

# Clients hétérogènes, serveurs disparates: une approche utilisant perl v5

**Laurent Bardi et Emmanuel Courcelle**

Institut de Pharmacologie et de Biologie Structurale, UPR 9062, CNRS, 205 route de Narbonne,  
F-31077 Toulouse CEDEX, France.

*lolo@ipbs.fr* (L. Bardi) ou *manu@ipbs.fr* (E. Courcelle)

## Résumé

Nous avons développé un ensemble de scripts en perl v5, afin d'administrer efficacement le réseau local de l'IPBS : celui-ci est en effet constitué de serveurs Unix et NT d'une part, de clients Macs et PC d'autre part.

Les serveurs sont regroupés en « domaines », les domaines constituant l' « unité d'administration » (mêmes utilisateurs et mêmes groupes d'utilisateurs sur toutes les machines d'un même domaine). Nous pouvons ainsi ajouter ou retirer utilisateurs et groupes sur tous les serveurs d'un ou plusieurs domaines simultanément.

Pour ce qui est des clients, nous avons particulièrement travaillé les processus de connexion et de déconnexion des Macs et des PC, avec deux objectifs :

- Certains paramètres des applications sont configurés automatiquement lors de la connexion.
- Le menu (menu démarrer ou menu pomme) est régénéré lors de chaque connexion, en fonction des logiciels auxquels a accès l'utilisateur
- Le micro-ordinateur devient ainsi une « station de travail banalisée », puisque les logiciels se trouvent sur les serveurs, ainsi que les fichiers et la configuration de chaque utilisateur.

## 1. Introduction

L'IPBS dispose d'un réseau local fortement hétérogène, aussi bien du côté clients (Macs, PC) que serveurs (Unix et Windows NT). Afin de simplifier le travail des utilisateurs, nous avons intégré un maximum d'automatisation à la configuration du réseau. En particulier, les fonctionnalités suivantes sont présentes:

- Les applications les plus couramment utilisées, ainsi que les fichiers de données des utilisateurs, résident sur un ou plusieurs serveurs.
- Il existe des volumes de données partagés par les différents membres d'un même groupe, ceci afin de simplifier le travail en équipe. Il est possible de créer de nouveaux groupes, au fur et à mesure des besoins (collaborations entre équipes de recherche).
- Les répertoires d'accueil des principaux serveurs Unix sont vus par les micro-ordinateurs comme de nouveaux volumes, qui sont montés lors de la connexion au réseau. Cela simplifie l'échange de données entre machines UNIX et micros.
- Les micros sont considérés comme des « stations de travail banalisées »: un utilisateur peut se connecter indifféremment à partir d'un poste ou d'un autre, il retrouvera le même environnement de travail, pour ce qui est des fonctionnalités offertes par le réseau et des fichiers de données.
- Le changement de mot de passe se fait par l'intermédiaire d'un navigateur html, et l'utilisateur a la possibilité de fixer le même mot de passe sur les différents serveurs sur lesquels il a un compte.

Il est évident que l'administration d'un tel système ne va pas sans poser de nombreux problèmes, en particulier à cause du manque de compatibilité entre les 4 grandes familles de systèmes d'exploitation que nous avons à gérer : par exemple, il existe des outils de gestion globaux aussi bien sur NT que sur Unix, cependant les outils UNIX ne sont pas implantés (de manière native) sur NT, et réciproquement ; quand ils le sont, ils sont relativement onéreux.

Nous avons donc décidé de développer un ensemble de scripts nous permettant de mener à bien la plupart des tâches d'administration système. Nous avons utilisé à cet effet le langage perl (version 5) [perl v5] .

## 2. Portabilité et règles de développement

perl est un langage de script présent sur tous les environnements qui nous préoccupent actuellement [perlw32] [macperl] . Il est disponible sur internet [perlsrc] , soit en tant que produit gratuit, soit comme logiciel, et bien supporté, grâce à l'existence de listes de distribution sérieuses et dynamiques [listes] . Cependant, il est nécessaire de prendre en compte certaines règles de développement, pour arriver à écrire des scripts fonctionnant sur tous les environnements.

## 2.1 Ecriture modulaire

L'ensemble de nos développements est implémenté sous forme de packages `perl`. Les programmes principaux devront donc inclure (mot réservé du langage `require`) ces modules avant de pouvoir faire appel aux fonctions et aux variables exportées qu'ils contiennent.

## 2.2 Modules spécifiques à certains systèmes

Certains modules, particulièrement proches de la machine, sont conçus pour ne fonctionner que sur un système particulier. D'autres, au contraire, doivent pouvoir fonctionner sur tous les systèmes. D'autres pourront fonctionner sur deux systèmes seulement. Afin de repérer aisément ces spécificités, nous avons choisi pour chaque système une mnémonique particulière :

- `nt` pour symboliser Windows NT.
- `ux` pour symboliser Unix
- `wi` pour symboliser Windows 95
- `ma` pour symboliser Mac OS.

Ainsi, les noms de fichiers commenceront systématiquement par une indication d'architecture utilisant ces mnémoniques, permettant d'indiquer sur laquelle ou lesquelles de ces architectures le module en question fonctionne. Par exemple, le préfixe `nt_` indique un module ne fonctionnant que sous Windows NT, tandis que `ux_` indique un module ne fonctionnant que sous Unix, alors que `ntuxwima_` indique un module conçu pour fonctionner sur tous les environnements. Puisque `perl`, contrairement à des langages compilés tels que le C, permet d'inclure un module ou un autre en fonction d'une variable, nous utilisons une variable globale définissant le type générique d'architecture pour inclure le module dont nous avons besoin, suivant la machine sur laquelle le programme est exécuté.

## 2.3 Le package `ntuxwima_sys.pl`

Ce module implémente des sous-programmes permettant de lire le nom de machine sur lequel nous nous trouvons, ainsi que le nom de machine, etc. Cela, bien sûr, sans avoir d'idée a priori sur le type de système. Il affecte en particulier la variable `gtype`, qui donne le type générique de la machine (`nt`, `unix` etc). Ce module sera donc inclus par tous les exécutables.

## 2.4 Ecriture de primitives « standard »

Nous avons écrit des primitives standard (module `ux_lib.pl`, `nt_lib.pl` etc) pour effectuer certaines actions particulièrement utiles, comme la copie de fichiers ou d'arborescences, leur destruction, etc. En effet, nous nous interdisons de faire appel pour cela aux utilitaires du système, pour les raisons suivantes :

- Sur les différents systèmes, ces utilitaires ne portent pas le même nom et n'ont pas les mêmes conventions pour ce qui est du passage de paramètres.
- Il est parfois compliqué de savoir avec précision ce que fera la machine lorsqu'on appellera telle ou telle commande externe. Puisque nous avons réécrit des programmes de toute pièce, nous pouvons savoir précisément ce qui se passe, et surtout garantir qu'il se passera la même chose quelque soit la machine.

## 3. Format de fichiers

Nous avons conçu un format de fichier de données, appelé `xconf`, qui est utilisé pour tous les programmes sur tous les environnements, et pour tous les types de données . Cela nous a permis d'écrire une seule fois les routines de lecture et d'écriture associées à ce format. Il s'agit d'un package `perl`, appelé `ntuxwima_xconf`, et qui est utilisé en tant qu'objet.

