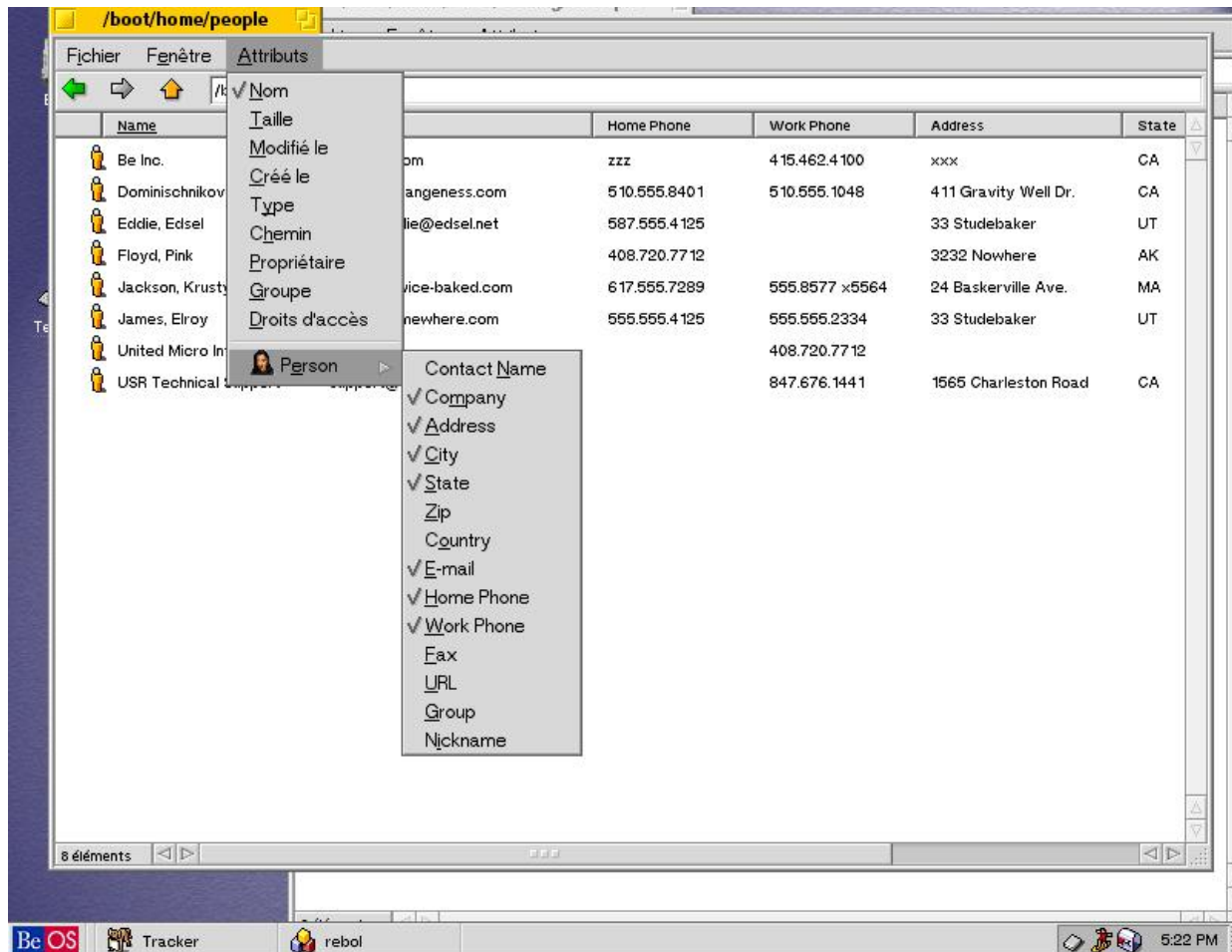


## **Rebol et BeOS : une association exemplaire pour la gestion des Bases de Données**

**François Jouen**

Malgré sa disparition prématurée, BeOS reste un des meilleurs systèmes d'exploitation pour PC. Supportant le SMP (Symetric Multi Processing), compact, rapide et se contentant de petites configurations (un modeste P133 fait l'affaire), le micro-noyau de quelques Ko est certainement avec QNX un des produits les plus rapides et les plus stables pour une architecture x86. BeOS possède de plus des caractéristiques exemplaires. BeOS dispose d'abord d'un gestionnaire de fichiers qui permet d'associer à un fichier des attributs de différents types (string, boolean, byte, character, unsigned character, integer 16 bit, integer 32 bit, integer 64 bit, double, float et time) , et de taille illimitée. En plus des attributs classiques comme le nom, la taille, la date de création, la date de modification, le type, le chemin d'accès, le propriétaire, le groupe et les droits d'accès du fichiers, BeOS permet de gérer ces « attributs supplémentaires » de façon extrêmement simple via le "tracker" qui offre une interface flexible et intuitive permettant un accès facile aux différents attributs. BeOS dispose en outre d'une gestion très élaborée des "MIME type" qui permet de créer potentiellement tout type de structure pour décrire la relation fichier/attributs. De plus BeOS offre un moteur de recherche très puissant qui permet une recherche sophistiquée des différents attributs associés à un fichier. Enfin, BeOS intègre un langage de script puissant et rapide le "bash shell". En d'autres termes, la gestion des fichiers et de leurs attributs telle qu'elle a été conçue par les ingénieurs de Be représente un noyau de base de données qui est directement intégré au système. Potentiellement, chaque fichier ou chaque répertoire peut être utilisé comme une base de donnée dont les éléments sont accessibles de façon instantanée. Les fichiers contenus dans le répertoire /boot/home/people/ ou dans le répertoire /boot/home/mail/in sont un bon exemple de ce type de gestion. Chaque fichier contenu dans ce répertoire

