

# **Diplôme de Recherche Technologique**

présenté par **Nicolas Tachker**

**Spécialité Informatique**

**Extension des fonctions d'un logiciel "pare-feu"**

**Date de la soutenance : 29 octobre 1999**

**composition du jury :**

**Roland Balter  
Jacques Briat  
Nouar Garcia  
François -Xavier Le Dimet  
Serge Lacourte  
Michel Riveill  
Frederic Soinne**

Diplôme de Recherche Technologique préparé chez Bull S.A. Echirrolles

## Remerciements

Tout d'abord, je tiens à remercier toutes les personnes de Bull, de Dyade et de l'Inria qui m'ont accueilli et assisté durant ces deux années.

Je remercie plus particulièrement :

Roland Balter, responsable du projet Sirac  
Daniel Monges, responsable du service Internet et Sécurité  
Serge Lacourte, responsable du projet AAA  
Frédéric Soinne, responsable du projet NetWall

Ainsi que tous les membres des équipes NetWall, Bronco et Gestion de Configuration pour la confiance et l'intérêt qu'ils m'ont témoigné tout au long de mon contrat.

## Table des matières

<b>CHAPITRE 1 : INTRODUCTION .....</b>	<b>6</b>
1.1 CONTEXTE DU PROJET .....	7
1.1.1 <i>SIRAC, le laboratoire</i> .....	7
1.1.2 <i>Bull, centre de développements industriels</i> .....	7
1.1.3 <i>Dyade, partenariat Bull - Inria</i> .....	8
1.2 CONSTATS .....	8
1.2.1 <i>Limitations actuelles de NetWall</i> .....	8
1.2.2 <i>Technologie développée dans Sirac</i> .....	9
1.3 OBJECTIFS DU TRAVAIL .....	10
1.3.1 <i>Ajout d'un filtre pour SQL*Net (protocole dynamique)</i> .....	10
1.3.2 <i>Définition d'une architecture d'ouverture pour NetWall</i> .....	10
1.3.3 <i>Intégration des fonctions d'extension par agents</i> .....	10
<b>CHAPITRE 2 : DESCRIPTION DE L'EXISTANT .....</b>	<b>12</b>
2.1 NETWALL .....	13
2.1.1 <i>Description de NetWall</i> .....	13
2.1.2 <i>Résumé des fonctionnalités de NetWall</i> .....	15
2.1.3 <i>Limitation de NetWall</i> .....	16
2.2 AAA "AGENT ANYTIME ANYWHERE " .....	17
2.2.1 <i>Introduction</i> .....	17
2.2.2 <i>Architecture globale</i> .....	18
2.2.3 <i>Utilisation pour enrichir NetWall</i> .....	19
<b>CHAPITRE 3 : DESCRIPTION DU TRAVAIL RÉALISÉ.....</b>	<b>20</b>
3.1 PRINCIPES GÉNÉRAUX DE L'APPROCHE .....	21
3.1.1 <i>Introduction</i> .....	21
3.1.2 <i>SQL*Net</i> .....	21
3.1.3 <i>L'open framework</i> .....	21
3.1.4 <i>Liens entre l'open framework et H.323</i> .....	22
3.1.5 <i>SDK</i> .....	23
3.2 ARCHITECTURE .....	24
3.2.1 <i>NetWall</i> .....	24
3.2.2 <i>Open framework</i> .....	27
3.2.3 <i>H.323</i> .....	33
3.2.4 <i>Librairie SDK</i> .....	34
3.2.5 <i>Schéma de l'architecture OFW, SDK et H.323</i> .....	35
3.2.6 <i>Le multi plates-formes</i> .....	36
3.3 RÉALISATION.....	37
3.3.1 <i>Protocoles dynamiques</i> .....	37
3.3.2 <i>librairie open framework</i> .....	46
3.3.3 <i>AAA, H.323 et les logs</i> .....	46
3.3.4 <i>librairie SDK</i> .....	47
3.4 EXPÉRIMENTATIONS .....	47
3.4.1 <i>SQL*Net</i> .....	49
3.4.2 <i>Sun RPC</i> .....	49
3.4.3 <i>OFW</i> .....	49
3.4.4 <i>Netmeeting et Netscape conférence</i> .....	50
3.4.5 <i>librairie SDK et les proxies</i> .....	51
<b>CHAPITRE 4 : BILAN ET PERSPECTIVES .....</b>	<b>52</b>
4.1 RAPPEL DES OBJECTIFS ET BILAN TECHNIQUE .....	53
4.2 PERSPECTIVES .....	54
4.3 BILAN PERSONNEL .....	56
<b>GLOSSAIRE .....</b>	<b>58</b>
<b>RÉFÉRENCES .....</b>	<b>60</b>

---

<b>ANNEXES .....</b>	<b>62</b>
1.1 FONCTIONS COUVERTES PAR NETWALL .....	63
1.1.1 Filtrage dynamique au niveau IP.....	63
1.1.1 Fonctions de log et d'audit. ....	64
1.1.2 Configuration et administration via une interface graphique conviviale. ....	64
1.1.3 Authentification des utilisateurs au niveau des relais applicatifs.....	64
1.1.4 Fonctions de coopération des relais applicatifs avec des produits externes d'Antivirus.....	65
1.1.5 Contrôle à distance sécurisé.....	65
1.1.6 Fonction d'alerte dynamique.....	65
1.2 AGENT ANYTIME ANYWHERE .....	66
1.2.1 Réalisation .....	66
1.2.2 Extensions.....	75
1.2.3 Outils de construction.....	77
1.3 CMVC ET ODE, GESTION DES SOURCES .....	80
1.3.1 Introduction .....	80
1.3.2 Gestion de la configuration de NetWall.....	80
1.3.3 Glossaire.....	85
1.4 FORMATION POUR LE DRT .....	86
1.4.1 Description des enseignements sélectionnés en 1997-1998.....	86
1.4.2 Description des enseignements sélectionnés en 1998-1999.....	87
1.4.3 Formation suivie chez Bull .....	88

Durant la dernière décennie, le réseau informatique mondial Internet a connu une croissance exponentielle. En effet il permet d'échanger de très grande quantité d'informations dans des délais extrêmement courts, ce qui permet, notamment, d'augmenter la productivité des entreprises. Cependant, le raccordement de ces dernières au réseau mondiale ainsi que le caractère confidentiel des informations qu'elles envoient, nécessite de développer des logiciels nouveaux permettant d'assurer une protection vis-à-vis du piratage ou de la malveillance. L'ensemble de ces logiciels constituent le domaine de la Sécurité Informatique.

Afin de répondre à l'attente de ses clients, la société Bull a développé le pare-feu NetWall à partir de 1995. C'est dans le cadre de ce projet que j'ai effectué mon Diplôme de Recherche Technologique du 3 novembre 1997 au 2 novembre 1999, au sein de l'équipe Internet et Sécurité de la société Bull à Echirolles.

L'objectif de ce rapport est donc de présenter l'environnement de travail et l'existant lors de mon arrivée dans l'équipe, d'expliquer les motivations et les enjeux de la collaboration entre Bull et l'Inria et de décrire le travail réalisé et l'expérience que cette formation m'a permis d'acquérir.

**Chapitre 1 : INTRODUCTION**

Ce chapitre présente le contexte structurel et scientifique du projet. Dans un premier temps, nous donnons une brève présentation du laboratoire et de l'entreprise qui ont servi de cadre à ce travail. Dans un deuxième temps, nous décrivons la problématique scientifique qui a conduit à ce travail. Les limitations actuelles sur le logiciel de pare-feu NetWall sont d'abord exposées. Mon travail a porté sur la conception et la réalisation d'extensions à ce logiciel pour lever ces limitations. La technologie nécessaire à ces extensions a été développée en partie au sein de l'équipe NetWall et en partie par une opération de transfert technologique depuis le laboratoire Sirac.

## 1.1 Contexte du projet

### 1.1.1 SIRAC, le laboratoire

SIRAC (*Systèmes Informatiques Répartis pour Applications Coopératives*) est un laboratoire commun à l'Université Joseph Fourier, à l'Institut National Polytechnique de Grenoble, et à l'Institut National de Recherche en Informatique et Automatique (Unité de Recherche Rhône-Alpes). Le laboratoire comprend une trentaine de chercheurs, dont neuf permanents (chercheurs de l'INRIA et enseignants-chercheurs de l'université), une dizaine de doctorants et des ingénieurs contractuels. Le laboratoire mène depuis plusieurs années des recherches dans le domaine des *systèmes et applications répartis*. Il entretient de longue date une tradition de partenariat étroit avec des industriels, en particulier avec le Groupe Bull.

L'objectif général du laboratoire SIRAC est de **concevoir et réaliser des services et outils pour le développement et l'exécution d'applications réparties**. A cet effet, le laboratoire développe des technologies répondant à deux besoins : a) construire des applications réparties en combinant des techniques de programmation à base d'objets et des techniques de programmation par agents ; b) faciliter la configuration, le déploiement, l'administration et l'évolution de ces applications sur des plates-formes d'usage courant. Le travail de DRT présenté ici s'inscrit dans ce programme de recherche et vise à transférer des résultats du laboratoire vers le groupe Bull.

### 1.1.2 Bull, centre de développements industriels

L'équipe NetWall fait partie de la division Bull Soft et est intégrée dans le département AccessMaster.

AccessMaster est un progiciel qui permet d'instaurer une politique de sécurité centralisée et homogène de l'ensemble des systèmes d'information distribués et hétérogènes de l'entreprise ainsi que des accès aux réseaux de télécommunications externes et à l'Internet. L'administrateur peut gérer le réseau global Internet/Intranet de l'entreprise depuis un point de contrôle unique. Cette capacité repose sur la gestion centralisée des utilisateurs. Ceux-ci disposent, en effet, d'un mot de passe unique (single sign-on) qui leur permet d'accéder de façon transparente à toutes les applications auxquelles l'administrateur leur donne accès.

Le logiciel NetWall, qui fait partie de la suite AccessMaster, est un pare-feu TCP/IP. Un pare-feu est un composant (matériel et/ou logiciel) utilisé dans un environnement de réseau pour protéger une machine sensible ou un réseau. En outre, il contrôle l'accès entrant et sortant de ce réseau par le biais de mécanismes de filtrage du trafic TCP/IP.

NetWall comprend à la fois des capacités de filtrage IP dynamique et un ensemble de passerelles applicatives permettant aux responsables de la sécurité ou aux administrateurs de contrôler les accès entre leurs réseaux privés et Internet, d'isoler les sous-réseaux internes et de protéger les serveurs sensibles.

De plus, NetWall prend en charge toutes les méthodes d'authentification standards.

Un antivirus associé à NetWall renforce la sécurité des réseaux locaux et des postes de travail.

### *1.1.3 Dyade, partenariat Bull - Inria*

Le partenariat stratégique entre Bull et l'Inria est organisé sous la forme d'un Groupement d'Intérêt Économique (GIE) d'une durée de cinq ans, à partir du 1er avril 1996.

La vocation de Dyade est de concevoir et de développer de nouveaux composants technologiques pour les futurs systèmes d'information. Dyade a pour origine une vision commune, entre Bull et l'Inria, de la nouvelle dimension des technologies de la communication et du traitement de l'information.

Les objectifs de Dyade sont de conjuguer l'expertise en recherche amont de l'Inria et la capacité industrielle de Bull ; de définir et exploiter de nouvelles filières technologiques répondant aux besoins des systèmes d'information et d'intermédiation.

Dyade doit pour cela identifier les domaines d'action à fort potentiel innovant, démontrer la maturité et l'intérêt des technologies développées avec des utilisateurs pilotes, transférer et valoriser les résultats dans les unités opérationnelles de Bull et dans les filiales ou "start-up" de Bull ou de l'Inria.

Bull peut ainsi renforcer son portefeuille technologique et accéder à un marché à fort potentiel de croissance. L'Inria peut valider et valoriser ses résultats et identifier de nouveaux axes de recherche stratégiques pour l'industrie française.

## **1.2 Constats**

### *1.2.1 Limitations actuelles de NetWall*

#### *1.2.1.1 Filtre sur des protocoles dynamiques complexes*

Les protocoles de la couche TCP ou UDP se répartissent en deux catégories. D'une part, les protocoles simples qui utilisent une seule connexion pour assurer la communication entre le client et le serveur. Cette connexion s'établit entre un port fixe du serveur (associé au service TCP ou UDP considéré) et un port que le client s'alloue dynamiquement. D'autre part, les protocoles dynamiques qui, à partir d'une première connexion entre un port fixe du serveur et un port dynamique du client, en ouvrent d'autres après échange des valeurs des ports associées à ces connexions. Par conséquent, filtrer des protocoles dynamiques complexes peut être très compliqué, voir impossible avec l'architecture actuelle de NetWall. Nous avons pu le constater pour SQL\*Net et H.323 (vidéo conférence sur Internet).

Pour le protocole SQL\*Net, nous avons pu ajouter le filtre au niveau du driver de NetWall, mais nous imposons certaines restrictions sur l'utilisation de ce protocole, comme par exemple l'impossibilité de filtrer des données cryptées, de plus le "listener" (démon d'écoute du protocole SQL\*Net) doit être le serveur de données, etc.

Pour le protocole H.323, le problème est plus compliqué, car les données échangées sont codées en ASN.1. Les machines utilisant ce protocole échangent des numéros de ports alloués dynamiquement afin d'ouvrir des connexions permettant le transfert des données, par exemple de type audio et vidéo.

Or, tout comme dans le noyau des systèmes d'exploitation, il n'est pas possible de décoder de l'ASN.1 dans le driver de NetWall ce qui permettrait de trouver les numéros de port alloués dynamiquement.

Pour résoudre ce problème, il faut alors filtrer H.323 au niveau applicatif c'est-à-dire dérouter la totalité du trafic dans l'espace utilisateur. Une telle solution rend le protocole inexploitable du fait du trop faible débit du son et de l'image.

Par conséquent, il faut analyser les connexions de contrôle (celles sur lesquelles transitent les numéros de ports dynamiques) dans l'espace utilisateur et laisser passer les données (sons et images) sans les dérouter mais l'architecture de NetWall ne le permet pas sans certaines améliorations.

### 1.2.1.2 *Gestion des logs*

NetWall fournit plusieurs fichiers de logs : un pour les traces du driver de NetWall, un pour les proxies, un pour le démon d'administration, etc. De plus, les formats d'écriture dans ces fichiers de logs sont différents. Il est donc difficile de regrouper l'ensemble des traces associées à une connexion, surtout s'il y a un déroulement proxy, car il faut mettre en relation plusieurs fichiers de logs qui contiennent des verbes différents. Ces problèmes ne seraient pas très gênants si les fichiers de logs ne faisaient que quelques dizaines de lignes. Or ce n'est pas le cas, NetWall fournit un volume moyen de logs d'environ 200 Mo par jour. Bien sûr ce chiffre dépend de la fréquentation sur les serveurs protégés par NetWall mais aussi du niveau de logs fixé par l'administrateur. Nous voyons bien que l'exploitation des fichiers de log, pour un administrateur en charge de la sécurité, est difficile et fastidieuse sans outils et sans harmonisation des fichiers de logs.

## 1.2.2 *Technologie développée dans Sirac*

L'action Dyade AAA (Agents Anytime Anywhere) vise à fournir des outils et des services pour faciliter l'extension d'applications existantes, soit par enrichissement des fonctions de l'application cible (par exemple pour définir des filtres à la demande pour une application de pare-feu), soit par interconnexion d'applications existantes (par exemple pour la mise en coopération d'applications interactives mono utilisateur). L'approche suivie s'appuie sur une technologie à agents. Un agent est une unité d'exécution autonome mono-localisée qui communique avec l'extérieur par un mécanisme événement-réaction. Le comportement d'un agent est décrit par une classe Java qui hérite d'une classe prédéfinie. L'intégration d'applications existantes est réalisée par des agents d'interfaçage (wrappers). La spécificité de AAA résulte des propriétés de l'environnement d'exécution des agents : communication asynchrone par messages typés, garantie de délivrance des messages, ordre causal de délivrance des messages, persistance des agents et atomicité de la réaction exécutée à l'arrivée d'un message. Cet environnement d'exécution est lui-même réalisé en Java, ce qui assure sa portabilité. Il est utilisé dans deux domaines applicatifs : le pare-feu (NetWall) et l'administration de systèmes (ISM/Open-Master).

La contribution du projet Sirac à l'action AAA concerne les outils et services pour la configuration et la reconfiguration des agents. Ces outils facilitent la réutilisation des agents, leur personnalisation et leur intégration au sein d'une application. Ils permettent de faire évoluer une application en fonction des besoins, en apportant une perturbation minimale à son fonctionnement.

Une application réelle (la gestion du fichier journal (log) créé par NetWall) a mis en évidence la puissance d'expression et la souplesse d'utilisation des outils de configuration développés dans AAA.

Le système AAA ainsi que les outils et les services de configuration et reconfiguration développés par Sirac sont écrits en Java et sont donc disponibles sur toute plate-forme supportant une machine virtuelle Java. Pour information, l'outil graphique de configuration représente environ 30 000 lignes de code Java et le système AAA, étendu avec les services de configuration et de reconfiguration, représente environ 40 000 lignes de code Java.

### **1.3 Objectifs du travail**

L'objectif global de mon travail a été de trouver et de mettre en œuvre des solutions techniques pour apporter des solutions aux problèmes qui viennent d'être présentés. Trois types d'action ont été plus particulièrement réalisées pour atteindre cet objectif : la mise en œuvre de protocole dynamiques ; la définition d'une architecture ouverte pour NetWall afin de permettre plus aisément l'intégration de nouvelles fonctions ; l'intégration d'un environnement à agents, développé au sein du GIE Dyade, qui permet de définir à la demande de nouvelles fonctions de traitement des informations enregistrées par le pare-feu. Ces points sont brièvement introduits ci-dessous et présentés plus en détail dans la suite du document.

#### *1.3.1 Ajout d'un filtre pour SQL\*Net (protocole dynamique)*

Une forte demande des utilisateurs de NetWall était le filtrage de SQL\*Net. L'ajout d'un filtre pour SQL\*Net dans le driver de NetWall était un des objectifs premiers de mon DRT.

SQL\*Net est un nom générique désignant l'ensemble des couches nécessaires à la communication entre clients et serveurs Oracle. Ce protocole passe notamment au travers des couches TCP et IP. Il utilise des adresses IP et des ports de communication pour se connecter aux serveurs de données. La difficulté de ce protocole est qu'il échange des numéros ports de communication durant la phase d'initialisation de la connexion. Ces ports sont appelés ports dynamiques.

NetWall a besoin de préparer une liste de contrôles d'accès autorisant le passage de la connexion. Il est donc obligatoire de trouver et de stocker la valeur du port dynamique pour autoriser le passage des données.

#### *1.3.2 Définition d'une architecture d'ouverture pour NetWall*

L'objectif du travail est de définir une architecture d'ouverture pour NetWall, ou plutôt d'étendre l'architecture actuelle du pare-feu. Ce travail permet l'ajout de fonctionnalités comme l'audit, la détection d'intrusion et la mise en œuvre de contre mesures mais aussi de définir et d'implémenter une infrastructure d'intégration ouverte et extensible basée sur la technologie à agents pour des applications tierces telles que des solutions spécifiques de contrôle et de filtrage.

L'objectif est d'assurer ainsi l'évolutivité de NetWall en vue du support de nouveaux protocoles tels que l'audio et la vidéo sur Internet.

#### *1.3.3 Intégration des fonctions d'extension par agents*

Une plate-forme à agents a été réalisée par l'équipe AAA de Dyade et permet l'échange de messages de façon asynchrone entre applications. Un débouché industriel possible à l'utilisation des agents AAA est une application de traitement des logs. Les agents reçoivent de façon asynchrone les notifications d'alarmes des différents équipements du réseau et filtrent les informations.

Les machines à base d'agents sont capables d'analyser de nouveaux protocoles tels que l'audio et la vidéo sur Internet ou tout autre protocole. Elles peuvent réagir à des demandes d'ouverture de connexion et se charger de la détection des ports dynamiques.

Cette technologie à base d'agents intéresse l'industrie. Ainsi, un des objectifs du travail est de réaliser un transfert de technologie entre l'Inria et Bull. Ce transfert comprend l'intégration dans NetWall du développement effectué par l'action AAA.

**Présentation de la structure du document.**

Le chapitre II est une description de l'existant, présentée en deux étapes : le produit NetWall qui constitue le contexte applicatif de l'étude d'une part, et la plate-forme à agents AAA qui constitue le support technologique issu du laboratoire d'autre part.

Le chapitre III détaille le travail réalisé dans le DRT. Dans un premier temps les principes généraux de l'approche suivie sont présentés. Puis est décrite l'architecture de la solution retenue pour étendre les fonctions du logiciel NetWall. Ensuite sont exposés quelques éléments sur la mise en œuvre de cette architecture et sur les expérimentations qui ont été réalisées.

Le chapitre IV dresse un bilan du travail réalisé et propose quelques perspectives d'extensions pour le court terme.

**Chapitre 2 : DESCRIPTION DE L'EXISTANT**

## 2.1 NetWall

### 2.1.1 Description de NetWall

Par définition, NetWall, plus généralement un garde-barrière (passerelle de sécurité) ne peut protéger que le trafic qui le traverse. Cela signifie par exemple que :

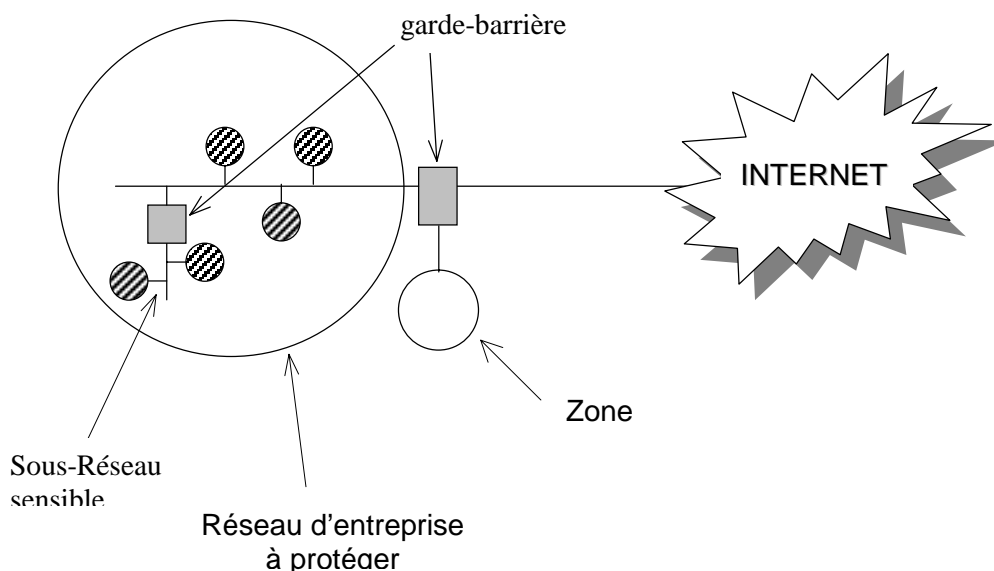
- NetWall ne peut pas protéger les accès entre deux machines situées sur un même réseau, si ces accès ne se font pas au travers de NetWall.

NetWall ne peut pas protéger un réseau vis à vis des autres réseaux, s'il existe des chemins d'accès ne passant pas par NetWall.

- NetWall ne peut protéger l'accès aux biens que par les services réseaux qu'il contrôle et dans la limite des contrôles effectués sur ces services. L'accès à ces serveurs, effectué par les services réseaux prévus et au travers du garde-barrière est garanti, s'il est réalisé dans le respect de la politique de sécurité définie par l'administrateur.

#### 2.1.1.1 Qu'est-ce qu'un garde-barrière ?

Un garde-barrière est un logiciel, qui installé sur une machine à l'entrée d'un réseau d'entreprise ou d'un sous-réseau local, permet d'assurer sa sécurité en le protégeant de toute attaque ou tentative d'intrusion. Il a pour rôle de filtrer les accès en provenance de l'extérieur. Il est là pour protéger les données sensibles et les services du réseau local.



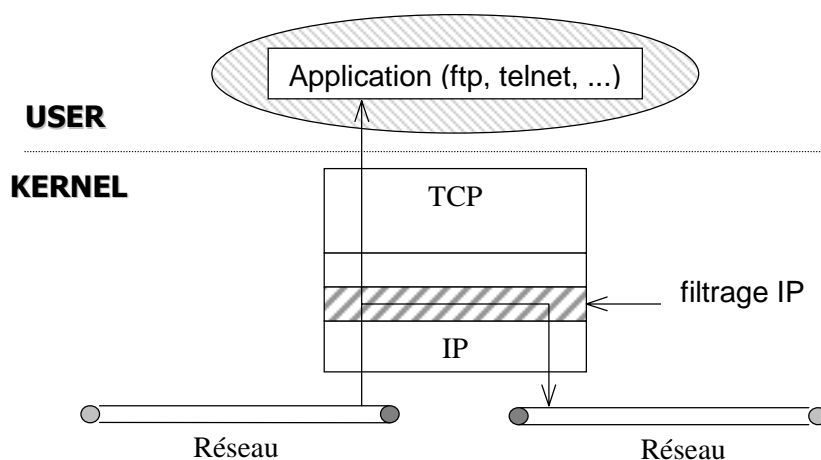
Il y a deux approches principales dans la réalisation du garde-barrière pour combattre les intrusions dans un réseau TCP/IP :

- Le filtrage IP
- Les passerelles applicatives (Applications proxies)

Le filtrage IP :

Chaque paquet essayant de traverser la couche IP est comparé avec une liste de contrôle d'accès. Cette liste de règles est utilisée pour déterminer si le paquet est autorisé à traverser ou non. Ces règles sont basées sur les adresses sources et destinations contenues dans les paquets et les interfaces réseaux sur lesquelles ils arrivent (internes, externes ou zone démilitarisée (DMZ)).

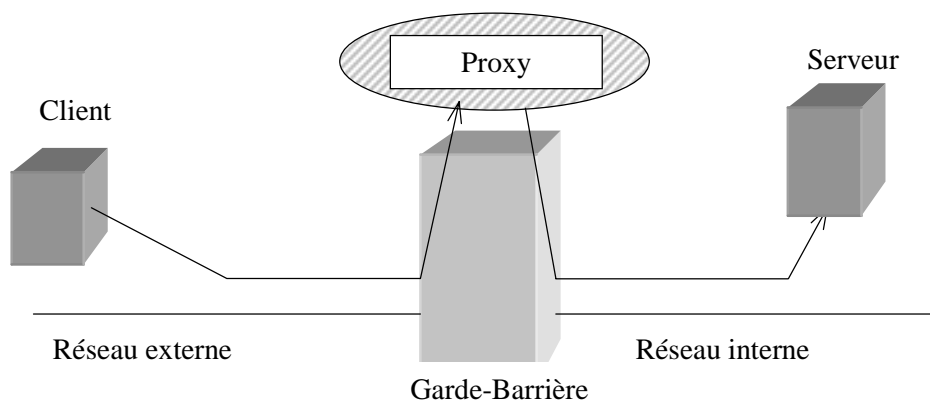
Le mécanisme de filtrage est implémenté dans le noyau (kernel).



Les passerelles applicatives (proxies) :

Un démon proxy est implémenté sur la passerelle pour chaque service TCP/IP supporté. Les proxies sont implémentés dans l'espace utilisateur.