### 3.1 Un fichier `ascii`

Les fichiers `xconf` sont des fichiers `ascii`, structurés d'une manière particulière. En effet, utiliser des fichiers `ascii` offre plusieurs avantages :

- Le débogage est plus simple.
- Il est toujours possible de lire les données, sans recourir à un programme particulier.
- Il est possible d'éditer le fichier pour le modifier, bien que cela soit déconseillé, sauf à des fins de débogage.

### 3.2 Un fichier structuré en sections

Un fichier `xconf` est composé de lignes, chaque ligne comptant un, deux ou trois champs. Les lignes blanches, ainsi que les lignes commençant par un signe `#` (commentaires) sont ignorées.

Un fichier contient des données concernant une série homogène d'objets : par exemple les données qui caractérisent les utilisateurs, ou les groupes d'utilisateurs, ou les machines constituant le réseau.

A chaque objet correspond une section, qui contiendra les données se référant à cet objet. Une section est identifiée par une première ligne donnant le nom de section (nom encadré par deux caractères `!`), et une dernière ligne constituée des caractères `!EOS!`. Le corps de la section est constitué des données se rapportant à cet objet. Chaque ligne de données comporte deux champs :

- Le premier champ est un identificateur
- Le second champ est la valeur prise par cet identificateur.

Les identificateurs, ainsi que les valeurs possibles pour chaque identificateur, sont spécifiés ainsi qu'on le verra ci-dessous.

### 3.3 Une section spéciale

Pour chaque fichier `xconf`, il existe une section spéciale, qui possède les caractéristiques suivantes :

- Son nom est `__DEF__` (« nom réservé »).
- Les lignes constituant cette section sont constituées de trois champs :
  - Un identificateur
  - La lettre `L` ou `S`
  - Une expression régulière, écrite selon la syntaxe reconnue par `perl`.

Le rôle de cette section, est de préciser les identificateurs pouvant apparaître dans les fichiers `xconf`, ainsi que les valeurs pouvant leur être affectées ; en effet :

- seuls les identificateurs de cette section peuvent apparaître dans les autres sections du fichier.
- Ces identificateurs peuvent être soit des scalaires, (second champ : `S`), soit des listes (second champ : `L`).
- Les valeurs affectées à ces identificateurs, ou dans le cas d'une liste chaque item de la liste, doit répondre au modèle donné par l'expression régulière du troisième champ.

Ce système nous permet d'assurer une grande robustesse au système, dans la mesure où le fichier contient lui-même ses règles de cohérence.

### 3.4 Un ou deux fichiers ?

`ntuxwima_xconf` prévoit la possibilité de lire en fait deux fichiers différents : le premier contenant la section `__DEF__`, le second les autres sections. Cela donne une grande souplesse, dans la mesure où la section `__DEF__` est de nature différente des autres sections : les données qu'elle contient sont définies par le programmeur, tandis que les autres sections peuvent contenir de « vraies » données (résultats des exécutable).

### 3.5 Vérification de cohérence

Pour des raisons de performance, l'objet `ntuxwima_xconf` ne vérifie pas normalement la cohérence du fichier lors de sa lecture. Par contre, les primitives d'écriture de section feront ces vérifications avant toute modification du fichier. On est ainsi assuré qu'un fichier `xconf` restera parfaitement cohérent, à moins bien sûr qu'il n'ait été édité manuellement.

Cependant, nous avons également une option du constructeur de `ntuxwima_xconf` permettant de vérifier la cohérence des fichiers existants.

### 3.6 Les primitives de lecture et d'écriture

Les sous-programmes `r_section` et `w_sections` permettent respectivement de lire une section, et d'écrire une ou plusieurs sections simultanément. Dans chaque cas, la représentation en mémoire de la section est un tableau associatif dont les clés sont les identificateurs. Ces primitives sont utilisées dans les cas suivants :

- Écriture d'une ou plusieurs nouvelles sections : on commence par générer le ou les tableaux associatifs, puis on appelle `w_sections` pour écrire le fichier avec les nouvelles sections.
- Modification d'une section : on récupère le tableau associatif correspondant à la section à modifier par un appel à `r_sections`, on effectue les modifications voulues, puis on écrit le tableau par `w_sections`.

### 3.7 Considérations de performance

#### 3.7.1 Lecture

Il y a une différence importante entre les sous-programmes `r_section` et `w_sections` : en effet, le fichier n'est réellement lu qu'au moment de la création de l'objet. La primitive `r_sections` ne fait en fin de compte que de renvoyer une référence vers un tableau associatif existant déjà en mémoire. Cette manière de procéder permet de simplifier le code, mais l'inconvénient est clairement un risque de gaspillage de mémoire, puisque l'intégralité du fichier est lue et stockée en mémoire, alors qu'une toute petite partie sera éventuellement utilisée (fichier `u.conf`, par exemple, qui contient les données pour tous les utilisateurs du système, alors qu'un seul utilisateur est concerné par le travail en cours).

Pour pallier cet inconvénient, il est possible de lire seulement une partie du fichier : soit certaines sections, soit certains identificateurs, soit les deux. Dans le premier cas, les sections explicitement spécifiées seront lues entièrement, et dans le second les identificateurs spécifiés seront lus dans toutes les sections où ils apparaissent.

#### 3.7.2 Ecriture

Par contre, le fichier est effectivement écrit lors de l'appel à `w_sections`. Chaque ligne est en réalité relue, et immédiatement réécrite, après modification si nécessaire. Enfin, les nouvelles sections sont ajoutées en fin de fichier. Ce processus est nécessaire pour deux raisons :

- Il permet de conserver dans le nouveau fichier les lignes de commentaires qui sont ignorées lors de la lecture par le constructeur
- Il permet de conserver les données qui n'auraient pas été lues, dans le cas où seuls certaines sections ou certains identificateurs sont lus lors de la construction.

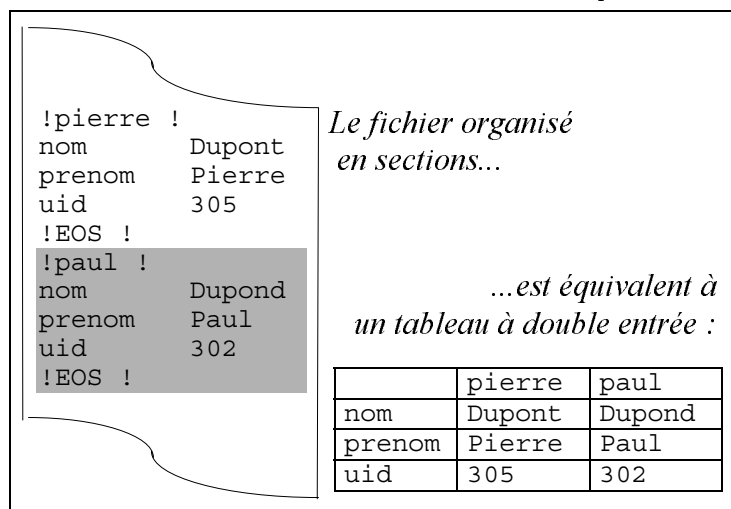
Pour garder des performances acceptables, on passe en fait une liste de noms de sections et de tableaux associatifs associés à `w_sections`, de sorte que les différentes sections sont toutes écrites en même temps.

