figure E.5 de l'Annexe montre la fenêtre principale du service de contrôle.

– l'offre d'une interface graphique aux participants pour pouvoir bénéficier de ce contrôle, selon la politique mise en place.

Dans la suite, nous décrivons la solution apportée par le prototype à ces deux problèmes. Nous commençons par une courte présentation du concept de "politique du contrôle de *floor*".

Les politiques de droit d'entrée *control floor*

Dans notre contexte de travail, une politique de contrôle gère les droits de générer des événements d'entrée au serveur VNC. Il y a un ensemble de politiques de contrôle qui sont les plus utilisés. Celles-ci sont présentées dans le Tableau 5.1.

Politique	**Description**
centralisée	il y a un président de session coopérative qui gère le droit d'entrée *floor*
transmission explicite	le droit d'entrée *floor* est explicitement passé à une autre personne
relâchement explicite	le droit d'entrée *floor* peut être obtenu après son relâchement explicite
file d'attente	les demandes du droit d'entrée *floor* sont enregistrées dans une file d'attente (d'habitude de type FIFO)
préemption	à la demande, le droit d'entrée *floor* est obtenu tout de suite
découpage en tranches de temps	il y a une limite de temps pour détenir le droit d'entrée *floor*
détection de pause	une pause dans l'activité conduit à la perte du droit d'entrée *floor*

TAB. 5.1 – Exemples de politiques de contrôle de droit d'entrée *floor*

L'interface de contrôle

Pour la clarté de la présentation, nous décrivons d'abord la solution concernant l'interface graphique de contrôle de droit d'entrée *floor* offerte aux utilisateurs.

Nous avons cherché à étudier la possibilité de fournir cette interface graphique sur l'écran de chaque participant en prenant en compte l'indépendance de la session de collaboration et de la plate-forme ainsi que la transparence par rapport au système VNC.

Une solution à ce problème peut être offerte par un serveur VNC dédié, qui utilise l'infrastructure VNC pour offrir les éléments de contrôle à son utilisateur. Ils prennent en compte la

possibilité pour un utilisateur d'effectuer les opérations suivantes sur le droit d'entrée *floor* :
- le *demander* ;
- le *relâcher* ;
- l'*accorder* à un autre participant ;
- le *retirer* à un autre participant.

Ces opérations sont traduites par des messages adressés au service de contrôle de droit d'entrée *floor*. L'échange des messages entre le service de contrôle et un tel serveur VNC d'un participant (noté *UI*) suit des règles définies dans un protocole de communication *FloCtrl* présenté dans l'Annexe, Section D.

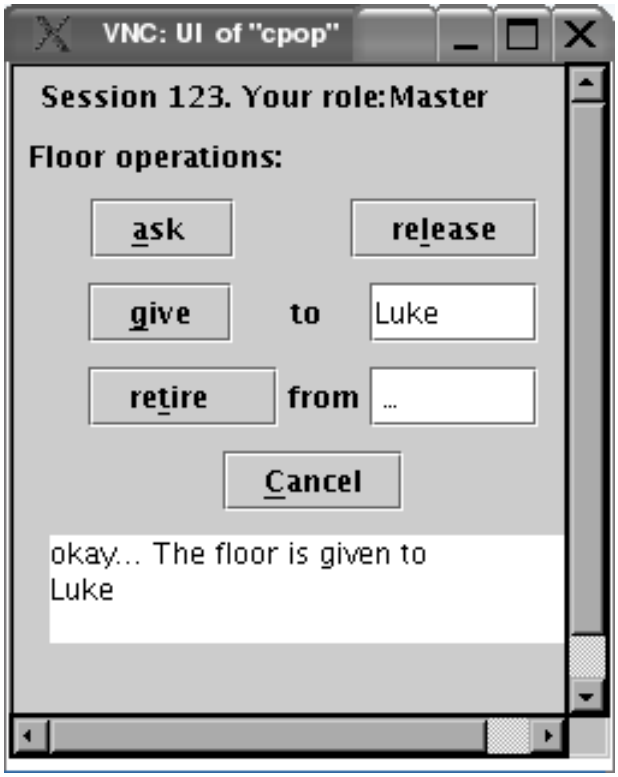

FIG. 5.4 – L'interface graphique de contrôle de droit d'entrée *floor* UI affichée sur l'écran d'un utilisateur.

Pour l'implémentation de ce serveur dédié (UI), nous avons utilisé une boîte à outil pour créer des serveurs VNC en Java, nommé VNCj[10] [68] [37]. En utilisant un des modèles proposés, nous avons implémenté un petit serveur VNC qui offre l'interface graphique illustrée dans la Figure 5.4.

[10]VNCj est un logiciel libre sous licence GPL.

En plus d'éléments de contrôle élémentaires, dont nous avons parlé ci-dessus, l'interface offre également une fenêtre de messages par laquelle le participant est informé en permanence sur l'état de droit d'entrée *floor*, sur les résultat de ses opérations et les raisons d'éventuels échecs.

Notre UI peut s'afficher sur l'écran de l'utilisateur à distance ou sur la même machine. Son utilisation est très flexible : il est possible d'initier la connexion RFB du serveur UI[11] vers un client en mode "à l'écoute" mais, également, du client vers le serveur VNC *UI* [12]. La Figure E.6 de L'Annexe illustre toutes les options en ligne de commande offertes par l'UI. Pour le moment, un serveur UI qui correspond à un participant à la session, est lancé automatiquement en exécution par le service de contrôle de droit d'entrée *floor* suite à sa configuration pour une session. Cela implique l'existence d'un client en mode "à l'écoute" sur la machine de chaque participant. La Figure 5.5 illustre l'architecture du système VNC enrichi avec le contrôle de *floor*.

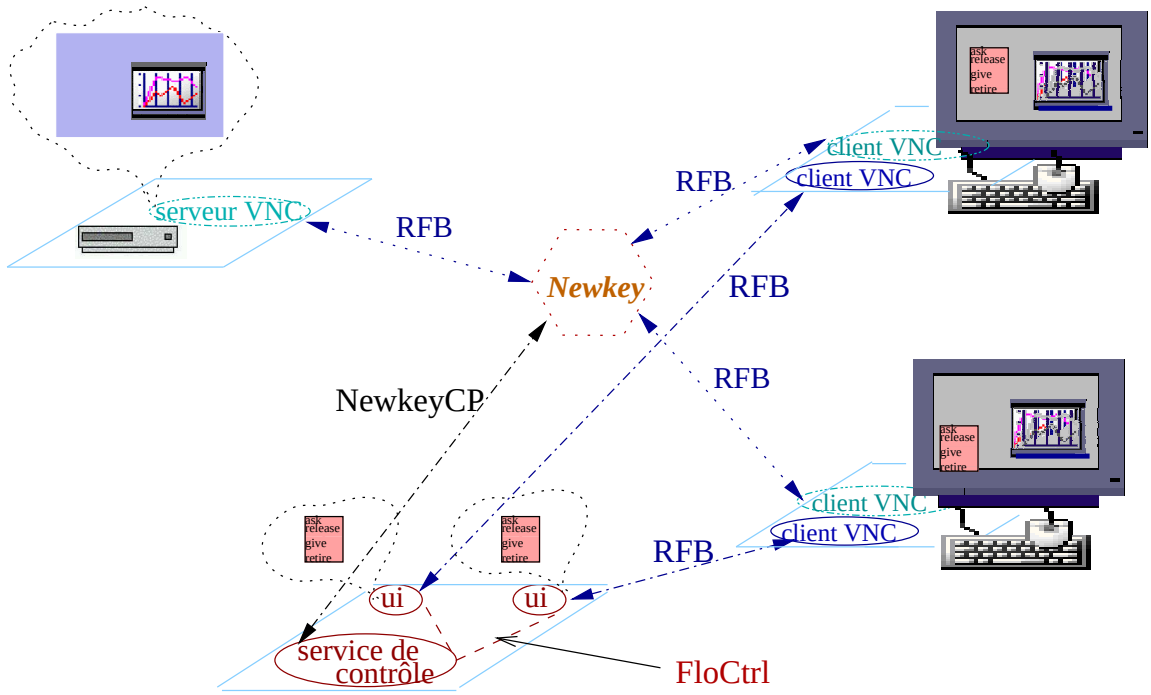

FIG. 5.5 – L'architecture du système NC enrichit avec le contrôle de droit d'entrée *floor*.

Gestion du contrôle de droit d'entrée *floor* de Newkey

Notre service de contrôle est capable de fournir plusieurs politiques de contrôle à la

[11] Il s'agit du style de connexion "connectToServer".
[12] Il s'agit du style de connexion "connectToClient".

demande. La politique de contrôle de droit d'entrée *floor* est générée sur la base des règles qui sont définies en fonction de la présence des participants dans la session.

Quand le service reçoit un message d'un participant qui demande une opération sur le droit d'entrée *floor*, le programme vérifie d'abord la pertinence de la demande selon la politique mise en place. Si la demande est acceptée, le service la transforme dans un message de contrôle de droit d'entrée *floor* adressé à Newkey[13] et envoie un événement aux participants en les informant de l'opération. Dans le cas contraire, le service envoie un événement à la source de la demande pour expliquer le refus et les autres participants sont informés également au sujet de cette tentative de manipulation de droit d'entrée *floor*. La Figure 5.6 illustre la transformation d'un message qui demande une opération sur le droit d'entrée *floor* en un message de contrôle de Newkey.

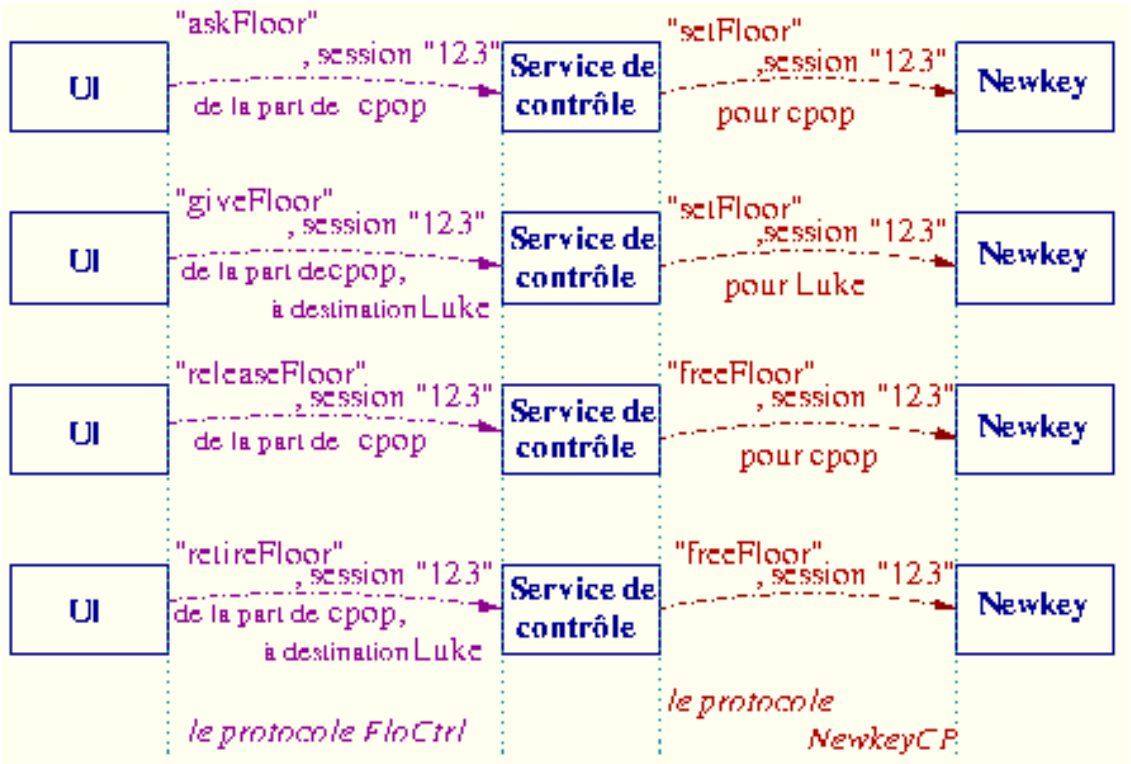

FIG. 5.6 – La translation d'un message de contrôle de droit d'entrée *floor* du protocole *FloCtrl*, en un message de commande de droit d'entrée *floor* du protocole *NewkeyCP*.