Un utilisateur extérieur, désirant se connecter à un serveur interne pour un service donné (smtp, http, ftp...), se connecte d'abord au proxy auprès duquel, éventuellement, il s'authentifie, avant de se connecter au serveur destination. Tout le trafic doit passer au travers du proxy, qui procède aux vérifications et aux filtrages basés sur les commandes spécifiques du service.



NetWall combine ces deux approches : filtrage IP et passerelles applicatives. Il est constitué des composants logiciels suivants.

#### *2.1.1.2 Le composant de filtrage IP dynamique*

Ce composant se divise en 2 parties :

- un module de filtrage (driver), implanté dans le noyau des systèmes d'exploitation, est chargé du traitement des paquets en fonction des règles de filtrage. Ce driver est un driver chargeable, utilisant la notion d'extension noyau. Il est intégré dynamiquement au noyau du système d'exploitation, en dérivation des couches TCP/IP du système. Il permet d'intercepter les paquets IP avant leur entrée dans les couches TCP/IP.

- un processus démon, dans l'espace utilisateur du système d'exploitation, en liaison avec le driver ci-dessus, est chargé de transmettre à ce dernier les règles de filtrage telles que définies par l'administrateur, ainsi que de recevoir du driver les informations relatives à la sécurité pour les enregistrer.

#### *2.1.1.3 Un ensemble de relais applicatifs (proxy)*

L'ensemble de ces logiciels est implanté dans l'espace utilisateur du système d'exploitation. Ils reçoivent la connexion issue de la machine cliente et, après contrôle, ouvrent une nouvelle connexion à destination de la machine serveur.

#### *2.1.1.4 L'interface Graphique d'administration et de configuration*

C'est un composant situé "au-dessus" des autres composants du logiciel NetWall. Il contrôle le composant de filtrage IP dynamique et les relais applicatifs, pour ce qui concerne la configuration des règles de filtrage, le démarrage et l'arrêt, la configuration et l'exploitation des informations enregistrées relatives à la sécurité.

### *2.1.2 Résumé des fonctionnalités de NetWall*

NetWall est un logiciel garde-barrière (firewall), permettant une protection aussi bien face à Internet qu'à Intranet. Il permet de réaliser un cloisonnement des réseaux et de définir quelles machines peuvent accéder à quelles machines et pour quels services. Il peut traiter les services au-dessus des protocoles TCP et UDP, le protocole ICMP, et tout protocole au-dessus de IP. Il combine une technologie de filtrage IP dynamique de haut niveau, implémentant une gestion du contexte de l'état des connexions TCP et pseudo connexions UDP, et un ensemble de relais applicatifs, permettant une authentification des utilisateurs et des filtres internes aux applications. Une technologie de translation d'adresses, statique et/ou dynamique permet le masquage des adresses des réseaux protégés vis-à-vis de l'extérieur. NetWall offre également des fonctions d'audit et d'alertes complètes et ergonomiques. Il peut être utilisé en conjonction avec d'autres produits de sécurité de Bull, tels que

- SecurWare VPN pour réaliser des fonctions de chiffrement et de réseaux virtuels privés,
- l'authentification des utilisateurs distants par cartes à puce (CP8),
- ISM/AccessMaster pour une gestion centralisée des utilisateurs (authentification et privilèges) et la possibilité d'authentification unique ("Single Sign On").

Grâce à son infrastructure de relais transparents, NetWall permet à un utilisateur sur un poste client d'adresser directement le serveur final, sans connaître l'adresse du garde-barrière. L'étape classique de double identification/authentification, auprès du garde-barrière et auprès du serveur final, peut être effectuée en une seule fois avec NetWall. En effet, une syntaxe particulière, reconnue par les relais applicatifs, permet à l'utilisateur de saisir les données d'identification et d'authentification, en une seule fois. Ces données seront transmises au serveur final.

### 2.1.3 Limitation de NetWall

- Chargement dynamique de la politique de sécurité impossible

L'administrateur de NetWall définit une politique de sécurité. Il peut autoriser des connexions entre des machines, des groupes de machines, des réseaux, des sous réseaux... et il doit définir un service et une action pour ces différentes connexions. Le service se présente sous la forme d'un numéro de port. Une fois qu'il a défini toutes ses règles, qui sont de type

règle n	machine A	vers	Machine B	port XX	ACTION	Log
---------	-----------	------	-----------	---------	--------	-----

il peut démarrer NetWall. En fait, il démarre le démon « netwalld » qui se charge de transférer les règles au driver de NetWall. Le driver quand à lui est chargé à chaque boot. Quand le démon n'est pas en marche, il applique une règle par défaut. Cette règle est : soit je rejète tous les paquets IP, soit je les accepte tous.

Si l'administrateur décide de rajouter une règle pour autoriser ponctuellement le passage d'une connexion, il doit ajouter cette règle en utilisant l'interface graphique, puis arrêter le démon « netwalld » et le redémarrer. Nous voyons dans ce cas là que l'on va appliquer la politique de sécurité défini lorsque le démon ne marche pas. Il faut savoir que la durée du démarrage de NetWall est proportionnelle aux nombres de règles, plus il y a de règles plus le démarrage est lent, surtout sur les plates-formes AIX et Solaris.

Ceci nous montre une limitation de NetWall, nous ne pouvons pas ajouter des règles de filtrage de manière dynamique (en fonctionnement de NetWall), il faut obligatoirement passer par une phase d'arrêt de NetWall.

- Gestion des ports dynamiques

Lorsqu'un client se sert d'une application spécifique pour dialoguer entre plusieurs entités de sa société, il veut sécuriser ses connexions en installant un pare-feu. Il peut se retrouver bloqué car son application spécifique négocie des ports dynamiques comme le fait ftp, SQL\*Net, rpc, ... La seule façon qu'il a de traverser le pare-feu est d'autoriser le passage de tous les services, donc tous les ports entre ces deux machines client et serveur. Mais cette solution n'est pas très sécurisée. En fait nous nous rendons vite compte de la limitation qu'implique l'utilisation d'un pare-feu avec des applications dialoguant avec des ports dynamiques.

NetWall gère plusieurs services qui utilisent des ports dynamiques, mais il ne les gère pas tous. Nous voyons que les fabricants de pare-feu s'alignent en proposant des produits filtrants les services les plus couramment utilisés.

Pourtant dans le cas de certains protocoles, la sécurité est très dure à réaliser, car nous ne pouvons pas toujours extraire les numéros de port dynamique. Dans certaines applications, les données sont encodées et le décodage n'est pas toujours possible dans le driver de NetWall. Nous pouvons citer comme exemple netmeeting qui utilise le protocole H.323.

La solution à ce problème est d'accepter d'ouvrir son pare-feu, c'est à dire de laisser passer tous les ports, mais dans ce cas le pare-feu n'est pas très utile.

Une autre solution est de dérouter tout le trafic et de le faire passer à travers un proxy qui se chargera de résoudre les problèmes de port. Mais les connexions deviennent très lentes et peuvent ne pas fonctionner (ex : netmeeting).

Enfin nous observons un problème supplémentaire, à savoir que le déROUTement proxy n'est pas transparent, c'est à dire que la connexion se fait entre la machine cliente et NetWall puis NetWall et la machine serveur.

- Gestion des logs

Il existe un manque de clarté et de précision dans l'information se trouvant dans les fichiers de log de NetWall. En effet, nous prenons des traces au niveau du driver et au niveau des proxies et nous les stockons dans des fichiers de log. Or, il n'est pas évident de mettre en relation les logs provenant du driver et les logs provenant des proxies, car les informations contenues dans ces fichiers ne sont pas suffisamment corrélées. Ainsi, dans le driver nous n'avons pas d'information sur les utilisateurs et dans les proxies nous n'avons pas d'information sur les adresses IP (translatées ou réelles), les ports réels etc... Les informations de ce type sont récupérées par les proxies à l'aide des sockets donc après translation.

Exemple : dans les traces driver nous pouvons avoir A vers B' ouverture de connexion et tout de suite derrière A vers B' fermeture de connexion. Nous n'avons pas vu que l'utilisateur Durand n'a pas le droit de se connecter au serveur B'.

Alors que dans les traces provenant du proxy nous avons A' vers N (NetWall) utilisateur Durant rejeté.

Donc pour suivre une connexion il serait utile d'avoir accès à la même information. Au niveau du driver, il faut connaître toutes les informations des proxies, et au niveau des proxies il faut connaître toutes les informations sur la connexion vu par le driver. Dans ce cas, les traces, et donc les fichiers de log, seraient d'une très grande précision, ils seraient alors entièrement corrélés et donc très faciles à exploiter.

- Réactivité contre des attaques

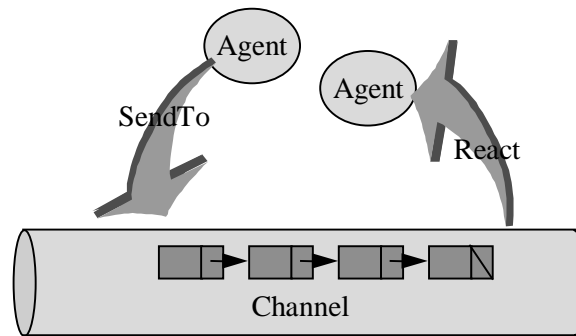
NetWall n'est pas capable de réagir aux attaques en modifiant de manière dynamique ses actions de protections. Pour palier à cette limitation, des agents spécifiques de Dyade seraient implantés et ils seraient capables de traiter et d'analyser, les logs et les alertes remontées par le démon de NetWall. Ces agents pourraient centraliser les informations provenant de plusieurs NetWall, il serait par conséquent possible de détecter des attaques qui passeraient inaperçues sur chaque NetWall. Dans l'état actuel, pour enrayer une attaque, il faut modifier ou ajouter des règles en utilisant l'interface graphique de NetWall, puis arrêter le démon et le redémarrer. De plus, il faut déployer cette modification sur tous les NetWall, alors que les agents pourraient créer des règles de rejet ou de redirection vers des machines leurre, et ce, en fonctionnement de NetWall.

Une autre idée serait d'observer et de loguer, de façon ponctuelle et beaucoup plus précise, le trafic d'une règle donnée. Ces traces seraient prises ponctuellement pour ne pas saturer les log. A l'heure actuelle, il n'est pas possible de changer une action associée à une règle, sans modifier les règles définies dans l'interface graphique. De même nous avons vu qu'il faut arrêter et redémarrer le démon pour que le changement soit pris en compte.

## **2.2 AAA "Agent Anytime Anywhere "**

### *2.2.1 Introduction*

Les agents sont des objets "réactifs" qui se comportent conformément au modèle "événement → réaction" : un événement est une transition d'état significative à laquelle un ou plusieurs agents vont réagir.



### Modèle de base

Dans notre modèle, un événement est représenté par une notification ; une notification est un objet passif qui est signalé à l'agent destinataire afin qu'il exécute la réaction appropriée. L'envoi de notifications représente le seul moyen de communication entre agents ; les notifications transitent entre agents via un bus de message.

## 2.2.2 Architecture globale

### 2.2.2.1 Principes de réalisation

Un agent est un objet passif, instance d'une classe qui hérite de la classe *Agent* ; la classe *Agent* définit le comportement commun à tous les agents. Le comportement réactif est mis en œuvre au moyen de la méthode *react* qui définit le comportement de l'agent lors de la réception d'une notification ; cette méthode est appelée par le moteur d'exécution. Chaque classe d'agents est, en particulier, caractérisée par l'ensemble des réactions qu'elle traite dans la méthode *react*. Les agents sont des objets persistants : les données qui composent leur état ont une durée de vie indépendante de toute exécution. L'architecture distribuée est composée d'un ensemble de serveurs d'agents répartis sur différents sites ; chaque agent est attaché à un moteur d'exécution, les seules entités mobiles sont les notifications. Chaque agent est désigné par un identificateur unique qui permet sa localisation.

Les notifications sont des objets passifs organisés comme une hiérarchie de classes dont la racine est la classe notification.

L'interface du bus de message est réalisée par la classe *Channel*, cette classe définit la méthode *sendTo* qui permet l'envoi d'une notification à un agent désigné par son identificateur. Les notifications sont alors acheminées vers une file de messages où elles sont stockées dans l'ordre chronologique ; le bus de message a la responsabilité de la localisation de l'agent destinataire.

### 2.2.2.2 Caractéristiques du bus de message

Dans un premier temps l'envoi d'une notification est une opération "point à point", l'émetteur précisant le ou les destinataires. Le bus de message assure les propriétés suivantes :

- toute notification acceptée sera délivrée une fois et une seule à chaque destinataire ;
- l'ordre de réception des notifications entre 2 agents est conforme à l'ordre causal d'émission :
  - si un agent A émet successivement les notifications n1 et n2, alors aucun agent ne recevra n2 avant n1 ;
  - si un agent A1 émet une notification n1, et qu'un agent A2 après avoir reçu n1 émet une notification n2, alors aucun agent ne recevra n2 avant n1.

L'exécution des réactions est réalisée au moyen d'un moniteur transactionnel : soit l'ensemble des actions d'une réaction (Constitué des notifications envoyées et des changements de l'état de l'agent) est validé, soit l'ensemble de ces actions est défait.

La réalisation, l'extension et la construction des agents sont détaillées en annexe.

### *2.2.3 Utilisation pour enrichir NetWall*

Nous pouvons enrichir NetWall en nous appuyant sur un développement fait par l'action AAA. En utilisant la technologie à base d'agents et les bibliothèques fournies par NetWall, nous sommes capables de répondre aux difficultés qu'a NetWall de filtrer un protocole comme H.323. AAA pourrait concevoir une bibliothèque (en java) s'appuyant sur les bibliothèques de NetWall et ainsi fournir des outils qui faciliteraient le développement d'un intégrateur pour l'ajout de protocole dynamique.

L'outil de configuration à base d'agents permet à un administrateur non initié, de créer son application de gestion de log. Il peut transférer ses connaissances et son expérience dans le domaine de l'analyse et de l'exploitation des logs, en utilisant une interface qui est fournie par AAA. Il peut ainsi directement extraire les informations qui lui semblent pertinentes. Il n'a plus besoin de faire un pré traitement des fichiers de log de NetWall avant l'exploitation des résultats.

**Chapitre 3 : DESCRIPTION DU TRAVAIL REALISE**

### 3.1 Principes généraux de l'approche

#### 3.1.1 Introduction

Dès mon arrivée dans le projet NetWall, il a fallu me familiariser avec le driver et regarder l'architecture de NetWall. J'ai commencé par lire le code réalisé pour le filtrage du service ftp. Ceci m'a permis de voir la difficulté d'une telle implémentation.

La meilleure façon d'appréhender NetWall était de coder le filtre pour SQL\*Net. J'ai donc commencé l'analyse de ce service qui communique en utilisant des ports dynamiques.

Lors du développement, j'ai pu constater la difficulté d'ajouter un filtre où les connexions sont assez complexes.

D'autres complications peuvent résulter de la définition des services que l'on trouve dans des RFC qui peuvent être assez ouvertes, c'est à dire que l'interprétation qui peut en être faite peut différer suivant les personnes. De plus, les versions des services peuvent évoluer et rendre l'utilisation des filtres de NetWall incompatible avec ces nouvelles versions, ceci engendre un coût de développement pour se réaligner, nous l'avons vu pour le service des sun-RPC.

Pour ouvrir l'architecture de NetWall et répondre aux limitations, par exemple, à l'impossibilité de filtrer dans le driver certains services comme H.323, il a été nécessaire de développer une librairie permettant un dialogue (interrogation sur des connexions, actions de création et de destruction de cache, ...) entre l'espace utilisateur et le driver.

De plus une extension de NetWall a été indispensable pour permettre une prise de log cohérente entre le driver et les proxies. Cette extension est fournie sous forme de librairie. Elle permet entre autre, de gérer les utilisateurs dans le driver.

#### 3.1.2 SQL\*Net

L'ajout du filtre pour SQL\*Net ne demande pas de modification de l'architecture de NetWall. L'approche a été relativement simple puisqu'il suffisait de s'inspirer du développement fait pour le filtre de ftp. En effet l'analyse du protocole SQL\*Net a mis en évidence un échange de port dynamique entre le client et le serveur. Le paquet contenant ce port dynamique est facilement repérable car il contient le mot "PORT", c'est le même mot que pour ftp.

La description complète du travail réalisé pour l'ajout du filtre de SQL\*Net est détaillée dans un chapitre suivant.

#### 3.1.3 L'open framework

##### 3.1.3.1 Fonctionnement dynamique de NetWall

Nous avons vu dans les chapitres précédents que le fonctionnement dynamique de NetWall n'était pas possible sans une modification de son architecture. Or, nous voulons obtenir des informations sur certaines connexions, et agir sur les règles de filtrage en autorisant ou rejetant les connexions. Cependant, une forte interaction avec NetWall est nécessaire car toutes ces actions doivent être faites pendant son fonctionnement. Ainsi, il n'est pas possible d'arrêter NetWall pour autoriser le passage de certains ports dynamiques en ajoutant des règles leur autorisant le passage. Car les numéros de ports dynamiques, qui peuvent être lus dans les fichiers de log produits par NetWall, sont en effet dynamiques et peuvent changer à chaque connexion. Le fait d'arrêter NetWall coupe les connexions.

Dans le domaine de la gestion des logs, il est intéressant de pouvoir modifier la précision des logs pour certaines connexions qui attirent notre attention, et ce bien sûr sans arrêter le filtrage.

L'approche qui a été choisie est de rajouter un point d'entrée dans le driver de NetWall, ainsi nous pouvons lui passer des commandes sans arrêter le filtrage. Ces actions peuvent être exécutées par des services, démons ou tout autre programme s'appuyant sur la librairie d'open framework de NetWall.

### 3.1.3.2 Manipulation d'entrées caches en fonctionnement

Le cache de NetWall est constitué de listes d'entrées caches. Des décisions très rapides peuvent être prises grâce à ces entrées caches. Car lorsqu'elles ont été créées pour une connexion, nous pouvons appliquer à tous les paquets les actions qui ont été décidées lors de l'ouverture de cette connexion.

Pour répondre aux limitations de NetWall, il est nécessaire de disposer d'outils permettant la manipulation d'entrées caches. Nous avons besoin des opérations suivantes : création, destruction, recherche et prise d'informations sur une connexion.

La destruction d'entrées caches est limitée au seul créateur de celles-ci. Ainsi, un accès est autorisé en lecture seule, sur toutes les entrées caches du driver. Cela permet d'éviter d'éventuelle maladresse au cours d'une destruction d'entrée cache. Par conséquent, nous sommes sûr que l'extension de NetWall, grâce aux manipulations d'entrées caches, ne perturbera pas le bon fonctionnement du driver.

Pour la réalisation, nous avons opté pour une paire d'identificateurs, l'un provenant du driver et l'autre provenant de l'application s'exécutant dans l'espace utilisateur.

Pour le traitement et les échanges entre les démons, services, etc... et le driver, nous devons créer une nouvelle structure "VCACHE" qui est indépendante de la structure "entrée cache". Cette structure "VCACHE", nous permet de fournir des informations très complètes sur une connexion. Nous fournissons plus d'information que ce que l'on peut obtenir d'une socket. En effet, par exemple les adresses IP dans les différents domaines du NetWall, ne sont connues que du driver. Grâce à la librairie de l'open framework ces informations peuvent être utilisées par les programmes s'exécutant dans l'espace utilisateur. Il faut savoir que ces informations sont indispensables pour la réouverture des connexions.

La librairie "ofw" permet d'occulter les différentes créations d'entrées caches, effectivement elles peuvent être de type statique ou dynamique suivant la présence d'un joker à la place d'un numéro de port.

### 3.1.3.3 Déroutement d'une partie d'une connexion

En s'appuyant sur les propriétés de NetWall, le fonctionnement dynamique de NetWall et la manipulation d'entrées caches, il devient possible de dérouter une partie des connexions de façon transparente. C'est à dire que le serveur et le client ne connaissent pas la présence de NetWall. Nous pouvons dérouter le flux dit de contrôle et laisser passer les données. Nous avons donc tous les éléments pour filtrer des nouveaux protocoles dynamiques comme H.323. Les taux de transfert sont alors optimisés, et les difficultés dues au protocole TCP/IP sont alors écartées car dans l'espace utilisateur nous sommes au dessus de la couche TCP.

## 3.1.4 Liens entre l'open framework et H.323

### 3.1.4.1 Plugin H.323 de AAA

Le développement du plugin H.323 est fait par l'équipe de AAA. Le développement est réalisé en java de même que l'architecture des agents sur laquelle ce plugin est basé. Pour fonctionner, le plugin s'appuie sur une librairie dynamique (*lib\_ofw*) fournie par NetWall. Cette librairie procure tous les outils de manipulation des entrées caches.

L'idée générale du déploiement et du déclenchement du plugin est la suivante.

Pour le déploiement, le plugin doit être mis en route à chaque démarrage de NetWall de manière automatique. Il doit être démarré comme un proxy ou service de NetWall.

Pour le déclenchement, nous ajoutons une règle de filtrage grâce au GUI de NetWall. Nous verrons par la suite que le déclenchement de cette règle de filtrage crée une entrée cache. Le plugin a besoin de s'approprier cette entrée cache et de la référencer comme étant parent de toutes les autres demandes d'ouverture qui découleraient de cette connexion. Comme le service H.323 est démarré, il se trouve en attente sur un numéro de port. Dès que les paquets arrivent dans NetWall, ils sont alors transmis au plugin. Le plugin peut alors commencer son analyse protocolaire et déclencher la création d'entrées caches qui permettront le passage direct du son et de la vidéo.

#### 3.1.4.2 Interface native des agents

La difficulté du lien entre la librairie open framework et le plugin H.323 est due aux deux langages utilisés. D'une part le langage C pour la librairie "OFW" et d'autre part le langage Java pour le plugin H.323. NetWall étant présent sur trois plates-formes à savoir AIX, Windows NT et Solaris, il y a donc un code natif sur chaque plate-forme car la librairie NetWall est programmée en C. Par contre, le code java des agents est le même sur les trois plates-formes. Il est donc nécessaire d'avoir une interface entre la librairie "OFW" et le plugin H.323. Cette interface se présente sous la forme d'une librairie java. Elle permet de faire abstraction des plates-formes utilisées. Cette librairie peut être utilisée par des intégrateurs pour simplifier le développement et l'ajout de nouveau filtre. De plus, la programmation étant la même sur les trois plates-formes le nombre de lignes de code est divisé par trois.

#### 3.1.5 SDK

L'évolution de NetWall dans sa version 5.0 a rendu possible la connaissance des utilisateurs dans le driver. Ainsi, nous avons dû créer des nouvelles structures d'échanges car la librairie "ofw" ne pouvait pas transférer ces informations.

Nous avons créé une librairie "sdk" en nous appuyant sur des spécificités qui n'étaient pas présentes lors du développement de la librairie "ofw".

De plus, la librairie "sdk" devait répondre à des besoins très précis de la part de tous les proxies de NetWall. C'est pourquoi nous n'avons pas adapté les structures de l'open framework car le réaligement avec les agents de AAA aurait été plus long. D'autre par, l'ajout des fonctions de la librairie "sdk" s'appuie sur le travail fait pour l'open framework. Les échanges de structures avec le driver sont très simples à mettre en place car nous nous appuyons sur l'architecture ouverte de l'open framework.

La librairie "sdk", doit nous donner la possibilité de retrouver l'information d'une connexion par un balayage des listes de contrôles d'accès. Mais aussi, par un balayage des listes d'entrées caches et enfin de donner les valeurs des adresses IP de NetWall ainsi que ses interfaces d'entrée ou de sortie.

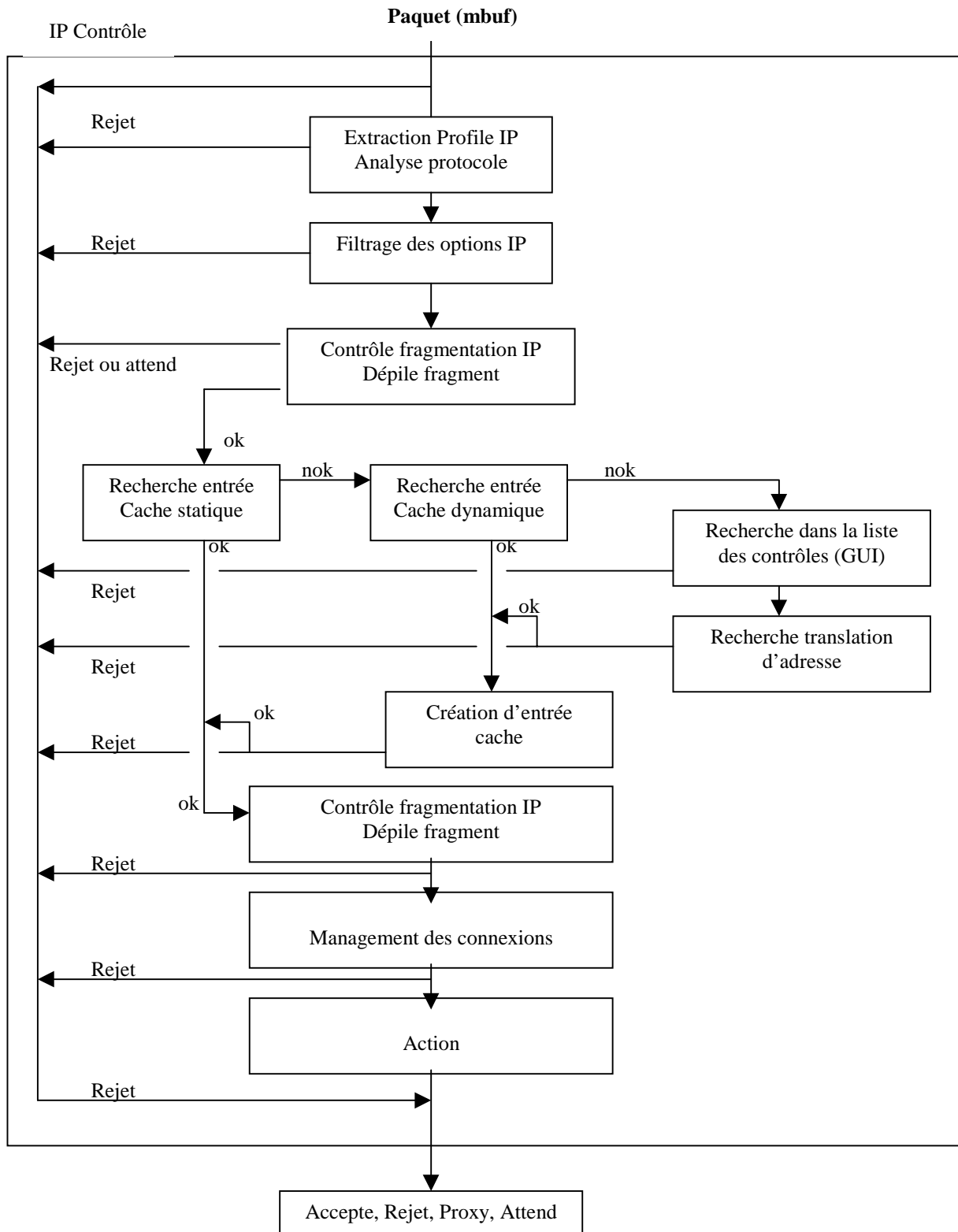
Bien entendu, les listes de contrôles d'accès contiennent dorénavant les identifiants des utilisateurs.

Les fonctions fournies par la librairie "sdk" vont nous permettre d'avoir des traces fortement corrélées entre le driver et les proxies. Ainsi, nous aurons des fichiers de logs cohérents et plus faciles à exploiter. Elles vont aussi nous permettre de prendre des décisions sur l'ouverture de connexion.