#### 3.7.3 Un tableau à double entrée

Ainsi que le montre la figure 1, un fichier `xconf` peut être vu comme un tableau à double entrée, dans lequel chaque colonne correspond à une section, chaque ligne à un identificateur. Cette représentation conduit d'ailleurs à une utilisation des données parfaitement naturelle ; d'où la présence de la primitive : `relation_directe`.

Celle-ci permet, étant donné un nom de section et un nom d'identificateur, de lire la valeur de cet identificateur dans cette section.

Une autre primitive, `relation_inverse`, permet, quant à elle, de connaître toutes les sections dans lesquelles un identificateur donné prend une valeur donnée.



#### 3.7.4 Exemples

L'appel `relation_directe(domaine, machine)` renvoie le nom du domaine dont fait partie la machine. Inversement, l'appel `relation_inverse(domaine, val)` renvoie la liste des sections dans lesquelles l'identificateur `domaine` prend la valeur `val`. C'est-à-dire la liste des machines constituant le domaine `val`. D'ailleurs, cette fonction renvoie effectivement une liste, au sens `perl` du terme (c'est-à-dire un tableau).

## 4. La gestion des serveurs

### 4.1 Quelques notions de base

Avant de détailler le processus de gestion des comptes utilisateurs, il convient de préciser certaines notions de base. C'est-à-dire expliquer précisément le sens que nous donnons, dans le cadre de notre réseau, à des concepts comme : machine maître, domaine, groupe d'utilisateurs, utilisateur.

#### 4.1.1 *Le maître des serveurs*

Ainsi qu'on le verra ci-dessous, toutes les commandes visant à modifier un compte utilisateur ou un groupe sont orchestrées à partir d'une machine unique. Cette machine est appelée maître des serveurs (pour la distinguer du maître des micros dont le rôle est explicité ci-dessous).

On peut bien sûr discuter la pertinence de cette notion : en particulier une caractéristique de notre système d'administration réside dans l'existence d'un maître unique ; il serait probablement meilleur, du point de vue de la sécurité comme de l'efficacité du système, de disposer de maîtres secondaires. Cependant, il nous a semblé préférable d'adopter un point de vue plus pragmatique : il est bien plus simple, du point de vue de la programmation, de disposer d'un maître unique. Si l'efficacité peut s'en trouver amoindrie, il convient de noter que cela ne porte pas vraiment à conséquence tant que le réseau est de taille relativement moyenne (une vingtaine de serveurs environ). Enfin, le découpage en domaines, ainsi qu'il est explicité ci-dessous, nous permet de rationaliser le processus, et finalement d'arriver à une performance satisfaisante de l'ensemble. Enfin, le maître peut être n'importe quel serveur, soit `unix` soit `nt` pourvu qu'il dispose des deux fichiers qui la caractérisent en tant que maître des serveurs : il est donc toujours possible, en cas de panne, de remplacer momentanément le maître des serveurs par une autre machine.

#### 4.1.2 *Domaines*

On peut considérer le domaine comme l'unité d'administration. Cela signifie qu'il est impossible, pour un utilisateur, d'avoir un compte sur une machine isolée d'un domaine. Les comptes seront ouverts ou fermés simultanément sur toutes les machines du domaine. Les machines seront donc groupées en domaines suivant leurs principales fonctionnalités : par exemple, les machines appartenant à une équipe de recherche déterminée constitueront un domaine.

##### 4.1.2.1 *Relation avec les domaines NT et les zones Appletalk.*

Windows NT connaît lui aussi la notion de domaines, tandis que le protocole Appletalk définit les zones. La notion de domaine NT est basée sur l'acquiescement des utilisateurs par le serveur (Contrôleur Principal de Domaine) ; la notion de zone AppleTalk agit quant à elle uniquement au niveau de chaque machine (elle est similaire à un groupe de travail (workgroup) sous Windows). Le domaine au sens où nous l'entendons ici est un compromis entre ces deux fonctionnalités ; en effet, un domaine est constitué de machines (serveurs), mais sur un domaine sont déclarés groupes et utilisateurs. Toutes ces notions sont donc suffisamment proches pour que, sans pour autant les confondre, nous installions une coïncidence entre elles : autrement dit, les domaines, les domaines NT et les zones Appletalk que l'on trouvera sur notre réseau porteront les mêmes noms, et seront constitués des mêmes machines.

#### 4.1.3 *Utilisateurs et groupes*

Les utilisateurs ont un nom d'utilisateur, un `user-id` et un `group-id` (ces derniers ne sont utilisés que par `unix`).

##### 4.1.3.1 *Uid et gid :*

Chaque utilisateur a un `user-id` et un `group-id`. Ces données sont indispensables pour `unix` ; surtout, parce que nous désirons partager des répertoires par `nfs` entre plusieurs machines, il est indispensable que chaque utilisateur ait le même `uid` et le même `gid` sur toutes les machines `unix` du réseau. Nos scripts d'administration garantissent ce fait, ils respectent également la politique suivante :

- Tous les `gid` des utilisateurs sont des multiples de 100
- Les `uid` des utilisateurs d'un groupe de `login` donné sont compris entre le `gid` correspondant et le `gid + 100`. Cela simplifie grandement la gestion des comptes.

##### 4.1.3.2 *Les groupes d'utilisateurs :*

Nous distinguons trois sortes de groupes :

- Les groupes de `login`
- Les groupes supplémentaires
- Les groupes Extra.

La différence entre ces trois sortes de groupes est la suivante : chaque utilisateur appartient à un et un seul groupe de `login`. Chaque utilisateur peut appartenir à un ou plusieurs groupes supplémentaires. Les groupes supplémentaires seront également déclarés au niveau du système, soit dans la base de données des utilisateurs

pour Windows NT, soit dans le fichier `/etc/group` pour Unix. Chaque utilisateur peut par ailleurs appartenir à un ou plusieurs groupes extras, cependant les groupes extras ne sont pas connus du système ; Une application des groupes extra pourrait être, par exemple, l'écriture d'un script de génération automatique d'une page d'annuaires pour le serveur www, qui éviterait de faire figurer les utilisateurs appartenant au groupe « pas\_annuaire ». Cela permet de générer aisément une telle page, tout en respectant les recommandations de la CNIL, qui demande à ce que chacun puisse refuser de figurer sur l'annuaire.

#### 4.1.3.3 Répertoires de groupes et partages :

A chaque groupe de `login`, ou à chaque groupe supplémentaire, peut être associé un répertoire de ce groupe. Par ailleurs, ce répertoire peut être partagé, afin d'être vu par les micro ordinateurs. Cela nous permettra, par exemple, de regrouper les utilisateurs travaillant sur un même sujet de recherche dans un même groupe, la présence d'un répertoire associé à ce groupe, ainsi que le fait que ce répertoire soit partagé, permettra des échanges de fichiers simplement.

#### 4.1.3.4 Utilisateurs et groupes déclarés sur le réseau ou sur un domaine.

Les utilisateurs et les groupes sont déclarés sur le réseau tout entier. Cela nous permet en particulier de nous assurer de ce que chacun a un unique `uid` et `gid` sur toutes les machines Unix du réseau ; tous les montages `nfs` seront donc possibles. D'autre part, toutes les machines gardent dans les fichiers systèmes des données sur les utilisateurs, comme le nom, le prénom, etc. Nos scripts d'administration permettent de s'assurer que toutes ces données sont les mêmes quelle que soit la machine du réseau.