5.4.4 Sécurité

Dans cette section, nous analysons notre système coopératif du point de vue de la sécurité.

[13]Il peut s'agir soit d'un message *setFloor*, soit de *freeFloor* (voir la Section C).

Concernant l'authentification VNC, nous pouvons avoir deux configurations de sessions dans notre implémentation. D'abord, le contrôleur à distance accepte l'association à une session d'un mot de passe qui correspond au serveur impliqué. Dans ce cas là, le client n'a pas besoin de chiffrer le défi (*challenge*) envoyé par le serveur : cela reste la tâche du contrôleur. Si le contrôleur est configuré pour une session sans mot de passe associé, on demande au client impliqué dans la session un mot de passe. La première configuration de session se prête donc bien à des situations qui impliquent plus de confiance. Simultanément elles favorisent la mise en place rapide de sessions. Dans la version présente du prototype, le mot de passe d'une session est envoyé en clair à Newkey. Cela pourrait poser des problèmes si le contrôleur se trouve sur une machine à distance par rapport à Newkey. Encore une fois, une solution à ce problème est donnée par l'utilisation d'une connexion ssh (discuté déjà dans la Section 4.1.4).

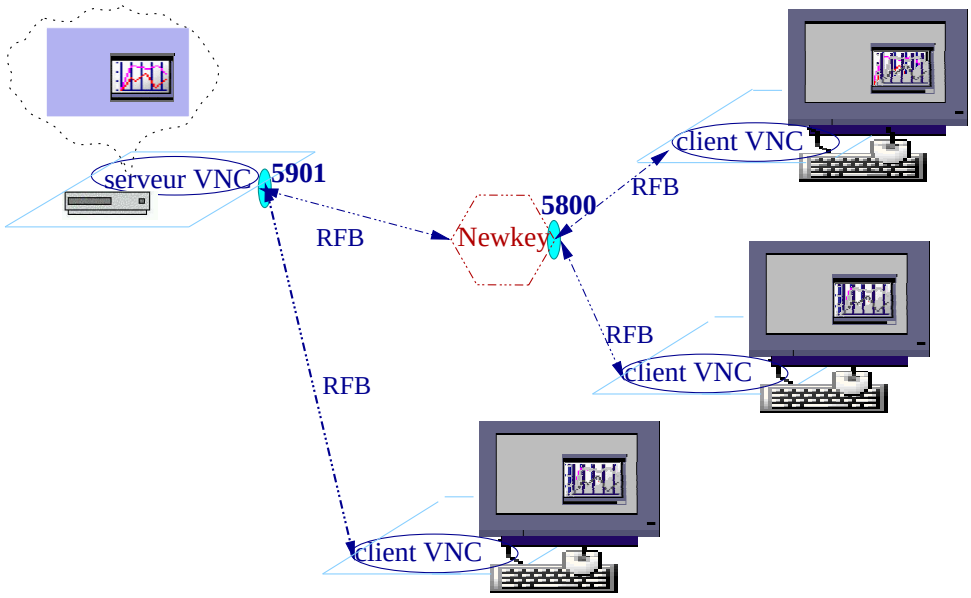

FIG. 5.7 – Le court-circuit du Newkey. Un client VNC en connaissant le mot de passe d'accès à un serveur VNC peut s'y connecter en court-circuitant Newkey.

Interposé entre le client et le serveur, Newkey est configuré pour recevoir ou initier des connexions RFB de/vers des clients identifés par leurs machines. Cela renforce la sécurité du système VNC à condition que celui-ci possède un contrôle d'accès[14]. Dans ce cas, il est suffisant que l'accès aux serveurs soit accordé seulement à la machine sur laquelle se trouve Newkey. Cela se réduit à l'interdiction de l'accès de l'extérieur si Newkey se trouve sur la même machine que les serveurs. Dans le cas où le système VNC n'a pas de contrôle d'accès intégré, il est malheureusement possible de court-circuiter Newkey. La Figure 5.7 illustre ce

[14]Comme nous l'avons vu dans la Section 4.1.4, certaines versions de la distribution libre RealVNC incluent ce mécanisme.

serveur \ système	VNC	VNC + Newkey
x0rfbserver	5.18s	5.49s
Xvnc	3.92s	4.09s

Tab. 5.2 – Résultats de mesures.

cas. À condition de connaître le mot de passe, cela rend possible la connexion à un serveur VNC impliqué dans une session en dehors du cadre de celle-ci. Heureusement, les dernières versions de la distribution libre du système VNC incluent l'option qui demande la permission de l'utilisateur pour accepter une nouvelle connexion : un contrôle qui n'est pas automatique, mais qui permet de régler le problème décrit ci-dessus.

5.4.5 Comportement du prototype

Dans la version actuelle, en utilisant le contrôleur à distance, on peut contrôler Newkey pour mettre à profit toutes les fonctionnalités décrites. Si on utilise le contrôleur pour configurer une session sur Newkey qui se trouve sur la machine locale, le proxy n'étant pas encore chargé en mémoire, celui-là le lance en exécution automatiquement et le configure. Le contrôleur de droit d'entrée *floor* peut être lancé en exécution après la mise en place d'une session sur Newkey.

En ce qui concerne le mécanisme du multiplexeur, nous avons le problème classique généré par la différence entre les caractéristiques physiques d'écrans : Newkey fonctionne bien avec des clients qui utilisent le même type de visuel. Les mesures effectuées montrent des différences acceptables en vitesse entre l'utilisation du système VNC avec et sans Newkey. Le Tableau 5.2 illustre ces résultats. Pour réaliser les tests, nous avons choisi l'application *ghostview* et un ensemble de documents riches en figures. Les mesures ont été réalisés sous Linux en utilisant les serveurs Xvnc et x0rfbserver.

Pour le moment, les prototypes sont en phase de tests, qui pourront dépister des problèmes de fonctionnement.

Concernant le fonctionnement, nous pouvons faire les remarques suivantes :

- L'utilisation du contrôleur à distance, avec toutes ses fonctionnalités est rendu difficile par des options relativement riches offertes en ligne de commande. Ainsi, l'implémentation complète de l'interface graphique du contrôleur s'impose. Sa réalisation demande de la réflexion sur des moyens à faciliter son utilisation.
- Dans le fonctionnement d'un UI en tandem avec le service de contrôle de droit d'entrée *floor*, nous avons un délai en ce qui concerne la réponse aux opérations demandées.
- En utilisant l'interface graphique d'UI, on remarque l'absence des informations concernant la session : par exemple les participants.
- L'interface UI et le service de contrôle de droit d'entrée *floor* devraient permettre des

opérations concernant le changement dynamique de la politique du contrôle de droit d'entrée *floor*.

5.4.6 Travail futur

Le travail futur sur les prototypes impliquera la résolution des problèmes présentés dans la section précédente. Nous remarquons évidement la nécessité d'inclure le service de contrôle de droit d'entrée *floor* comme fonctionnalité du contrôleur à distance : celle-ci peut être utilisé à la demande dans le cadre d'une session. Dans ce sens, il faut exploiter aussi toutes les fonctionnalités mises à disposition par UI (illustrées dans la Figure E.6) : la possibilité d'un UI de se connecter au service de contrôle et également de configurer la mise en place de la gestion de politique de contrôle de droit d'entrée *floor* pour une session sur Newkey.

D'un autre point de vue, toutes ces idées conduisent à la conclusion qu'il faut renforcer le contrôle d'accès à Newkey. Pour le moment, la seule restriction concerne le propriétaire d'une session : il doit faire partie des participants de la session qu'il a configuré.

5.5 Conclusion

Dans ce chapitre nous avons présenté notre proposition de support coopératif au système VNC. Cette proposition a voulu rendre plus flexible ce système, un aspect intéressant pour des environnements spontanés de communication.

Les prototypes réalisés mettent en évidence les fonctionnalités qui transforment VNC en un système adaptable à diverses situations de collaboration qui peuvent survenir dans un environnement de collaboration spontané. Une vue critique sur ces prototypes souligne principalement l'importance de l'interface offerte à l'utilisateur et indique son amélioration.

Chapitre 6

Conclusion

6.1 Bilan du travail

Dans le contexte des idées présentées dans ce mémoire, nous nous sommes proposés d'étudier deux infrastructures coopératives construites sur deux systèmes très répandus : le système de fenêtrage X et le système d'affichage VNC. Nous analysons dans cette section les travaux réalisés.

6.1.1 Le système de partage d'applications X

Nous avons vu dans le Chapitre 2 que pour réaliser un système de partage d'applications X, beaucoup des efforts se sont focalisés sur l'intégration d'un *proxy* d'une façon transparente dans le système X Window. À cause de la complexité du protocole X, à notre avis cette tâche présente en soi un sujet d'étude. Nous nous sommes limités dans notre travail à implémenter un multiplexeur avec des fonctionnalités de base.

Nous n'avons pas pu implémenter le mécanisme de contrôle de droit d'entrée *floor*, mais nous avons fourni une interface d'utilisateur à ce mécanisme. À notre avis, cette interface se distingue des autres solutions proposées par sa simplicité et la façon naturelle dont elle s'intègre dans le système X : l'interface est offerte par un client X qui attache sa fenêtre à la fenêtre principale de l'application partagée. Nous avons vu que son point faible, tel qu'il est conçu pour le moment, est le fait que l'on exécute une instance de ce client pour chaque participant à la session. Cette opération pourraient consommer beaucoup de ressources de la machine locale dans le cas d'un grand nombre de participants[1].

Notre multiplexeur est implémenté en tant qu'un service actif qui peut être chargé dyna-

[1] Cela dépend également de la complexité du client.

miquement sur une plate-forme générique de noeud actif. De cette manière, le multiplexeur devient un élément actif du réseau. Nous considérons qu'il apporte une nouvelle contribution grâce à l'idée de traitement de paquets dans le réseau pour supporter une session de collaboration. L'utilisation de ce service actif dans le contexte d'un environnement spontané de communication est appropriée grâce à la caractéristique de la plate-forme choisie pour son implémentation : elle permet de charger dynamiquement à la demande de l'utilisateur ou d'un autre programme le service actif souhaité. De plus, la plate-forme a été conçue dans un but d'adaptation au contexte : nous n'avons pas encore exploité cette fonctionnalité dans notre travail.

6.1.2 Le support coopératif ajouté au système VNC

Nous avons vu dans le Chapitre 4 qu'une partie des contributions existantes apportées pour améliorer les caractéristiques coopératives du système VNC se résument à des modifications qui s'intègrent d'une façon transparente au système. Les contributions majeures dans ce sens se basent sur la modification du système entier.

Par rapport à ces contributions, nous nous sommes orientés d'une part vers des opérations sur des flots RFB et d'une autre part sur l'exploitation de l'existence des clients VNC en mode "à l'écoute". Le système coopératif gagne ainsi en flexibilité par la programmation dynamique des sessions de collaboration sur un *proxy* (*Newkey*) introduit d'une façon transparente dans le système VNC. Les scénarios donnés au cours de la présentation du Chapitre 5 illustrent la façon dont le système VNC peut s'adapter à différentes situations de collaboration grâce à *Newkey*. Dans le but de contrôle du *proxy*, nous avons défini un protocole de contrôle (*NewkeyCP*). En plus de la programmation des sessions de collaboration, le protocole inclut des primitives pour le contrôle d'entrée et une partie de gestion.