## 3.2 Architecture

### 3.2.1 NetWall

#### 3.2.1.1 Chemin d'un paquet TCP/IP dans la fonction de contrôle de NetWall



### 3.2.1.2 *Le cache de NetWall*

Le cache de NetWall est constitué de listes d'entrées caches. Des décisions très rapides peuvent être prises grâce à ces entrées caches. Car lorsqu'elles ont été créées pour une connexion, nous pouvons appliquer à tous les paquets les actions qui ont été décidées lors de l'ouverture de cette connexion. A savoir, si on accepte le passage du paquet TCP/IP, si on le rejète ou si on l'envoie à la couche TCP pour qu'il ne soit pas routé par la couche IP (déroutement proxy). En fait, il y a beaucoup d'actions qui peuvent être faites sur un paquet, par exemple la translation d'adresse, les log, etc... le fait de stoker ces actions nous évite un recalcul pour chaque paquet ce qui ralentirait considérablement le trafic.

Dans NetWall, il y a deux grandes classes d'entrées caches. La première est la classe des entrées caches statiques. Ce sont des structures où tous les champs, comme l'adresse IP de la source, le port source, l'adresse IP de la destination, le port de la destination, le protocole etc... sont connus et renseignés. Une fois quelles ont été créées plus rien n'est modifié dans la structure principale. Elles sont alors insérées, en utilisant des méthodes de hachage, dans des listes de petite taille pour optimiser la recherche.

La deuxième classe est celle des entrées caches dynamiques. Ce sont des structures identiques aux structures des entrées caches statiques, mais certains champs ne sont pas connus donc non renseignés. Il peut s'agir des champs ports de la source ou ports de la destination qui prennent alors la valeur zéro. Il se peut aussi qu'un des champs adresse ait la valeur 127.0.0.1 (local host), ceci est dû à des problèmes de spoofing dans le déROUTement proxy. Les entrées caches dynamiques sont des entrées caches statiques contenant un joker. Elles sont rangées dans une même liste, et lorsque tous les champs sont connus, elles sont détruites et recrées dans les listes d'entrées caches statiques.

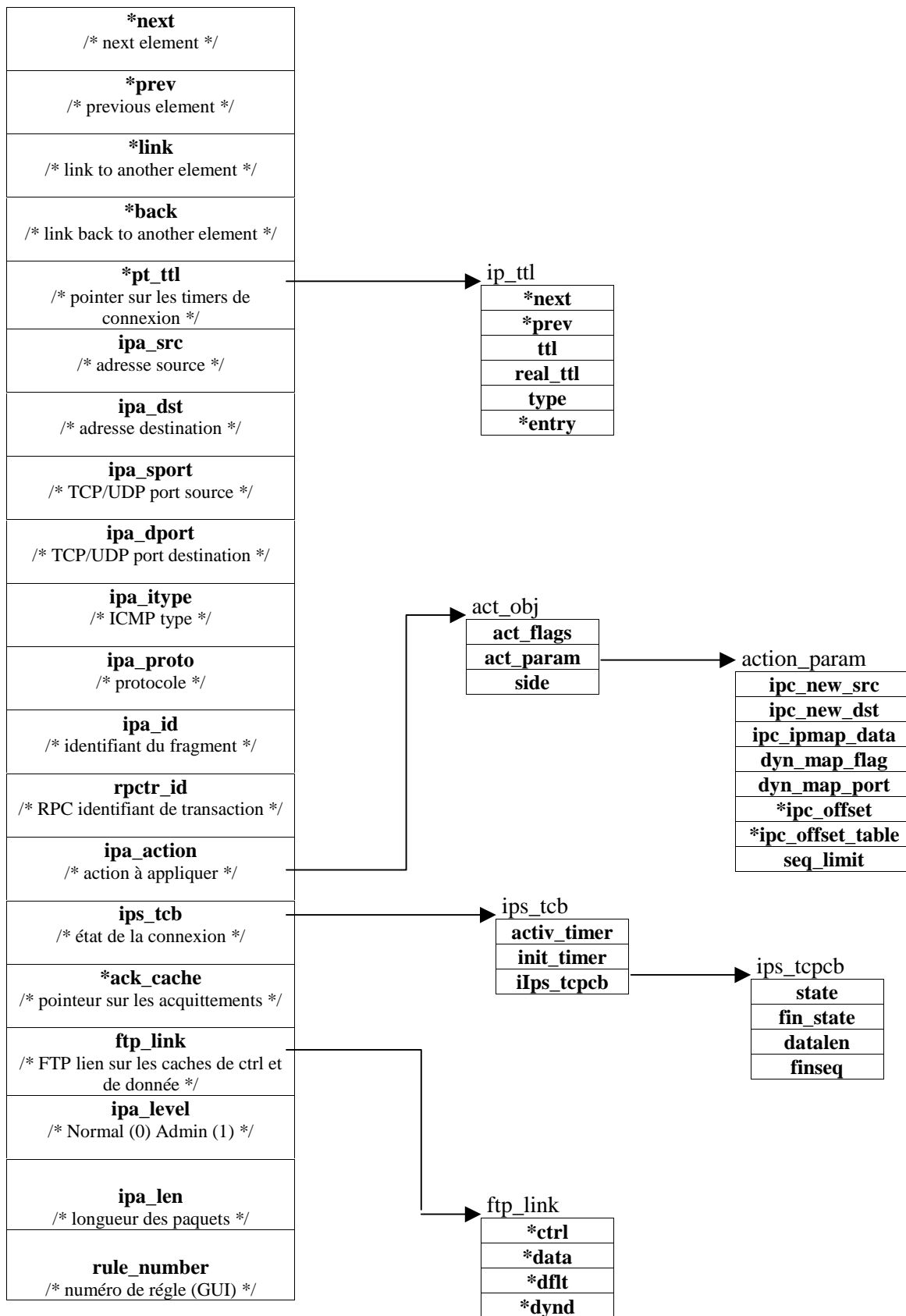
L'utilité des entrées caches dynamiques est la suivante. Par exemple, dans le cas des services dialoguant au travers de port dynamique, nous extrayons les numéros de ports des données contenu dans les paquets, et nous créons une entrée cache dynamique avec bien sûr un joker. Le client va contacter le serveur en utilisant le port dynamique (port de la destination) que le serveur lui a donné. Sur la machine NetWall, nous n'avons aucun moyen de connaître le port source que le client va utiliser, et pourtant, grâce à l'entrée cache dynamique, NetWall laisse passer la connexion. Nous obtiendrons la valeur du port source lors de l'envoi du premier paquet, et nous pourrons alors créer l'entrée cache statique.

Un élément important pour NetWall est qu'il est capable de faire une recherche très rapide dans son cache, pour garder des taux de transfert optimums. Ainsi, nous retrouvons l'entrée cache correspondant à une connexion, en utilisant une fonction de hachage. Il y en a plusieurs dans NetWall, cela dépend du protocole ou de la fragmentation (TCP, UDP, ICMP, Fragment et autre). Ces fonctions de hachage nous positionnent en début de liste, il ne nous reste plus qu'à parcourir cette liste pour trouver l'entrée cache. Cette opération est très rapide car les listes correspondantes à une entrée possible sont très petites.

Pour ne pas mélanger les entrées caches, il est nécessaire d'identifier de manière unique une connexion. Donc nous nous servons des éléments suivants : l'adresse source, l'adresse destination, dans le cas d'une connexion TCP ou UDP du port source et du port destination, dans le cas d'une connexion ICMP du type et dans les autres cas, du protocole ou de l'id du fragment.

Voici le détail de la structure d'une entrée cache, nous voyons toutes les informations qui sont stockées et qui n'auront pas besoin d'être recalculées.

Structure d'une entrée cache (ip\_cache).



<b>rsh_dtg</b>
<b>ipc_chstate</b>
<b>ipa_src_side</b>
<b>rpc_persistent</b>

### 3.2.1.3 *Le déroulement proxy*

Lorsque l'action proxy est positionnée, tous les paquets d'une connexion sont déroulés. C'est à dire, une connexion entre un client A et un serveur B est en fait déroulée par NetWall (tout le trafic passe par l'espace utilisateur), nous avons donc une connexion de A vers N (NetWall) puis N vers B. Cette action est nécessaire pour filtrer par exemple certaines commandes de FTP comme le put, le get, ... ou pour faire un filtrage particulier de certains services, par exemple pour rediriger une connexion mail (pop) vers un anti-virus.

Le déroulement proxy ralentit considérablement le trafic entre deux machines. Ceci est normal car les paquets traversent plusieurs couches pour passer du noyau du système d'exploitation à l'espace utilisateur.

## 3.2.2 *Open framework*

### 3.2.2.1 *Insertion dans NetWall*

Nous devons établir un dialogue entre le driver de NetWall qui se trouve dans le noyau du système d'exploitation et l'espace utilisateur. Nous utilisons des ioctl pour ces dialogues.

Avant de faire passer des structures de données entre l'espace utilisateur et le noyau, il faut ouvrir un file descriptor (fd). Les files descriptor sont des points d'entrée dans le driver. Ces points d'entrée sont positionnés au boot de la machine lors du chargement du driver.

Jusqu'à présent il y avait sept points d'entrée dans le driver de NetWall.

/dev/netwall_adm	=> Administration
/dev/netwall_amt	=> Translation d'adresse
/dev/netwall_ipc	=> Chargement et déchargement driver
/dev/netwall_acl	=> Administration des listes de control d'accès
/dev/netwall_alert	=> Administration des alertes
/dev/netwall_auth	=> Administration de l'authentification
/dev/netwall_log	=> Administration des logs

Quand le démon de NetWall "*netwalld*" démarre, il ouvre ces sept fd. Ceci a pour effet au niveau du driver, de positionner un état ouvert pour ces points d'entrée. Donc aucune autre application ou démons de l'espace utilisateur ne pourront ouvrir de nouveau ces sept points d'entrées car pour le driver l'état ouvert est déjà utilisé (par le démon de NetWall). C'est un verrou qui est positionné dans le driver.

Pour permettre à plusieurs programmes d'accéder au driver via un point d'entrée particulier, il a fallu ajouter une méthode d'ouverture de file descriptor qui ne verrouille pas l'accès à d'autres applications ou démons. Ce point d'entrée particulier est appelé entrée clone (/dev/netwall).

Nous avons besoin de cette entrée clone car l'architecture d'ouverture de NetWall ne se limite pas à une seule application ou démon s'exécutant dans l'espace utilisateur et souhaitant accéder au driver.

Décrivons maintenant la liste des fonctions qui peuvent être exécutées en utilisant l'entrée clone.

Ces fonctions sont codées dans la librairie open framework. Elles répondent au besoin de manipulation d'entrées caches et de recherche d'information sur les connexions.

- `create_cache( union cache_info *cache )`

Cette fonction est appelée pour créer une entrée cache statique ou dynamique, cela dépend de la présence ou non de joker dans la structure *Virtual\_Cache* qui est le paramètre d'entrée de cette fonction. La structure *Virtual\_Cache* fait partie de l'union *cache\_info* ainsi que la structure *CacheID* qui est la structure retournée après la création de l'entrée cache.

- `remove_cache( struct CacheID *id )`

Cette fonction est appelée pour détruire une entrée cache. S'il s'agit de l'entrée cache parent alors elle sera détruite ainsi que tous ses enfants, sinon elle est la seule détruite. Cette fonction est appelée avec la structure *CacheID* qui nous permet de trouver l'entrée cache de manière instantanée car l'adresse de cette dernière se trouve dans la structure.

- `get_cache( union cache_info *cache )`

Cette fonction est appelée pour recueillir une structure *ip\_cache* telle quelle est dans le driver. Le paramètre d'entrée est une structure *CacheID*.

- `get_virtual_cache( union cache_info *cache )`

Cette fonction est appelée pour recueillir une structure *Virtual\_Cache* telle quelle est dans le driver. Le paramètre d'entrée est une structure *CacheID*.

Ces deux dernières fonctions sont le moyen le plus simple de trouver toutes les informations sur une connexion et sur le traitement qui est effectué par NetWall.

- `get_mapping_real_fict( struct addr_real_fict *rf )`

Cette fonction est appelée pour connaître l'adresse IP d'une machine par rapport au domaine de NetWall. Car une adresse IP peut être différente suivant le côté qui est regardé. Il faut donc donner une adresse IP et un côté ("side") ainsi la fonction nous donne l'adresse IP connue de ce côté.

- `get_mapping( struct Profile *prof )`

Cette fonction est très proche de la fonction `get_mapping_real_fict`. Nous passons juste deux adresses IP au lieu d'une.

- `set_cacheid( union cache_info *cache )`

Cette fonction est appelée pour s'approprier une connexion. Elle est utilisée pour s'attribuer l'entrée cache créée par le GUI. Elle a en entrée une structure *Virtual\_Cache* qui ne contient que les informations suivantes : adresse IP de la source et de la destination, le port source, le port destination et le protocole. Elle a en sortie une structure *Virtual\_Cache* complètement remplie.

- `get_addr_intf_info( struct InetAddr *addr_side )`

Cette fonction est appelée pour connaître le côté ("side") d'une adresse IP. Nous lui passons en entrée une adresse IP et en sortie nous récupérons le côté ("side") correspondant.

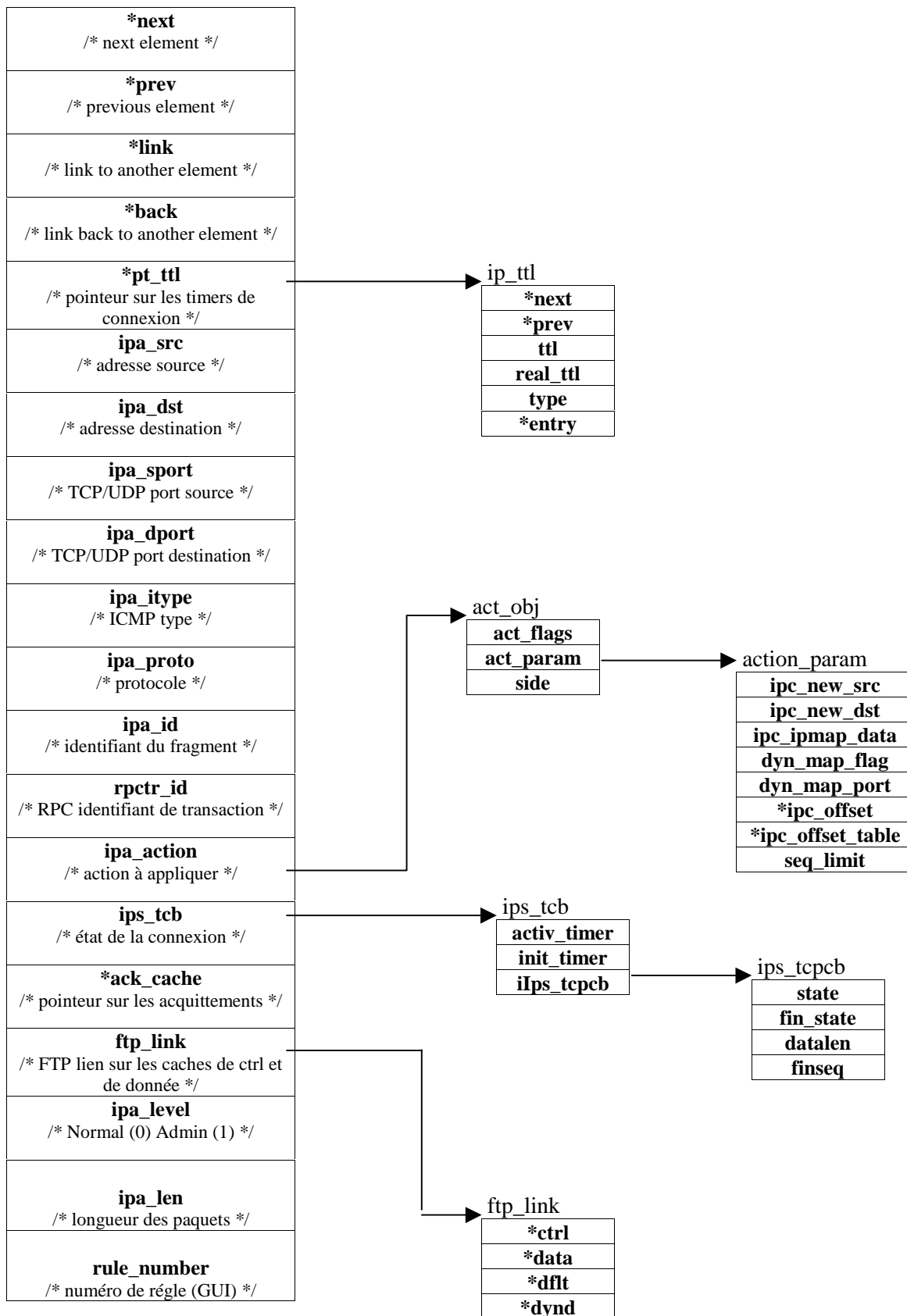
- `isLocalAddr(ulong addr)`

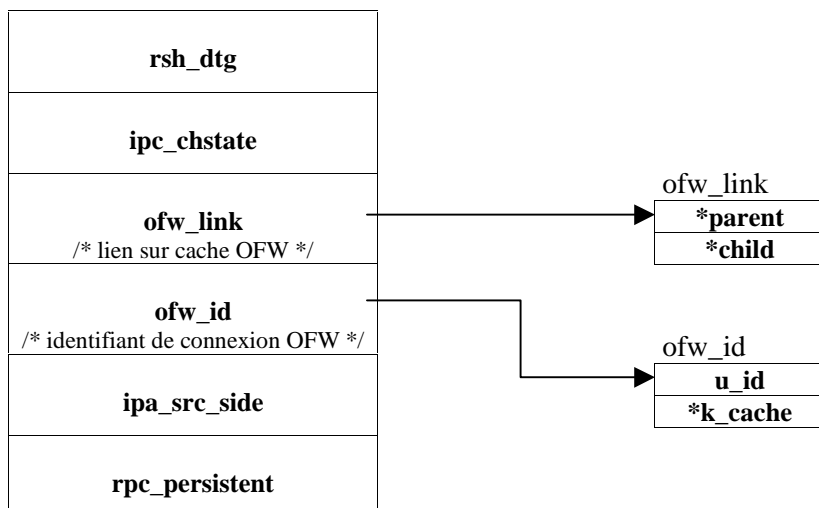
Cette fonction est appelée pour savoir si une adresse IP est une adresse locale de NetWall.

La librairie open framework utilise l'entrée clone `"/dev/netwall"`. Dans le driver, cette entrée clone (`IPSECU_CLONE_MINOR`) déclenche la fonction `ofw_ioctl(commande, argument, &size)`. Elle permet alors d'exécuter une des fonctions open framework définies ci-dessus suivant la commande qui est envoyé comme paramètre d'appel.

Par conséquent, l'ajout de fonctions pour échanger des informations entre le driver et l'espace utilisateur devient très simple. Toute la difficulté du point d'entrée clone est résolue.

## 3.2.2.2 Modification de la structure de cache





Pour l'open framework, nous raisonnons en terme de connexion et surtout de parent et d'enfant. Une connexion TCP (parent) peut très bien engendrer des connexions TCP et UDP (enfant). C'est le cas pour des dialogues basés sur le service H.323 et T120 de netmeeting. Nous pouvons ainsi accéder à une entrée cache de façon instantanée. Ceci est utile pour la recherche et surtout pour la destruction d'entrée cache lors d'une fin de connexion, il suffit de donner le parent de la connexion et nous détruisons tous les enfants de cette connexion sans avoir à faire de recherche.

Pour cette raison, nous avons ajouté dans la structure du cache de NetWall, deux champs.

Le premier est une structure `ofw_link` qui permet de lier entre elles les entrées caches créées pour une connexion. Elles sont créées par un démon, un proxy ou un programme s'exécutant dans l'espace utilisateur. Cette structure contient un pointeur sur l'entrée cache parent, ce qui permet un accès direct à l'entrée cache parent. L'autre élément de la structure est un pointeur sur l'entrée cache enfant. Ceci est très utile lors de la fermeture de connexion ou de la destruction du parent car il entraîne la destruction de tous les enfants.

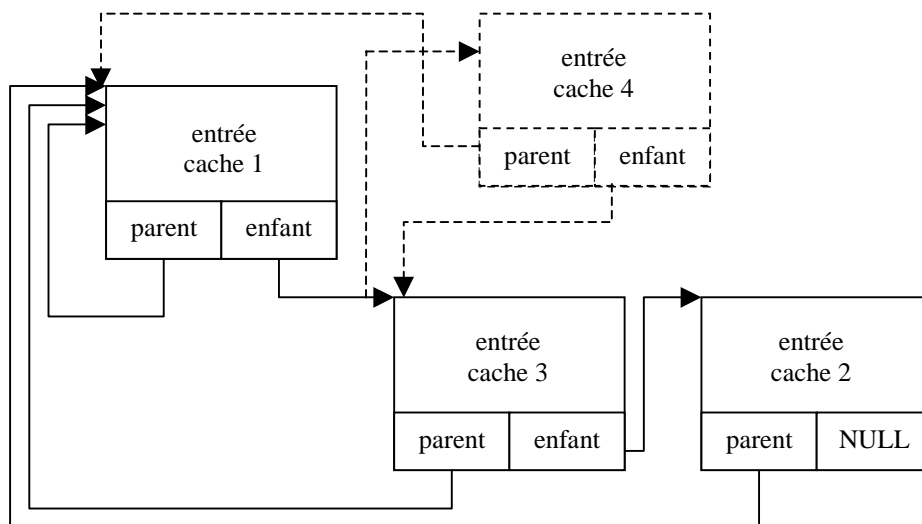
Le deuxième champ, est une structure `ofw_id`. Cette structure permet l'identification unique d'une entrée cache open framework. Elle contient un identifiant provenant de l'application s'exécutant dans l'espace utilisateur et un identifiant unique provenant du driver qui n'est autre que l'adresse de cette entrée cache.

### 3.2.2.3 Lien entre les entrées caches pour l'open framework

Dans le driver de NetWall les entrées caches sont liées entre elles grâce aux quatre premiers éléments de la structure `ip_cache`. Ce lien dépend du protocole utilisé : les entrées caches TCP (protocole 6) sont liées entre elles, les entrées caches UDP (protocole 17) sont liées entre elles etc. ; mais les liens ne sont pas croisés (TCP et UDP ou autre).

Dans le cadre de l'open framework, lorsque nous créons une entrée cache, elle est soit parent, si elle n'a pas de parent ce qui est très rare car elle découle normalement d'une règle qui provient du GUI de NetWall, soit enfant.

Toute nouvelle entrée cache enfant qui est créée, est insérée en tête de la liste enfant (structure `ofw_link`) et son pointeur parent pointe sur l'entrée cache parent.



### 3.2.3 H.323

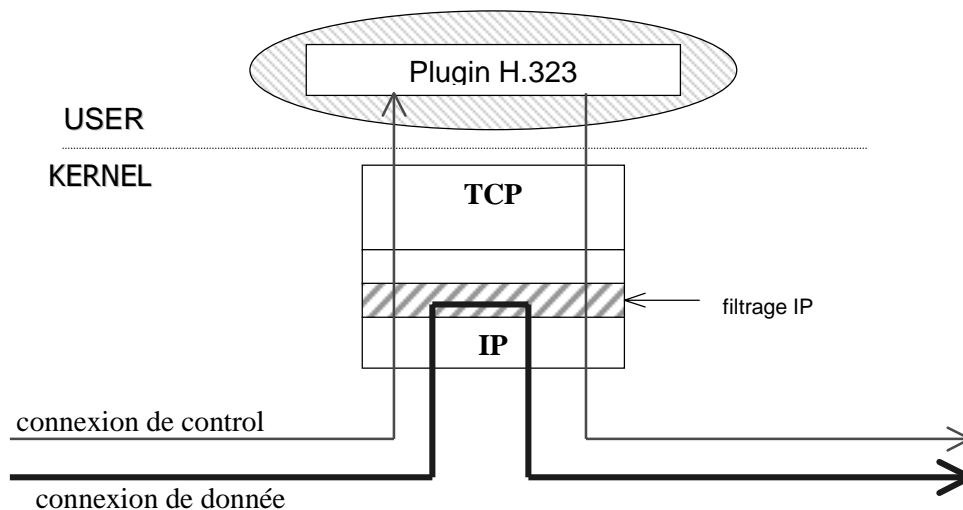
Dans une conférence H.323, les participants s'échangent des messages dont les formats rentrent dans l'une des trois catégories suivantes :

- Des messages constitués d'une longueur variable d'octets suivis d'une structure ASN.1  
Par exemple, les messages Q.931 servant à l'établissement de l'appel.
- Des messages constitués d'une structure ASN.1.  
Par exemple, les messages H.245 pour la négociation des paramètres.
- Des messages constitués d'une suite d'octets de longueur variable.  
Par exemple, les paquets RTP et RTCP, audio et vidéo.

Les messages ASN.1 sont encodés en conformité avec les PER (*Packed Encoding Rules*). Il existe deux variantes d'encodage PER, à savoir l'encodage aligné et l'encodage non aligné. H.323 utilise la variante alignée.

Tous les messages ASN.1 de H.323 contiennent des champs optionnels. En conséquence, il est impossible d'extraire une information pour l'examiner ou la modifier à un endroit précis de la structure encodée ; Il est nécessaire de décoder une partie ou la totalité du message. Ceci est indispensable pour pouvoir extraire les numéros de port échangés au cours d'une connexion.

La solution choisie et implémentée consiste à dérouter la connexion de contrôle et laisser passer les connexions de données, comme le montre le schéma ci-dessous, en utilisant les fonctions de manipulation des entrées caches décrites dans les paragraphes précédents.



### 3.2.4 Librairie SDK

L'extension de NetWall via la librairie "*sdk*" nous permet des échanges de structures entre les proxies et le driver. Pour le développement de cette librairie nous nous sommes appuyés sur l'architecture de l'open framework.

Nous utilisons donc l'entrée clone de l'open framework pour accéder au driver.

Les fonctions suivantes sont codées dans la librairie "*sdk*". Elles sont indispensables pour le fonctionnement des proxies sur NetWall 5.0.

- `get_rule_obj(struct scan_obj *scan, struct rule_obj *rule, object_id *tab)`

Cette fonction est appelée pour parcourir les listes de contrôles d'accès dans le driver. Elle prend en entrée une structure "*scan\_obj*", contenant tous les éléments nécessaires à la recherche du meilleur élément des listes de contrôles d'accès. Cette fonction prend aussi en entrée un tableau d'utilisateurs, s'il y a des utilisateurs déclarés. Ainsi, nous utilisons la même méthode que le driver pour chercher un élément dans les listes de contrôles d'accès. Cette recherche est faite à l'aide d'un "flag" qui détermine ce que l'on doit regarder ou ignorer. C'est pourquoi, s'il y a des utilisateurs déclarés, le "flag" sera positionné pour faire une recherche des utilisateurs en plus de la recherche habituelle.

Quand la recherche est terminée, nous avons le meilleur élément trouvé, et nous appliquons, au niveau des proxies, l'action retournée pour cet élément. Cette action est stockée dans la structure "*rule\_obj*" (retournée par la fonction `get_rule_obj`), qui contient aussi toutes les informations sur cette connexion. Nous aurons notamment les domaines d'entrées et de sorties de NetWall, les adresses IP de la machine source et de la machine destination dans chacun de ces domaines ainsi que leurs identifiants, le port source et le port destination, la période de validité, l'identifiant du service (protocole), l'identifiant unique de la règle et l'identifiant unique de la liste du contrôle d'accès du driver.