Cependant, cela ne signifie nullement que les utilisateurs ont un compte sur toutes les machines du réseau : en effet, il est possible d'ouvrir un compte à un utilisateur sur seulement un ou plusieurs domaines. Simplement, le fait qu'il soit déclaré au niveau du réseau implique que si, ultérieurement, on ouvre un compte à cet utilisateur sur un nouveau domaine, les mêmes paramètres seront automatiquement repris pour cet utilisateur (`uid`, `gid`, nom, prénom, etc).

De même, un groupe peut fort bien n'exister que sur un domaine. Mais si, par la suite, on étend le groupe à un autre domaine, il aura automatiquement les mêmes paramètres sur tous les domaines ; en particulier, sa composition sera la même, avec bien sûr la restriction suivante : tous les utilisateurs de ce groupe sur ce domaine doivent également être déclarés sur le domaine. Par contre, sur toutes les machines d'un même domaine, la base de données des groupes et des utilisateurs est la même<sup>1</sup>.

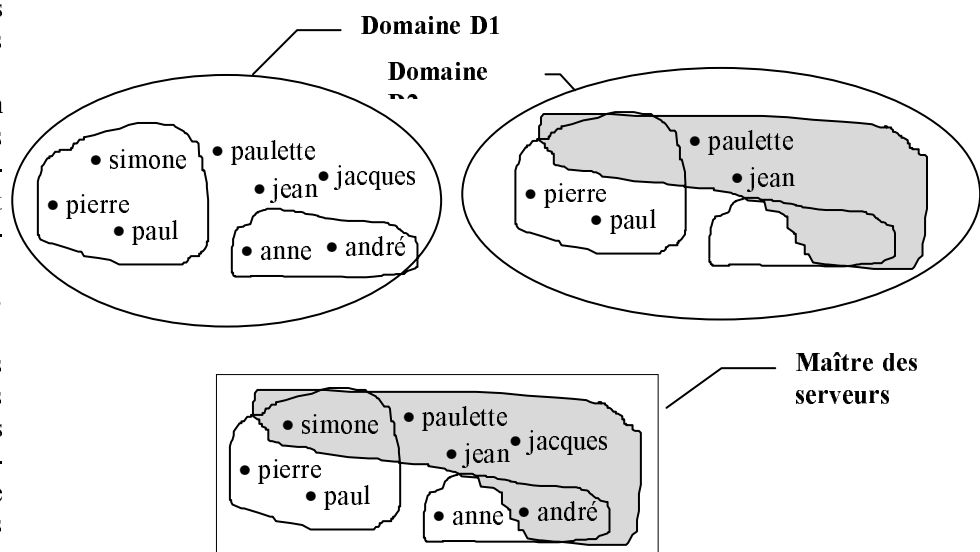


Figure 2. Tous les groupes et les utilisateurs sont déclarés sur le Maître des Serveurs. Tous les utilisateurs ont un compte sur le domaine D1, mais seuls deux groupes sont déclarés sur ce domaine. Tous les groupes sont déclarés sur le domaine D2, mais seuls 4 utilisateurs sont déclarés sur ce domaine. La composition des groupes est donc bien « la même » sur tous les domaines.

<sup>1</sup> Cela exclut cependant, bien entendu, les groupes et les utilisateurs propres au système d'exploitation (`root`, `lp`, par exemple), qui ne sont probablement pas les mêmes suivant le constructeur de la machine. Ces groupes et ces utilisateurs ont généralement un identificateur inférieur à 100, qui par principe sera ignoré par nos scripts.

#### 4.1.4 Profils

Toutes les opérations d'administration, en particulier l'ajout de nouveaux utilisateurs, ou de nouveaux groupes d'utilisateurs, ont besoin de connaître un nombre important de paramètres : par exemple, le shell par défaut, ou encore le système de fichiers sur lequel se trouve le répertoire d'accueil, voire même le nom de ce répertoire ; pour la création de groupes, il est nécessaire dans certains cas d'associer un répertoire à un nouveau groupe, mais cela est également parfois superflu. De plus, il n'est pas toujours nécessaire de partager le répertoire associé au groupe. D'autres paramètres peuvent être assimilés à des drapeaux permettant d'influer sur le fonctionnement des routines d'administration elles-mêmes. Nous nous trouvons en définitive devant plusieurs possibilités :

- Il aurait été possible de coder directement ces valeurs de paramètres au sein des scripts d'administration. Cependant, il est évident que cette solution avait l'énorme inconvénient de figer une fois pour toutes des données qui, somme toute, sont liées à une configuration donnée, et peuvent évoluer au cours du temps.
- Une autre solution était de demander à l'administrateur de fournir l'ensemble de ces paramètres à chaque exécution des routines. Il est clair que cela aurait vite conduit à une utilisation extrêmement fastidieuse des outils d'administration, étant donné le nombre important de paramètres à entrer.

Nous avons préféré adopter une solution intermédiaire entre ces deux extrêmes : un des paramètres que devra entrer l'administrateur est une chaîne de caractères, appelée profil. Les routines d'administration pourront alors sélectionner un jeu de paramètres prédéterminé, correspondant à cette chaîne de caractères. Les différents profils possibles sont décrits dans un fichier au format `xconf`, de sorte qu'une modification de configuration du réseau, qui entraînerait une modification des paramètres utilisés par les routines d'administrations, se traduirait uniquement par une modification du fichier de profil .

##### 4.1.4.1 Un profil par domaine

Puisque le domaine est l'unité d'administration, il est cohérent de considérer que les profils sont définis pour chaque domaine, de sorte qu'il y a un et un seul fichier de profil par domaine. Par ailleurs, les profils sont définis indépendamment pour les différentes primitives d'administration, ainsi qu'on le voit sur l'exemple ci-dessous :

##### 4.1.4.2 Un profil par primitive d'administration

sur le domaine `D1`, les profils suivants peuvent être spécifiés lors de l'ajout d'un groupe d'utilisateurs :

- `rg0`, `rg1`, `rgp`

`rg0` signifie qu'il n'y a aucun répertoire de groupe associé à ce groupe, `rg1` signifie qu'il y a un répertoire de groupe associé (et les paramètres permettant de déterminer le nom du répertoire seront dans le fichier de profil), `rgp` signifie que ce répertoire est partagé pour les PC (là encore, les paramètres correspondants se trouvent dans le fichier de profil).

Les profils suivants peuvent être spécifiés lors de la suppression d'un groupe :

- `dfr`, `000`

`dfr` signifie que le répertoire associé à ce groupe sera supprimé également, `000` signifie que l'on ne touchera pas à ce répertoire.

Une section du fichier de profils est ainsi associée à chaque primitive d'administration. Suivant les primitives, il est évident que l'importance des paramètres spécifiés dans cette section sera plus ou moins grande. Cependant, un paramètre qui peut toujours être mis dans le fichier de profils est l'existence de « scripts externes » : il s'agit de définir un ou plusieurs scripts, qui seront appelés par le système lors de l'exécution de la primitive. Cela permet par exemple de configurer un logiciel particulier sur une machine donnée, et ce quelque soit la primitive considérée.