Quand nous avons conçu et implémenté notre prototype, il n'y avait pas de support pour le contrôle de droit d'entrée pour le système VNC. *Collaborative VNC* [1] apporte plusieurs politiques de contrôle du droit d'entrée. Cependant, nous pensons avoir apporté une réponse originale à ce problème dans le contexte du système VNC : notre travail offre un mécanisme du contrôle du droit d'entrée et laisse la liberté d'utiliser la politique la mieux adaptée au contexte de collaboration. Dans le mémoire, dans la Section 5.4.3, nous avons montré comment les opérations de contrôle de droit d'entrée sont traduites en deux primitives du protocole *NewkeyCP*.

L'interface utilisateur au mécanisme de contrôle du droit d'entrée est assurée par un serveur VNC dédié (*UI*) qui s'affiche sur l'écran d'un participant. Celui-ci offre la possibilité d'opérer sur le droit d'entrée. Ainsi, il communique avec le service qui gère la politique de contrôle d'entrée.

Dans la conception d'un support coopératif au système VNC, nous avons essayé de tirer des conclusions de l'analyse de la conception du système de partage d'applications X :

– le support est basé également sur un *proxy* inséré d'une façon transparente dans le système VNC, mais celui-ci est contrôlable dynamiquement à distance. Cela permet le contrôle du droit d'entrée, mais également la gestion plus flexible de l'entrée et la sortie d'un participant à une session.
– d'une façon analogue à celle concernant le système X, nous avons exploité les caractéristique du système de base pour offrir un interface utilisateur au mécanisme de contrôle d'entrée : cette fois-ci, le programme qui s'affiche sur l'écran de chaque utilisateur peut être lancé en exécution soit par le *proxy*, soit par l'utilisateur lui-même ; de cette façon, on évite la charge d'une seule machine.

On peut remarquer que par rapport au système coopératif de partage d'applications X, celui-ci est plus flexible. Cependant, des travaux futurs doivent apporter des améliorations au système surtout en ce qui concerne l'interface de programmation du *Newkey*. En plus, nous n'avons pas encore exploité la caractéristique du *Newkey* d'être programmé à distance. Nous avons introduit cette caractéristique pour bénéficier à distance des fonctionnalités d'un proxy *Newkey* présent dans un environnement spontané de communication, par exemple sur une machine proche physiquement d'un point d'accès sans fil. Malheureusement, cela n'est pas possible dans les conditions actuelles, à cause du manque d'informations sur la disponibilité d'un tel logiciel.

6.2 Perspectives

Tout au long de ce document, nous avons mis en évidence les limites de nos propositions et les possibilités d'amélioration de nos prototypes, ce qui constitue des perspectives de continuation de notre travail. En conséquence, dans cette partie nous allons nous orienter vers la présentation de quelques nouvelles directions d'études que nous souhaitons prendre.

On prévoit dans le futur une implication active et pro-active de réseaux de communication dans la vie de chaque personne [21] [69]. Selon cette vision, l'environnement inclura beaucoup de dispositifs intelligents capables de communiquer avec d'autres équipements informatiques à travers différents types de réseaux de communication au bénéfice de l'utilisateur, d'une façon autonome et adaptable au contexte. Cela constitue un terrain riche de réflexion et d'étude concernant le support que le réseau peut offrir pour soutenir la collaboration entre des personnes.

En partant d'une observation pratique de l'utilisation de nos systèmes coopératifs dans un environnement spontané de communication, nous commençons par exploiter une nouvelle façade du problème. Cette direction principale ouvre d'autres perspectives qui sont présentées dans la suite. À notre avis, toutes ces voies constituent un pas en avant vers la re-considération de l'infrastructure de collaboration que le réseau pourrait offrir dans future.

6.2.1 Découverte des systèmes sur le réseau

Dans notre vision, une personne dispose sur son ordinateur portable du système de partage X basé sur le multiplexeur et/ou le système VNC enrichi avec des caractéristiques coopératives. Au cours de ses déplacements, elle peut se trouver dans un environnement spontané de communication qui pourra lui permettre de partager des applications ou l'écran avec d'autres personnes. Malgré cela, actuellement chaque personne a tendance à utiliser son ordinateur d'une façon isolée. Même dans des situations de travail en collaboration, l'utilisation d'un support informatique est limitée par le manque d'informations concernant l'existence des composants sur des machines d'autres participants. Donc, le problème qui en résulte est la découverte des personnes qui ont des composants de ces systèmes disponibles pour être partagés.

Il y a un nombre grandissant d'applications ou de systèmes dont les composants sont capables de se découvrir tous seuls (par exemple l'application Hydra [30]). Ce mécanisme est inclus soit au niveau des communications (comme la technologie Bluetooth, ou qui utilisent des dispositifs spéciaux, comme Active Badge [5] ou Thinking Tag [46]), soit au niveau des applications[2]. Au niveau application, le mécanisme est mis en place par l'utilisation d'un *protocole de découverte de services* [43].

Dans notre cas, nous sommes adeptes de l'intégration transparente du mécanisme de la découverte de services dans nos systèmes coopératifs. Ceci implique donc l'exclusion d'éventuelles modifications des systèmes X et VNC pour l'intégrer.

Une solution évidente est l'ajout de la fonctionnalité de découverte des composants de chaque système dans les *proxy*. La nouvelle fonctionnalité d'un *proxy* consiste en un "balayage" des ports à la recherche des machines qui écoutent sur les ports associés aux proxy. Cette solution présente néanmoins un inconvénient : le manque d'informations concernant les clients/serveurs découverts. De plus, en utilisant cette méthode on ne détecte pas, par exemple, l'existence des clients VNC[3] sur les autres machines.

Une solution plus élégante est la considération de ces composants comme services réseau et, donc, l'utilisation d'un protocole de découverte de services pour les annoncer et les découvrir en réseau. Un support extérieur pourrait fournir cette fonctionnalité, indépendemment du système. Dans la suite de cette section, nous présentons le protocole de découverte de service adéquat à notre contexte de travail.

Multicast DNS Service Discovery (mDNS-SD)

Nous considérons que le protocole de découverte de services *Multicast DNS Service Discovery* (*mDNS-SD*) [49] est le plus approprié à notre contexte de travail. Nous présentons ici

[2]Par exemple, l'application ProxyLady [18] ou des applications construites sur des plates-formes qui incluent la découverte de services (comme exemple JXTA [32]).

[3]Nous ne référons aux clients qui ne sont pas en mode "à l'écoute".

l'utilisation de DNS pour la découverte de services et nous justifions ce choix.

DNS [39] s'est déjà ouvert pour pouvoir enregistrer des services par l'introduction d'un nouveau type d'enregistrement (*record*) DNS nommé *SRV* [4]. Celui-ci fait l'association entre le type de service et l'hôte où le service est localisé. Malheureusement, de cette manière, on ne peut pas faire la distinction entre différentes instances de service du même type.

DNS-SD [14] introduit un niveau d'indirection en utilisant le type d'enregistrement *PTR*[4] : à chaque type de service correspond une liste d'instances de service. Chaque nom d'instance de service a son enregistrement *SRV* et *TXT*. Si l'enregistrement *SRV* spécifie la localisation du service, celui de *TXT*[5] est utilisé pour sa description (sous forme de pairs attribut-valeur).

DNS-SD comporte la sémantique et la syntaxe d'un service qui utilise l'enregistrement DNS SRV auquel s'ajoute le nom d'instance de service :

<*Instance* >. < *TypeDeService* >.<*domaine* >, où <Instance > est une étiquette DNS unique dans le < domaine >DNS. De point de vue pratique, ce nom d'instance est signifiant pour les utilisateurs finaux (*end users*).

En plus, pour mieux préciser une catégorie d'un type de service, DNS-SD accepte l'utilisation de sous-type :

<*Instance* >._<*Sous-TypeDeService* >.< *TypeDeService* >.<*domaine* >;

et, conformément aux règles fournies par DNS SRV :

< *TypeDeService* >= _<*ProtocoleDeApplication* >._<*ProtocoleDeTransport* >.,

où le <ProtocoleDeApplication > est le protocole qui donne l'accès au service.

De cette manière le modèle arborescent du système de nommage DNS est préservé, comme illustré dans la Figure 6.1.

Le processus de requête/réponses des services suit le format DNS standard pour le nommage et la résolution de noms. Pour obtenir une liste d'instances d'un type de service, on utilise la requête DNS de type PTR sur un nom dans un certain domaine. Ensuite, des requêtes DNS de type SRV et TXT, sur un nom d'instance, retournent des informations supplémentaires sur le service et son instance. La figure 6.2 montre un échange des messages correspondant à la hiérarchie des noms DNS présentée dans la figure 6.1.

Ainsi, le système de nommage DNS est utilisé pour la découverte de services. L'in-

[4]Dans le DNS, l'enregistrement *PTR* est un pointeur d'un nom vers un autre nom dans l'espace de noms DNS (dans le texte original : "The PTR record is used to let special names point to some other location in the domain tree." (RFC 1033)).

[5]Dans DNS l'enregistrement *TXT* a un rôle informationnel et il n'y a aucun format du contenu texte imposé.

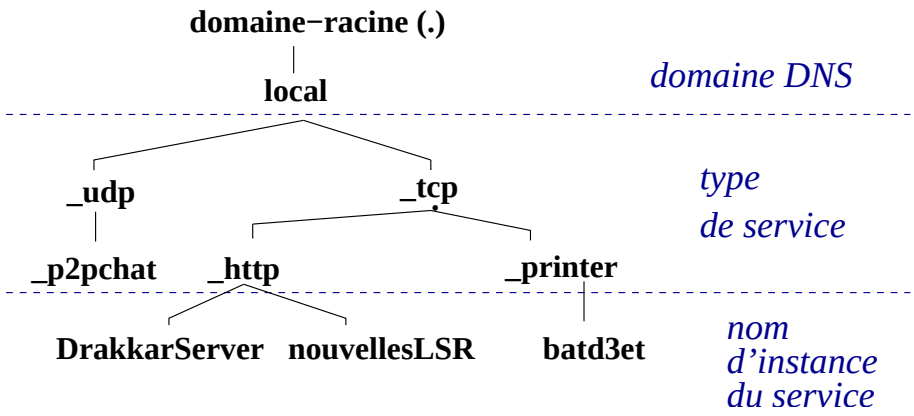

FIG. 6.1 – L'extension de la structure arborescente de DNS par les noms de services. La figure illustre la hiérarchie d'un domaine DNS, des protocoles de transport et d'application (qui entrent dans la composition des types de service) et des noms des instances.

Requête: **._http._tcp.local.PTR?**
Réponse: **nouvellesLSR._http._tcp.local.**
serveurDrakkar._http._tcp.local.
Requête: **serveurDrakkar._tcp.local.SRV?**
Réponse: **SRV − 0 0 80 delos.imag.fr**
Requête: **serveurDrakkar._tcp.local.TXT?**
Réponse: **TXT − http_URL = "http://drakkar.imag.fr"**

FIG. 6.2 – Exemple d'échange des Requêtes/Réponses en DNS-SD. L'exemple montre la découverte d'un site web.

convénient majeur est l'opacité du protocole en ce qui concerne le contenu de l'enregistrement TXT. Celui-ci contient des informations qui décrivent le service. Ces informations ne peuvent être retrouvées que par des requêtes de type TXT.

Dans un réseau de communication qui se forme spontanément, l'utilisation d'un système de nommage centralisé est inappropriée. Le protocole Multicast DNS (mDNS) [15] propose l'utilisation des opérations similaires au DNS traditionnel dans un réseau local en absence de tout serveur conventionnel DNS fonctionnant en unicast. Le protocole s'appuie sur le mécanisme de IP Multicast en utilisant l'adresse 224.0.0.251 et le port UDP 5353, et permet l'envoie de requêtes de différents types contenus dans une seule requête mDNS. En dépit du fait qu'il n'assure pas l'unicité du nom d'un hôte dans le domaine, il présente une solution d'utilisation du système DNS d'une façon distribuée[6]. Chaque noeud du réseau est capable de gérer lui-même des informations DNS en jouant dans le même temps le rôle de serveur et client DNS.