est considéré par l'OS comme un enregistrement d'une base de donnée et chaque attribut associé à ce fichier comme un champ de l'enregistrement (figure 1).



### Accéder aux attributs

Outre l'accès par une interface graphique comme le Tracker, les attributs des fichiers sont directement accessibles via le bash shell en mode console.

la commande "listattr nom\_fichier" renvoie une liste de tous les attributs non vides associés au fichier. Par exemple listattr '/people/Be Inc.' renvoie les informations suivantes:

| File Be Inc. |      |               |
|--------------|------|---------------|
| Type         | Size | Name          |
| -----        |      |               |
| MIME str     | 21   | BEOS:TYPE     |
| Text         | 7    | META:name     |
| Text         | 12   | META:nickname |
| Text         | 9    | META:company  |
| Text         | 4    | META:address  |
| Text         | 11   | META:city     |
| Text         | 3    | META:state    |
| Text         | 4    | META:zip      |
| Text         | 4    | META:country  |
| Text         | 4    | META:hphone   |
| Text         | 13   | META:wphone   |
| Text         | 5    | META:fax      |
| Text         | 12   | META:email    |
| Text         | 18   | META:url      |
| Text         | 8    | META:group    |

f

La commande "catattr nom\_attribut nom\_fichier" quant à elle renvoie les données contenues dans l'attribut du fichier sous la forme nom\_fichier: type: données. Par exemple catattr META:nickname '/people/Be Inc.' retourne "Be Inc. : string : Main number".

La commande "rmattr nom\_attribut nom\_fichier" supprime l'attribut. Enfin la commande " addattr nom\_attribut "contenu" nom\_fichier" permet d'ajouter ou de modifier l'attribut du fichier.