##### 4.1.4.3 Validation du profil

Le fichier de profil contient lui-même l'information permettant aux routines d'administration de vérifier la validité du profil entré par l'utilisateur : en effet, il existe un identificateur, appelé profil, dont la valeur est une expression régulière. Le profil passé aux routines sera comparé à cette expression régulière, le paramètre sera refusé si la comparaison se révèle négative.

##### 4.1.4.4 Syntaxe

La syntaxe pour passer aux routines d'administration à la fois l'indication de profil et de domaines est la suivante :

```
domaine=profil
```

Ainsi, dans l'exemple ci-dessous, un paramètre légal sera `D1=rgp`

## 4.2 Les protocoles de communication

Pour l'instant, nous utilisons les protocoles de communication suivants :

- `rcp` ou `ftp` pour transférer des fichiers entre serveurs
- `rsh` pour synchroniser l'exécution de programmes entre serveurs

Nous avons cependant conscience que cela n'est pas parfaitement satisfaisant, au moins du point de vue de la sécurité : nous envisageons d'écrire des protocoles de communication reposant sur les sockets [`rifflet`], afin de pallier cet inconvénient. Les appels aux routines de communications sont encapsulés dans des sous-programmes de transfert de fichiers, ce qui rendra le passage d'un protocole à l'autre relativement aisé.

## 4.3 Les primitives de gestion d'utilisateurs

Un dialogue s'établit entre le maître des serveurs et les serveurs des domaines concernés. Ce dialogue est détaillé ci-dessous :

Un programme est exécuté sur le maître des serveurs : appelé `mrap`, son rôle est de valider les paramètres passés par la ligne de commande, puis de contacter l'un après l'autre tous les serveurs concernés, et de provoquer, sur chacun d'entre eux, l'exécution d'un autre programme, `rap`.

A chaque fois qu'un tel dialogue s'établit, on a une « requête » : un numéro de requête est attribué, ce qui permettra de garder une trace ultérieurement, afin de ne perdre aucune donnée si un incident se produit ; l'incident le plus probable étant tout simplement le fait qu'un serveur soit arrêté, et n'ait donc pas reçu les données.

### 4.3.1 Les fichiers de données :

6 sortes de fichiers, tous au format `xconf`, sont impliqués dans ce mécanisme :

- `U.conf`, `G.conf` : Ces fichiers sont maintenus par le maître des serveurs, ils centralisent toute l'information concernant groupes et utilisateurs, pour l'ensemble du réseau. En particulier, pour chaque utilisateur ou chaque groupe, on connaît ainsi les domaines sur lesquels ils sont déclarés, ainsi que les profils utilisés lors de la création.
- `u.conf`, `g.conf` : Ces deux fichiers sont maintenus par chaque serveur, ils contiennent l'information concernant utilisateurs et groupes enregistrés sur cette machine. En ce sens, ils sont redondants par rapport aux fichiers systèmes de la machine. Le fait que leur format, `xconf`, soit unique quelque soit le système (Unix ou NT), et qu'ils contiennent des données supplémentaires par rapport aux fichiers système rend toutefois leur existence indispensable.
- `u.conf.xxx`, `g.conf.xxx`, où `xxx` représente un numéro de requête. Ces fichiers contiennent un extrait de `U.conf` ou `G.conf` (une ou plusieurs sections, éventuellement incomplètes), ils sont transmis par le maître lors de chaque requête

### 4.3.2 Les requêtes :

A chaque appel du programme `mrap`, et à condition que les paramètres aient été validés, et qu'un dialogue doive s'établir avec d'autres machines, un numéro de requêtes sera calculé et associé à cet appel. La séquence est donc la suivante :

1. analyse des paramètres spécifiés par la ligne de commande, validation du profil, etc.
2. Modification des fichiers `U.conf` ou `G.conf`
3. Lecture du fichier `requetes`, incrémentation du numéro de requête qu'il contient
4. Ecriture de `u.conf.xxx` ou `g.conf.xxx`
5. Détermination des machines à contacter (les serveurs qui se trouvent dans les domaines spécifiés)
6. Ecriture du fichier `requete.xxx` : ce fichier contient le nom de la primitive exécutée, ainsi que la liste des machines concernées par cette requête.
7. Détermination des machines opérationnelles (par un ping).
8. Sur chaque machine opérationnelle, envoi du fichier `u.conf.xxx` ou `g.conf.xxx` (éventuellement les deux) et exécution du programme `rap`.

Le programme `rap` sera exécuté sur chaque serveur concerné par une requête donnée. Son rôle est de lire le fichier transmis (`u.conf.xxx` ou `g.conf.xxx`), de modifier en conséquence le fichier local correspondant

(`u.conf`, `g.conf`), et de faire les actions qui se révéleront nécessaires pour exécuter la requête : écriture dans les fichiers systèmes, création de répertoires d'accueil, etc.

Ensuite, un fichier, appelé `machine.xxx`, où `machine` est le nom du serveur, sera créé et envoyé sur le maître. Ce fichier contient simplement une ligne spécifiant la date et l'heure d'exécution de la requête.

#### 4.4 Nettoyage ou archivage

Etant donné le protocole décrit ci-dessus, le maître sait à chaque instant où en sont les choses : si une requête a été générée et correctement exécutée sur chaque serveur concerné, on trouvera les fichiers `u.conf.xxx` ou `g.conf.xxx`, le fichier `requete.xxx`, et un fichier `machine.xxx` par serveur concerné. `Requete.xxx` contenant la liste des serveurs concernée, il est possible de savoir si la requête a été exécutée partout, ou s'il y a des « retardataires » (`machine` en panne lors de l'exécution de `rap`, par exemple). Dans le premier cas, on peut soit supprimer les fichiers correspondants à cette requête, soit les archiver en les déplaçant dans un autre répertoire. C'est cette dernière solution que nous avons adoptée, la commande `crap` se chargeant de cette tâche.

#### 4.5 La mise en place du système

Le système que nous venons de décrire permet de gérer simplement un réseau complet, comprenant de nombreux serveurs `unix` ou `nt`, à condition que le système soit en phase de production.

Cependant, lors de la mise en place du système, il a été nécessaire de générer artificiellement les fichiers `U.conf` et `G.conf` : en effet, nous n'avons pas attendu d'écrire ces routines d'administration pour mettre en service nombre de machines `unix` ou `nt` ; nous devons donc partir d'une base installée relativement importante. Il faudra également générer des fichiers `g.conf` et `u.conf` sur chaque serveur du réseau. Indépendamment des procédures de démarrage, cette dernière tâche sera très utile lors de l'installation dans le réseau d'une nouvelle machine.

##### 4.5.1 Génération des fichiers `U.conf` et `G.conf`

La génération des fichiers `U.conf` et `G.conf` n'est pas entièrement automatisée, essentiellement parce qu'il est très difficile de remonter de la configuration existant aujourd'hui pour tel utilisateur ou tel groupe jusqu'au profil correspondant : d'une part, cela n'aurait pas pu être généralisé, car les possibilités sont infinies en termes de définitions de profils ; d'autre part, certaines définitions de profils n'ont pas vraiment de signification à l'échelle d'une machine, mais bien plutôt à l'échelle de tout le domaine ; nous nous contentons donc de générer `G.conf` à l'aide l'utilitaire `group2G`, de générer `U.conf` à l'aide de `passwd2U`, en affectant un profil unique par défaut à chaque groupe ou utilisateur, puis d'éditer les fichiers ainsi générés pour modifier les profils enregistrés de certains groupes ou utilisateurs.