Dans cette perspective, la proposition DNS-SD, en combinaison avec le *multicast DNS* (*mDNS*) devient intéressante pour un environnement spontané de communication[7].

Support extérieur de découverte de services

Comme soutien pour la découverte de services de systèmes X/VNC, nous pensons à un programme qui s'exécute comme un processus de fond. Il reçoit comme entrée la description des composants se trouvant sur la machine locale qu'il peut annoncer sur le réseau en utilisant le protocole mDNS-SD. Simultanément, il reçoit des annonces d'autres services en gardant seulement celles qui correspondent aux composants se trouvant sur la machine locale. Pour faciliter la présentation, nous nommons ce programme *Nemo*.

Les services qui correspondent à un service donné s'appellent "compatibles" avec celui-ci. La compatibilité de services se résume à trois conditions :

- *un service peut offrir un flot de données et l'autre en recevoir*. Par exemple, un serveur X n'est pas compatible avec un autre serveur X, car tous les deux offrent une interface d'entrée. Le service qui l'initialise induit la *direction de la communication* ;
- *les deux services offrent le même format de données, X ou VNC* : par exemple, un client X ne peut pas communiquer avec un serveur VNC ;
- *les protocoles de communication sont compatibles* : les composants de système X ne sont pas compatibles avec ceux du système VNC.

Un composant qui se trouve sur la machine locale peut être annoncé sur le réseau comme

[6]Pour plus de détails, voir [15].

[7]L'exemple le plus concluant est donné par la technologie Apple *Bonjour* [7] qui fournit la configuration automatique et la détection dynamique des services dans un réseau local ou ad-hoc. En plus, il y a beaucoup de fabriquants d'imprimantes qui ont inclus cette technologie dans leurs produits.

un service[8], ou rester un service local[9]. Nemo bénéficie seulement des services annoncés sur le réseau qui lui sont compatibles. Nemo affiche à la demande des services mDNS-SD compatibles avec ceux de la machine locale et marque la direction possible de la communication. Nemo peut afficher également des informations supplémentaires sur les services incluses dans les champs *SRV* et *TXT*. La Figure 6.4 et 6.3 illustrent une proposition d'affichage des services compatibles avec ceux existants sur la machine locale.

Pour bien identifier le service, nous suggérons que le nom d'instance inclut le nom du propriétaire du service. Sur les Figures 6.4 et 6.3, on peut remarquer des exemples des noms d'instance qui contiennent les noms de propriétaires des services.

Annonce des systèmes par mDNS-SD

En utilisant le protocole mDNS-SD, les informations concernant les protocoles de communication utilisés par un service sont incluses dans la spécification du <TypeDeService>. Des informations concernant la configuration, le nom d'hôte et son port de communication, se trouvent dans l'enregistrement SRV. Toutes les autres informations sont incluses dans l'enregistrement TXT[10]. Ci-dessous nous présentons les composants de deux systèmes coopératifs comme services mDNS-SD.

Le système de partage d'applications X

Pour annoncer les composants du système de partage d'applications X en utilisant mDNS-SD, nous suggérons l'utilisation du type de service *_X._tcp*. Cela indique l'utilisation du protocole d'application X et du protocole de transport TCP. La distinction entre les deux composants du système X peut se faire en utilisant des sous-types de service : *client* et *serveur*. Pour la description du multiplexeur nous utilisons le sous-type de service : *mux*.

Voici des exemples de noms d'instance de ces types de services :

clientXDuCPop._client._X._tcp.local,

serveurXDuDupont._server._X._tcp.local,

multiplexeurXDuCPop_mux._X._tcp.local.

On considère que les services communiquent entre eux en utilisant le format du protocole X. Les trois services présentent les interfaces de communication suivantes :

[8]On considère un tel service comme étant "public".

[9]On peut considère un tel service comme étant "privé".

[10]La spécification du protocole DNS-SD en ce qui concerne la structure des informations de l'enregistrement TXT est très ouverte. Il y a un ensemble de recommandations, mais chaque type de service peut définir sa propre sémantique pour les paramètres.

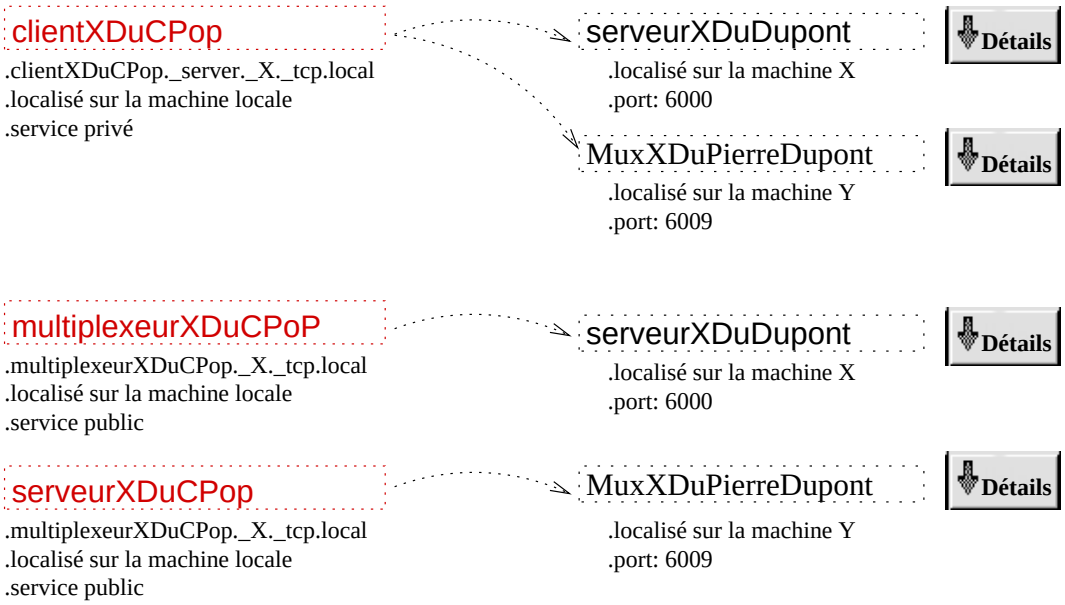

FIG. 6.3 – Proposition d'une interface graphique présentée par Nemo. La figure illustre des services de type *_X._tcp* compatibles. Les services compatibles sont liés par des arcs de cercle qui marquent également la direction de la communication.

- *un client X* : offre une interface de sortie ;
- *un serveur X* : offre une interface d'entrée ;
- *un multiplexeur X* : offre une interface d'entrée et une interface de sortie.

La Figure 6.3 montre une proposition de l'interface graphique du Nemo qui indique également les directions de communications.

Le système coopératif de partage de l'écran

Les composants du système coopératif de partage de l'écran peuvent être annoncés comme services mDNS-SD en utilisant le type de service *_RFB._tcp*. Cela montre l'utilisation du protocole d'application RFB et du protocole de transport TCP. La distinction entre les composants du système peut se faire en utilisant des sous-types de service : *client*, *serveur* et *Newkey*. Voilà des exemples de noms d'instance de ces types de services :

clientVNCDuCPop._client._RFB._tcp.local,

serveurVNCDuPierreDupont._server._RFB._tcp.local,

NewkeyDuCPop_newkey._X._tcp.local.

Dans le cas de ce système, nous considérons que les composants communiquent en utilisant le format de données *VNC* et ils présentent les interfaces de communication suivantes :

- *un client VNC* : offre une interface d'entrée ;
- *un serveur VNC* : offre une interface de sortie ;
- *Newkey* : il offre une interface d'entrée et de sortie.

En général on utilise le client pour initialiser la communication. Mais à l'aide d'un outil spécial (comme par exemple, vncconnect, de la distribution TightVNC) on peut initialiser une connexion vers un client en mode "à l'écoute". Pour distinguer ce type de clients, on peut utiliser un indicateur "on stand-by" dans le champ TXT de leurs présentations comme services.

Des interfaces de communications offertes sont illustrées dans la Figure 6.4.

Une fois un système coopératif de partage d'écran mis en place et la session de collaboration ouverte vers d'autres participants, elle peut être également annoncée sur le réseau comme un service mDNS-SD qui a les caractéristiques suivantes :

- le champ SRV donne des indications sur l'adresse et le port du *proxy* ;
- le champ TXT offre des indications concernant la session ;
- il a le type de service *_collaboration._RFB._tcp* ;
- le service offre une seule interface d'entrée : cellle de contrôle du *proxy*.

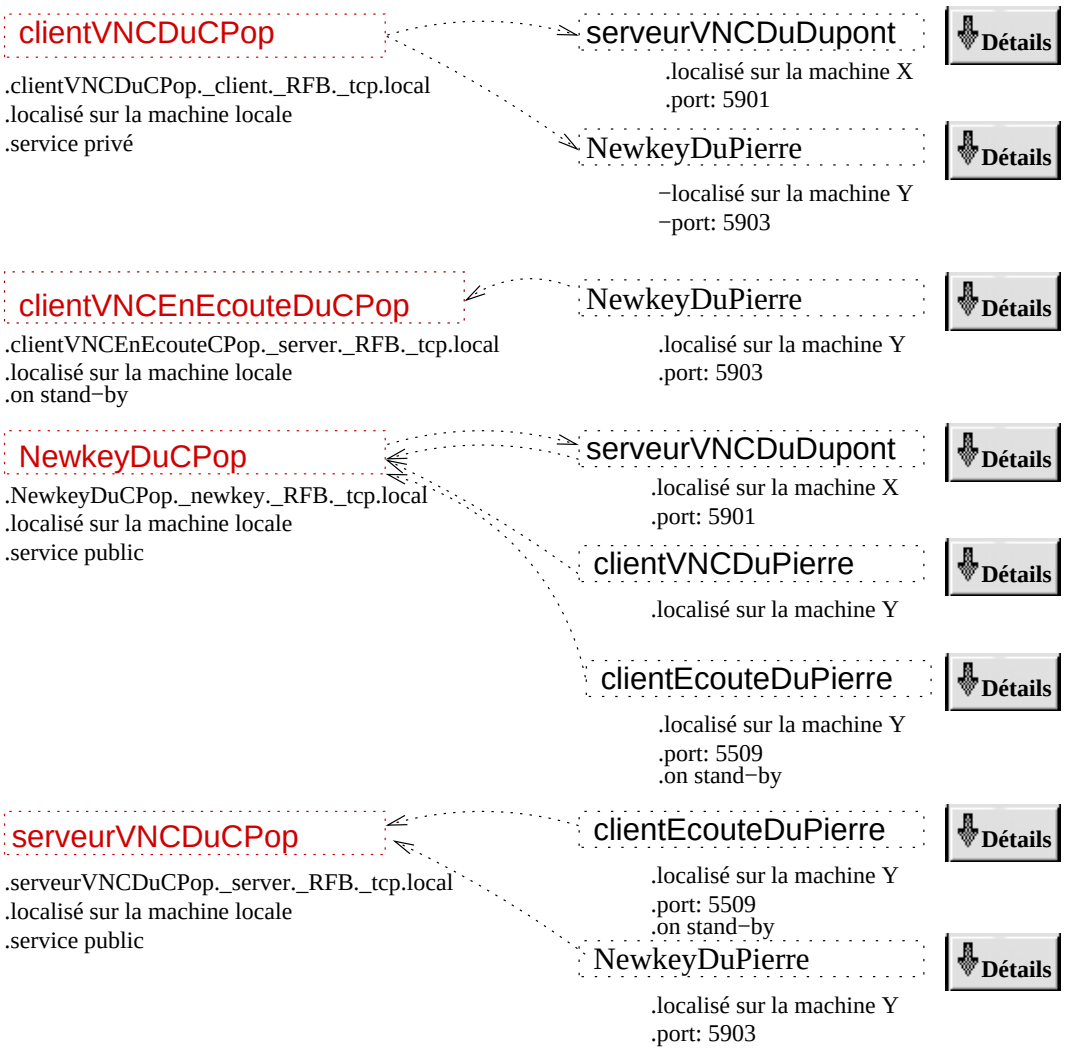

FIG. 6.4 – Proposition d'une interface graphique présentée par Nemo (2). La figure illustre des services de type *_RFB._tcp* compatibles. Les services compatibles sont liés par des arcs de cercle qui marquent également la direction de la communication. Nous remarquons les services qui sont des clients VNC en mode "à l'écoute".