### **Accéder aux attributs à partir d'un autre système d'exploitation**

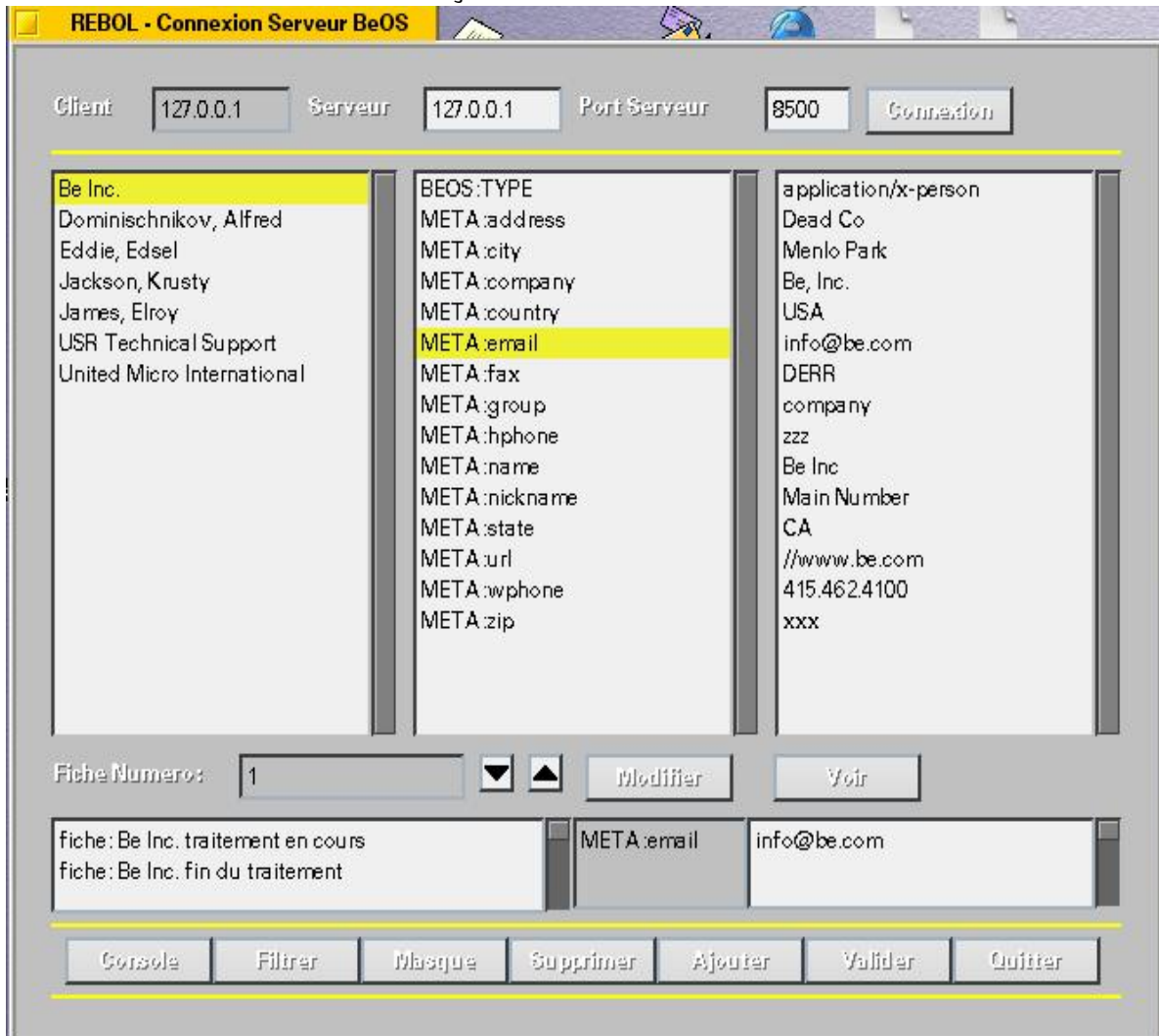
Lors de sa conception BeOS a été pensé comme un OS ouvert. La pile TCPIP est intégrée de façon native au noyau et il est dès lors tentant de vouloir partager les ressources insoupçonnées de BeOS avec des clients hétérogènes. On peut par exemple souhaiter que sa base de données des contacts contenue dans le répertoire /boot/home/people/ soit accessible au sein d'un réseau intranet. Il faut simplement un client capable d'interagir avec le noyau via un protocole TCPIP afin de pouvoir avoir accès aux attributs des fichiers. Par exemple une connexion via telnet est envisageable. C'est ici qu'intervient Rebol avec toutes ses caractéristiques : puissant langage réseau indépendant des plate-formes.

### **Le Serveur Rebol**

Il était tout à fait possible de concevoir en C++ un serveur natif BeOS pour faire l'interface entre le noyau de BeOS et les clients. Pour notre part nous avons choisi de gérer directement le serveur avec Rebol de façon à illustrer la polyvalence de Rebol. Le serveur est minimaliste mais fonctionne de façon optimale. Sa tâche est très simple : le serveur analyse les requêtes arrivant sur le port TCP 8500 (paramétrable), demande l'exécution des ces requêtes et renvoie le résultat au client qui va assurer leur mise en forme. L'exécution des requêtes se fait via un appel à la fonction «call» de Rebol. Par conséquent le serveur ne fonctionnera qu'avec la version pro de Rebol View.