- `get_itf_obj(struct itf_obj *itf)`

Cette fonction est appelée pour trouver les interfaces ("*sides*") d'entrées et de sorties de NetWall, ainsi que les adresses IP de ces interfaces et leurs identifiants associés.

- `get_cnx_obj(int sock, struct cnx_obj *cnx_obj)`

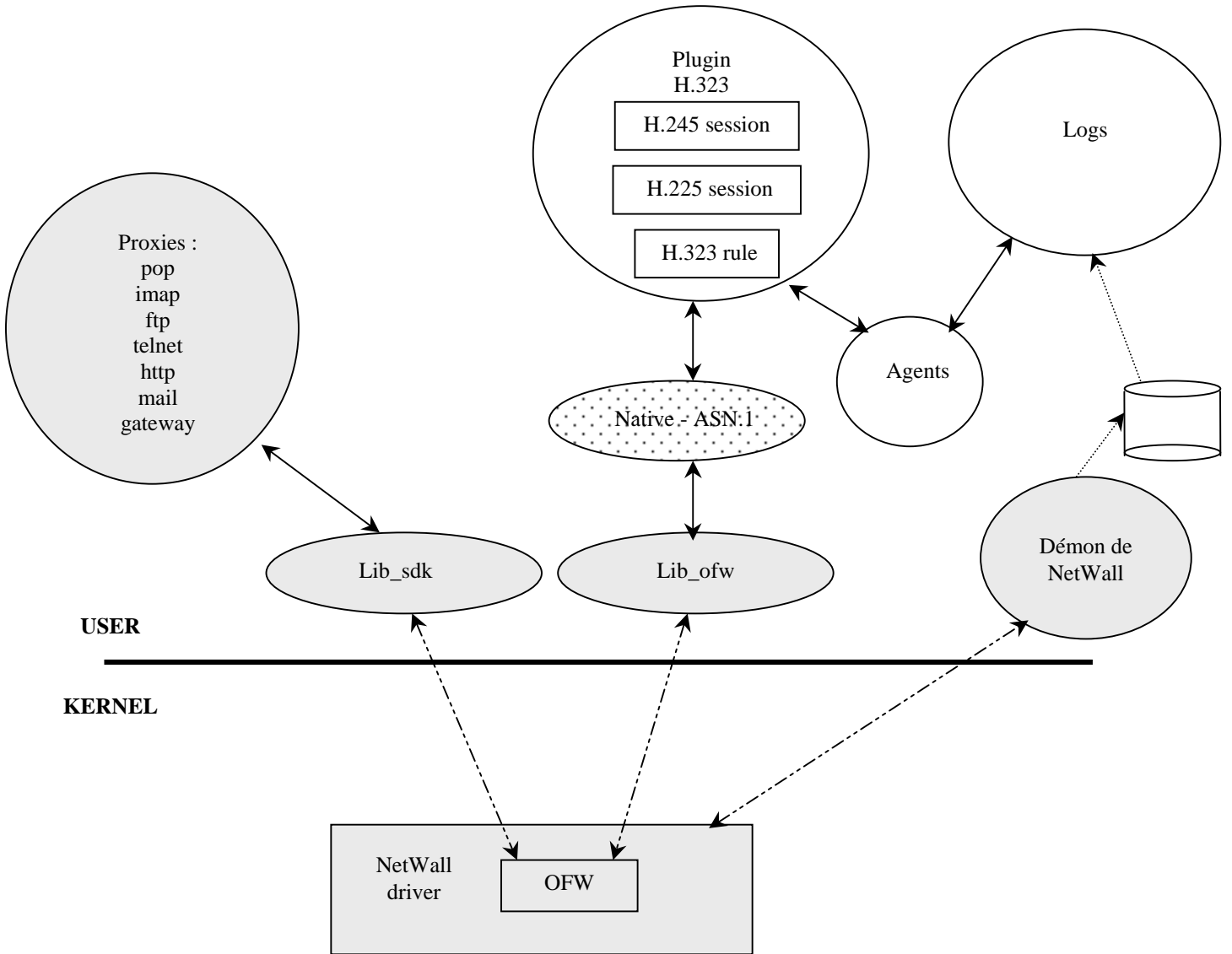
Cette fonction est appelée pour trouver l'entrée cache correspondant à la socket. Nous lui donnons en entrée une socket, et elle nous donne en sortie une structure "*cnx\_obj*" qui contient de nombreuses informations sur cette connexion.

Ces informations sont indispensables aux proxies. Grâce à elles, ils peuvent ouvrir une connexion auprès du serveur final.

Les éléments suivants sont présents dans la structure "*cnx\_obj*" : les domaines d'entrées et de sorties de NetWall, les adresses IP de la machine source et de la machine destination dans chacun de ces domaines ainsi que leurs identifiants, le port source et le port destination, l'identifiant du service, l'identifiant unique de la règle, le protocole et l'action (accepter ou rejeter).

Comme nous le voyons, dans la version 5.0 de NetWall, nous travaillons beaucoup avec des identifiants. Ce sont des longs non signés, ceci évite de descendre dans le driver des chaînes de caractères qui seraient beaucoup plus délicates à manipuler. Nous faisons la correspondance entre les identifiants et les noms lorsque nous prenons en compte les logs dans l'espace utilisateur.

3.2.5 Schéma de l'architecture OFW, SDK et H.323



..... fichier de log cnx.ulm

- - - - - ioctl

○ Langage C

● Langage C et Java

○ Langage Java

### 3.2.6 *Le multi plates-formes*

#### 3.2.6.1 *Le déROUTement proxy*

Le déROUTement proxy dans NetWall, est un flag dans la structure action (*act\_obj*). Lorsque le flag "IP\_CONTROL\_PROXY" est positionné, cela a pour conséquence de court-circuiter la couche IP (couche de routage) et de transmettre directement le paquet à la couche TCP comme si le paquet était à l'intention du NetWall et non du destinataire.

Le problème est que nous travaillons sur plusieurs plates-formes, AIX, Solaris et Windows NT. L'implémentation du déROUTement proxy est différent suivant les plates-formes.

Pour une version AIX de NetWall, nous avons une fonction qui envoie le paquet directement à la couche TCP. Il n'y a donc aucun problème de routage du paquet car nous court-circuitons la couche IP. L'adresse IP de destination peut être n'importe laquelle de toute façon le paquet sera pour NetWall.

Pour une version Solaris ou Windows NT de NetWall, nous n'avons pas l'équivalent de la fonction d'AIX qui nous permet l'envoi direct à la couche TCP.

Donc sur ces deux plates-formes, il nous faut faire une translation de l'adresse destination en une adresse locale de NetWall et une transformation du port source en un port que l'on alloue dynamiquement et ce pour assurer l'unicité de la connexion TCP/IP. Une fois ces translations faites, nous envoyons ce paquet à la couche IP qui va le router naturellement à la couche TCP locale du NetWall. Nous obtenons par cette méthode un déROUTement proxy. Mais cela engendre quelques inconvénients. A savoir un dysfonctionnement de la translation d'adresse.

En cas de translation statique d'adresse et de déROUTement proxy, les connexions ne franchissent pas NetWall. Nous perdons la valeur traduite de l'adresse IP de destination, uniquement sur les plates-formes Solaris et Windows NT.

Ce problème est résolu dans la version 5.0 de NetWall.

#### 3.2.6.2 *Le Routage*

Une difficulté majeure pour les versions Solaris et Windows NT de NetWall, est qu'il n'y a pas de fonction système pouvant s'exécuter dans le driver et permettant le calcul des routes. Ainsi, nous avons besoin de connaître l'interface ou la carte de sortie d'un paquet pour une adresse destination donnée. Cela nous permet de préparer les entrées caches et de positionner les valeurs des interfaces de sortie pour le spoofing.

Nous avons trouvé un moyen de contourner ce problème. Nous formons un paquet test et nous l'envoyons à la couche IP. Cette dernière va calculer le routage et il ne nous reste plus qu'à récupérer les informations. Cette méthode fonctionne très bien quand on a un paquet. Si nous n'avons pas de paquet, nous sommes incapable d'en formater un pour récupérer les informations de routage. Or dans la cas de la librairie open framework, nous créons des entrées caches par anticipation, nous n'avons pas de paquet "sous la main" pour calculer le routage.

Ce problème est résolu dans la version 5.0 de NetWall.

### 3.3 Réalisation

#### 3.3.1 Protocoles dynamiques

##### 3.3.1.1 Analyse et implémentation de SQL\*Net

Cette partie est la description complète du travail qui a été fait dans NetWall pour permettre le filtrage de SQL\*Net.

##### 1) Oracle et les réseaux (SQL\*Net)

SQL\*Net est un nom générique désignant en fait l'ensemble des couches nécessaires à la communication entre clients et serveur oracle.

SQL\*Net assure la transparence du réseau c'est-à-dire qu'une base de données peut être accédée indifféremment quelle que soit sa localisation physique (locale ou distante) cette abstraction réseau implique la transparence vis à vis :

- des systèmes d'exploitation et des architectures matérielles
- des protocoles réseaux utilisés
- des médias et de la topologie du réseau, assurées par le "Transport Network Substrate "(TNS)
- de la localisation physique des bases, celles-ci seront donc désignées par des noms ou adresses IP, un numéro de port et le SID désignant l'instance de la base sur ce serveur.

Nous ne parlerons que du protocole TCP/IP.

On connaît l'intérêt de désigner les ressources réseau par des noms indépendants de leur adresse physique (convivialité, déplacement transparent de services,...). Chaque base de données, chaque serveur, chaque listener (démon en charge de la réception des requêtes des clients) aura un nom unique.

On utilise un fichier de configuration TNSNAMES.ORA (équivalent à /etc/host) sur chaque poste client ou serveur.

##### a) SQL\*Net connexion

SQL\*Net établit une connexion à une base de données quand le client ou une autre base de données serveur envoie une demande de connexion à une base distante. Cette demande est faite de manière automatique ou manuelle.

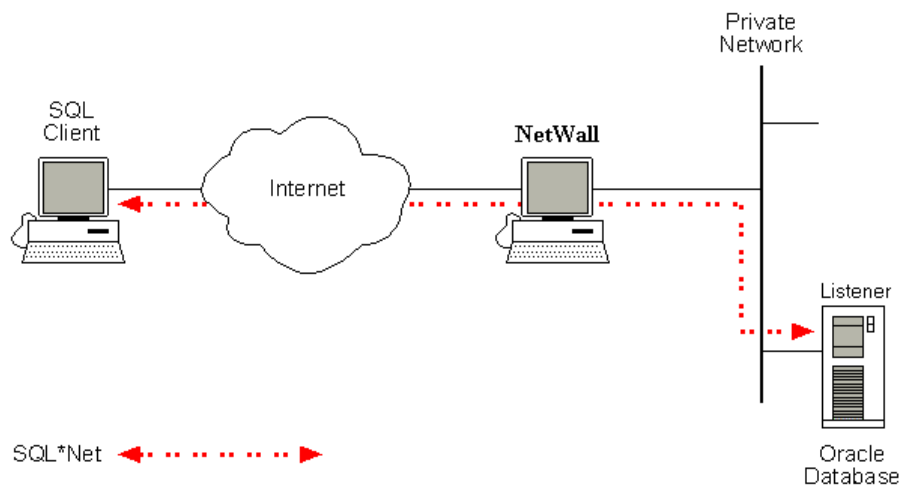
Il y a trois composants dans toutes les connexions SQL\*Net :

- Client : Logiciel qui amorce la connexion, qu'il soit client ou serveur oracle7.
- TNS Listener : Démon qui écoute sur un port fixe dans le cas de TCP/IP. Dans notre cas, c'est le port 1521.
- Serveur : Logiciel qui sert le client, il s'agit d'un serveur oracle7 dédié ou d'un serveur oracle7 multi-thread (avec un dispatcheur), nous détaillerons plus loin la différence entre ces deux serveurs.

Classiquement :

Le client appelle le serveur par l'intermédiaire du listener qui écoute sur le port 1521. Le client envoie une requête au listener qui contient l'adresse cible (serveur), le port (1521) et l'identifiant de la base de données (SID).

Le listener accepte la demande de connexion et la passe au serveur approprié.



Description de la requête SQL\*Net (tnsname.ora) :

```
(DESCRIPTION=
  (ADDRESS=
    (COMMUNITY= community_name)
    (PROTOCOL=tcp)
    (HOST=database_server)
    (PORT=1521)
  )
  (CONNECT_DATA=
    (SID=database_identifieur)
  )
)
```

Description de la requête réponse (si port dynamique) :

```
(ADDRESS= (PROTOCOL=tcp) (DEV=15) (HOST=129.183.155.24) (PORT=4257))
```

### b) Serveur dédié

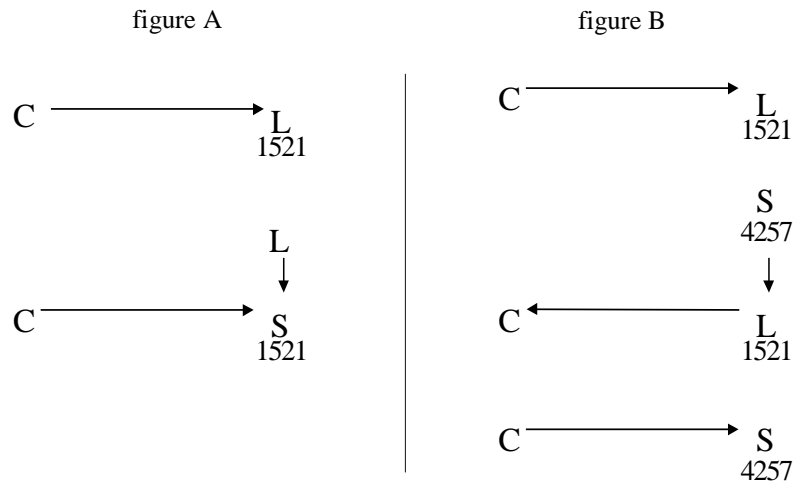
Pour utiliser un serveur dédié avec SQL\*Net, il faut que le démon oracle (Listener) soit démarré sur le serveur. De plus SQL\*Net permet de filtrer l'accès à des bases de données en regardant les autorisations définies dans le fichier *protocol.ora*. Ce fichier contient une liste d'adresses IP et la règle associée. Ce contrôle est indépendant de NetWall.

Les étapes de la connexion :

- Le client initialise la connexion en contactant le LISTENER sur le port 1521 comme décrit ci-dessus.
- Le listener reçoit la demande et détermine si le client a le droit de se connecter en vérifiant les droits dans le fichier *protocol.ora*. Si le listener refuse la connexion, il renvoie une erreur au client et continue l'écoute en vue d'une nouvelle demande de connexion.
- Si la connexion est acceptée, le listener engendre un processus serveur.
  - Il lègue la connexion du client au processus serveur et ils se partagent efficacement le port 1521 (figure A). Le plus souvent, le listener lègue la connexion au lieu de la rediriger vers un autre port.
  - lorsque le listener engendre un processus serveur (figure B), celui-ci contacte le système qui lui donne un numéro de port disponible (comme le fait le port-mapper). Il transmet l'information au client et ferme la connexion. Le client ouvrira une nouvelle connexion sur ce nouveau port.
- Le listener continue à écouter les connexions entrantes.

schéma :

C : client  
L : listener  
S : serveur



### c) Serveur multi-thread

Un serveur multi-thread supporte plusieurs clients par processus serveur. Ceci permet de réduire la mémoire utilisée. De plus, un nombre plus important d'utilisateurs peuvent se connecter à la base de données.

C'est le listener qui sert de relais pour une connexion à un dispatcher, donc le client contacte le serveur en passant par le listener qui lui renvoie automatiquement un numéro de port dynamique (celui du dispatcher).

Pour pouvoir faire des accès à une base qui se trouve sur un serveur multi-thread, il est impératif de démarrer le listener, le serveur multi-thread et le dispatcher.

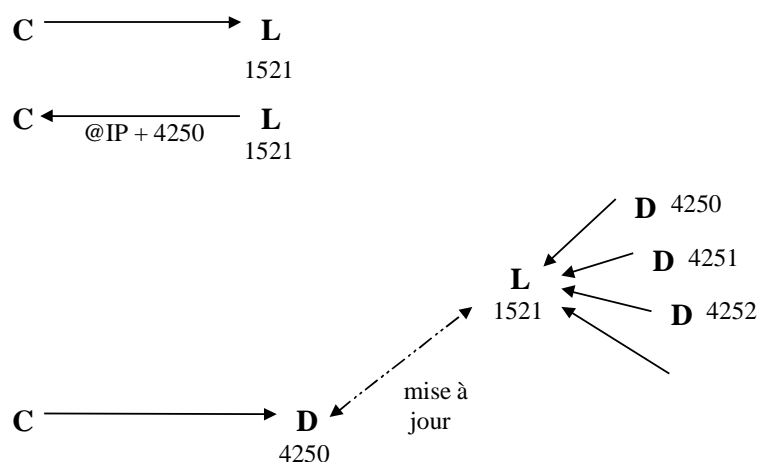
SQL\*Net permet de filtrer l'accès à des bases de données en regardant les autorisations définies dans le fichier *protocol.ora*. Ce fichier contient une liste d'adresses IP et la règle associée. Ce contrôle est indépendant de NetWall.

Les étapes de la connexion :

- Le client initialise la connexion en contactant le LISTENER sur le port 1521 comme décrit ci-dessus.
- Le Listener reçoit la demande et détermine si le client a le droit de se connecter en vérifiant les droits dans le fichier *protocol.ora*. Si le listener refuse la connexion, il renvoie une erreur au client et continue l'écoute en vue d'une nouvelle demande de connexion.
- Le listener renvoi au client une requête qui contient l'adresse IP du dispatcher et le numéro de port sur lequel il doit le contacter.
- Le client perd la connexion avec le listener et ouvre une nouvelle connexion avec le dispatcher en utilisant l'adresse et le numéro de port contenu dans la requête réponse du listener.
- Le listener et le dispatcher crée un lien pour se mettre à jour sur les nouvelles connexions. Le listener peut envoyer une nouvelle connexion à un dispatcher qui tourne.
- Le listener continu à écouter les connexions entrantes.

schéma :

C : client  
L : listener  
S : serveur  
D : dispatcher



## 2) Information nécessaire pour filtrer SQL\*Net avec NetWall

L'administrateur définit ses règles de sécurité, il autorise un client A à faire des connexions vers un serveur B en utilisant le service SQL\*Net. Au niveau du GUI de NetWall, il va ajouter une règle A vers B sur le port 1521 et préciser qu'il accepte cette connexion.

La difficulté pour NetWall, réside dans le fait qu'il faut autoriser le passage des informations de la machine client A vers la machine serveur B pour un trafic SQL\*Net, et ce pour suivre la politique de sécurité définie par l'administrateur. Or ce trafic s'initialise forcément avec le quintuplé adresse source (machine client A), port source, adresse destination (machine serveur B), port destination (port=1521) et protocole TCP (numéro=6), mais le serveur peut décider de ne plus dialoguer sur le port 1521 mais sur un autre port qu'il détermine de façon dynamique. Il y a donc dans les connexions SQL\*Net, un échange de numéro de port au cours de la connexion.

L'administrateur ne peut pas connaître le numéro de port dynamique échangé au cours d'une connexion. Pourtant il veut autoriser la connexion du client A vers le serveur B pour le service SQL\*Net.

NetWall doit donc extraire le numéro de port dynamique, pour créer une règle autorisant le client A à se connecter au serveur B conformément à la politique de sécurité établie par l'administrateur. Sans cette nouvelle règle la connexion entre les deux machines est impossible.

En résumé :

- Le client initialise la connexion sur le port 1521.

- Le serveur renvoie au client, dans certains cas (décrit précédemment), l'adresse et le port sur lequel il doit le contacter. Ces informations sont contenues dans les data d'un paquet TCP/IP.

ex :

```
.J . . . . . @ ( ADDRE
SS = ( PROTOCOL = tcp
) ( DEV = 15 ) ( HOST = 1
29 . 183 . 155 . 24 ) ( P
ORT = 4257 ) )
```

Il faut regarder le contenu des paquets réponse pour extraire le numéro de port dynamique (dans l'exemple c'est le port 4257). Cette information est indispensable pour créer une règle dynamique qui permettra d'autoriser le passage des paquets suivants sur le port 4257.

Dans la suite, nous expliquons le traitement appliqué aux paquets si l'administrateur de NetWall a associé une translation d'adresse statique ou dynamique, à la source ou à la destination des paquets SQL\*Net.

### a) Translation statique d'adresse

Nous pouvons traduire statiquement l'adresse IP des clients ou des serveurs (voir Fonctions couvertes par NetWall / filtrage dynamique au niveau IP).

La translation statique d'adresse, consiste à masquer l'adresse réelle d'une machine interne à toutes les machines (externes) qui se trouvent de l'autre côté de NetWall. La seule contrainte est de préciser le chemin (routage) sur les machines externes pour quelles puissent atteindre la machine dont l'adresse a été traduite.

ex : 129.183.155.24 est traduite en 222.222.222.22

### i. Translation statique de l'adresse du serveur

La difficulté pour le garde barrière est de modifier les en-têtes IP des datagrammes et l'adresse du serveur qui sont contenues dans la réponse du serveur.

Pour l'en-tête IP il faut modifier les adresses qui ont été traduites ou les adresses qui ont besoin d'être traduites et recalculer le checksum IP de ce datagramme. Il faut faire cette opération pour tous les paquets échangés entre le client et le serveur.

Dans le cas où il y a une modification à faire dans les données :

Exemple :

Données émises par le serveur (sans l'en-tête IP et TCP)

```
====( 128 bytes transmitted on interface en0 )====
16:30:04.592962799
ETHERNET packet : [ 08:00:3e:30:33:93 -> 00:00:c0:35:3f:00 ] type
800 (IP)
IP header breakdown:
  < SRC = 129.183.155.24 > (rioga.frec.bull.fr)
  < DST = 129.183.155.15 > (onyx.frec.bull.fr)
  ip_v=4, ip_hl=20, ip_tos=0, ip_len=114, ip_id=32388, ip_off=0
  ip_ttl=60, ip_sum=c66b, ip_p = 6 (TCP)
TCP header breakdown:
  <source port=1521, destination port=2661 >
  th_seq=cb5e8510, th_ack=55eebaa
  th_off=5, flags<PUSH | ACK>
  th_win=16060, th_sum=fdc7, th_urp=0
Data :
  .J.....@(ADDRE
  SS=(PROTOCOL=tcp
  )(DEV=15)(HOST=1
  29.183.155.24)(P
  ORT=4257))
```

mais le client doit recevoir les données suivantes :

```
IP header breakdown:
  < SRC = 222.222.222.22 >
  < DST = 129.183.155.15 > (onyx.frec.bull.fr)
  ip_v=4, ip_hl=20, ip_tos=0, ip_len=114, ip_id=32388, ip_off=0
  ip_ttl=60, ip_sum=xxxx, ip_p = 6 (TCP)
TCP header breakdown:
  <source port=1521, destination port=2661 >
  th_seq=cb5e8510, th_ack=55eebaa
  th_off=5, flags<PUSH | ACK>
  th_win=16060, th_sum=xxxx, th_urp=0
Data :
  .J.....@(ADDRE
  SS=(PROTOCOL=tcp
  )(DEV=15)(HOST=2
  22.222.222.22)(P
  ORT=4257))
```

Problématique posée dans NetWall :

- si la longueur de l'adresse traduite est égale à la longueur de l'adresse réelle :

L'adresse réelle du serveur dans les données, doit être remplacée par l'adresse traduite. Mais il faut aussi recalculer le checksum TCP car les données ont changé, puis mettre à jour les adresses contenues dans l'en-tête IP et pour finir recalculer le checksum IP.

- si la longueur de l'adresse traduite est différente de la longueur de l'adresse réelle :

Il faut changer certaines informations au niveau IP et TCP comme décrit dans le cas précédent. Mais il faut aussi recalculer les valeurs donnant la longueur des données calculées pour et par SQL\*Net (ici J =74 et @ =64). Dans l'exemple, la longueur totale du paquet est de 114 (c'est la

longueur de l'en-tête IP + la longueur de l'en-tête TCP + la longueur des données), mais il faut retrancher 40 (longueur de l'en-tête IP + longueur de l'en-tête TCP) ce qui nous donne une longueur des données qui vaut 74.

Cette modification de longueur des paquets engendre des problèmes supplémentaires, pour la cohérence des numéros de séquences et des numéros d'acquittements. Si ces numéros de séquences ne correspondent pas à ce que la couche TCP attend, les paquets seront rejetés par cette dernière. Il faut donc gérer des listes de numéros de séquence pour rester cohérent par rapport aux deux couches TCP (client et serveur) qui dialoguent entre elles. Ces listes seront indispensables pour assurer le dialogue entre le client et le serveur, et ce pour tous les paquets qui seront émis après cette modification de donnée.

Attention il faut modifier le fichier *tnsname.ora* sur le client pour qu'il puisse se connecter au serveur dont l'adresse est traduite.

#### *ii. Translation statique de l'adresse du client*

Les données des paquets émis par le client ne contiennent pas d'information à modifier. Donc, il y aura juste une modification des en-têtes IP au passage du NetWall. Il faut remplacer les adresses traduites par les adresses réelles ou vis et versa, puis recalculer le checksum IP.

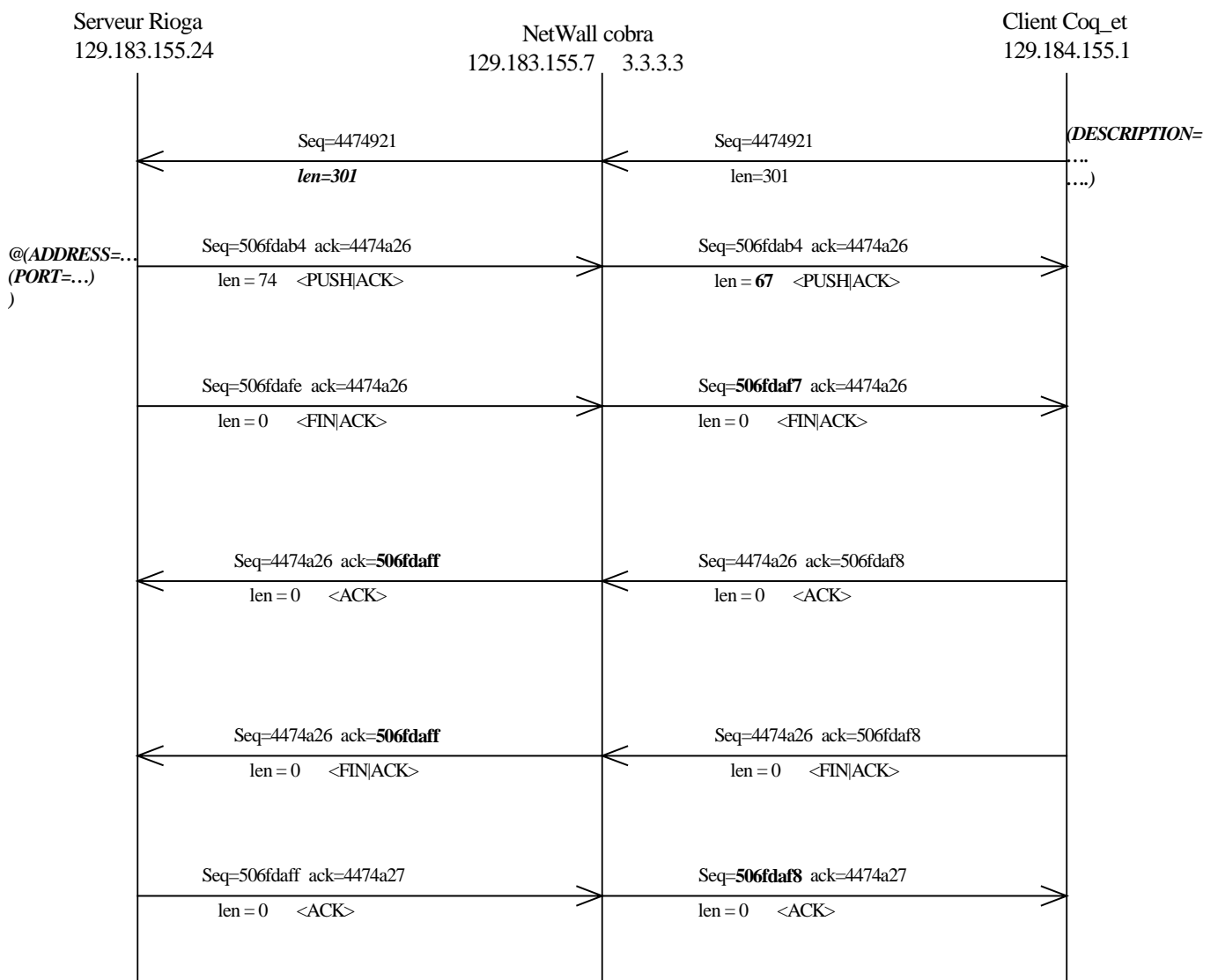
iii. Exemple de connexion avec translation statique d'adresse (serveur)

Connexion entre le client "coq\_et" et le serveur "rioga", on passe à travers NetWall qui est sur "cobra".