Scénario d'utilisation

La possibilité de découvrir des services compatibles avec ceux disponibles sur l'ordinateur personnel dans l'environnement spontané de communication, favorise des collaborations qui ne sont pas planifiées et même l'initialisation d'interaction avec des personnes inconnues. Nous montrons maintenant un scénario allant dans ce sens.

Disons qu'une personne se trouve dans une salle d'attente où elle peut profiter d'un

réseau de communication sans fil en utilisant son ordinateur personnel portable. Elle aime bien jouer un jeu (*multi-joueur*) sous Unix, par exemple un jeu d'échec et elle souhaiterait avoir un partenaire pendant la période d'attente. Dans cette condition, elle configure Newkey pour dédier un port pour la duplication de l'affichage en utilisant son serveur XVnc[11]. Elle configure aussi son serveur XVnc pour assurer un nouvel environnement graphique qui inclue une seule application : le jeu[12]. En utilisant Nemo, il annonce Newkey comme un service sur le réseau :

PierreDupont :quiVeutJouerAuxÉchecsAvecMoi ?._newkey._RFB._tcp.local .

La champ SRV du service contient le port associé à la session, l'adresse de la machine-hôte et le champ TXT apporte des informations supplémentaires sur le jeu, comme par exemple, le mot de passe. En découvrant ces informations sur le réseau à l'aide de Nemo, un éventuel amateur peut partager le plaisir d'une partie d'échec avec notre personnage. Par la politique adéquate, le système de contrôle de droit d'entrée peut assurer que chaque joueur a l'accès exclusif à l'interface graphique du jeu à son tour.

Discussions

En analysant la description de cette proposition de continuation du travail présenté dans le mémoire, nous pouvons faire quelques observations :

- **Nous proposons une vue plus générale sur la notion de "services réseau"**. On peut remarquer que dans cette proposition, nous considérons comme les serveurs et les clients en mode "à l'écoute" aussi bien que les clients Vnc habituels comme des services réseau [13]. Comme nous avons vu, cela est nécessaire pour pouvoir découvrir les composants des systèmes coopératifs et même des systèmes X/VNC sur des ordinateurs portables d'autres utilisateurs. Donc, cette proposition change l'image classique d'un service réseau.
- **Il se pose un problème d'accès concurrent**.
  Nous avons montré dans cette section comment les composants peuvent être mis à disposition à d'autres utilisateurs et, en même temps, comment on peut découvrir des services utiles dans un environnement spontané de communication, par l'utilisation du protocole mDNS-SD. Le scénario présenté montre qu'il reste encore des questions ouvertes concernant l'utilisation de ce protocole dans le but présenté. Par exemple, il y a des services qui acceptent l'accès concurrent de plusieurs services en même temps par une interface de communication, tels que les serveurs X et VNC[14]. Cependant, il y a d'autres services qui peuvent accepter seulement une connexion à la fois, comme

[11]Il s'agit de pconfigurer une session coopérative de type "connectToServer" sur la base de duplication de flots RFB, voir la Section 5.2.1.

[12]Cette configuration est possible pour un serveur XVnc en précisant le nom du programme dans le fichier $HOME/.vnc/xstartup.

[13]Il s'agit des clients lancés en exécution pour se connecter à un serveur X.

[14]En général, tous les serveurs acceptent plusieurs connexions au même moment.

par exemple le client VNC en mode "à l'écoute" ou le Newkey dans notre scénario : notre personnage a besoin d'un seul partenaire de jeu. Dans le deuxième cas, Il faudrait signaler le fait qu'un tel service n'est plus disponible, suite à la réception d'une connexion. Le protocole mDNS-SD n'offre pas la possibilité de retirer une annonce d'un service une fois qu'il n'est plus disponible. Une solution possible serait l'utilisation d'un indicateur dans la description d'un service qui montre sa disponibilité à un moment donné : par exemple, une fois qu'un client en mode "à l'écoute" a reçu une connexion, il peut être annoncé sur le réseau à nouveau avec la description indiquant son indisponibilité. Un autre problème qui se pose dans le même contexte est le contrôle effectif de l'accès à un service. Le service peut gérer le contrôle lui-même ou, dans le cas contraire, il faudrait utiliser un mécanisme extérieur.

– **Il faut définir des interfaces qu'un service peut avoir**.
Dans la présentation faite dans cette section, nous avons pris en considération seulement les interfaces de communications dont ces services peuvent avoir par rapport à des autres composants du système. Mais, *Newkey* peut communiquer également avec le contrôleur à distance. La continuation du développement de cette direction de recherche doit prendre en compte aussi ce type d'interface. De plus, il faut définir la présentation d'interfaces de communication qu'un service peut avoir. Cela revient à la définition de ses paramètres spécifiques pour le champ TXT.

6.2.2 La généralisation de l'utilisation de la découverte de services

Le développement naturel de la proposition faite dans la section antérieure est la généralisation de l'utilisation de la découverte de services pour mettre au profit d'autres systèmes coopératifs.

L'annonce des composants d'autres systèmes sur le réseau en utilisant mDNS-SD et la découverte des services compatibles suivent les mêmes indications générales que celles faites dans la section antérieure.

Dans ce contexte, un but serait de mettre à disposition d'un utilisateur tout l'équipement informatique qui se trouve dans l'environnement spontané de communication comme support de collaboration.

Table des figures

Liste des tableaux

Bibliographie

[1] Collaborative VNC. http ://benjie.org/software/linux/vnc-collaborate.html.

[2] SharedAppVNC. http ://www.cs.princeton.edu/ gwallace/shareappvnc/sharedappvnc.html.

[3] TightVNC. http ://www.tightvnc.com/.

[4] P. Vixie A. Gulbrandsen. A DNS RR for Specifying the Location of Services (DNS SRV). Technical report, IETF RFC 2052, Network Working Group, October 1996.

[5] The Active Badge System. http ://www.uk.research.att.com/ab.html, accédé la dernière fois en Juin, 2005.

[6] H.M. Abdel-Whahab and M.A. Feit. XTV : A Framework for Sharing X Window Clients in Remote Synchronous Collaboration. In *Proc. IEEE Conference on Communications Software : Communications for Distributed Applications & Systems*, Chapel Hill, North Carolina, April 1991.

[7] Apple Rendezvous Technology. http ://www.apple.com/macosx/features/rendezvous/.

[8] John Eric Baldeschwieler, Thomas Gutekunst, and Bernhard Plattner. A Survey of X Protocol Multiplexors. *ACM SIGCOMM Computer Communication Review*, 23(2) :16–24, 1993.

[9] Daniel J. Barrett and Richard Silverman. *The Secure Shell : The Definitive Guide*. O'Reilly & Associates, Inc., 2001.

[10] John Bazik. XMX - An X Protocol Multiplexor, Project Webpage. http ://www.cs.brown.edu/software/xmx/.

[11] L'extension X BIG-REQUESTS, X Consortium Standard. http ://www.x.org/X11R6.8.1/docs/Xext/bigreq.pdf, accédé la dernière fois en Janvier, 2005.

[12] Carsten Bormann and Gero Hoffmann. Xmc and Xy-Scalable Window Sharing and Mobility, or, From X Protocol Multiplexing to X Protocol Multicasting. In *Proc. of the 8th Annual X Technical Conference, Issue 9*. O'Reilly & Associates, January 1994.

[13] John Bradly. xv, logiciel dans le domaine publique. http ://www.trilon.com/xv/index.html, accédé la dernière fois en Janvier, 2005.

[14] S. Cheshire and M. Krochmal. DNS-Based Service Discovery. Technical report, IETF draft, February 2004. Expires August 14, 2004.

[15] Stuart Cheshire and Marc Krochmal. Multicast DNS. Technical report, IETF draft, February 2005. Expire en Décembre, 2005.

[16] G. Chung, K. Jeffay, and H.Abdel-Wahab. Accomodating late-comers in shared window systems . *IEEE Computer*, 26(1) :72–74, 1993.

[17] Inc. Copyright (C) 2001-2003 HorizonLive.com. VNC Reflector. http ://www.advogato.org/proj/VNC

[18] P. Dahlberg, F. Ljungberg, and J. Sannenblad. Proxy lady : Mobile support for opportunistic communication. 1999.

[19] Thomas W. Yip Daniel Garfinkel, Bruce C. Welti. HP SharedX : A Tool for Real-Time Collaboration. *Hewlett-Packard Journal*, 1994.

[20] Hans-Peter Dommel and J. J. Garcia-Luna-Aceves. Floor control for multimedia conferencing and collaboration. *Multimedia Systems*, 5(1) :23–38, 1997.

[21] Andrzej Duda. Ambient Networking. In *Proc. Smart Objects Conference, "SOC 2003"*, France, Grenoble, May 2003.

[22] ethereal, logiciel dans le domaine publique. http ://www.ethereal.com/, accédé la dernière fois en Janvier, 2005.

[23] Jim Fulton and C. Kantarjiev. An update on low bandwidth X (LBX) ; a standard for X and serial lines. In *Proc. of the 7th Annual X Technical Conference*. O'Reilly & Associates, January 1993.

[24] Simson Garfinkel, Daniel Weise, and Steve Strassmann. Unix Haters Handbook, Chapter 7, The X-Windows Disaster. http ://catalog.com/hopkins/unix-haters/x-windows/disaster.html, accédé la dernière fois en Janvrier, 2005.

[25] James Gettys and Keith Packard. The (Re)Architecture of the X Window System. In *Proc. 2004 Ottawa Linux Symposium*, Ottawa, Canada, July 2004.

[26] Jim Gettys. The Future is Coming : Where the X Window system Should Go. In *Proc. FREENIX Track : 2002 USENIX Annual Technical Conference*, Monterey, California, SUA, June 2002.

[27] Jim Gettys, Phil Karlton, and Scott McGregor. The X Window System, Version 11. *Software Practice and Experience*, 20(S2), 1990.

[28] Thomas Gutekunst, Daniel Bauer, Germano Caronni, Bernhard Plattner, and Hasan. A distributed and policy-free general-purpose shared window system. *IEEE/ACM Transactions on Networking (TON)*, 3(1) :51–62, 1995.

[29] H.Abdel-Wahab and K. Jeffay. Issues, Problems and Solutions in Sharing X Clients on Multiple Displays. *Internetworking - Research and Practice*, 5(1) :1–15, 1994.

[30] L'application Hydra - le site web de présentation. http ://www.apple.com/downloads/macosx/productivity_tools/hydra.html, accédé la dernière fois en Avril, 2005.

[31] JavaTM 2 SDK, Standard Edition. http ://java.sun.com/j2se/1.4.2/docs/, accédé la dernière fois en Decembre, 2004.

[32] JXTA - le site web de présentation. http ://www.dhcp.org/, accédé la dernière fois en Juin, 2005.

[33] Cristopher Kent Kantarjiev, Alan Demers, Ron Frederick, Robert T. Krivacic, and Mark Weiser. Experiences with X in a Wireless Environment. In *Proc. USENIX Mobile & Location-Independent Computing Symposium*, Cambridge, Massachusets, SUA, August 1993.

[34] KDE Desktop Sharing. http ://www.kde.org/info/3.1/feature_guide_3.html.