##### 4.5.2 Génération des fichiers `u.conf` et `g.conf` et modification des fichiers systèmes

Par contre, la génération des fichiers `u.conf` et `g.conf`, à partir de `U.conf` et `G.conf`, peut être entièrement automatisée, puisqu'il s'agit en fin de compte d'exécuter en une seule fois des requêtes qui normalement sont distribuées au cours du temps. Les utilitaires `G2g` et `U2u` se chargent ainsi de générer plusieurs fichiers de requêtes, l'utilitaire `rap` sera alors appelé pour exécuter les requêtes et créer groupes et utilisateurs.

##### 4.5.3 Mise en place

Ainsi, la mise en place du système consiste donc dans les étapes suivantes :

1. Collecte de l'information :
  - a) Exécuter `group2G` et `passwd2U` sur chaque serveur d'un domaine donné, en transférant les fichiers obtenus d'une machine sur l'autre, afin que les résultats obtenus soient cumulatifs.
  - b) Editer le fichier, pour modifier éventuellement les indications de profils.
  - c) Recommencer les opérations a). et b) sur tous les domaines, en utilisant toujours le même fichier `G.conf` et `U.conf`,
2. Reconstitution des fichiers de configuration individuels :
  - a) Transférer `U.conf` et `G.conf` sur un serveur
  - b) Exécuter `U2u` et `G2g` afin de générer les requêtes pour `rap`
  - c) Exécuter `rap` afin d'exécuter les requêtes créées en b), et ainsi générer `u.conf` ou `g.conf`, et faire les actions nécessaires.
  - d) Recommencer a) b) c) sur tous les serveurs du réseau.

#### 4.6 Distribution de fichiers systèmes(Unix)

Une partie importante du travail de l'administrateur système consiste à maintenir plusieurs fichiers permettant de configurer les principaux sous-systèmes. Cette tâche est compliquée par le fait que ces fichiers résident sur toutes les machines du réseau, avec éventuellement un contenu différent sur chacune d'entre elles, mais l'ensemble correspond bien à une configuration globale du réseau : un exemple typique de cela est le système nfs, en particulier lorsque des montages croisés sont réalisés entre plusieurs stations de travail. Nous avons résolu ce problème de la manière suivante : chaque fichier est maintenu de manière centralisée sur le maître des serveurs. Nous avons ajouté à la syntaxe normale de chaque fichier des directives permettant de déclarer sur quelles machines certaines portions de fichiers seront envoyées. Les directives occupant une ligne, il est à la limite possible de déclarer une machine de destination différente pour chaque ligne du fichier.

Peuvent apparaître dans les directives aussi bien des noms de machines que des noms de domaines, ou encore des noms de types génériques de machines ou de systèmes d'exploitation, ainsi que la combinaison des deux : nous pouvons ainsi écrire que telle portion d'un fichier système devra être écrite sur toutes les machines Silicon Graphics du domaine D1. Cela permet à l'administrateur d'avoir une vue globale des fichiers systèmes des machines unix dont il s'occupe.

### 5. La gestion des clients

Notre objectif, vis-à-vis des clients, est que chaque micro ordinateur joue le rôle de station de travail banalisée ; les utilisateurs doivent pouvoir passer d'un micro à l'autre, que ce soit un macintosh ou une machine Windows, tout en retrouvant leur environnement : programmes les plus utilisés, notamment Bureautique, bureau, fichiers utilisateurs, etc. D'autre part, nous souhaitons assurer que certains paramètres de configuration des logiciels soient automatiquement sélectionnés, sans donner à l'utilisateur la possibilité de les changer... ou à défaut, en les remettant en place lors de la connexion suivante : par exemple, l'adresse de retour pour le logiciel Eudora, le nom de machine jouant le rôle de cache pour netscape, etc. En effet, la mauvaise configuration de ces paramètres a une influence non seulement sur l'utilisateur concerné, mais aussi sur l'ensemble du réseau. C'est donc de la responsabilité de l'administrateur système de positionner ces paramètres.

Nous avons donc défini un processus de connexion, ayant pour objet de positionner ces différents paramètres au moment de la connexion. Ce processus s'appuie dans le cas des machines Windows sur le protocole issu de Lan Manager, dans le cas des Macs il le simule.

#### 5.1 Le Maître des Micros

Le Maître des Micros est un serveur Windows NT qui joue un rôle particulier dans le réseau :

- Le bureau des machines sous Windows 95 est déporté sur ce serveur
- Il a en charge le processus de connexion (ressource netlogon).
- Le répertoire partagé sous le nom de netlogon contient le script netlogon.bat, mais également les programmes écrits en perl qui seront utilisés par les micros lors de la connexion : puisque ceux-ci sont installés sur le serveur, et non pas sur chaque client, tout processus de mise à jour ou de correction de bogue sera extrêmement simplifié.

Le Maître des Micros est obligatoirement le serveur primaire du domaine NT par défaut. Il est possible, afin de distribuer la charge, de disposer de plusieurs Maîtres des Micros : cela suppose que tous les micros n'appartiennent pas au même domaine par défaut. En effet, il s'agit là d'un paramètre statique, qui doit être choisi lors de l'installation de la machine.

#### 5.2 Les fichiers utilisés :

Nous utilisons à nouveau plusieurs fichiers xconf lors du processus de login et logout :

- u.conf (fichier des utilisateurs déclarés sur cette machine) .
- S.conf
- wi\_os.conf, ma\_os.conf, wi\_1.cons et ma\_1.conf

##### 5.2.1 Le fichier S.conf

Ce fichier définit l'ensemble des partages utilisés sur le réseau, et pour chacun d'entre eux donne les informations permettant de les utiliser : en particulier le nom de la machine concernée, le nom du partage, la lettre de lecteur utilisée pour les machines sous Windows, ainsi qu'une liste de clients autorisés à se connecter ; ceci afin de tenir compte des situations où un logiciel n'est accessible qu'à partir d'un nombre fixe de machines, à cause d'un problème de licences (licences fixes et non flottantes).

### 5.2.2 Les fichiers *wi\_os.conf*, *ma\_os.conf*, *wi\_l.conf*, *ma\_l.conf*

Dans chaque partage qui sera monté lors de la connexion, on trouve un répertoire particulier, (de nom *h1b\_config*), qui contiendra entre autres les fichiers : *wi\_l.conf* et *ma\_l.conf*. Ces fichiers décrivent la configuration des logiciels associés à ce partage, ainsi que nous le décrivons plus loin. Lorsque le partage est un répertoire d'accueil, puisque l'utilisateur a le droit d'écrire dans ces fichiers, nous y mettons des paramètres initialisés lors du *logon*, mais qui peuvent être modifiés par l'utilisateur. Enfin, les fichiers *wi\_os.conf* et *ma\_os.conf* se trouvent sur la ressource *netlogon*, ils ont pour but de configurer les systèmes d'exploitation eux-mêmes.