On fait une translation statique de l'adresse de "rioga", son adresse réelle est 129.183.155.24, son adresse virtuelle est 3.3.3.3

Cet exemple illustre les modifications des numéros de séquences (décrit ci-dessus).

En gras, les valeurs à modifier au passage de NetWall.



### *b) Translation dynamique d'adresse*

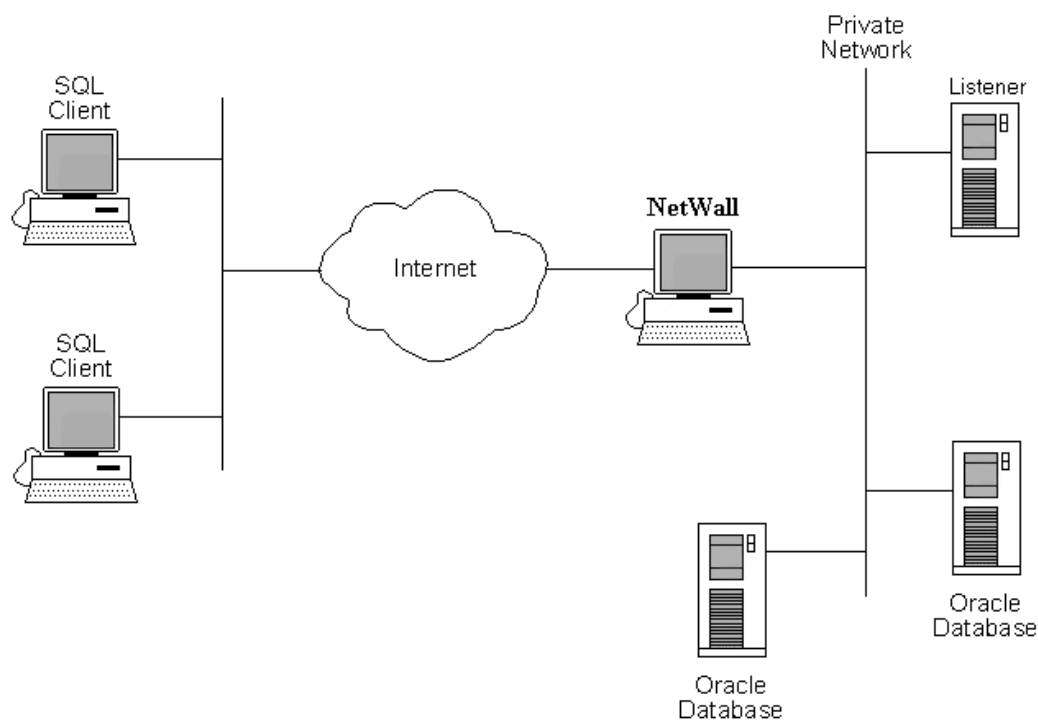
Pour faire de la translation dynamique d'adresse, cela ne peut se faire que par la translation dynamique des clients, car faire de la translation dynamique des serveurs n'a pas de sens.

Dans ce cas nous sommes très proches de la translation statique de l'adresse du client, sauf que le client prend l'adresse IP du NetWall lorsqu'il le traverse et comme port source un port défini par NetWall. Nous pouvons donc retrouver l'adresse du client grâce à ce numéro de port.

Il n'est pas nécessaire de modifier les tables de routage pour faire de la translation d'adresse dynamique.

### 3) Restriction sur l'utilisation de SQL\*Net avec NetWall

- Il ne faut pas faire de l'encodage de données avec SQL\*Net si l'on veut passer à travers un NetWall, car NetWall ne sera pas capable de lire le numéro de port dynamique contenu dans les data des paquets TCP/IP.
- Le listener doit être en écoute sur le port 1521 ou 1526.
- Le listener, le dispatcheur et le serveur de données doivent se trouver sur la même machine, ils doivent avoir la même adresse IP. Donc cette configuration ne marche pas avec NetWall :



### 3.3.1.2 *Modification de l'implémentation des sun-RPC*

Le service sun-rpc est un service qui utilise des ports dynamiques. Ces ports sont fournis par le port mapper. Nous pouvons accéder au port mapper en utilisant le protocole TCP ou UDP.

NetWall filtre les sun-rpc. Jusqu'à présent lorsqu'un programme appelait le port mapper il utilisait le protocole TCP (ou UDP). La connexion qui s'établissait par la suite était sur le même protocole TCP (ou UDP).

Le problème est le suivant, l'application contacte le port mapper en utilisant le protocole UDP et ouvre, dès qu'il a la connaissance du port qui lui a été assigné, une connexion TCP. Or, NetWall crée une entrée cache dynamique pour autoriser la connexion sur ce nouveau port, mais en utilisant le protocole UDP et non TCP. Donc l'application cliente ne peut pas fonctionner, car les paquets arrivant avec le protocole TCP ne seront pas autorisés à traverser NetWall, parce que le protocole n'est pas celui de l'entrée cache.

La correction adoptée a été d'autoriser la connexion sur UDP ou sur TCP en positionnant un joker pour le protocole. La vérification du protocole n'est plus faite quand il s'agit d'une entrée cache dynamique créée à la suite d'une demande d'ouverture de connexion des sun-RPC.

### 3.3.2 *librairie open framework*

La librairie open framework a été réalisée en deux étapes. Le développement devait dans un premier temps répondre à des attentes de clients sur la plate-forme AIX. La demande du marché était de filtrer le protocole H.323, ce qui rendait possible l'utilisation de logiciel comme Netmeeting au travers de NetWall. Au début de l'analyse de l'extension des fonctionnalités de NetWall, nous n'étions présents sur le marché des "pare-feu" seulement sur la plate-forme AIX, en effet la version de NetWall sur Windows NT était en cours de développement et le portage de NetWall sur Solaris allait débiter.

La deuxième étape était donc le portage de l'open framework sur les plates-formes Windows NT et Solaris. En effet, NetWall devait avoir un fonctionnement identique sur les trois plates-formes AIX, Windows NT et Solaris.

La librairie, a été faite en utilisant le langage C, elle est dépendante des plates-formes. Le code est en grosse partie le même sur AIX, Windows NT et Solaris, seuls les appels vers le driver de NetWall sont différents, mais nous n'avons pas mis les sources dans la partie commune du projet. En effet, le portage sur Windows NT et Solaris n'était pas prévu.

La librairie open framework, répond aux attentes exprimées dans les chapitres précédents, à savoir la manipulation d'entrée cache lorsque NetWall fonctionne. Cette librairie est livrée sur les trois plates-formes et elle a un fonctionnement identique sur chacune d'elles.

De plus, cette librairie a permis à NetWall de filtrer le protocole H.323 de façon optimum, ce qu'aucun fabricant de pare-feu ne savait faire.

### 3.3.3 *AAA, H.323 et les logs*

La réalisation du plugin H.323, a été faite à l'INRIA par l'équipe AAA. Ce plugin s'appuie comme nous l'avons décrit dans des chapitres précédents sur une architecture basée sur les agents et sur la librairie open framework. Le travail qui a été réalisé à Bull est un travail d'intégration de code. Nous nous sommes heurtés à de nombreux problèmes pour transférer le code et pour l'archiver dans le projet NetWall. En effet, le développement du plugin est fait dans le langage Java. Or, l'équipe de gestion des sources de NetWall, et plus généralement de tous les projets présents à Echirrolles, n'avait pas d'outils permettant de compiler et de générer des classes Java.

Il nous a fallu mettre en place tout l'environnement permettant de compiler le code Java et nous avons dû refaire tous les makefiles permettant de compiler le plugin. Cette partie a été un très gros travail, mais nous sommes parvenu à archiver, à compiler et à générer des classes Java. Nous faisons ceci de manière transparente pour le projet. Que l'on travaille sur un code c ou un code java, les outils de compilation et d'archivage sont les mêmes.

De plus le travail qui a été fait pour la compilation du code java dans le projet peut être dorénavant utilisé dans tous les projets de Bull à Echirrolles.

Nous avons pu constater l'utilité de ce travail lorsque nous avons transféré les sources de l'application de gestion des logs de NetWall. Le développement de l'application de gestion des logs a été fait par l'équipe AAA et le code est en Java. L'intégration dans le projet NetWall a été très simple car tout l'environnement de développement était déjà présent, seul les makefiles ont été modifiés.

### 3.3.4 *librairie SDK*

La librairie "*sdk*" a été réalisée pour la version 5.0 de NetWall. Cette librairie était indispensable pour le fonctionnement des proxies livrés avec le pare-feu.

D'un point de vue réalisation, la librairie est identique sur les trois plates-formes. Nous avons développé la librairie "*sdk*", en utilisant le langage C. Le code est commun aux trois plates-formes ; seul les appels aux *ioctl*, c'est à dire aux points d'entrées dans le driver sont différents. En effet ces points d'entrées sont dépendants des plates-formes.

Pour la partie driver de NetWall, le code est lui aussi dans la partie commune du projet ; il se trouve au même endroit que le code de l'open framework.

La librairie "*sdk*", répond aux attentes exprimées dans les chapitres précédents. A savoir, un échange d'information entre le driver et les proxies, pour des prises de décision sur l'ouverture de connexion et pour l'uniformisation des logs.

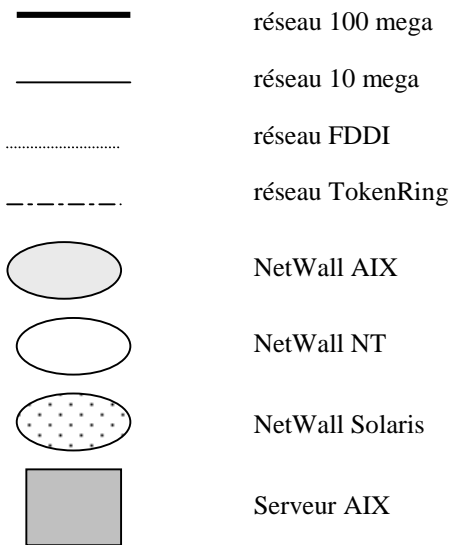
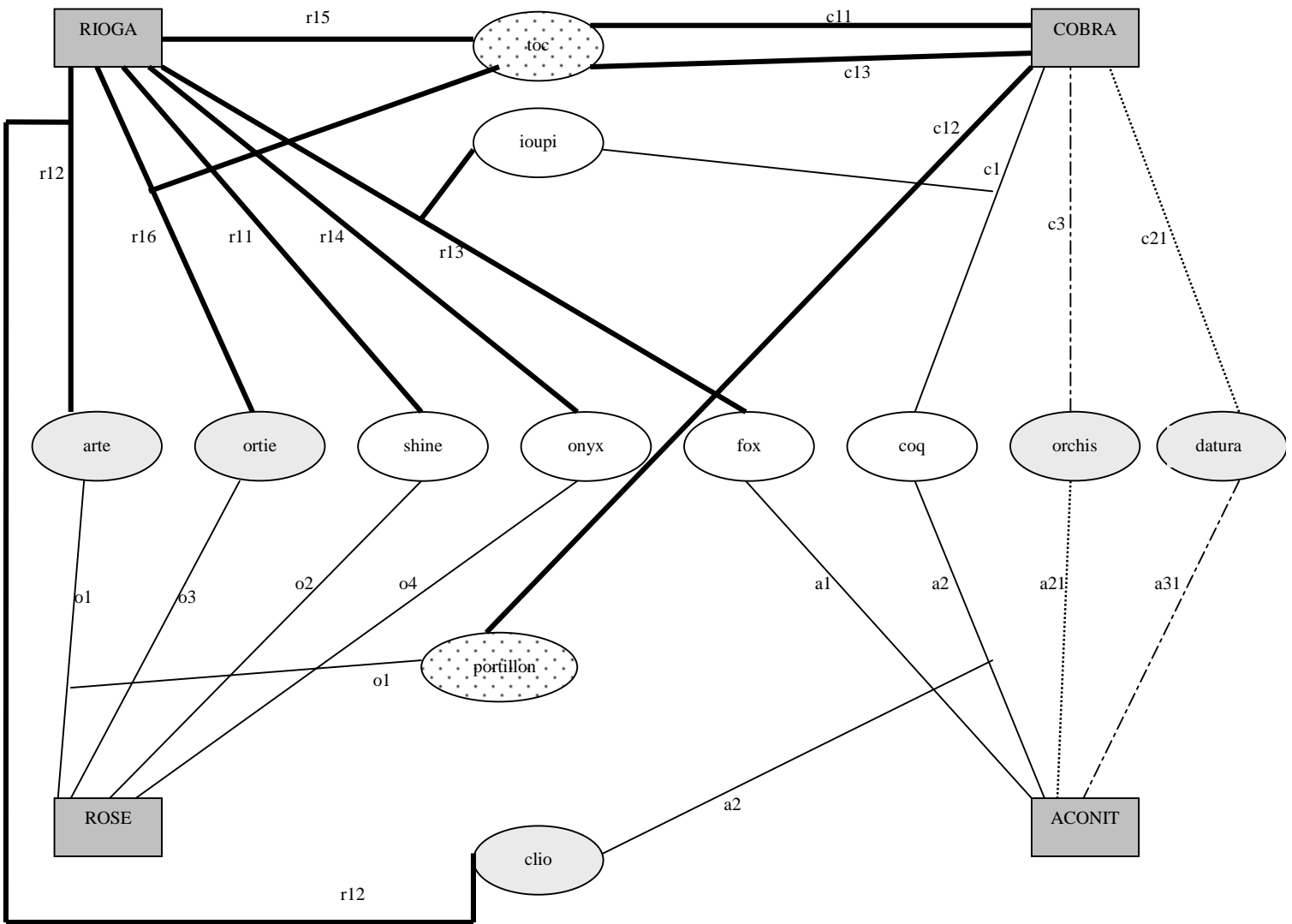
Cette librairie est livrée sur les trois plates-formes, elle a un fonctionnement identique sur chacune d'elles. De plus, elle a permis à NetWall dans sa version 5.0 de simplifier les règles de filtrage des proxies qui gèrent des droits d'accès pour les utilisateurs.

## 3.4 **Expérimentations**

L'équipe NetWall dispose d'une plate-forme de test, comprenant de nombreuses machines. Notamment, des machines AIX mono processeur, bi-processeurs, quadri-processeurs, mais également plusieurs PC installés avec Windows NT client et serveur, ce sont des PC mono processeur ou bi-processeurs ainsi que des machines Solaris sparc et intel.

Toutes les machines sont équipées d'au moins trois cartes réseaux, ce qui nous permet d'avoir de nombreux réseaux indépendants.

Le schéma suivant représente notre salle de test.



Toutes les machines ont en plus une carte réseau les reliant au réseau d'entreprise.

Grâce à cette plate-forme de test, nous avons pu expérimenter et tester les différentes modifications de NetWall. Les modifications les plus importantes ont été réalisées dans le driver de NetWall. Le moindre problème de pointeur ou le moindre problème d'allocation mémoire entraîne un crash de la machine et il faut parfois réinstaller le système d'exploitation.

La suite décrit les expérimentations qui ont été réalisées, et ce dans un ordre chronologique.

### 3.4.1 *SQL\*Net*

Pour pouvoir commencer les tests, Nous avons dû installer des serveurs oracles supportant SQL\*Net version 2. Nous avons installé deux types différents de serveurs sur des machines AIX. Le premier est un serveur dédié et le second est un serveur multi-thread.

Après avoir pris connaissance du fonctionnement de Oracle SQL\*Net, nous pouvions interroger les serveurs de données à partir d'un client. Les clients pouvant être des clients AIX ou des clients Windows NT. En effet, il est important de tester les connexions avec ces deux types de clients car les requêtes de connexion sont différentes.

Les essais de filtrage ont été réalisés en utilisant un NetWall version 3.3.

Nous avons vérifié le bon fonctionnement des connexions lorsque celles-ci étaient acceptées au niveau des règles de filtrage de NetWall, et le rejet dans le cas contraire.

L'étape suivante est la translation statique d'adresse. Tout d'abord la translation de l'adresse du client, c'est le cas le plus facile, car il n'y a pas de décalage dans les offsets de l'en-tête TCP. Puis la translation statique de l'adresse du serveur (qui dans ce cas là crée un décalage dans les offsets).

Et enfin, la translation dynamique d'adresse.

Les tests de fonctionnalités étant effectués (tests L1), les binaires sont testés par l'équipe de test qui effectue les tests d'intégration et de robustesse.

### 3.4.2 *Sun RPC*

Le filtrage du service Sun RPC était déjà présent dans NetWall, mais une correction du code a été nécessaire pour le fonctionnement des montages NFS entre des machines AIX et des machines Solaris. Cette modification faite nous avons vérifié qu'elle n'entraîne pas de régression dans le fonctionnement et dans le filtrage de NetWall. Les tests ont été réalisés via un montage NFS entre un client AIX et un serveur Solaris, en passant au travers de NetWall. Le but étant atteint, des tests de translation d'adresse statique et dynamique ont été effectués.

### 3.4.3 *OFW*

Pour vérifier le fonctionnement de la librairie "*lib\_ofw*", nous avons écrit deux programmes de tests créant des entrées caches dynamiques, et effectuant des recherches et des destructions d'entrées caches.

Le premier programme établit une connexion entre un client et un serveur telnet en passant au travers de NetWall, qui a pour seule règle de filtrage, une règle rejetant toutes les connexions. Dans ce cas là, NetWall est vraiment un mur infranchissable, sauf pour la connexion telnet car le programme de tests a créé une entrée cache dynamique acceptant cette connexion. Après la connexion nous détruisons l'entrée cache, et nous vérifions qu'aucune connexion n'est possible.

Le deuxième programme de test est un peu plus compliqué, il s'agit d'un plugin ftp. En effet NetWall filtre le service ftp uniquement sur le port 21, c'est pourquoi nous avons pensé au plugin ftp qui est

capable de filtrer le service ftp sur n'importe quels ports. Nous déroutons seulement la connexion de contrôle pour l'analyse et la découverte du port dynamique.

Ce programme a permis de tester la quasi-totalité des fonctions de la librairie open framework, sans s'intéresser aux problèmes de translation d'adresse dans ce deuxième programme de test car la translation d'adresse est réalisée dans le cadre de l'expérimentation du plugin H.323.

### *3.4.4 Netmeeting et Netscape conférence*

Nous avons installé sur plusieurs PC, Microsoft NetMeeting afin d'effectuer des conférences. Ces PC sont munis de cartes sons et de vidéos caméras. NetMeeting est un logiciel basé sur le protocole H.323.

Nous avons dans un premier temps effectué les tests sur un NetWall AIX version 4.0. Les expérimentations se déroulent suivant des méthodes d'approches classiques. C'est à dire, nous vérifions le fonctionnement de l'application sans NetWall, puis nous rajoutons NetWall, si tout se déroule bien nous passons aux tests de translation d'adresse IP sur chacunes des machines clients et serveurs.

Pour nos tests nous réalisons deux conférences simultanées avec deux utilisateurs dans chacune des conférences. Nous avons vérifié le bon fonctionnement des conférences lorsque celles-ci étaient acceptées au niveau des règles de filtrages de NetWall, et le rejet dans le cas contraire. En exploitant les traces fournies par NetWall, nous constatons que seules, les connexions de contrôle sont déroutées dans l'espace utilisateur où le plugin fait son analyse protocolaire. Les connexions de l'audio et de la vidéo ne sont quant à elles pas déroutées. Après une initialisation de connexion légèrement plus importante lorsque nous filtrons le protocole H.323, nous obtenons les mêmes performances pour la suite de la conférence, qu'il y est filtrage ou non.

Nous avons par la suite procédé à des tests de vidéo conférence avec de la translation statique d'adresse IP est constater le bon fonctionnement de ces conférences.

Ainsi s'achevaient nos tests sur AIX, nous avons complètement répondu à la demande qui était de filtrer le protocole H.323. Un petit plus a même était rajouté, car nous filtrons complètement le logiciel NetMeeting. En effet, NetMeeting utilise deux protocoles le premier H.323 et le second T120, ce dernier est utilisé pour des conversations de type "chat", pour l'utilisation de tableau blanc et pour le partage d'application, transfert de fichiers, etc... Au vu des expérimentations réalisées et des résultats nous pouvons dire que nous filtrons totalement le protocole H.323 et le protocole T120.

Une fois finis les tests du plugin sur NetWall AIX, nous sommes passés aux tests sur les plateformes Windows NT et Solaris. Nous avons rencontré des difficultés lors des tests de translation statique d'adresse IP. En effet, nous nous sommes heurtés à un défaut connu de NetWall (ce défaut était considéré comme un cas d'école).

La solution à ce problème arrive avec la version 5.0 de NetWall, où toutes les translations d'adresses et tous les déroutements proxies ont été revus et gérés différemment. Les tests fait sur la version 5.0 de NetWall NT et Solaris atteste du bon fonctionnement du plugin H.323. Ainsi nos trois plateformes ont le même fonctionnement.

Nous avons poussé les expérimentations un peu plus loin, en faisant des conférences avec trois, quatre ou cinq intervenants. Les conférences fonctionnent, mais il ne faut pas faire de translation d'adresse IP. De toute façon NetWall garanti le fonctionnement qu'avec des conférences à deux.

Des tests ont été faits pour réaliser des conférences à partir du logiciel Netscape conférence, il s'appuie sur le protocole H.323. Nous avons même testé des conférences entre Netscape conférence et NetMeeting.

### 3.4.5 librairie SDK et les proxies

Il n'a pas été écrit de programme de test pour vérifier le fonctionnement de la librairie "*lib\_sdk*". En effet, les expérimentations réalisées sur la librairie ont été faites au travers des tests d'intégrations des proxies de NetWall version 5.0 et sur chacune des trois plates-formes (AIX, Solaris et Windows NT). Tous les proxies ont été impactés par la librairie "*lib\_sdk*".

La vérification du fonctionnement des proxies devait être faite, car un changement important a été opéré dans cette version. La décision de l'ouverture de connexion par rapport à un utilisateur se fait dans le driver de NetWall. Cela se passe en deux étapes, la première, le driver accepte le passage de la connexion vers l'espace applicatif. La deuxième, le proxy vérifie si l'utilisateur a le droit de se connecter en se servant de la librairie "*lib\_sdk*".

Lorsqu'un client contacte un serveur, il ne sait pas forcément qu'il traverse un proxy applicatif qui peut effectuer un filtre sur la connexion. Ce type de proxy s'appelle un proxy transparent car il n'est visible ni du client ni du serveur. L'autre cas possible est que le client n'a pas connaissance du serveur de destination. Il ouvre une connexion avec le proxy de NetWall et ce dernier se charge de le connecter au serveur. La situation est la suivante, le client et le serveur ne se voient pas, ils s'échangent des données au travers du proxy de NetWall. Nous parlons alors de proxy non transparent.

Le test des proxies transparents et non transparents a été réalisé grâce à la librairie "*lib\_sdk*". Elle fournit l'information nécessaire aux proxies pour qu'ils puissent connecter le serveur.

La plupart des proxies ont été testés en ouvrant des connexions à l'aide de telnet sur les numéros de port de chaque service. Le refus d'ouverture de connexion de certains utilisateurs non autorisés est ainsi testé. Il en est de même pour le filtre de certains verbes (GET, PUT, ...). Les tests de fonctionnement des proxies suivants : pop, imap, ftp, telnet, http, mail et gateway sont effectués en utilisant de nombreux shell script mis à notre disposition.

**Chapitre 4 : BILAN ET PERSPECTIVES**

#### 4.1 *Rappel des objectifs et bilan technique*

Le principal objectif de ce travail de DRT était de permettre au milieu industriel d'intégrer dans un produit finalisé (commercialisable) une technologie développée au sein d'un laboratoire de recherche. La technologie à base d'agents proposée par l'Inria devait nous permettre, d'une part, de définir et d'implémenter une infrastructure d'intégration ouverte et extensible, d'autre part, de développer un système d'audit et de gestion des logs pour le produit NetWall.

L'aboutissement de ce travail devait assurer l'évolutivité de NetWall en vue, notamment, de supporter de nouveaux protocoles tels que l'audio et la vidéo conférence sur Internet ou encore de filtrer plus finement l'accès aux bases de données.

La principale difficulté d'un tel transfert de technologie réside aussi bien dans les problèmes techniques d'intégration dans le produit existant que dans les contraintes économiques et les délais liés à la fabrication du produit.

Il est à noter que les objectifs de départ ont évolué au contact de cette problématique. Le travail de DRT s'est recentré, au regard de ces contraintes, autour de l'ouverture de l'architecture de NetWall et du réalignement des fichiers de logs. De fait, le développement d'un système d'audit et de gestion des logs a été réalisé par l'équipe AAA.

Au cours de ces deux années de travail plusieurs réalisations concrètes ont été mises en place. Elles peuvent être regroupées dans trois ensembles rappelés brièvement ci-après.

Le premier ensemble correspond à la mise en œuvre de solutions pour filtrer les protocoles dynamiques. Ainsi, un filtre pour l'accès aux bases de données a été implémenté au niveau du driver de NetWall. En effet, NetWall a été enrichi par le filtrage du protocole SQL\*Net. Ce protocole est utilisé par une machine cliente pour accéder aux bases de données d'un serveur Oracle. Les connexions entre ces deux machines s'établissent au travers de ports alloués dynamiquement. La difficulté, comme nous l'avons indiqué précédemment, est d'extraire ces numéros de ports des données et de préparer l'environnement pour cette connexion. Une autre réalisation fut le filtrage du protocole H.323. En conséquence, NetWall a été un des premiers pare-feu à filtrer réellement le protocole H.323 d'audio et vidéo conférence sur Internet. A cet effet, la technologie à base d'agents développée par l'équipe AAA fut incorporée dans NetWall.