[35] L.Gannoun and Jacques Labetoulle. Scalable Shared Window System for Large Groups Based on Multicast. In *Proc. of the Internet Global Summit, INET'98.*

[36] Sheng Feng Li, Quentin Stafford-Fraser, and Andy Hopper. Integrating Synchronous and Asynchronous Collaboration with Virtual Network Computing. *IEEE Internet Computing*, 4(3) :26–33, May/June 2000.

[37] Tal Liron. Remote-control Java : Use VNCj to export your Java user interface to VNC viewers . apparu dans la publication électronique online "Java World", http ://www.javaworld.com/javaworld/jw-07-2002/jw-0712-remote.html. accédé la dernière fois en Mai, 2005.

[38] Mitre Desktop Collaborative Desktop. http ://collaboration.mitre.org/docs/rdcwrel002.pdf.

[39] P. Mochapetris. Domain Names - Implementation and Specification. Technical report, IETF RFC 1035, Network Working Group, November 1997.

[40] Hoa-Binh Nguyen. *Services Actifs et Passerelles Programmables.* PhD thesis, Institut National Polytechnique Grenoble, January 2004.

[41] NoMachine - Introduction to NX technology. http ://www.nomachine.com/documentation/intr-components.php, accédé la dernière fois en Décembre, 2004.

[42] NX X Protocol Compression. http ://www.nomachine.com/documentation/html/NX-XProtocolCompression.html, accédé la dernière fois en Décembre, 2004.

[43] Justinian Oprescu. *Découverte et Composition de Services dans des Réseaux Ambiants.* PhD thesis, Institut National Polytechnique Grenoble, Décembre 2004.

[44] Keith Packard. Design and Implementation of LBX : an Experiment Based Standard. In *Proc. of the 8th Annual X Technical Conference.* O'Reilly & Associates, Issue Nine 1994.

[45] Keith Packard and James Gettys. X Window Network Performance. In *Proc. FREENIX Track : 2003 USENIX Annual Technical Conference*, San Antonio, Texax, SUA, June 2003.

[46] R.Borovoy, M. McDonald, F. Martin, and M. Resnick. Things that blink : Computationally augmented name tags. 35(3-4) :488–495, 1996.

[47] Real VNC. http ://www.realvnc.com.

[48] Remote Destop Sharing. http ://msdn.microsoft.com/library/default.asp ?url=/library/en-us/wcerdp/html/cmconRemoteDesktopProtocolforWindowsCE40.asp.

[49] Apple Rendezvous (Dossier Technique). http ://images.apple.com/ca/fr/education/technicalresources/pdf/L24638B-CF_Rendezvous_TB.pdf, accédé la dernière fois en Decembre, 2004.

[50] Tristan Richardson, Frazer Bennett, Glenford Mapp, and Andy Hoppe. Teleporting in an X Window System Environment. *IEEE Personal Communications Magazine*, 1(3) :6–12, August 1994.

[51] Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1) :33–38, 1998.

[52] Tristan Richardson and Kenneth R. Wood. The RFB Protocol. http ://www.realvnc.com/docs/rfbproto.pdf, January 1998.

[53] M. Roseman and S. Greenberg. Groupkit : A groupware toolkit for building real-time conferencing applications. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'92)*, pages 43–50, Toronto, Ontario, 1992. ACM Press.

[54] David Rosenthal. *Inter-Client Communication Conventions Manual, version 2.0*. X Consortium standard, 1993, Stuart W. Marks SunSoft, Inc. edition.

[55] Robert W. Scheifler. X Window System Protocol, version 11. Technical report, IETF RFC 1013, Network Working Group, June 1987.

[56] Robert W. Scheifler and James Gettys. *X Window System : The Complete Reference to Xlib, X Protocol, Icccm, Xlfd*. Digital Press, Newton, MA, 1990.

[57] Robert W. Scheifler and Jim Gettys. The X Window System. *ACM Transactions on Graphics*, 5(2) :79–100, 1986.

[58] WiredX.net - le site web. http ://wiredx.net/, accédé la dernière fois en Janvier, 2005.

[59] B. Shizuki, M. Nakasu, and J. Tanaka. VNC-based Access to Remote Computers from Cellular Phones. In *Proc. IASTED International Conference on Communication Systems and Networks (CSN 2002)*, pages 74–79, Septembre 2002.

[60] I. Smith and E. Mynatt. What You See Is What I Want : Experiences with the Virtual X Shared Window System, 1991.

[61] OpenOffice, logiciel dans le domaine publique. http ://fr.openoffice.org/, accédé la dernière fois en Janvier, 2005.

[62] Ethan Solomita, James Kempf, and Dan Duchamp. XMOVE : a Pseudoserver for X Window Movement. *The X Resource*, (11) :143–170, 1994.

[63] tcpdump , logiciel dans le domaine publique. http ://www.tcpdump.org/, accédé la dernière fois en Janvier, 2005.

[64] The performance of Java's Lists. http ://www.onjava.com/pub/a/onjava/2001/05/30/optimization.html, June 2001.

[65] Boutell. Com Inc. Thomas Boutell, Copyright 1999. rinetd, logiciel dans le domaine publique. http ://www.boutell.com/rinetd/, accédé la dernière fois en Janvier, 2005.

[66] Virtual Room Videoconferencing System. http ://www.vrvs.org.

[67] Le projet VNC. http ://www.uk.research.att.com/archive/vnc/extras.html.

[68] Description de la boîte à outil pour la construction des serveurs VNC en Java. http ://www.amherst.edu/ tliron/vncj/index.html, accédé la dernière fois en Mai, 2005.

[69] Mark Weiser. The computer for the 21 century. *ACM SIGMOBILE Mobile Computing and Communications*, 3(3), July 1999.

[70] John Wilson. VNC Proxy. http ://www.wilson.co.uk/Software/vnc/proxy/VncProxy.htm.

[71] x11perf, logiciel dans le domaine publique. http ://www.xfree86.org/4.2.0/x11perf.1.html, accédé la dernière fois en Janvier, 2005.

[72] xbench, logiciel dans le domaine publique. http ://www.ntlug.org/archive/tp/Xbench/ xbench.html, accédé la dernière fois en Janvier, 2005.

[73] Xnee - GNU Project. http ://www.gnu.org/software/xnee/, accédé la dernière fois en Septembre, 2004.

[74] xterm, logiciel dans le domaine publique. http ://dickey.his.com/xterm/xterm.html, accédé la dernière fois en Janvier, 2005.

Annexe

Annexe A

Annexe : Exemple d'échange des messages X

Ensuite nous présentons la communication X entre un client est le serveur X. Le client est un simple application de type "Hello world" :

1. Requête 55 : création d'un *contexte graphique*
2. Requête 98 : demande d'une extension
3. Requête 20 : demande d'une propriété
4. Réponse positive pour la Requête 98
5. Réponse positive pour la Requête 20
6. Requête 98 : demande d'une extension
7. Réponse positive pour la Requête 98
8. Requête 98 : demande d'une extension
9. Réponse positive pour la Requête 98
10. Requête 45 : création d'une *ressource de type police*
11. Requête 47 : demande des informations sur une police .
12. Réponse positive pour la Requête 47 .
13. Requête 45 : création d'une *ressource de type police*
14. Requête 47 : demande des informations sur une police .
15. Réponse positive pour la Requête 47 .
16. Requête 16 : InternAtom
17. Réponse positive pour la Requête 16
18. Requête 16 : InternAtom
19. Réponse positive pour la Requête 16 .
20. Requête 20 : demande d'une propriété

21. Réponse positive pour la Requête 20 .
22. Requête 92 : recherche d'une couleur
23. Réponse positive pour la Requête 92
24. Requête 84 : allocation d'une entrée dans une colormap
25. Réponse positive pour la Requête 84
26. Requête 92 : demande de recherche d'une couleur
27. Réponse positive pour la Requête 92
28. Requête 84 : allocation d'une entrée dans une colormap
29. Réponse positive pour la Requête 84
30. Requête 1 : création d'une *fenêtre*
31. Requête 18 : changement d'une propriété
32. Requête 18 : changement d'une propriété
33. Requête 18 : changement d'une propriété
34. Requête 18 : changement d'une propriété
35. Requête 18 : changement d'une propriété
36. Requête 55 : création d'un *contexte graphique*
37. Requête 2 : changement des attributs
38. Requête 2 : changement des attributs
39. Requête 8 : demande que la fenêtre devient visible
40. Événement 12 : expose
41. Requête 14 : demande la géométrie d'une fenêtre ou pixmap
42. Réponse positive pour la Requête 14
43. Requête 61 : affranchir un secteur d'une fenêtre
44. Requête 74 : demande de dessiner un texte

Annexe B

Annexe : La programmation du service actif

Nous présentons un fragment de la programmation du service actif réalisée dans cette deuxième version :

```
(...)

Transmit(0) ;

if (( comInfo.serversNumber != 1 ) && ((opcode == 55)||(opcode == 56))){

    if ( 4*(length-4) != 0){

        value_list = Tools.treat55(2,value_mask,value_list,this) ;

    }

    if ( Tools.existField("gid",this) == 1 ){

        gid = Tools.addIds(1,gid,this) ;

        gid = Tools.addIds(2,gid,this) ;

    }

}

(...)
```

Annexe C

Annexe : Présentation du protocole NewkeyCP

Le protocole Newkey est un protocole synchrone : à chaque message envoyé par un contrôleur à distance corresponde un réponse qui infirme ou confirme la réalisation de l'opération. Nous présentons ci-dessous la description des messages du protocole. La description utilise les notations : *byte*, *int*, *byte[dim]*. Elles désignent les types de données : un octet et un entière signés, et un tableau d'octets de dimension "dim".

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 0
dimClient	byte	
deClient	byte[dimClient]	le client impliqué dans la session
dimServeur	byte	
serveur	byte[dimServeur}	le serveur impliqué dans la session
portServeur	int	
existe	byte	(existe = 1) => il suit un mot de passe
dimPasswd	byte	
mot_de_passe	byte[dimPasswd]	mot de passe d'accès au serveur

FIG. C.1 – Le message "connectToAServer" qui permet la mise en place d'une connexion d'un client à un serveur VNC.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 1
proxyPort	int	le port demandé pour la session
dimServeur	byte	
serveur	byte[dimServeur}	le serveur impliqué dans la session
portServeur	int	
IDsession	int	ID de la session
existe	byte	
dimPasswd	byte	(existe = 1) => il suit un mot de passe
mot_de_passe	byte[dimPasswd]	mot de passe d'accès au serveur

FIG. C.2 – Le message "connectToServer Mux" qui permet la mise en place d'une session de type "connectToServer" basée sur le mécanisme de multiplication du flot RFB.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 2
IDsession	int	ID de la session
dimServeur	byte	
serveur	byte[dimServeur}	le serveur impliqué dans la session
portServeur	int	
NoClients	byte	
dimClient	byte	
auClient$_1$	byte[dimClient]	client impliqué dans la session
...		
dimClient	byte	
auClient$_{NoClients}$	byte[dimClient]	client impliqué dans la session
existe	byte	(existe = 1) => il suit un mot de passe
dimPasswd	byte	
mot_de_passe	byte[dimPasswd]	mot de passe d'accès au serveur

FIG. C.3 – Le message "connectToServer Router" consacré aux sessions de type "connectToServer" basées sur le mécanisme d'acheminement de flots RFB.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 5
dimClient	byte	
auClient	byte[dimClient]	le client impliqué dans la session
portClient	int	
dimServeur	byte	
serveur	byte[dimServeur}	le serveur impliqué dans la session
portServeur	int	
existe	byte	(existe = 1) => il suit un mot de passe
dimPasswd	byte	
mot_de_passe	byte[dimPasswd]	mot de passe d'accès au serveur