## 5.3 Comment configurer les logiciels et le système ?

Le processus de configuration des logiciels est légèrement différent pour des machines sous Windows et pour les Macintoshes, bien que les idées générales soient les mêmes :

### 5.3.1 Les machines Windows

Les logiciels Windows se configurent essentiellement de deux manières, suivant qu'ils ont été développés de manière native pour Windows 95, ou qu'il s'agit de logiciels écrits pour Windows 3. Certains logiciels, résultat de portages encore incomplets, combinent les deux méthodes.

#### 5.3.1.1 Logiciels Windows 3

Les anciennes versions de Windows utilisaient des fichiers *.ini* pour définir les différents paramètres. La configuration de ces logiciels nécessite donc d'identifier la section du fichier *.INI* dans laquelle se trouvent les paramètres à modifier, les identificateurs à utiliser, et les valeurs que nous désirons affecter à ces identificateurs.<sup>2</sup> Il n'y a alors plus qu'à écrire ou modifier le fichier *ini*.

#### 5.3.1.2 Logiciels Windows 95

Cette nouvelle génération de logiciels utilise la base de registres de Windows 95 afin de stocker les valeurs des différents paramètres. Or, la version Win32 de *perl* fournit des primitives capables d'écrire dans la base de registres.

#### 5.3.1.3 La configuration des logiciels

Il est alors simple de configurer un logiciel sous Windows [*mskb*] : il faut soit générer le fichier *.ini*, soit écrire des couples (clés, valeurs) dans la base de registres. Les données permettant ces opérations se trouvent stockées dans un fichier appelé *l.conf* sous forme de plusieurs listes : une liste pour les clés, une pour les valeurs associées. Par ailleurs, il est possible de prévoir l'appel de scripts externes, afin de tenir compte des logiciels qui, par malchance, ne pourraient être configurés entièrement de cette manière.

### 5.3.2 Les macintosh.

D'après la documentation "Inside Macintosh", La méthode « orthodoxe » pour configurer les logiciels Mac consiste à générer des fichiers situés dans le dossier "préférences", du dossier système, et écrire dans la partie ressource [*inmac*]. Il existe plusieurs types de ressources, cependant nous ne configurons directement que les ressources de types *STR* et *STR#*. Dans les autres cas, nous utilisons un fichier modèle (avec des paramètres fixes), qui est modifié à la volée lors de la connexion.

### 5.3.3 La configuration du système :

Dans le cas de Windows 95 essentiellement, il est possible de protéger certains aspects du système (interdiction de modifier certains paramètres de drivers, etc.). Cela se fait, de même que pour les logiciels, par des écritures dans la base de registres. Le fait d'assurer cette configuration lors de chaque connexion (alors que cela pourrait être fait lors de l'installation de la machine), présente plusieurs avantages :

- une robustesse accrue, puisque la configuration est reconstruite régulièrement
- L'administrateur peut modifier la configuration de l'ensemble des machines du réseau simplement en modifiant un fichier situé sur le serveur.

---

<sup>2</sup> On l'aura compris, nos fichiers *xconf* s'inspirent de ces anciens fichiers de configuration.

## 5.4 Les menus

Lorsqu'un utilisateur a accès à un ou plusieurs logiciels, il désire bien entendu disposer des menus du système (menu démarrer sous Windows, menu pomme sur les Macs) ; Puisque les logiciels, ainsi que les fichiers l.conf, se trouvent a priori dispersés sur plusieurs serveurs du réseau, il est logique que les menus soient éclatés : cela permet d'avoir sur le même serveur, logiciel fichier de configuration et menus associés.

### 5.4.1 Qu'est-ce qu'un menu ?

Que ce soit sous Windows ou sous Mac Os, le menu est en fait le pendant d'une hiérarchie de répertoires, contenant des raccourcis (Windows) ou des alias (Mac). Il est donc a priori simple de reconstituer le menu complet lors du login, puisque cela consiste en une copie d'un ensemble de répertoires contenant relativement peu de données (les raccourcis sont de petits fichiers).

### 5.4.2 Où est le menu ?

Le menu, c'est-à-dire l'ensemble de ces répertoires, se trouve donc éclaté sur les différentes ressources auxquelles sont connectés des utilisateurs. Lors de la connexion, le poste de travail devra copier chacun de ces répertoires en-dessous du répertoire correspondant au menu système (c'est-à-dire soit en local s'il s'agit d'un Mac, soit sur le serveur s'il s'agit d'une machine Windows). Enfin, une partie du menu correspondant aux applications locales sera copiée depuis le poste lui-même. Ce système nous procure en définitive une très grande souplesse : en effet, le menu peut être constitué de nombreux sous-menus présentant diverses caractéristiques ; certains sont formés par l'administrateur réseau, tandis que d'autres (en particulier le sous-menu du répertoire d'accueil) sont remplis par l'utilisateur, avec ses propres applications installées sur le réseau. On peut également imaginer un sous-menu correspondant à un répertoire partagé par un groupe, pour les applications installées sur le réseau, mais spécifiques à un groupe d'utilisateurs. Enfin, le sous-menu des applications locales correspond aux applications installées sur une machine particulière.

### 5.4.3 Génération du menu pilotée par l'administrateur ou l'utilisateur.

Le menu lui-même (Windows), ou une copie de celui-ci (Macs), se trouvera sur le répertoire d'accueil de l'utilisateur, afin d'accélérer le processus de login : il est en effet plus rapide de copier une seule hiérarchie que de reconstituer un menu à partir de plusieurs sources. Cette dernière opération n'a donc lieu que si la copie de sécurité du menu est supprimée. L'administrateur aussi bien que l'utilisateur peut supprimer cette copie afin de forcer la reconstruction du système de menus lors de la prochaine connexion, par exemple lorsqu'un nouveau logiciel est installé.

## 5.5 Le processus de connexion des micros

Le processus de connexion d'un micro ordinateur aux ressources du réseau est schématisé sur le tableau 1. Le serveur qui intervient lors de ce processus est le maître des micros. Ce serveur contiendra entre autres tous les scripts ou fichiers conf utilisés par les scripts de connexion.

	Windows	Mac
1	Ecran de login de Windows 95	Programme local login.exe
2	montage automatique de netlogon	Montage explicite de netlogon
3	appel automatique de logon.bat	
4	appel du script wi_logon.pl	appel du script ma_logon.pl
5	Lecture du fichier u.conf	Lecture du fichier u.conf
6	Lecture du fichier S.conf	Lecture du fichier S.conf
7	Connexion aux partages	Connexion aux partages
8	Configuration de l'O.S.	
9	Configuration des logiciels	Configuration des logiciels
10	Génération des menus	Génération des menus
11		Ecriture de logout.conf
12	Fin de wi_logon.pl	Fin de ma_logon.pl
13	Fin de logon.bat	
14	démontage automatique de netlogon	démontage de netlogon
15	Fin du processus de logon	Fin de login.exe

Tableau 1. Le protocole de connexion pour une machine Windows et pour un Mac

Les différentes étapes

sont explicitées ci-dessous, les numéros renvoient à la figure . Pour plus de clarté, nous avons séparé le texte explicatif, suivant que le client est une machine Windows ou un Mac :

### 5.5.1 Pour un client Windows :

1. L'écran de login est inclus dans Windows 95 ;
2. Le protocole Lan Manager de Windows 95 assure que la ressource netlogon est automatiquement montée.