Le deuxième ensemble concerne la définition d'une architecture ouverte pour NetWall rendant réalisable l'intégration de nouvelles fonctions en s'appuyant sur deux bibliothèques. En effet, une infrastructure ouverte et extensible réalisée au travers de la bibliothèque open framework, offre la possibilité à des partenaires de développer des solutions spécifiques de contrôle et de filtrage. Le développement du filtre H.323 fut possible grâce à cette bibliothèque qui permet la manipulation directe d'éléments du driver (entrées caches), au niveau des couches applicatives. Une autre bibliothèque a été développée pour effectuer des échanges d'informations entre le driver et l'espace applicatif. Cette bibliothèque SDK est indispensable pour le fonctionnement des proxies et pour la cohérence des fichiers de log. Ces deux bibliothèques offrent une réelle valeur ajoutée au pare-feu de Bull.

Enfin, La dernière réalisation concrète a été l'insertion dans NetWall du prototype de gestion des logs. Ce prototype collecte des fichiers de logs géographiquement dispersés car les pare-feu peuvent être présents dans plusieurs agences d'une même société. Ce prototype permet de définir à la demande des nouvelles fonctions de traitement des informations enregistrées par les NetWall. Grâce à lui, NetWall se munit d'une technologie de pointe en intégrant des fonctionnalités à base d'agents et de programmation objet en Java.

La difficulté conceptuelle dans la mise en œuvre des nouvelles fonctions a résidé principalement dans le fait que l'architecture existante fournissait peu de liberté quand à l'insertion de code permettant son ouverture. De plus, il était risqué de réaliser d'importantes modifications dans le code avant la sortie du produit. C'est pourquoi, il a été inséré dans l'architecture un code qui permet l'ouverture sans impacter le fonctionnement et la stabilité de NetWall. Ceci nous a limité pour le développement de certaines fonctions comme la création de listes de contrôle d'accès dans le driver. En outre, la conception de fonctions de filtrage, utilisant la librairie open framework, est plus compliquée et demande une certaine connaissance de NetWall. Ces contraintes nous ont conduits à réaliser une librairie qui a une forte adhérence à NetWall notamment pour la gestion de la translation d'adresse.

Par ailleurs, la mise en œuvre sur différents systèmes d'exploitation s'est révélée compliquée car certaines fonctions nécessaires au fonctionnement de NetWall n'étaient pas présentes sur toutes les plates-formes. Nous avons donc été contraints d'effectuer quelques modifications dans l'architecture de l'open framework.

Toutes nos réalisations ont été basées sur l'utilisation des méthodes et outils de développement standards de Bull, en particulier l'environnement ODE (OSF Développement Environnement) et l'outil CMVC (Configuration Management Version Contrôle) pour la gestion des modifications sur le code source. Le principe de fonctionnement de ces outils est résumé ici. Pour travailler sur des fichiers, le développeur effectue une copie de la dernière version des fichiers dans son environnement de développement, en utilisant une commande de l'outil ODE. Il peut alors effectuer tous les travaux nécessaires sans perturber les autres personnes travaillant sur le projet. Lorsqu'il a terminé son codage et les tests d'intégrations, il utilise une commande ODE afin de soumettre ses fichiers dans la base commune. Si, entre temps, un autre développeur a modifié et soumis les mêmes fichiers, un mécanisme permet de fusionner les deux versions. La soumission se fait sous le contrôle de l'administrateur de l'outil CMVC.

La procédure de validation est la suivante, le développeur doit faire les tests unitaires de fonctionnement, puis il fait l'intégration de son code et teste le fonctionnement de son application ou de ses modifications. Une fois ces tests accomplis, l'équipe de validation effectue des tests de "stress" et de non-régression sur la totalité du produit. Elle vérifie aussi que le programme répond bien au cahier des charges ou au document de spécification. Si les testeurs détectent un dysfonctionnement, ils assignent un défaut au développeur en utilisant l'outil CMVC. Ce dernier est chargé de la correction, il resoumet ensuite son programme à l'équipe de validation.

## 4.2 Perspectives

Dans cette section, nous présentons essentiellement quelques perspectives d'extension à court terme, dont l'objet est de compléter directement le travail réalisé dans mon projet de DRT. Pour conclure, il est fait état de quelques perspectives à plus long terme sur un usage plus approfondi de l'environnement à agents pour la détection des attaques.

- **Librairie en Java pour la gestion des entrées caches**

NetWall fournit la librairie *"lib\_ofw"* et la librairie *"lib\_sdk"*. Ce sont des bibliothèques codées en langage C, dépendantes des plates-formes sur lesquelles elles sont utilisées. De plus, elles sont relativement adhérentes au driver de NetWall. Leur emploi demande une certaine connaissance et compétence dans la gestion des connexions, proposée par le driver de NetWall. Ainsi, les translations d'adresses sont disponibles au travers de ces bibliothèques mais cela demande une expertise indéniable de NetWall pour mettre en place un tel mécanisme.

Il serait intéressant de fournir une librairie en langage Java basée sur les librairies open framework et sdk. Elle pourrait offrir une couche d'abstraction supplémentaire et ainsi se détacher complètement du driver. La gestion des connexions serait plus simple et tous les problèmes de dépendance du driver seraient écartés. Cette librairie serait directement utilisable dans des programmes Java, ce langage devenant incontournable du fait de sa grande portabilité.

- **Ajouter des listes de contrôle d'accès dans le driver grâce à l'open framework**

L'ajout d'éléments dans les listes de contrôle d'accès du driver est une évolution attrayante de la librairie open framework. En effet, cela permettrait à l'open framework de gérer les connexions de bout en bout, ce qui éviterait l'ajout d'une règle au niveau de l'interface graphique de NetWall, puis une appropriation de cette dernière par l'open framework. La gestion des entrées caches deviendrait alors beaucoup plus simple, car dans ce cas, il ne serait plus nécessaire de créer des entrées caches en se préoccupant :

- des actions à appliquer aux paquets,
- de la translation d'adresse,
- d'autres mécanismes, telle que la gestion des états de la connexion TCP.

Toute la gestion des connexions serait faite par le driver.

L'inconvénient d'un tel ajout résiderait dans l'impossibilité de visualiser les règles de filtrage ajoutées par l'open framework. En effet, L'ajout de ces règles étant dynamique, lorsque NetWall fonctionne, il faudrait prévoir une communication entre le driver et l'interface graphique.

- **Gestion des logs**

- Outil de configuration

Une grande entreprise dont les locaux sont distants géographiquement peut être sécurisée de façon globale et centralisée grâce à la version 5.0 de NetWall. Ainsi, cette nouvelle version n'est plus un pare-feu isolé sans relation avec les autres pare-feu de l'entreprise. Il est capable de créer des domaines de sécurité non plus restreints à une zone géographique déterminée, mais englobant un domaine de sécurité virtuel. Ceci implique une forte coopération entre tous les systèmes NetWall de l'entreprise qui peuvent être gérés de manière globale et centralisée.

Dans ce nouvel environnement de sécurité où NetWall se présente comme un des leader du marché, il est indispensable d'intégrer une notion de gestion globale et centralisée de logs. C'est pourquoi, fort d'une expérience en gestion d'applications réparties et distribuées, l'équipe du projet AAA fournit des outils à base d'agents et de Java pour cette gestion.

Cependant, il manque un outil de configuration permettant d'exploiter la totalité des fonctionnalités livrées avec l'outil de gestion de logs. En effet, seules trois configurations standards sont disponibles dans NetWall. Or, cet outil a des capacités bien supérieures. Par exemple, il offre la possibilité de gérer les logs par rapport à des domaines de sécurité. Les logs pris sur un domaine interne, non critique pour l'entreprise, n'ont pas une importance primordiale. Il serait donc intéressant de disposer d'un minimum de logs et de les conserver durant une courte période. Par contre, un domaine jugé sensible, comme les finances ou Internet, demande une attention particulière. Il est donc intéressant d'avoir un maximum de logs et de les conserver plus longtemps.

- Fichier de log

L'application de gestion des logs de l'équipe AAA utilise le fichier "cnx.ulm" généré par NetWall pour réaliser le traitement demandé par l'administrateur. Une analyse du fichier de logs est effectuée, localement, sur chaque NetWall, puis le résultat est envoyé à un collecteur, chargé de traiter de

manière globale les informations qu'il reçoit. Le collecteur est une machine indépendante dont la localisation géographique dépend de l'administrateur.

Il serait possible d'optimiser la collecte d'informations en se branchant directement sur le driver et sur les proxies de NetWall. Ceci éviterait l'étape d'écriture et de lecture dans le fichier "cnx.ulm", ainsi que la prise de verrou avant l'écriture dans ce fichier. L'information serait collectée directement et non plus toutes les x minutes. Il serait alors possible de réagir plus vite à d'éventuelles attaques.

Une perspective à plus long terme concerne l'utilisation des agents AAA pour la détection en ligne de certains types d'attaques et pour la mise en place immédiate de réactions associées. En effet, l'environnement AAA permet de définir et de configurer à la demande des agents de filtrage qui peuvent donc être associés à la détection d'événements particuliers.

### **4.3 bilan personnel**

Pour conclure ce document, je donne ci-après quelques réflexions personnelles sur l'expérience acquise à travers ce projet de DRT. Ceci concerne plus particulièrement le savoir-faire en matière de noyau de système d'exploitation et de technologie pare-feu d'une part, et le bilan d'une opération de transfert de technologie d'autre part.

- **acquisition de connaissances**

L'extension des fonctionnalités de NetWall, m'a permis de découvrir des systèmes d'exploitation, de me former sur AIX, Solaris, Windows NT et linux. Une part importante de mon travail a concerné le développement de fonctionnalités se trouvant dans le noyau des systèmes d'exploitation et ainsi de me familiariser avec le fonctionnement de ces noyaux.

Cet apprentissage a été très enrichissant, notamment au niveau de l'importante rigueur et du caractère spécifique associés au développement d'extensions d'un driver. En effet, la recherche d'une erreur au travers d'un environnement d'exécution pas à pas n'est pas possible. De plus, le moindre problème d'adressage en mémoire peut avoir des conséquences importantes. Un pointeur non initialisé nécessite souvent un redémarrage de la machine quand il ne s'agit pas d'une réinstallation complète du système d'exploitation. A la suite d'un tel problème, il faut rechercher la cause du dysfonctionnement, dans un fichier contenant l'image de la mémoire au moment de l'erreur. Ceci se fait à l'aide d'un débogueur. La convivialité limitée de ces outils rend l'analyse longue, compliquée et nécessite une grande expérience dans ce domaine. Il s'avère donc indispensable d'insérer dans le code source de nombreuses traces, qui permettent de suivre le déroulement du programme.

Un pare-feu est un système ou un groupe de systèmes qui permet de mettre en œuvre une politique de contrôle d'accès entre différents réseaux. Conceptuellement il y a deux types de pare-feu selon qu'on s'intéresse au niveau IP ou au niveau applicatif.

Les pare-feu de niveau IP prennent généralement leurs décisions en se basant sur les adresses source, destination et les ports indiqués dans chaque trame TCP/IP ou UDP/IP, pris individuellement. Ils gèrent des informations internes sur l'état de chaque connexion filtrée, le contenu des flux de données, etc. La caractéristique la plus importante en ce qui concerne la plupart des pare-feu de niveau IP est qu'ils routent le trafic directement à travers eux. Ils tendent à devenir très rapides et transparents pour les utilisateurs. A l'inverse, les performances d'un réseau peuvent être réduites de façon importante par un pare-feu applicatif. Ce type de pare-feu est surtout utilisé pour le filtrage de verbes ou d'actions associés à un service spécifique (telnet, ftp, http, ...)

Peu à peu, les pare-feu de niveau IP tendent à devenir plus "conscients" de l'information qui les traverse et les pare-feu de niveau applicatif acquièrent une certaine transparence. De plus en plus, les

pare-feu supporteront le cryptage afin de protéger le trafic qui passe entre eux par l'intermédiaire d'Internet.

NetWall est un pare-feu complet puisqu'il propose un filtrage IP et un filtrage au niveau applicatif. Il offre également une solution pour permettre de crypter les données qu'il échange avec un autre NetWall ou un autre progiciel de sécurité.

Travailler dans le domaine de la sécurité informatique et pour l'évolution d'un pare-feu implique de s'intéresser aux différentes attaques connues et à la possibilité d'engendrer des trous de sécurité lors de son développement.

Les attaques peuvent être classées comme suit :

Classe d'attaque	Protocoles incriminés
Ecoute passive	Slip, icmp, smtp, snmp, dns, http, X
Vol de mot de passe	Slip, ftp, telnet, r-command
Usurpation d'identité	ppp, ip, arp, tcp, udp, ftp, smtp, pop, telnet, r-command, rcp, nfs, snmp, dns, imap
Déni de service	ip, snmp, icmp, tcp, dns
Corruption du client ou du serveur	ip, udp, smtp, telnet, r-command, rcp, nfs, http, script java
Divulgateion illicite d'informations	ftp
Contrôle d'un hôte par exploitation de bogue	Tous les protocoles

J'ai été confronté à deux types d'attaques. La première était un trou de sécurité de NetWall. Il s'agissait d'un bogue dans la méthode de filtrage pour les usurpations d'adresses. La deuxième était une nouvelle attaque de type recouvrement de fragments. Les contrôles pour empêcher ces deux attaques ont été mis en place dans toutes les versions de NetWall.

- **Transfert de technologie Inria – Bull**

Le travail du Diplôme de Recherche Technologique s'inscrit dans un programme de recherche et réalise un transfert de technologie des résultats du laboratoire vers le groupe Bull.

En effet, le laboratoire SIRAC mène depuis plusieurs années des recherches dans le domaine des systèmes et applications réparties. L'objectif général du laboratoire est de concevoir et de réaliser des services et outils pour le développement et l'exécution d'applications réparties. A cet effet, le laboratoire développe des technologies utilisant une programmation à base d'objets et des techniques de programmation par agents.

Par conséquent, NetWall se pourvoit d'une technologie de pointe en intégrant, les fonctionnalités à base d'agents et de programmation par objets en Java. Ce transfert de technologie a été mis en œuvre à deux reprises, la première lors de la réalisation du plugin H.323. La deuxième, lors de l'évolution dans le traitement et la gestion des logs.

**GLOSSAIRE**

---

AAA	Agents Anytime Anywhere
ACL	Liste de Contrôle d'Accès
anti-spoofing	Anti-usurpation d'adresse IP
boot	Démarrage du système d'exploitation
datagramme	Paquet comprenant en-tête IP, en-tête TCP et les données de l'application
dispatcheur	Permet de dispatcher les ports de connexion des serveurs multi-thread
driver de NetWall	Composant qui effectue le filtrage IP
entrée cache	Structure permettant des contrôles d'accès très rapide
file descriptor	Point d'entrée dans le noyau du système d'exploitation
flag	Drapeau
ftp	Protocole de transfert de fichiers
ftpd	Démon du protocole ftp
GUI	interface graphique utilisateur
H.323	protocole d'audio et vidéo conférence sur Internet
ioctl	Commande d'échange de structure entre le noyau et l'espace utilisateur
kernel	noyau du système d'exploitation
listener	démon d'écoute du protocole SQL*Net
Mbuf	Paquet Internet
multi-thread	Serveur oracle gérant plusieurs port de connexion SQL*Net
Netbiosd	Démon de netbios
Netmeeting	Logiciel de visioconférence sur Internet
Netwalld	Démon de NetWall
Port	Porte d'entrée à un service
port-mapper	Fournisseur de porte (Sun-RPC)
Proxy, proxies	Passerelle applicatif
rshd/rexecd	Démon des r-commandes
Spoofing	usurpation d'adresse IP
SQL*Net	nom générique désignant l'ensemble des couches nécessaires à la communication entre clients et serveur oracle
Sun-RPC	Remote procedure call de Sun
TCP/IP	Transfer Control Protocol/Internet Protocol. Il s'agit du protocole de communication utilisé sur Internet. Il est simple, robuste et est supporté dans la plupart des environnements.
Timer	Minuterie

**REFERENCES**

Oracle®, White Paper SQL\*Net and Firewalls

Oracle®, White Paper SQL\*Net Support in Gauntlet® Internet Firewalls

<http://www.tis.com>

INRIA, Base Répartie configuration et Installation

Bull, External Interface Specification, NetWall Kernel

Bull, Software Design Sepcification, NetWall: Windows NT version

Bull, NetWall Administrator's Guide (référence :86 A2 51 KX 01)

Dyade, AAA spécification, Architecture d'agents.

Dyade, Agent Infrastructure: The Agent Anytime Anywhere Platform

<http://www.dyade.fr/actions/aaa/aaa.html>

<http://www.dyade.fr>

<http://dyade.inrialpes.fr>

SCSSI, Panorama des vulnérabilités d'Internet

<http://mirror.ox.ac.uk/rfc.html>

ITU-T Recommandation H.225.0, version 2, call signaling protocols and media stream packetization for packet based multimedia communications systems.

ITU-T Recommandation H.323, visual telephone systems and equipment for local area networks which provide a non-garanteed quality of service.

Intel, H.323 and firewalls, intel internet video phone.

<http://support.intel.com/support/videophone/trial21/interfaq.htm>

[http://support.intel.com/support/videophone/trial21/h323\\_wpr.htm](http://support.intel.com/support/videophone/trial21/h323_wpr.htm)

**ANNEXES**

## 1.1 Fonctions couvertes par NetWall

### 1.1.1 Filtrage dynamique au niveau IP

- Gestion de contextes concernant l'état des connexions TCP et des pseudo connexions UDP. NetWall garantit que les paquets de retour, correspondent véritablement à une connexion en cours. Pour ce faire, NetWall gère un automate d'état, qui suit l'état de la connexion TCP dans son évolution, ou utilise une technique de timer d'inactivité pour le trafic UDP.
- Contrôle dynamique des connexions allouées par les services tels que FTP, RPC, Netbios, SQL\*NET, rcp/rshell/rexec.  
NetWall, grâce aux capacités de filtrage dynamique intégrant des fonctionnalités d'analyse intra protocole détaillée, scrute le trafic à destination des processus d'allocation dynamique de ports (ftpd, portmapper, netbiosd, listener, rshd/rexecd). Il reconnaît les demandes d'ouverture de connexions annexes ou attachées, ouvre alors un contexte et peut dynamiquement autoriser une connexion secondaire vers le port alloué. Cette autorisation sera automatiquement supprimée dès que la connexion secondaire en cause sera refermée.

Exemple : Contrôle dynamique des connexions de données FTP en relation avec la connexion de contrôle.

NetWall scrute les échanges sur la connexion de contrôle de FTP, analyse les requêtes d'ouverture de ports (commandes PORT et PASV de FTP). Il gère alors un contexte décrivant l'ensemble de la session FTP et garantit ainsi que les connexions de données de FTP sont effectivement issues de la connexion de contrôle en cours.

- Gestion sécurisée de la fragmentation IP.  
En cas de fragmentation IP, les informations pertinentes en terme de filtrage sont situées dans le premier fragment. Les autres fragments doivent alors être traités en fonction du premier fragment. NetWall est capable de contrôler la cohérence de cette fragmentation, afin d'appliquer la même politique de sécurité sur le premier fragment que sur les autres fragments.
- Translation statique et dynamique des adresses.  
Permet le masquage des adresses des réseaux internes par rapport aux réseaux externes, et permet de cacher les adresses des postes clients initiant des connexions vers l'extérieur derrière l'adresse du garde-barrière. Ce masquage est effectué par les méthodes suivantes :

- Translation statique d'adresses IP.

Pour chaque adresse IP réelle, l'administrateur peut définir dans NetWall une adresse IP virtuelle, constituant ainsi des tables de translation.

Les adresses source et/ou destination dans les paquets IP sont modifiées à la volée lors du traitement des paquets, en fonction de ces tables de translation. Dans ces conditions, seules les adresses déclarées comme virtuelles (i.e. celles définies par l'administrateur dans la table de translation) sont visibles depuis les autres réseaux.

Cette caractéristique peut permettre de masquer à la fois des adresses de clients (initiateurs de connexions) ou de serveurs (receveurs de connexions).

- Translation dynamique d'adresses IP.

Une adresse est automatiquement allouée lorsqu'un utilisateur établit une connexion au travers de la cible d'évaluation depuis un réseau vers un autre. Cette adresse est celle de la plateforme supportant NetWall elle-même (adresse de l'interface réseau par laquelle doit sortir le paquet). L'adresse source présente dans l'en-tête du paquet est remplacée par cette valeur.

Pour le traitement des paquets de réponse, NetWall mémorise l'état de la connexion et peut ainsi faire l'opération en sens inverse et retrouver la machine qui a initié cette connexion.

De cette manière, la machine externe destinataire de la connexion ne voit que l'adresse de l'interface réseau de NetWall avec laquelle il communique et non l'adresse de la machine qui a initié cette connexion.

Par principe, la translation dynamique n'a de sens que pour des connexions sortantes, c'est à dire que la translation d'adresse ne peut s'appliquer que sur l'adresse de la machine qui initie l'échange, et en aucun cas sur celle de la machine destinataire de l'échange.

Cette translation dynamique ne s'applique qu'aux protocoles TCP et UDP.

- Anonymat des machines.  
L'administrateur peut configurer NetWall pour que la commande ping ne le traverse pas. Cela évite les attaques visant à connaître les noms des machines du réseau protégé par des requêtes ICMP de type "echo request", envoyés par la commande ping.
- Dispositif de protection contre l'usurpation d'adresses IP («anti-spoofing») conforme à la recommandation CERT (CA-95:01).  
L'attaque consiste pour une station d'un réseau, à se faire passer pour une machine d'un autre réseau, dans le but de bénéficier de ses droits.  
NetWall contrôle, à partir de l'adresse source telle qu'inscrite dans le paquet en cours de filtrage, la cohérence entre l'interface réseau sur laquelle le paquet est attendu et l'interface réseau par laquelle le paquet est entré. Pour chaque adresse ou groupe d'adresses de machine de sous-réseau ou de réseau, l'administrateur doit indiquer lors de la configuration de NetWall sur quelle interface les paquets issus de ladite adresse doivent arriver. En cas d'incohérence, NetWall détruit les paquets.
- Filtrage des options IP.  
NetWall peut rejeter les paquets contenant des options IP telles que par exemple les options de "routage selon la source" souvent à la base d'attaques.

### *1.1.1 Fonctions de log et d'audit.*

NetWall enregistre les informations relatives à sa propre activité. Il offre à l'administrateur la possibilité de filtrer et d'exploiter ces informations.

### *1.1.2 Configuration et administration via une interface graphique conviviale.*

Relais applicatifs permettant des contrôles ou filtrages internes aux protocoles applicatifs, comme par exemple le filtrage des commandes internes de FTP pour autoriser ou refuser les accès en lecture et/ou en écriture.

### *1.1.3 Authentification des utilisateurs au niveau des relais applicatifs*

- S/Key  
Une liste de mots de passe est générée à partir d'un mot de passe connu de l'utilisateur. La liste de mots de passe dérivée est connue de NetWall et de l'utilisateur. Ce dernier doit fournir comme mot de passe à chaque nouvelle demande d'authentification le mot de passe suivant dans cette liste.
- Mots de passe simples

- Challenges et Réponses MD5
- Authentification des utilisateurs par carte à puce CP8
- Authentification des utilisateurs par des serveurs d'authentification externes
  - Radius
  - SecureID
  - Active Card
  - Access Master

	<b>Password</b>	<b>MD5</b>	<b>SKey</b>	<b>SecurID</b>	<b>AccessMaster</b>	<b>ActivCard</b>	<b>RADIUS</b>
IP Authentication	X	X	X	X	X (no SSO)	X	X
FTP	X	X	X	X	X (full SSO)	X	X
Telnet	X	X	X	X	X (full SSO)	X	X
Generic	NO	NO	NO	NO	NO	NO	NO
POP	X	NO	NO	X	X (no SSO)	X	X
IMAP	X	NO	NO	X	X (no SSO)	X	X
Mail	NA	NA	NA	NA	NA	NA	NA
HTTP/FTP	X	NO	NO	NO	X (no SSO)	X	X
Reverse HTTP	X	NO	NO	X (Cookie Mode)	X (no SSO)	X (Cookie Mode)	X

X = Authentification possible.

NO = Pas d'authentification.

NA = Non applicable.

SSO = Single Sign On.

#### *1.1.4 Fonctions de coopération des relais applicatifs avec des produits externes d'Antivirus*

#### *1.1.5 Contrôle à distance sécurisé*

Fonctionnalité garantissant l'authentification mutuelle et l'intégrité des données échangées entre la station d'administration et le(s) garde-barrière(s) administré(s). Cette interface donne accès à l'ensemble des fonctionnalités d'administration de NetWall, et reste compatible avec l'administration locale par l'interface graphique sur une console directement raccordée (gestion de verrous contrôlant les conflits d'accès).

#### *1.1.6 Fonction d'alerte dynamique*

Fonctionnalité fournissant une infrastructure permettant à l'administrateur de recevoir des alertes dynamiquement, en fonction d'événements et de conditions de son choix.

En fonction de la configuration définie par l'administrateur, NetWall peut exécuter un certain nombre d'actions de manière à prévenir ce dernier de l'occurrence d'événements relatifs à la sécurité. Ces actions peuvent être l'envoi d'un trap SNMP, l'envoi d'un message SMTP, l'affichage d'un message sur la console d'administration ou l'exécution d'un programme fourni par l'administrateur.

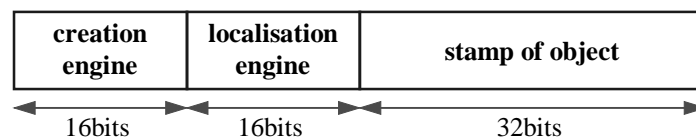
## 1.2 Agent Anytime Anywhere

### 1.2.1 Réalisation

#### 1.2.1.1 La classe AgentId

L'identificateur d'agent *AgentId* doit permettre d'identifier et de localiser de manière unique chaque agent dans le système. Les agents doivent pouvoir être créés à distance d'un serveur sur un autre ; afin de permettre la mémorisation de l'identification de l'agent créé par l'agent créateur, la génération des *AgentId* est réalisée par le serveur initiateur. En conséquence, il doit donc répondre à deux critères :

- localisation statique des agents afin de permettre l'acheminement des notifications ;
- génération locale des agents créés à distance.



#### Format des identificateurs d'objets

Afin de répondre à ces exigences, l'identificateur d'agent est constitué de trois parties : l'identification du serveur d'agents de l'agent créateur (attribut *from*), l'identificateur du serveur d'agents de résidence de l'agent créé (attribut *to*), une estampille locale au serveur d'agents de l'agent créateur (attribut *stamp*).

L'attribut *to* constitue l'indication de localisation de l'agent<sup>1</sup>, l'ensemble constitue l'identificateur global unique de l'agent. Actuellement chaque serveur d'agents gère deux zones d'estampilles : une zone pour les agents locaux, une zone pour les agents créés à destination d'autres serveurs<sup>2</sup>.

#### aaa/agent/AgentId.java

```
public class AgentId implements Serializable, Cloneable {
    private static int current[];
    static void init() throws IOException { ... }

    public short from;
    public short to;
    public int stamp;

    AgentId() { ... }
    AgentId(short to) { ... }

    AgentId(short from, short to, int stamp) { ... }

    public int hashCode() { ... }
    public boolean equals(Object obj) { ... }
}
```

La classe *AgentId* réalise l'interface *Serializable* afin de permettre la sérialisation des identificateurs d'agents pour les besoins de persistance et de communication, elle réalise l'interface *Cloneable* afin de permettre leur duplication. Ses méthodes *hashCode* et *equals* sont prévues pour en permettre une utilisation comme clé dans une table de hachage.