FIG. C.4 – Le message "connectToAClient" qui permet la connexion automatique d'un serveur à un client VNC.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 6
IDsession	int	ID de la session
dimServeur	byte	
serveur	byte[dimServeur}	le serveur impliqué dans la session
portServeur	int	
NoClients	byte	
dimClient	byte	
$auClient_1$	byte[dimClient]	premier client de destination
$portClient_1$	int	le port sur lequel le client écoute
...		
dimClient	byte	
$auClient_{NoClients}$	byte[dimClient]	
$portClient_{NoClients}$	int	
existe	byte	(existe = 1) => il suit un mot de passe
dimPasswd	byte	mot de passe d'accès au serveur
mot_de_passe	byte[dimPasswd]	

FIG. C.5 – Le message "connectToClients Router" qui a le rôle dans la mise en place d'une session de type "connectToClient" basée sur l'acheminent des flots RFB.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 7
IDsession	int	ID de la session
dimServeur	byte	
serveur	byte[dimServeur]	le serveur impliqué dans la session
portServeur	int	
NoClients	byte	
dimClient	byte	
auClient$_1$	byte[dimClient]	premier client de destination
portClient$_1$	int	le port sur lequel le client écoute
...		
dimClient	byte	
auClient$_{NoClients}$	byte[dimClient]	
portClient$_{NoClients}$	int	
existe	byte	(existe = 1) => il suit un mot de passe
dimPasswd	byte	mot de passe d'accès au serveur
mot_de_passe	byte[dimPasswd]	

FIG. C.6 – Le message "connectToClients Mux" qui permet la mise en place d'une session de type "connectToClient" basée sur la mécanisme de multiplication de flot RFB.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 12
IDsession	int	ID de la session
dimHost	byte	
hôte	byte[dimHost]	pour identifier l'utilisateur
portHôte	int	inutilisé

FIG. C.7 – Le message "setFloor" qui demande d'accorder le droit d'entrée (le *floor*) au participant désigné en cadre d'une session coopérative. Nous remarquons qu'un participant est identifié par sa machine. Pour la machine, on admit également le mot clé "all" pour désigner tous les participants à la session concernée.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 21
IDsession	int	ID de la session
dimHost	byte	
hôte	byte[dimHost]	pour identifier l'utilisateur
portHôte	int	inutilisé

FIG. C.8 – Le message "freeFloor" qui demande de retirer le droit d'avoir d'entrée (le *floor*) au participant désigné en cadre d'une session coopérative. Nous remarquons, encore une fois, qu'un participant est identifié par sa machine. Pour la machine, on admit également le mot clé "all" pour désigner tous les participants à la session concernée.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 3
objet	byte	(objet=-1) => toutes configurations
		(objet=0) => sessions générées par msgType 0
		(objet=1) => sessions générées par msgType 1
		(objet=2) => sessions générées par msgType 2

FIG. C.9 – Le message "showSettings" affiche des informations sur des sessions en cours sur Newkey.

Champ de message	Type de données	Commentaire
réponseType	byte	(réponseType = 1) => Succès (réponseType = 1) => Erreur
dimMSG	byte	
MSG	byte[dimMSG]	message conforme au type de réponse

FIG. C.10 – Le message de réponse. Celui-ci peut confirmer l'exécution du message envoyé ou signaliser un erreur. Le message inclut une explication conforme au message envoyé ou à l'erreur.

Annexe D

Annexe : Présentation du protocole FloCtrl

Préservant les mêmes notations comme dans la section précédente, nous présentons maintenant le protocole utilisé dans la communication entre les composants du système de contrôle de *floor* : le service de contrôle et les Uis de notre implémentation. Le protocole est composé de requêtes et des événements.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 0
IDsession	int	ID de la session
NoElList	byte	le nombre de participants dans la session
dimUser	byte	
nameUSer$_1$	byte[dimUser]	le nom du premier participant
dimUserAdr	byte	
userAdr$_1$	byte[userAdr]	la machine du premier participant
exist	byte	(exist=1) => il y a une adresse d'email
dimUserEmail	byte	
userEmail$_1$	byte[userEmail]	l'adresse d'email du premier participant
userRole$_1$		son rôle dana la collaboration
...		(la description d'autres NoElList-1 participants)
userRole$_{NoElList}$	byte	le rôle dans la session du NoElList-ème participant
dimOwner	byte	
owner	byte[dimOwner]	le proprietaire de la session
dimOb	byte	(dimOb=0)=> on a pas précisé l'objet à partager
obj	byte[dimObj]	l'objet à partager dans la session

FIG. D.1 – Le message "setSession" qui permet la mise en place d'une session de contrôle de *floor*, qui corresponde à un session coopérative du Newkey.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 1
IDsession	int	ID de la session
dimSource	byte	
source	byte[dimSource]	l'identification de la source se fait par sa machine

FIG. D.2 – Cette requête "askFloor" est utilisée pour demander au service de contrôle le *floor*.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 2
IDsession	int	
dimDest	byte	
àQui	byte[dimDest]	le destinataire du *floor* identifiée par sa machine
dimSource	byte	
duQui	byte[dimSource]	la source qui a donné le *floor* identifiée par sa machine

FIG. D.3 – Message "giveFloor" utilisé pour accorder à un autre participant le *floor*.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 3
IDsession	int	ID de la session
dimSource	byte	
source	byte[dimSource]	!identification de la source se fait par sa machine

FIG. D.4 – Message "releaseFloor" qu'informe le relâchement du *floor*.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 4
IDsession	int	
dimDest	byte	
àQui	byte[dimDest]	le destinataire du *floor* identifiée par sa machine
dimSource	byte	
duQui	byte[dimSource]	la source qui a donné le *floor* identifiée par sa machine

FIG. D.5 – Message "retireFloor" utilisé pour demander de retire la *floor* à un autre participant.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 9
dim	byte	
nouveauPossesseurFloor	byte[dim]	le nouveau possesseur du *floor*

FIG. D.6 – Événement "floorHolder" qui annonce le changement du possesseur du *floor*. Il est envoyé aux participants suite d'accomplissement d'un requête qui demande le *floor* pour soi-même ou pour un autre participant.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 10
dim	byte	
quiDemande	byte[dim]	le participant qui a demandé le *floor*

FIG. D.7 – Événement "demandeFloor" qu'informe que le *floor* a été demandé, mais il n'a pas été obtenu par un participant. Il est envoyé suite à une requête de *floor* qui n'a pas été accomplie.

Champ de message	Type de données	Commentaire
msgType	byte	msgType = 11
dim	byte	
deDemande	byte[dim]	le participant qui a débloqué le *floor*

FIG. D.8 – Événement "freeFloor" qui annonce que le *floor* n'as plus de possesseur. Il est envoyé aux participants suite au relâchement du *floor* ou à une demande de retirer la *floor* à un participant.

Annexe E

Annexe : Contrôleur à distance. Service de contrôle de *floor*

```
cpop@borneo:~> !ja
java remoteRFBProxyControl
  U have the possibility to send the follow messages from th
e command line:
 (connectToAServer): 0 proxyToControl fromVNCClient toVNCSer
ver VNCserverPort [password]
 (connectToAClient):5 proxyToControl client portClient serve
r portServer [passwd]
 (connectToServer  Mux): 1 proxyToCOnnect desiredProxyPort V
NCserver VNCserverPort shared(1/0) sessionIdentification [pa
ssword]
 (connectToServer  router): 2 proxyToConnect  idSession(no)
[password]/null  vncServer portServer noElements client_1 ..
. client_noElements
 (connectToClients router): 6 proxyToControl idSession(no) [
passwd]/null vncServer portServer  noElements client_1 port_
1 ... client_noElements portClient_noElements
(connectToClients mux): 7 proxyToControl idSession(String) [
passwd]/null vncServer portServer noElements client_1 port_1
 ... client_noElements portClient_ noElements
 (getFloor): 40 proxyToAsk connectionModeSession(1='setting
connection' / 2='connect to client')  typeSession(1=allocate
d/2=multiply desktop type) sessionId forWho(host)/all
 (releaseFloor): 41 proxyToAsk connectionModeSession(1='sett
ing connection' / 2='connect to client')  typeSession(1=allo
cated/2=multiply desktop type) sessionId forWho(host)/all
(show settings): 3 proxyToAsk  -1/0/1/2(-1: all settings/ 0:
 connection settings(msg0) / 1: allocated ports for sessions
(msg1) / 2: multiply settings(msg2)

cpop@borneo:~>
```

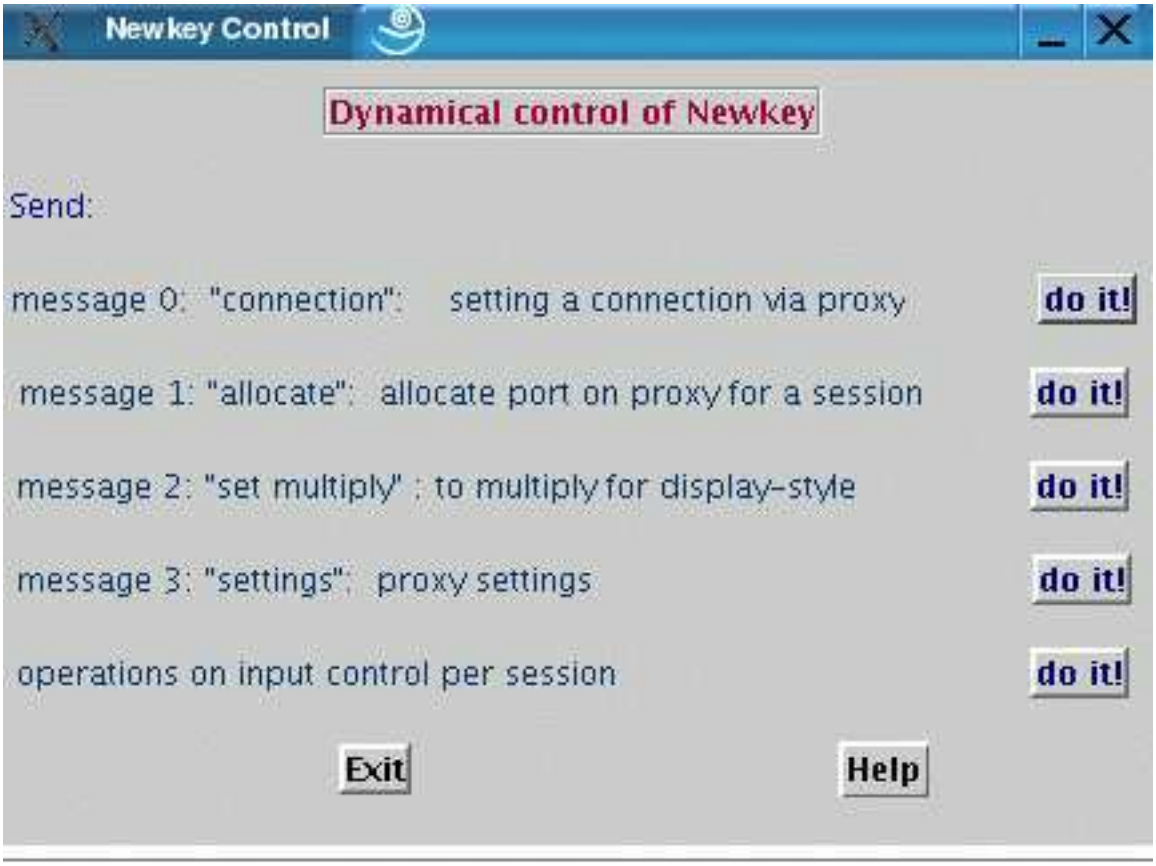

FIG. E.2 – La fenêtre principale de l'interface graphique du contrôleur à distance.