3. Le script `netlogon.bat` est appelé automatiquement.
4. `netlogon.bat` contient une seule ligne : l'appel du script `wi_logon.pl`
5. La section de `U.conf` correspondant à l'utilisateur connecté est lue, en particulier l'identificateur partages indique à quels partages l'utilisateur devra se connecter.
6. Les sections nécessaires de `S.conf` sont lues.
7. Les connexions aux différents partages sont établies.
8. Le système d'exploitation est configuré
9. Les logiciels sont configurés. Les étapes 8 et 9 fonctionnent de la manière suivante :  
les fichiers `wi_os.conf` dans `netlogon`, ou encore les fichiers `l.conf` dans chaque partage connecté, sont lus. Ces fichiers contiennent toute l'information concernant la configuration du système ou des logiciels, essentiellement les clés et leurs valeurs à insérer dans les fichiers `.ini` ou dans la base de registres
10. Génération du menu démarrer, si nécessaire.
11. Etape sautée sous Windows
12. Fin de `wi_logon.pl`
13. Fin de `netlogon.bat`.
14. `netlogon` est automatiquement démonté

#### 5.5.2 Pour un client Mac :

1. Nous avons développé un programme en perl qui se chargera du login. Ce programme commence par afficher une boîte de dialogue afin que l'utilisateur rentre son nom et son mot de passe.
2. Sur un Mac, le programme `login.exe` se charge de monter la ressource `netlogon` sur le maître des micros.
3. `login.exe` appelle le script `ma_logon.pl`, qui se trouve sur le maître.
4. à 9.voir ci-dessus
10. Génération du menu pommes, ou copie de celui-ci à partir du répertoire d'accueil de l'utilisateur sur la machine locale.
11. Un fichier, appelé `logout.conf`, est écrit en local sur le mac. Il contient des informations qui seront utilisées lors du `logout`.
12. La ressource `netlogon` est explicitement démontée

### 5.6 Le processus de déconnexion des micros

Pour un client Windows, le processus de `logout` est déjà implémenté dans le système, nous n'avons rien eu à modifier.

	Windows	Mac
1		Programme local <code>logout.exe</code>
2		Lecture du fichier <code>logout.conf</code>
3	Processus de <code>logout</code> de Windows 95	Sauvegarde des préférences
4		Démontage des volumes
5		Appel de <code>login.exe</code>

Tableau 2. Le protocole de déconnexion pour une machine Windows ou pour un Mac

Pour un client Mac, le tableau 2 montre le déroulement des opérations :

1. Appel de `logout.exe`
2. Lecture du fichier `logout.conf`
3. Sauvegarde des fichiers de préférence ou de configuration des utilisateurs sur le volume réseau de l'utilisateur (la liste des fichiers à sauvegarder se trouve dans `logout.conf`).
4. Démontage des volumes
5. Appel de `login.exe`, afin qu'un nouvel écran de login soit proposé.

## 6. Interfaces utilisateurs

Tous les programmes décrits ici fonctionnent en ligne de commande, sans aucun processus interactif. Cela nous laisse un maximum de souplesse pour développer par-dessus ces scripts des interfaces présentant un degré de convivialité plus ou moins avancé : nous avons en particulier écrit un programme permettant de fixer de manière interactive les (nombreux) paramètres intervenant lors de la création d'un utilisateur ou d'un groupe. Une version graphique est actuellement à l'étude. De la même manière, les utilisateurs peuvent modifier leur mot de passe simplement en remplissant les champs d'un formulaire `html`. Celui-ci est associé à un script `cgi` qui se chargera de contacter toutes les machines des domaines demandés afin de réaliser effectivement le changement de mot de passe.

## 7. Conclusion

Nous utilisons quotidiennement sur notre réseau<sup>3</sup> les programmes que nous venons de décrire, aussi bien pour ce qui est de la gestion des serveurs que de celle des clients. Les résultats sont satisfaisants, en particulier pour ce qui est des performances :

- Il faut environ 60 secondes pour ajouter ou retirer un utilisateur d'un domaine constitué de 6 serveurs.
- Il faut environ 5 secondes à une machine Windows (pentium 133) pour se connecter
- Il faut environ 30 secondes à un Mac (power ou 68000) pour se connecter.

Cependant, nous insistons sur le fait que nous ne décrivons pas ici une application d'administration système, mais bien plutôt une méthodologie qui nous semble intéressante : en effet, nous n'avons pas cherché à développer un logiciel supportant tous les cas de figure que l'on peut imaginer ; nous n'avons pas toujours bien séparé ce qui était politique locale (règles sur les `uid` et `gid`, par exemple) et ce qui était algorithme général ; certains fichiers de configuration (notamment les fichiers de profils) sont assez délicats à écrire

Ces développements nous ont permis d'acquérir un plus grand recul à propos des systèmes d'exploitation micros : il nous semble en fin de compte que ceux-ci ne sont que difficilement configurables pour plusieurs utilisateurs (surtout quand « plusieurs » est proche de 200 à 300)... mais il ne manquerait pas grand-chose pour qu'ils le deviennent ! De même, le manque de standardisation dans la manière de configurer les logiciels nous est apparu comme une grande difficulté. L'absence de langage de scripts, en particulier sur Windows et Windows NT, est également un handicap important ; heureusement, de ce point de vue, l'existence de langages tels que `perl`, qui plus est sous forme de produits non commerciaux, permet de travailler dans de bonnes conditions, même si les portages sur les différents environnements manquent parfois un peu de cohérence.

Tout cela montre que le micro-ordinateur reste, en grande partie, un ordinateur vraiment « personnel », et qu'il n'est pas simple de le transformer en « station de travail banalisée »... et pourtant c'est bien de cela que l'on a besoin dans le cadre d'une utilisation professionnelle, et c'est ce que nous pensons avoir réussi, en utilisant des développements relativement simples, et des outils non commerciaux extrêmement répandus.

## 8. Références

- [perl5] Le programmeur PERL5, SIMON ET SHUSTER, MACMILLAN (<http://www.ssm.fr/>)  
ou encore <http://www.perl.com>
- [perlsrc] <ftp://ftp.pasteur.fr/pub/Perl/CPAN/modules/01modules.index.html>
- [macperl] <http://err.ethz.ch/~neeri/macintosh/perl.html>
- [perlw32] <http://www.activeware.com/>
- [till] Technical Information Library, Apple <http://till.info.apple.com/>
- [mskb] Microsoft Knowledge Base, <http://www.microsoft.com/kb/>
- [listes] <mailto:Perl-Win32-Users@ActiveWare.com> et <mailto:mac-perl@iis.ee.ethz.ch>
- [inmac] Inside Macintosh : <http://devworld.apple.com/dev/insidemac.shtml>
- [rifflet] Jean-Marie Rifflet ; La communication sous UNIX: Applications réparties, 2nde édition ; Ediscience International, 1995

---

<sup>3</sup> Le réseau comprend actuellement une quinzaine de serveurs unix ou NT, 200 utilisateurs, une cinquantaine de clients MAC ou PC et est encore actuellement en phase de croissance