<sup>1</sup> Actuellement, les agents ne sont pas censés migrer, cette information est donc toujours à jour.

<sup>2</sup> On pourrait comme dans la version C++ gérer autant de zones que de serveurs.

### 1.2.1.2 La Classe Agent

La classe Agent est une classe abstraite qui définit l'interface et le comportement commun à l'ensemble des agents : tous les agents sont des instances de classes qui hérite de celle-ci ; ces fonctions se décomposent en plusieurs parties :

- L'interface d'accès au bus de message ;
- l'interface de la méthode de réaction aux notifications ;
- l'identificateur unique et le nom de l'agent ;
- les données et méthodes statiques permettant la gestion des agents en mémoire ;
- les méthodes de création des agents.

#### 1) L'interface d'accès au bus de messages

La méthode *sendTo* permet d'envoyer une notification à l'agent spécifié en paramètre. Elle permet de fixer l'identificateur de l'émetteur avant d'utiliser la méthode *sendTo* de la classe *Channel* ; cette méthode est protégée afin d'en empêcher l'utilisation par une entité autre que l'agent.

```
public abstract class Agent implements Serializable {
    ...
    protected void sendTo(AgentId to, Notification not);
    public void react(AgentId from, Notification not);
}
aaa/agent/Agent.java
```

#### 2) L'interface de réaction

La méthode *react* est exécutée par le moteur d'exécution au sein d'une transaction lors du traitement d'une notification destinée à l'agent. La méthode *react* définie dans la classe *Agent* détermine le traitement par défaut à apporter aux notifications d'erreurs de la machine à agents : destinataire d'une notification inconnu, aucun traitement connu à appliquer à la notification, etc.

#### 3) La gestion d'agents en mémoire

Les agents sont des entités persistantes et la classe Agent encapsule le mécanisme de "*swap-in/swap-out*" qui permet de charger (ou retrouver) en mémoire principale les agents à utiliser et de décharger sur disque les agents inutilisés depuis un certain temps<sup>3</sup>.

Ce mécanisme est mis en œuvre au moyen d'une table de hachage, *Agent.agents* dont la clé est l'*AgentId*, qui conserve la liste des agents actuellement présents en mémoire ; la date de dernier accès est elle conservée dans l'état de l'agent.

La méthode *load* permet de retrouver dans la table de hachage l'agent dont l'identificateur global est passé en paramètre ; si cet agent n'est pas présent, il est alors chargé depuis le disque. La méthode *save* permet de sauvegarder sur disque l'image d'un agent. Afin de permettre les opérations de chargement/déchargement, la classe Agent implémente l'interface Java *Serializable* qui permet la sérialisation (resp. désérialisation) d'un objet afin de le stocker (resp. charger) dans un fichier.

La méthode *garbage* est appelée lorsque le nombre d'agents en mémoire devient trop important, elle permet de vider tous les agents qui n'ont pas été accédés depuis un certain temps. Certains agents devant être chargés en permanence, l'attribut protégé *fixed* permet de les fixer en mémoire (cf. 5)).

Ces fonctions ne sont utilisées que dans la boucle de traitement du moteur d'exécution et ne nécessitent donc aucune synchronisation.

<sup>3</sup> A un instant donné, seul l'agent en cours d'exécution nécessite réellement d'être chargé.

```

aaa/agent/Agent.java
public abstract class Agent implements Serializable {
    static Hashtable agents;

    public static void init() {...}
    public static Agent load(AgentId id) {...}
    public void save() throws IOException {...}

    ...
}

```

#### 4) La création d'agents

La création des agents s'effectue en trois phases :

1. La création locale d'une image de l'agent via les constructeurs publics de sa classe : cet objet possède l'identificateur définitif du futur agent et peut-être manipulé comme n'importe quel objet mais il n'est pas persistant et il ne peut pas recevoir de notifications.
2. L'initialisation des attributs de l'agent au travers de l'interface de l'objet.
3. Le déploiement lancé par l'appel de la méthode *deploy*<sup>4</sup> sur l'objet. Cette méthode crée une notification *AgentCreateRequest* contenant l'état sérialisé de l'agent et l'envoie à l'agent *AgentFactory* chargé de la création des agents au sein du serveur de destination de l'agent. Une fois que l'agent a été déployé, il ne peut plus être utilisé qu'au moyen d'envoi de notifications ; la propriété de causalité du bus de message et l'atomicité de la réaction de l'agent *AgentFactory* permettent d'adresser des notifications à l'agent créé dès son déploiement.

Certains agents<sup>5</sup> sont créés via des constructeurs spéciaux de la classe *Agent* permettant de fixer leur identificateur, ou de positionner des attributs protégés. Ces constructeurs ne sont accessibles que par les classes du package *aaa.agent*.

```

aaa/agent/Agent.java
public abstract class Agent implements Serializable {
    ...

    private AgentId id;
    public String name;
    protected boolean fixed;

    public Agent(short to, String n) {...}

    Agent(short to, String n, boolean f) {...}
    Agent(String n, boolean f, AgentId id) {...}

    public void deploy() {...}
}

```

#### 5) Les agents "fixés" en mémoire

Certains agents pourront être fixé en mémoire, c'est à dire qu'ils seront chargés et initialisés lors de chaque démarrage du serveur, puis conservés en mémoire durant toute la durée d'exécution du serveur d'agents ; une méthode de finalisation sera appelée lors de la terminaison du serveur...

Les agents "fixés" en mémoire sont créés au moyen de constructeurs protégés de la classe *Agent* et accessibles uniquement depuis le package *aaa.agent*. Ces agents sont enregistrés dans une liste permettant de les charger lors du démarrage du serveur ; ils ne sont supprimés de la mémoire du serveur que lors de la terminaison de celui-ci.

<sup>4</sup> Cette méthode est définie dans la classe *Agent* et elle ne peut pas être surchargée.

<sup>5</sup> Par exemple, les agents *AgentFactory* afin qu'ils puissent être facilement identifiable en l'absence d'un service de désignation.

Lors de leur chargement, le système appelle la fonction `initialize` afin de permettre l'initialisation de leur contexte ; lors de la terminaison, la fonction `finalize` peut-être utilisée pour libérer les ressources des agents en mémoire.

Les agents "fixés" en mémoire sont utilisés par exemple pour la réalisation des agents qui gèrent les communications avec l'extérieur.

### 1.2.1.3 Les classes *Notification* et *Message*

La classe *Notification* définit la racine commune à l'ensemble des notifications ; elle réalise la sérialisation et la duplication de toute notification. Chaque notification est parfaitement identifiée par son émetteur, son destinataire et son contenu.

```
aaa/agent/Notification.Java
public class Notification implements Serializable, Cloneable {}
```

Une notification n'est jamais stockée en tant que tel, elle fait toujours partie d'un objet *Message* qui précise son émetteur et son destinataire, soit dans une queue de messages, soit dans une liste de messages en attente de délivrance dans le composant réseau. Les objets de la classe *Message* sont désignés par un identifiant local unique qui permet de les retrouver dans l'espace de stockage.

La classe *Message* est composée de deux parties, la première concerne la gestion des identificateurs de message ; outre l'aspect synchronisation, le seul vrai problème à résoudre est d'éviter la génération de deux identificateurs identiques entre deux sessions successives. Afin d'éviter d'avoir à sauvegarder l'identificateur courant lors de chaque allocation, on conserve un identifiant de session dans celui-ci<sup>6</sup>.

La seconde partie de la classe *Message* se compose des données des objets *Message*, des constructeurs et des opérations de sauvegarde et de restauration.

---

<sup>6</sup> On pourrait éventuellement factoriser l'écriture de l'identificateur courant lors du commit en utilisant une opération de sauvegarde explicite.

```

aaa/agent/Message.java
public class Message implements Serializable {
    static int current;

    static void init(short session) {
        current = ((int) session) << 16;
    }

    static synchronized int newStamp() {
        return (++current);
    }

    AgentId from;
    AgentId to;
    int stamp;
    Notification not;

    void save() throws IOException {
        Transaction.save(this, "Message" + stamp);
    }

    static Message load(long stamp) throws IOException {
        return (Message) Transaction.load("Message" + stamp);
    }

    /* Construct a new empty message in order to restore from disk. */
    public Message() {}

    /* Construct a new message. */
    public Message(AgentId _from, AgentId _to, Notification _not) {
    }
}

```

#### 1.2.1.4 Le bus de message

La classe *Channel* fournit l'interface d'envoi d'une notification d'un agent à un autre ; la classe *Channel* est responsable de la localisation de l'agent destinataire : en fonction de cette localisation, elle déposera le message soit dans la queue de message *qin* afin qu'il soit traité localement par le moteur d'exécution, soit dans la queue *qout* afin qu'il soit traité par la couche réseau.

```

aaa/agent/Channel.java
public class Channel {
    public static Channel channel;
    private Queue mq;
    ...
    public synchronized void
    sendTo(AgentId from, AgentId to, Notification not) throws IOException {
        ...
    }
}

```

La classe *MessageQueue* fournit l'implémentation de la partie destinataire du mécanisme de communication. Les messages<sup>7</sup> ayant une durée de vie faible, la liste de messages FIFO est conservée en mémoire principale au moyen d'un vecteur ; cette liste possède une image persistante.

##### 1) Persistance et atomicité

Les queues de messages *qin* et *qout* sont les seuls objets partagés entre les différents threads : moteur d'exécution et composants réseau ; cette particularité implique de nombreux problèmes de gestion de transaction : verrouillage, isolation, etc.

<sup>7</sup> Couple <notification, identificateur de l'agent destinataire>.

Afin d'éviter la gestion de transactions parallèles on restreint la durée de chaque transaction et on les sérialise. L'aspect transactionnel portant uniquement sur les images disques des objets, il suffit d'isoler les accès aux objets partagés (les queues de messages) et de réaliser les sauvegardes en section critique.

Afin de faciliter la gestion de l'atomicité, les notifications sont conservées dans une queue interne au bus de message jusqu'au point de validation ; elles sont alors recopiées dans les queues de messages qin et qout.

Pour diminuer le surcoût dû à la sauvegarde lors de chaque opération d'insertion (*push*) et de suppression (*pop*), les objets *Message* sont sauvegardés indépendamment et seule la liste des identificateurs de message est sauvée. La méthode *Restore* n'est utilisée que pour restaurer l'intégralité de la *MessageQueue* au lancement du programme.

### 1.2.1.5 Le moteur d'exécution

La classe *Engine* est la pièce centrale de l'architecture : dans sa boucle de traitement, elle distribue les notifications aux agents et elle s'assure de l'aspect transactionnel de la réaction à une notification. L'engine est un objet actif qui hérite de la classe *Thread* ; son rôle est de prendre successivement les messages de la queue qin et d'appeler la méthode *react* des agents destinataires avec la notification en question.

Afin d'assurer la propriété d'atomicité de la réaction, la séquence d'opérations est réalisée au sein d'une transaction :

- en cas de terminaison correcte, on assure la conservation des changements liés à la réaction : sauvegarde de l'état modifié de l'agent, enregistrement des notifications générées par la réaction et suppression de la notification traitée ;
- en cas d'erreur, annulation de tous les changements effectués.

```
public class Engine extends Thread {
    public Engine() {...}

    public void run() {
        while (true) {
            Message msg = qin.get();
            agent = Agent.load(msg.to);

            try {
                agent.react(msg.from, msg.not);
            } catch ( ... ) {
                ...
            };

            // Suppress the processed notification from message queue.
            qin.pop();
            // Push all new notifications in qin and qout, then saves changes.
            channel.dispatch();
            // Saves the agent state then commit the transaction.
            Agent.save();
        }
    }
}
```

aaa/agent/Engine.java

Le constructeur de la classe *Engine* initialise le thread et démarre la méthode *run* qui contient la boucle du moteur d'exécution.

#### 1) Traitement en cas d'erreurs

Si l'agent destinataire de la notification est inconnu, le moteur d'exécution retourne une notification *UnknownAgent* à l'agent source.

Si l'agent destinataire ne sait pas traiter la notification il génère une exception *UnknownNotificationException*, cette exception est récupérée par le moteur d'exécution qui retourne alors une notification *UnknownNotification* à l'agent source. Dans ces deux cas, le moteur d'exécution valide les résultats de la réaction.

Si une exception survient durant le traitement de la réaction, le moteur d'exécution restaure l'état de l'agent depuis son image persistante, puis il détruit la notification fautive et il efface les notifications générées par la réaction.

## 2) Gestion de l'atomicité

En cas de bonne terminaison de la réaction, le moteur d'exécution commence une transaction pour enregistrer les résultats obtenus ; la transaction assure l'atomicité et la persistance des différents changements :

- suppression de la notification traitée de la queue de messages *qin*,
- destruction sur disque du message associé,
- distribution des notifications générées par la réaction dans les queues de messages *qin* et *qout*,
- sauvegarde de l'état modifié de l'agent.

La transaction peut alors être validée. Le traitement atomique de ces actions est important afin que si une erreur survient durant cette phase, le système puisse repartir dans l'état précédant la transaction.

Une fois la transaction validée, on peut rendre visibles les notifications ajoutées dans les queues de messages *qin* et *qout* avant de libérer le verrou transactionnel.

En cas de problèmes durant la réaction, le moteur d'exécution commence une transaction pour réparer :

- restauration de l'état de l'agent,
- suppression de la notification fautive de la queue de messages *qin*,
- destruction sur disque du message associé,
- nettoyage du bus de messages de toutes les notifications générées par la réaction.

### 1.2.1.6 Les composants réseau

Le problème à résoudre est la mise en place d'un système de communication pour l'envoi et la réception de messages entre les serveurs ; pour différentes raisons ce système doit se situer au dessus de IP.

## 1) Architecture

La couche de communication est constituée de plusieurs composants actifs :

- le composant *NetServerIn* gère l'ensemble des messages reçus par le serveur : notifications, messages de service, etc ;
- le composant *NetServerOut* émet les notifications à destination des autres serveurs ;

Ces différents composants interagissent entre eux et avec le moteur d'exécution, la synchronisation et la gestion des transactions est donc cruciale.

## 2) Identification et localisation

Les serveurs d'agents sont identifiés statiquement lors du déploiement de l'application ; A tout moment un nouveau serveur peut être créé, la destruction d'un serveur implique la destruction préalable de tous ses agents.

## 3) Notifications distribuées

Hormis les propriétés de fiabilité du système de communication, il faut garantir le traitement transactionnel de la réaction à une notification. Pour ce faire, l'ensemble des changements relatifs à une réaction est enregistré dans une seule et même transaction du moteur d'exécution, le transfert des notifications distantes est ensuite réalisé en asynchrone par le composant *NetServerOut* :

1. Lorsqu'un agent signale une notification, celle-ci est stockée dans une file interne au bus de message.
2. Lors de la fin de la réaction, le contenu de cette file est diffusé dans la queue *qin* pour les notifications à destination d'agents locaux, dans la queue *qout* dans le cas contraire ; la transaction validée le traitement atomique de la réaction est alors assuré.
3. L'activité asynchrone du composant *NetServerOut* prend alors chaque notification dans la queue *qout* et insère le message correspondant dans une liste locale durant une transaction, puis les communique aux serveurs destinataires.
4. Associé à chaque serveur, le composant *NetServerIn* reçoit les notifications en provenance des autres serveurs et les incorpore dans la queue locale *qin* durant une transaction ; après avoir validé la transaction, et donc avoir assuré la permanence du message reçu, le récepteur envoie l'acquittement à l'émetteur.
5. Après avoir reçu l'acquittement, l'émetteur peut supprimer le message de la liste des messages à émettre.

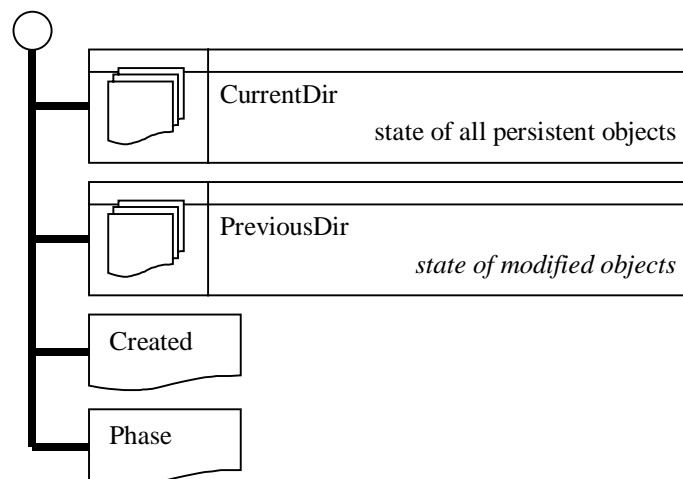
Les notifications sont transmises par valeur ; en conséquence, leurs identificateurs restent des identificateurs locaux et dès qu'un message a été transmis et acquitté la notification associée sur le site source peut être détruite.

Durant le transfert des messages d'un site à un autre il faut encoder/décoder les données transmises afin de résoudre les problèmes d'hétérogénéité. Cet encodage/décodage doit être fait pour les objets notifications, mais aussi pour les agents lors de leurs déploiements. Pour ce faire, on utilise le mécanisme de sérialisation/désérialisation d'objets Java (cf. interface *Serializable*).

#### 1.2.1.7 *Persistence et atomicité*

##### 1) FSTransaction

La persistance et les transactions sont réalisées en utilisant le système de fichiers de la machine hôte. Chaque serveur d'agents est associé à un répertoire de persistance organisé de la façon suivante :



#### **Structure du répertoire de persistance**

A chaque objet est associé un fichier dont le nom est simplement le nom unique de l'objet. Ce fichier contient l'état de l'objet tel qu'il est rendu par les primitives de sérialisation Java ; ce fichier reflète l'état courant de l'objet s'il réside dans le répertoire *current*, et l'état antérieur à la transaction courante s'il réside dans le répertoire *previous*.

Lors d'une opération d'annulation de la transaction, le système ne prend pas en charge la restauration des états des objets en mémoire ; cette étape est à la charge de l'utilisateur en se basant sur les images disques restaurées.

a) *Initialisation* : *init*.

Cette fonction initialise le répertoire de travail, puis teste l'état (fichier *Phase*) dans lequel se trouvait le système lors de l'arrêt :

- si le système a été arrêté durant un *commit* (resp. *rollback*), alors *init* termine la phase de validation (resp. annulation) à partir des informations stockées sur disque ;
  - validation : destruction de tous les fichiers du répertoire *previous*, nettoyage du fichier *Created* ;
  - annulation : restauration dans le répertoire *current* des fichiers modifiés par la transaction<sup>8</sup>, destruction de tous les fichiers créés par la transaction<sup>9</sup> ;
- si le système a été arrêté durant une transaction, alors *init* annule la transaction ;
- si le système a été arrêté normalement (en dehors de toute transaction), alors *init* n'a rien à faire.

b) *Création d'objets*

Lorsqu'un objet est sauvé pour la première fois, son image est écrite dans un fichier du répertoire *current*, et son nom est conservé dans le fichier *Created* afin de savoir quels fichiers détruire en cas de *rollback*. Lors de l'opération de validation, il suffit d'effacer le contenu de ce fichier afin d'entériner cette création.

L'opération de *rollback* consiste à détruire du répertoire *current* tous les fichiers dont le nom figure dans le fichiers *Created*.

Afin d'optimiser les opérations de validation et d'annulation "à chaud", la liste des objets créés est conservée en mémoire dans un vecteur : *created*.

c) *Lecture et modification d'objets* : *load*, *save*

L'opération de lecture retourne simplement l'objet contenu dans le fichier correspondant du répertoire *current* s'il existe.

Lors d'une sauvegarde, le système recopie le fichier correspondant à l'objet dans le répertoire *previous* si cela n'a pas encore été fait, puis il écrase ce fichier avec l'image courante de l'objet. La phase de validation consiste alors simplement à "oublier" les sauvegardes des objets modifiés, en détruisant tous les fichiers du répertoire *previous*.

Afin d'optimiser les opérations de validation et d'annulation "à chaud", la liste des objets modifiés est conservée en mémoire dans un vecteur : *used*.

Les primitives *load* et *save* peuvent être utilisées sans problème en dehors d'une transaction.

d) *Destruction d'objets* : *delete*

Le fichier correspondant à l'objet est simplement déplacé dans le répertoire *previous* ; lors du *commit*, le fichier est alors réellement détruit. Si une opération *rollback* est exécutée, le fichier est restauré dans le répertoire *current*.

La méthode de destruction de l'objet persistant doit être appelée explicitement.

e) *Validation et reprise* : *commit*, *abort*

Ces opérations commencent par "logger" dans le fichier *Phase* que la transaction entre dans une phase de validation ou de reprise.

L'opération *commit* détruit ensuite tous les fichiers du répertoire *previous*, puis elle efface le contenu du fichier *Created*.

L'opération *rollback* commence par restaurer l'état de tous les objets dont une copie apparaît dans le répertoire *previous*, puis elle détruit tous les fichiers dont le nom apparaît dans le fichier *Created*.

Lors d'une validation ou d'une annulation de transaction "à chaud", ces opérations utilisent les vecteurs *created* et *used* pour accélérer la détermination des fichiers à traiter. Si le système est arrêté durant ces

---

<sup>8</sup> à partir des images sauvegardées dans le répertoire *previous*.

<sup>9</sup> dont la liste est contenue dans le fichier *Created*.

opérations, le système termine la validation ou la reprise lors du démarrage suivant à partir des informations sauvegardées sur disque.

Le verrou de la transaction n'est pas libéré immédiatement par les primitives *commit* et *abort*, il est nécessaire d'appeler explicitement la primitive *release* pour le faire ; ceci permet, lors d'une validation ou d'une reprise, d'effectuer un certain nombre d'opérations en exclusion mutuelle sur les objets chargés en mémoire.

### 1.2.1.8 Compression, cryptage, etc

Il suffit d'utiliser un ou plusieurs filtres lors des opérations de sauvegarde et restauration des objets (GZIPInputStream vs. GZIPOutputStream par exemple).

## 1.2.2 Extensions

### 1.2.2.1 La communication avec l'extérieur

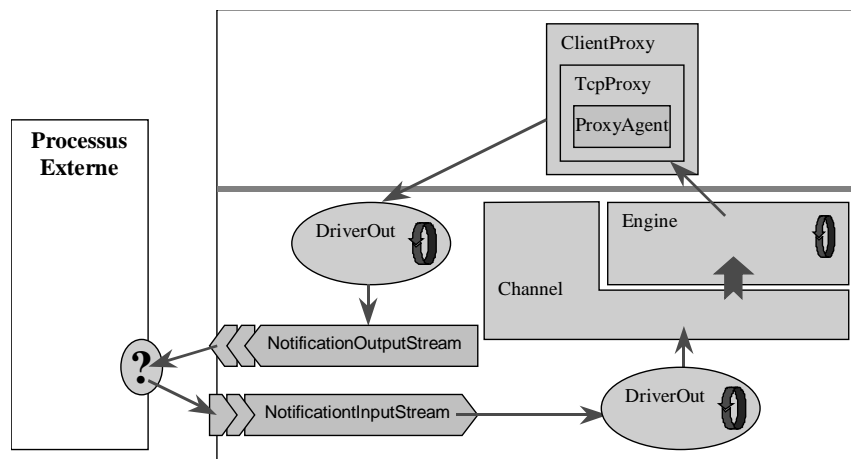
#### 1) Architecture globale

Deux propriétés essentielles ont été recherchées lors de la définition de ce mécanisme de communication : bidirectionnel et indépendant de la technologie Agent.

L'aspect bidirectionnel a été rendu possible par l'existence d'un agent proxy par flot de communication et donc par processus.

L'indépendance vis-à-vis de la technologie Agent est permise par le fait que les drivers manipulent des flots de données fournis par l'utilisateur ; en conséquence, l'utilisateur peut librement transformer les données transférées pour permettre le dialogue des deux entités :

- Notification → Format propriétaire
- Format propriétaire → Notification



#### Architecture globale

Outre l'agent proxy et les objets flots de données, le mécanisme met en œuvre deux composants actifs et une queue de messages :

- l'agent proxy reçoit les notifications destinées au processus extérieur et les insère dans une queue de messages locale ;

- le driver *DriverOut* prend les objets *Notification* dans la queue locale et les écrit sur le flot de données sortant ; l'objet *Notification* est alors transformé par l'objet *NotificationOutputStream* en un message compréhensible par le processus extérieur ;
- le driver *DriverIn* lit des objets *Notification*<sup>10</sup> sur le flot de données entrant, il les dirige alors vers la queue de messages correspondante : *qin* ou *qout* ; ce driver a la responsabilité de gérer l'atomicité de l'accès aux queues de messages ;
- Le driver *DriverConnect*, dont l'utilisation est optionnel permet la gestion asynchrone de la connexion.

#### a) L'agent proxy

Le rôle de la classe *AgentProxy* est de fournir un framework permettant de produire des agents proxy spécifique tout en maintenant la cohérence de fonctionnement de la machine à agents. Il doit donc permettre la spécialisation du protocole de communication et masquer l'accès aux mécanismes interne de la machine.

L'agent *AgentProxy* est un agent "fixé" en mémoire, il a la responsabilité de l'établissement de la connexion et de la création des drivers :

- La méthode *initialize* crée la queue de communication avec le driver *DriverIn*, appelle la méthode spécifique de connexion (création des flots de données d'entrée et de sortie), puis crée et démarre les drivers ;
- La méthode *reinitialize* permet de reconnecter les flots de données ;
- La méthode *finalize* appelle la méthode spécifique de déconnexion (fermeture des flots de données), puis arrête et détruit les drivers.

```

aaa/agent/ProxyAgent.java
public abstract class ProxyAgent extends Agent {
}

```

```

aaa/agent/ProxyAgent.java
class DriverIn extends Driver {
}

class DriverOut extends Driver {
}

class DriverConnect extends Driver {
}

```

#### b) L'interface NotificationInputStream

```

aaa/agent/NotificationInputStream.java
public interface NotificationInputStream {
    /**
     * Gets a Notification from the stream.
     */
    public Notification readNotification()
        throws ClassNotFoundException, IOException;

    /**
     * Closes the stream.
     */
    public void close() throws IOException;
}

```

#### c) L'interface NotificationOutputStream

<sup>10</sup> Créés par l'objet *NotificationInputStream* à partir des données en provenance du processus extérieur.

```
aaa/agent/NotificationOutputStream.java
public interface NotificationOutputStream {
    /**
     * Writes a <code>Notification</code> to the stream.
     */
    public void writeNotification(Notification msg)
        throws IOException;

    /**
     * Closes the stream.
     */
    public void close() throws IOException;
}
```

## 2) L'agent TcpProxy

L'agent TcpProxy est un exemple d'utilisation du mécanisme de communication décrit ci-dessus ; il permet, par héritage, d'écrire des Agents de gestion de flots TCP clients ou serveurs. La classe TcpProxy surcharge les méthodes initialize et reinitialize, elle définit les méthodes connect et disconnect, elle définit deux méthodes abstraites afin de fixer les filtres d'entrées et sorties :

**SetInputFilters** : permet de créer le filtre d'entrée à partir du flot d'entrée du socket.

**SetOutputFilters** : permet de créer le filtre de sortie à partir du flot de sortie du socket.

### a) Une utilisation de l'agent TcpProxy : aaa/ode

Il suffit de définir les filtres de transformation entre les notifications et les messages compréhensibles par l'application extérieures, puis de construire une classe d'agents héritant de TcpProxy définissant les méthodes abstraites *setInputFilters* et *setOutputFilters*.