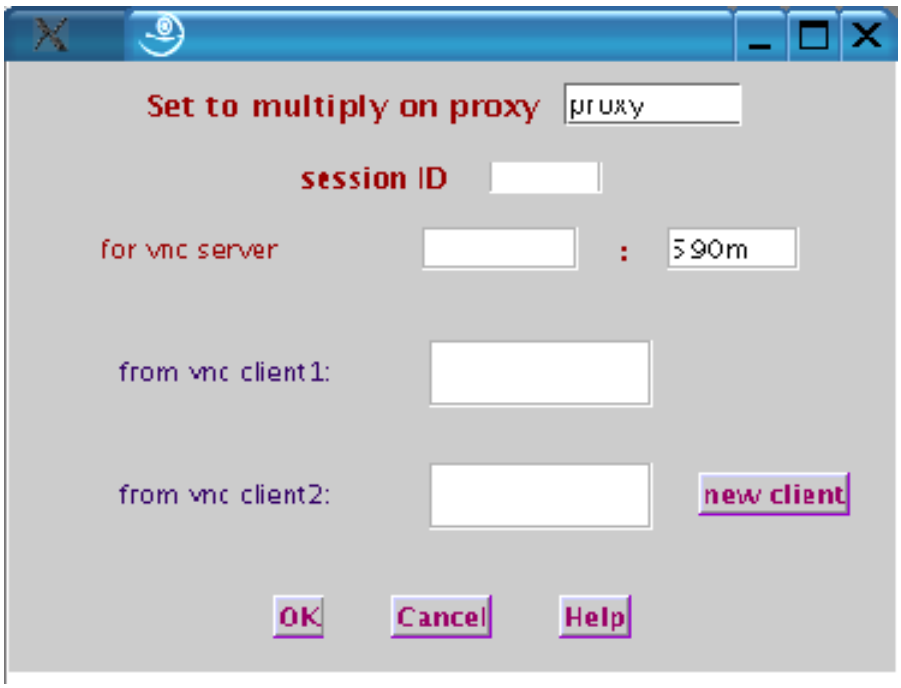

FIG. E.3 – La fenêtre d'une opération de contrôle offerte par l'interface graphique du contrôleur à distance.

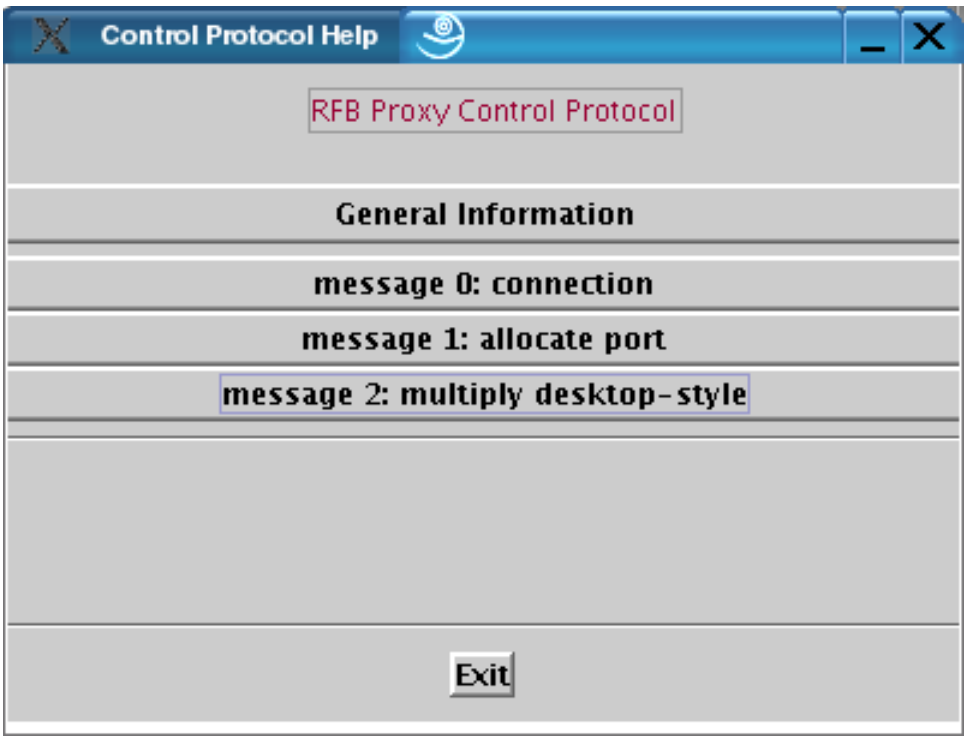

FIG. E.4 – La fenêtre principale d'aide offerte par l'interface graphique du contrôleur à distance.

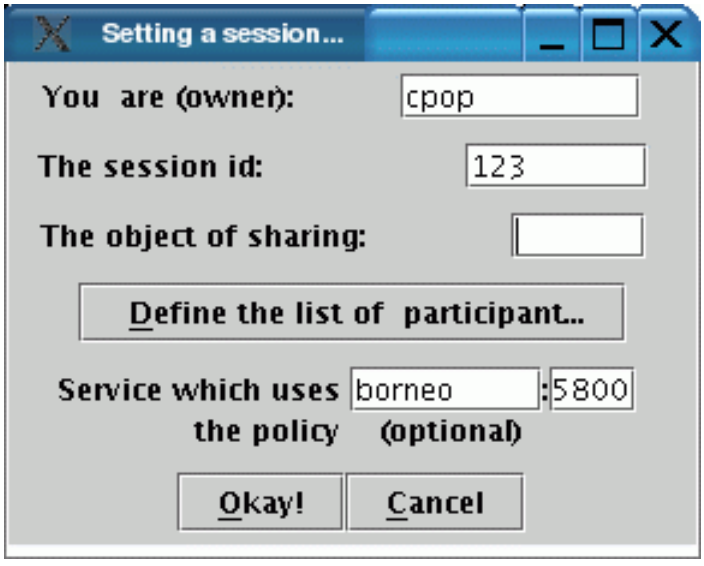

FIG. E.5 – La fenêtre principale du service de contrôle.

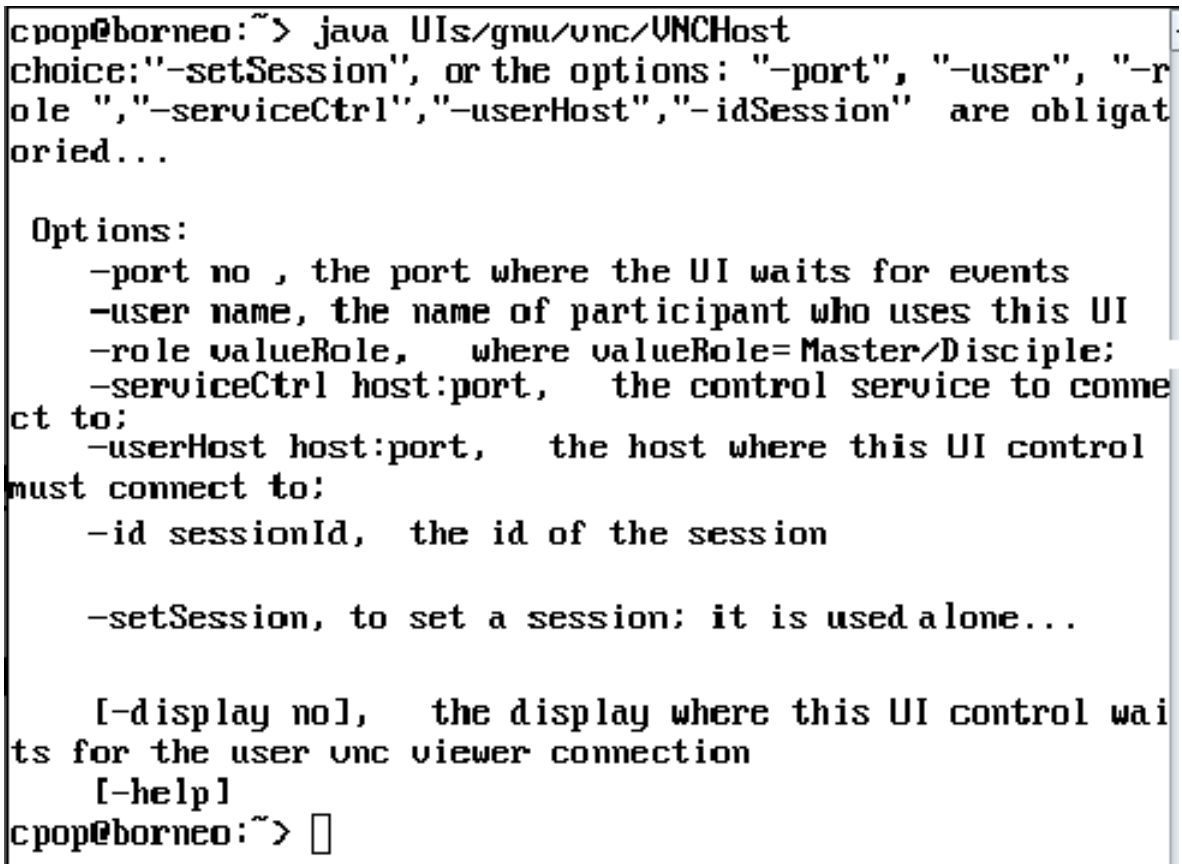

FIG. E.6 – L'interface en ligne de commande d'un serveur VNC *UI* qui offre l'interface graphique de contrôle de *floor* à un utilisateur.

Conception et développement d'une infrastructure de communication collaborative

Résumé. Ces dernières années le déploiement de réseaux sans fil a changé le paysage des réseaux informatiques. Grâce à l'évolution de la technologie informatique un utilisateur peut étendre son environnement de travail habituel en bénéficiant de la connectivité réseau accrue. La gestion souple de la mobilité lui permet d'accéder à différents environnements de collaboration. Cette thèse considère le problème de partage de l'affichage d'applications dans les conditions de la diversification des contextes de collaboration. Étant donné le déploiement du système de fenêtrage X et du système d'affichage à distance VNC, nous les avons choisi comme base pour notre travail. D'une part, nous avons conçu un système de partage d'applications X qui gagne en flexibilité par son interface du contrôle d'évènements d'entrée. Le système repose sur un multiplexeur de flots X conçu comme un service actif qui peut être chargé dynamiquement sur une plateforme générique. D'autre part, nous avons transformé le système VNC en un système coopératif flexible. Le nouveau système est construit autour d'un proxy dynamiquement contrôlé à distance. Le proxy peut être programmé pour gérer différentes sessions de collaboration. Chaque session de collaboration peut utiliser des flots de données de VNC multiples. L'ouverture du système vers des politiques de contrôle d'évènements entrée permet l'adaptation au contexte de coopération. Finalement nous présentons des nouvelles perspectives concernant les possibilités de collaboration assistée par la technologie informatique dans un réseau informatique quelconque.
Mots clé : environnement spontané de communication, X Window, VNC, TCAO, partage d'applications

Design and development of an infrastructure of collaborative communication

Abstract. Recently the deployment of the wireless technology changed the landscape of the computer networks. Due to the evolution of the computer technology an user can extend his working environment using the benefit of enlarged area of connection. The flexible management of the mobility permits the user to access different collaboration environments. This thesis considers the problematic of sharing the application displays from the angle of the diversification of the collaboration contexts. Taking into account the large popularity of the X Window System as well as of the remote display system VNC they have been chosen as the basis for our work. In the first part of the thesis we present a system for sharing X applications which provides more flexibility by its interface for input control. This system is based on a X multiplexer designed as an active service using a generic platform and can be loaded dynamically. In the second part of the thesis we transformed the VNC system into a flexible cooperative system. This new system was built around a proxy which can be remotely controlled. The proxy is programmable and can manage different collaboration sessions. Each collaboration session is able to use multiple VNC data streams. The openes of the system permits different input control policies and thus adaptation to the collaboration context. Finally new perspectives concerning the discovery of the collaboration possibilities in any computer network are presented.
Keywords : spontaneous environment of communication, X Window, VNC, CSCW, applications sharing.

Discipline : Informatique, Systémes et Communications
Laboratoire : LSR-IMAG - équipe Drakkar
BP 72, 38402 Saint Martin d'Hères Cedex, France