L'application aaa/ode permet l'utilisation de l'environnement de développement ODE en distribué depuis des PC's. Elle utilise le TcpProxy pour interfacier un éditeur emacs avec la machine à agents afin de lui permettre de transmettre les commandes de l'utilisateur et de récupérer les messages d'erreurs.

La classe SandboxInputStream permet de filtrer le flot de données en provenance du socket TCP et d'en extraire les commandes destinées à l'application ; ces commandes sont ensuite transformées en notification à destination des agents concernés.

La classe SandboxOutputStream permet de transformer les notifications de compte-rendu en messages afin d'être affichés dans une fenêtre.

La classe SandboxClientProxy définit le proxy serveur permettant la connexion de l'éditeur emacs.

## 1.2.3 Outils de construction

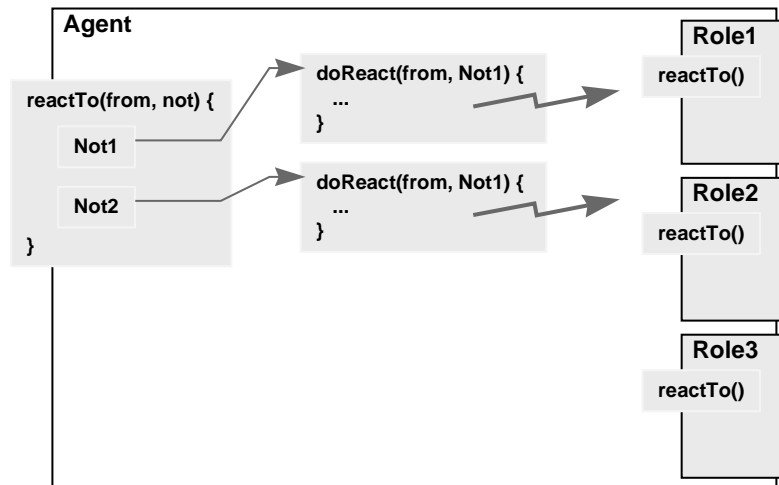
### 1.2.3.1 Agents configurables

La communication entre les agents se fait au moyen d'objets rôles dont la mission est la configuration de l'agent destinataire<sup>11</sup> et la transmission des notifications à celui-ci. Chaque rôle contient un ou plusieurs agents proxy dont l'interface est "conforme" à l'agent représenté.

---

<sup>11</sup> Ce qui permet la construction d'application à base d'agents par des outils visuels.

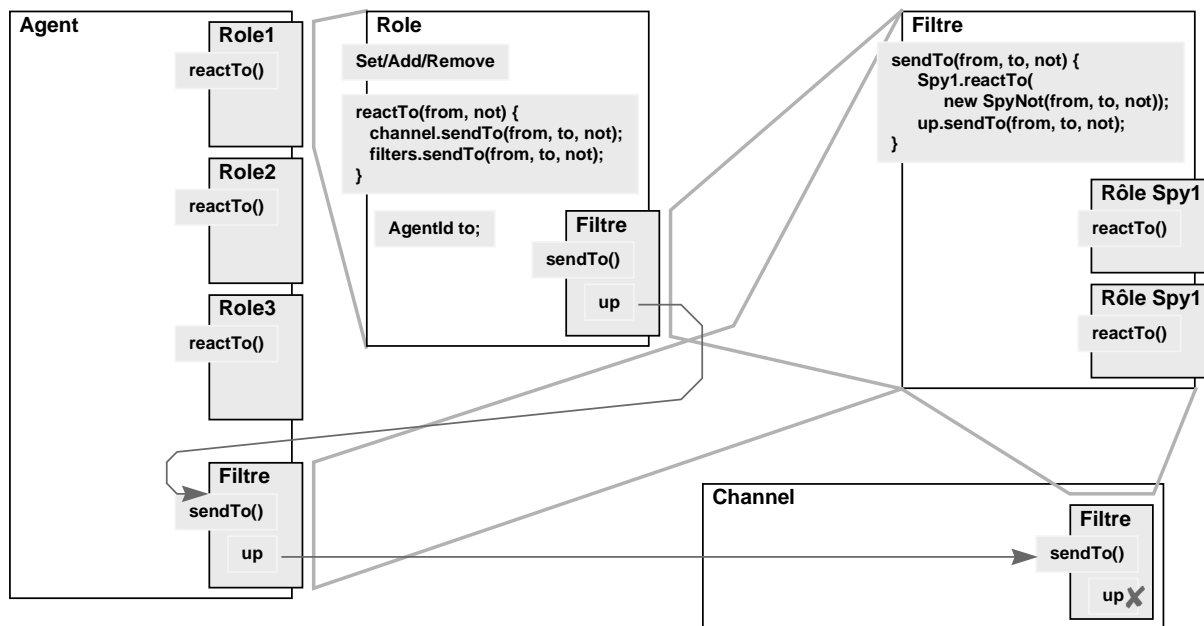




### Agents et réactions

La classe *Role* permet de spécifier le comportement attendu de la part d'un correspondant agent ; en particulier on spécifie l'ensemble des types de notifications que cet agent doit accepter de traiter. La classe *Role* réalise la même interface que la classe *Agent*, et bien que l'activité normale d'un rôle soit la transmission de la notification à un agent externe, on peut envisager des

En outre, l'objet Rôle permet la configuration dynamique du (ou des) destinataire(s) des notifications, ainsi que la mise en place des filtres.



### Rôles et filtres

Les filtres sont des objets passifs qui réalisent l'interface "sendTo" du Channel, chaque filtre possède une liste de rôle auquel il diffuse les notifications qui lui sont adressées. Les filtres sont organisés hiérarchiquement.

## 1.3 CMVC et ODE, gestion des sources

### 1.3.1 Introduction

La gestion des modifications sur le code source est assurée par l'outil CMVC (Configuration Management Version Control).

L'équipe NetWall assure :

- le développement et la maintenance des codes sources
- la rédaction des documents de développements

Le service " SHCM" (Software Hardware Configuration Management) assure :

- la gestion de la base de référence des codes sources (base ODE) et de la base d'archivage (base RCS)
- la fabrication de NetWall

Le service " Documentation" assure :

- la rédaction de la documentation d'exploitation

Le service " GRDOC "assure :

- la gestion de la base des documents de développements
- la gestion de la base des documents d'exploitation

### 1.3.2 Gestion de la configuration de NetWall

#### 1.3.2.1 Procédure de développement

##### 1) Généralités

La procédure de développement est basée sur l'utilisation des outils ODE et CMVC. Elle s'appuie sur le process du service SHCM. Le lecteur doit être familiarisé avec les termes de l'outil ODE pour mieux comprendre la suite. Un "GLOSSAIRE", donne une définition des termes les plus souvent employés.

##### 2) Initialisation de l'environnement de développement

L'initialisation consiste à mettre à disposition un espace pour le développement. Elle est assurée par le service SHCM. Les outils ODE et CMVC étant gérés de manière très étroite, le développement ne commence que lorsque les environnements sont prêts pour les deux outils. Dès que l'initialisation est terminée, tous les objets de la liste de configuration sont sous contrôle.

##### a) Initialisation de la base ODE

L'espace mis à la disposition des développeurs s'appelle un "latest". Au début du développement, le "latest" est vide et on a la visibilité sur tous les fichiers de référence contenus dans un "freeze". Au fur et à mesure du développement, le "latest" contiendra les fichiers modifiés.

##### b) Initialisation de la base CMVC

Pour permettre la saisie des défauts ("Defects"), il est créé des "composants" dans CMVC. Un "composant" CMVC n'a pas de contenu, c'est seulement une indication donnée par la personne qui ouvre le défaut. Cette indication est une "vue utilisateur" sur la partie de NetWall où le défaut est identifié. Le composant CMVC n'est pas lié au code source.

La saisie des défauts sur la documentation d'exploitation se fait sur le composant :  
netwall\_doc

La saisie des défauts sur les tests se fait sur le composant :  
netwall\_test

Il est également créé une "Release" c'est à dire la version dans laquelle les défauts sont corrigés, et un "Level" qui contiendra les "Defects" intégrés dans le prochain "freeze".

### 3) Fonctionnement

Chaque développeur possède un environnement de développement privé sur la machine de développement. Cet environnement privé s'appelle une "sandbox". La base des sources du projet NetWall est située sur la machine de l'équipe SHCM. Le lien entre les environnements privés et l'environnement commun (base ODE) est assuré par un montage NFS.

L'accès aux données est sécurisé via un système d'ACLs (Access Control List).

### 4) Développement

Pour travailler sur un fichier, le développeur commence par le "privatiser" : la commande bco d'ODE effectue une copie de la dernière version du fichier (soit depuis le latest, soit depuis le freeze) vers la sandbox du développeur.

Le développeur peut alors effectuer tous les travaux nécessaires sans perturber les autres personnes travaillant sur le projet. Il peut créer des versions intermédiaires dans sa sandbox avec la commande bci.

Lorsqu'il a terminé son codage et les tests d'intégrations, il utilise la commande bsubmit afin de soumettre son(ses) fichier(s) dans le latest. Si, entre temps, un autre développeur a modifié et soumis le(s) même(s) fichiers(s) un mécanisme de "merge" permet de fusionner les deux versions.

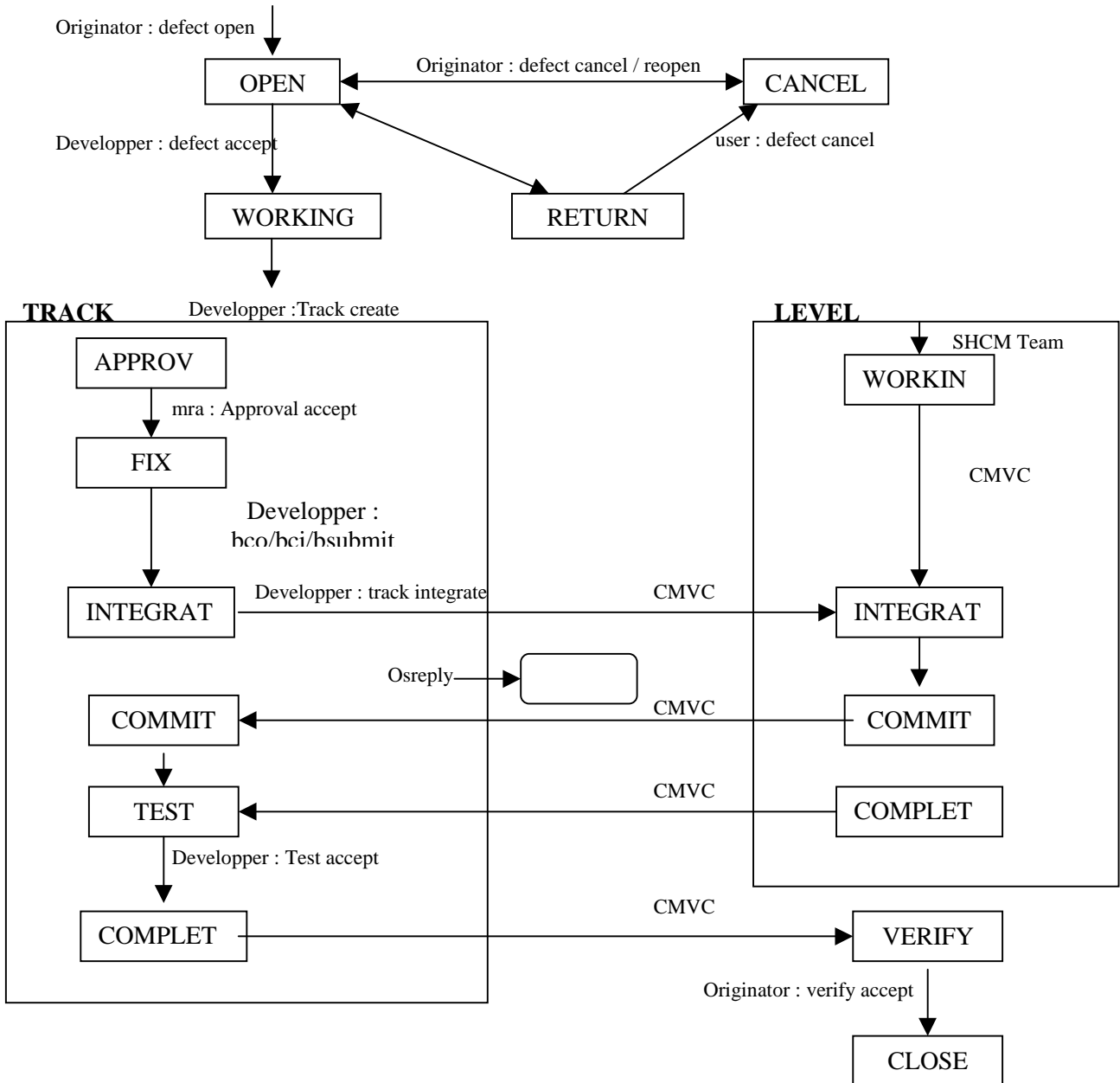
5) Gestion des versions

Les numéros et les changements de version sont complètement gérés par l'outil ODE. Toute soumission de fichier (quelle que soit la modification) entraîne un changement de version.

6) Contrôle des modifications

Toutes les modifications sont contrôlées : l'outil ODE a un lien avec l'outil CMVC afin que toutes les modifications de code soient liées à un défaut et que la modification n'ait lieu que lorsqu'elle est autorisée. Ce contrôle est fait au niveau de la commande bsubmit. Lors de la soumission, on associe une version de fichier à un "Defect" dans CMVC. La soumission (bsubmit) ne sera faite que si la "track" associée à ce Defect est dans l'état "fix".

7) Cycle de vie des défauts



#### *a) Fin de développement*

La fin de développement est appelée EOS (End of Submission). Quelques jours avant la date prévue pour le freeze, une "Demande d'agrément" est envoyée par le service SHCM au chef de projet et aux développeurs. Cette "Demande d'agrément" donne au chef de projet une vue sur le contenu de la version qui va être fabriquée ainsi que le travail restant à effectuer avant que la fabrication puisse commencer. Dès que la réponse à la demande d'agrément est positive et que toutes les tracks sont à l'état "integrate", on commence la fabrication de NetWall.

#### *b) Fabrication de NetWall*

NetWall est fabriqué par le service SHCM : c'est la phase de "freeze". Pendant cette phase les modifications ne sont plus autorisées. Si un problème survient pendant cette phase, SHCM crée un défaut dans CMVC. Ce défaut, qui a un cycle de vie très court, permet aux développeurs de fixer le problème rapidement, et la fabrication du freeze peut reprendre. Lorsque le freeze est terminé, le service SHCM émet un "EOF report" (End Of Freeze). Le freeze est alors disponible pour les tests.

### *1.3.2.2 Procédure de test*

L'entrée en vérification/validation est marquée par l'EOF (End Of Freeze).

L'activité de test est jalonnée par la création de "Defects" dans CMVC. La correction des "Defects" suit le cycle décrit précédemment, jusqu'à la fabrication d'un nouveau "freeze", et ce jusqu'au freeze final, qui marque le début de la livraison en clientèle de NetWall.

### *1.3.2.3 Procédure de maintenance*

#### *1) Initialisation de l'environnement de maintenance*

Cette opération est faite par le service SHCM.

#### *a) Initialisation de la base ODE*

L'espace mis à la disposition des développeurs s'appelle une "shared sandbox". Au début de la maintenance, la "shared sandbox" est vide et on a la visibilité sur tous les fichiers du "freeze". Au fur et à mesure de la maintenance, elle contiendra les fichiers modifiés. Les corrections sont délivrées dans ce que l'on appelle les "PTF"(Program Temporary Fix).

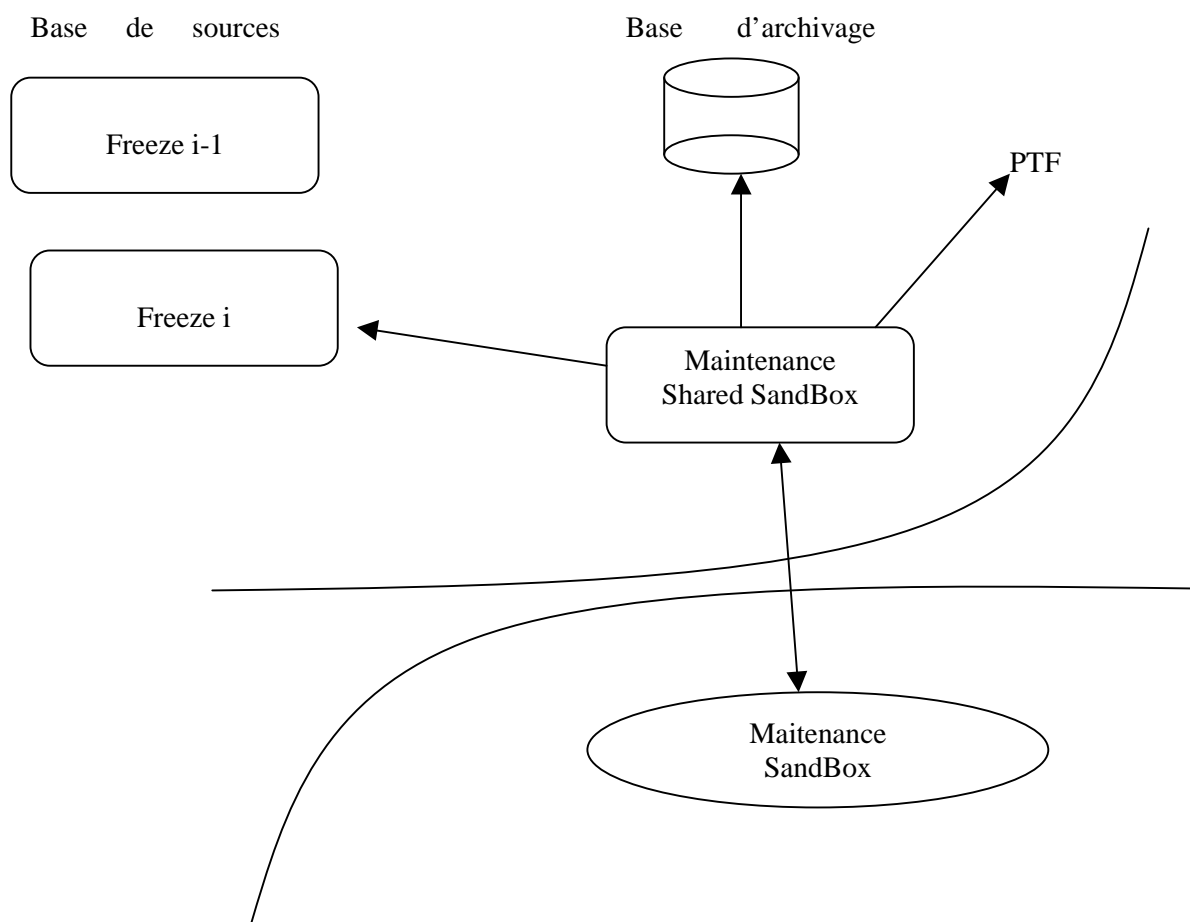
#### *b) Initialisation de la base CMVC*

Les défauts sont créés sur les mêmes composants que pour le développement.

Une "Release de PTFs " est créée pour les corrections.

## 2) Développement des corrections

Pour effectuer une correction, le développeur se connecte sur la machine développement. Il utilise la sandbox de maintenance, qui est "backée" sur la "shared sandbox" de maintenance, elle même "backée" sur le freeze pour lequel la correction doit être faite. Si la correction doit être reportée dans la version en cours de développement, le développeur utilise sa sandbox de développement et soumet sa modification dans le "latest".



### 3) Fabrication d'un PTF AIX

Un PTF est fabriqué dans l'environnement ODE, sous la responsabilité du développeur.

Un PTF est dit "incrémental" : il englobe tous les précédents.

Un PTF est constitué :

- d'un fichier contenant les binaires modifiés : c'est le fichier <numéro du PTF>.bff (par exemple Z000999.bff). L'outil "ptfpkg" affecte le numéro et fabrique le fichier.

- d'un fichier texte, contenant les informations relatives au PTF : c'est le fichier <numéro du PTF>.txt (par exemple: Z000999.txt). Ce fichier est fabriqué avec l'outil "mktxt".

Afin d'améliorer le repérage de ces PTFs, ils sont renommés en utilisant l'outil "ptf\_rename". La notation devient alors :

<lpp>.<vrnf>.<oldname>

où:

<lpp> est le nom du lpp modifié,

vrnf est la version telle qu'archivée dans ODE,

oldname est le nom donné par l'outil ptfpkg

Exemple :       Z002115.bff devient : NetWall.rte.3.3.0.5.Z002115.bff

                  Z002115.txt devient : NetWall.rte.3.3.0.5.Z002115.txt

### 1.3.3 Glossaire

ACL	Access Control List
Backing tree	Arbre de référence lisible depuis une sandbox (on dit qu'une sandbox est "backée" sur le latest ou sur une shared sandbox par exemple). Cela permet de voir des fichiers sans les privatiser.
CMVC	Configuration Management Version Control
EOS	End Of Submission of source files to start the building of a freeze
EVB	End of Validation Bulletin
Freeze	Image "gelée" d'un ensemble de fichiers sources extraits d'ODE
Latest	Arbre de référence contenant tous les fichiers sources en cours de développement soumis par les développeurs
ODE	OSF Development Environment - environment used to develop the source code; it allows the parallel development (merge feature), the compilation and the building of builds.
PDP	La "Production De Programme" contient tous les processeurs impliqués dans la fabrication d'un freeze
PTF	Correction sur un ou quelques fichiers envoyés en clientèle pendant la période de maintenance entre deux versions officielles.
PTF	Program Temporary Fix
RCS	Revision Control System - Unix source file manager used by ODE
Sandbox	Espace de travail privé à chaque développeur
Shared sandbox	Espace de travail partagé par une équipe. Son contenu est compilable et peut représenter une version de produit.
SHCM	Software Hardware Configuration Management

## **1.4 Formation pour le DRT**

### *1.4.1 Description des enseignements sélectionnés en 1997-1998*

#### *1.4.1.1 Systèmes Répartis (SR)*

DESS Génie Informatique

Responsable : S. Krakowiak

Volume horaire : 18 heures cours + 6 heures TP

Ce cours a pour but de présenter les concepts et méthodes qui servent de base à la conception, à la réalisation et à l'exploitation des systèmes informatiques répartis, en mettant l'accent sur les aspects traités au niveau du système d'exploitation et du logiciel de base.

#### *1.4.1.2 Option Systèmes Répartis (SR)*

DESS Génie Informatique

Responsable : S. Krakowiak

Volume horaire : 85 heures

L'option « Systèmes Répartis et Réseaux » du DESS Génie Informatique a pour objectif de former des ingénieurs compétents dans le domaine du logiciel de base et des applications pour les systèmes informatiques répartis.

Ce domaine connaît depuis plusieurs années un développement important, sous l'influence de deux facteurs :

Evolution technique. Amélioration constante du rapport coût-efficacité des ordinateurs, amélioration des performances des communications (réseaux à haut débit), développement de l'Internet et des réseaux d'entreprise (Intranets).

Nouvelles applications. L'accès généralisé aux réseaux favorise le développement de nouveaux secteurs d'applications. Ainsi, on observe un rapprochement entre les mondes du traitement de l'information, des télécommunications, et de la télévision. D'autres domaines en expansion sont ceux du travail coopératif et du commerce électronique.

Le domaine technique des systèmes répartis est caractérisé par plusieurs tendances :

- Emergence d'outils logiciels (middleware) pour faciliter le développement de nouvelles applications et l'intégration de l'existant.
- Emergence de nouveaux modèles de programmation et de structuration des applications (code mobile, agents)
- Importance des aspects liés à la tolérance aux fautes et à la haute disponibilité de l'information et des services.
- Importance des aspects liés à la sécurité des systèmes (confidentialité, authentification, protection d'accès).
- Importance de l'administration des systèmes et réseaux, notamment pour répondre aux impératifs de sécurité et d'évolutivité.

#### *1.4.1.3 Construction des Applications Reparties (CAR)*

DEA Informatique Systèmes et Communications

Responsable : Roland Balter

Volume horaire : 12 heures

Ce cours décrit les principes directeurs de la construction d'applications réparties. Il comprend deux parties :

- la première partie est consacrée aux outils logiciels disponibles pour le constructeur d'applications réparties. On décrit l'état de l'art industriel et quelques solutions avancées.
- la seconde partie présente les mécanismes sous-jacents utilisés par les systèmes pour la gestion de l'information répartie et la mise en œuvre des traitements répartis.

#### 1.4.1.4 *Serveur de Bases de Données et objets multimédia (SBD)*

DEA Informatique Systèmes et Communications

Responsable : M. Adiba

Volume horaire : 12 heures

Dans ce module, nous décrivons les aspects systèmes qui sont liés d'une part, à l'évolution des architectures des systèmes informatiques (par exemple le client/serveur), et, d'autre part, à l'évolution des modèles et systèmes de bases de données. Nous décrivons les architectures des SGBD et les techniques mises en œuvre pour créer, gérer et maintenir des bases de données réparties quelles soient relationnelles ou à objets. Un accent particulier est mis sur la notion de transaction et ses évolutions. Nous traitons également le problème de la modélisation, du stockage et de l'exploitation des objets multimédia (textes, Sons, Images, Vidéo) sous l'angle des bases de données.

#### 1.4.1.5 *Interconnexion de Systèmes (IS)*

Maîtrise MINF

Responsable : Roland Balter, J. Briat

Volume horaire : 18 heures cours + 18 heures TD

L'objectif de cette formation est de présenter les principes directeurs de la construction d'applications réparties selon le modèle client-serveur. Le cours est structuré en trois parties. La première partie est une introduction aux principes de fonctionnement des réseaux. La seconde partie présente le modèle client-serveur, les outils de construction d'application associés et le principe de réalisation à l'aide de l'appel de procédure à distance. La troisième partie décrit quelques services distribués essentiels tels que le service de désignation, le service de fichiers distribués, le service d'authentification, etc.

### 1.4.2 *Description des enseignements sélectionnés en 1998-1999*

#### 1.4.2.1 *Génie Logiciel (GL)*

DESS Génie Informatique

Responsable : F. Ouabdesselam, J. Parissis, J.M. Favre

Volume horaire : 156 heures

Objectifs :

Présenter les problèmes qui relèvent du Génie Logiciel et les diverses solutions envisagées pour toutes les phases du cycle de vie. L'enseignement vise en priorité à donner aux étudiants le réflexe d'une approche rigoureuse dans la recherche des solutions ; cette approche passe par l'emploi de moyens adaptés (méthodes, formalismes, outils). L'enseignement met également en évidence le fait

qu'une production semi-automatique de logiciel de qualité est possible sous certaines conditions et que la puissance des moyens disponibles varie selon les domaines d'application.

#### *1.4.2.2 Interface Homme-Machine (IHM)*

DESS Génie Informatique

Responsable : L. Nigay

Volume horaire : 30 heures

Objectifs :

Ce module présente les principes théoriques et pratiques nécessaires à la conception et à la réalisation des systèmes interactifs : la psychologie cognitive et le génie logiciel. Des études de cas sont présentées au moyen de techniques audiovisuelles.

#### *1.4.2.3 Multimédia*

DESS Génie Informatique

Responsable : G. Kuntz

Volume horaire : 21 heures

Objectifs :

Assimiler les spécificités des données multimédias et comprendre les problèmes soulevés par leur intégration au Web, concevoir une application multimédia en choisissant à partir du cahier des charges le système et le support le plus adéquat

#### *1.4.3 Formation suivie chez Bull*

1 semaine de formation sur le développement d'applications en Java.

2 jours de formation sur cmvc.