### **Le Client Rebol**

Le client ne nécessite pas l'usage du View Pro. Il s'agit d'un véritable gestionnaire de base de données qui permet (figure 2) d'accéder aux "bases" , de filtrer les enregistrements, de les afficher, de les modifier, de les supprimer ou d'en ajouter. En fonction des choix de l'utilisateur, le client envoie la requête au serveur TCP, attend le résultat et le traite localement les données recueillies. L'avantage de ce client rebol est qu'il fonctionne avec les différents OS supportés par Rebol View (Unix, Linux, MacOS, BeOS, Windows...) sans avoir besoin de réécrire le code.



### La communication Client-Serveur

Cette communication est la plus simple qui soit. Elle est fondée sur l'existence d'un dictionnaire de codes commun au serveur et au client. Le client fournit un code et un argument (en général le nom du fichier BeOS). Le serveur analyse le numéro de code et demande l'exécution de la requête à l'OS en lui passant en paramètre l'argument envoyé par le client (cf la fonction analyse-demandes). Enfin le serveur renvoie le résultat du traitement au client qui va assurer la mise en forme des données.

```
analyse-demandes: does [  
switch code-demande [  
  1; toute la base  
    [  
      destination: join rep-serveur "dir.txt"  
      change-dir to-file argument  
      call/wait join "ls " [argument " > "destination]  
      insert connexion read to-file destination  
      call/wait join "rm " destination]  
    ]  
  2 ; le filtre  
    [  
      destination: join rep-serveur "dir.txt"  
      call/wait join "ls " [argument " > "destination]  
      insert connexion read to-file destination  
      call/wait join "rm " destination]  
    ]  
  3 ; obtention des attributs du fichier passé en paramètre  
    [  
      ;récupere la liste des champs documentés  
      prog: to-string join rep-serveur ["getfields " argument]  
      probe prog  
      call/wait prog  
      insert connexion read %champs.txt  
      call/wait "rm attribute.txt"  
      call/wait "rm champs.txt"  
    ]  
  4; suppression de la fiche  
    [call/wait join "rm " argument]  
  5; mise à jour de la fiche  
    [call/wait join "addattr " argument]  
  6; créer une nouvelle fiche  
    [call/wait join "echo > " argument]  
  7; lecture du fichier  
    [  
      destination: join rep-serveur "contenu.txt"  
      call/wait join "cat " [argument " > "destination]  
      insert connexion read to-file destination  
      call/wait join "rm " destination]  
    ]  
  ]; fin switch  
]
```

Les deux premières demandes (codes 1 et 2) sont basiques : le serveur demande l'affichage du contenu du répertoire dans lequel sont contenues les

fiches en fonction du filtre passé en paramètre, copie ce contenu dans un simple fichier texte qui est renvoyé au client et enfin supprime le fichier texte. Comme BeOS est un « unix-like » les requêtes de filtre peuvent être simples ou très sophistiquées (par exemple [aA-zZ]\* : toutes les fiches qui commencent par une lettre majuscule ou minuscule). De la même façon il serait possible d'utiliser ici les fonctions "query" de BeOS. Par exemple une commande du type query "{(name=="\*\*\*)&&(BEOS:TYPE=="application/x-person")}" pourrait être utilisée de façon à lister toutes les fiches.

La demande 3 (obtention des attributs du fichier passé en paramètre) est un peu plus complexe. En fait le serveur demande l'exécution d'un simple shell script (getfields) qui assure une double fonction :

- 1°) récupérer la liste des attributs du fichier;
- 2°) afficher pour chaque attribut documenté le contenu de l'attribut

```
#!/bin/sh
listattr "$1" |grep :|cut -c30- |tr -d ' ' > attribute.txt

head -1 attribute.txt > tmp1
xx=`grep BEOS tmp1`
if [ "$xx" = "BEOS:TYPE" ]; then
    debut=0
else
    debut=1
fi

rm tmp1
{
    foo=0
    while read ThisLine; do
        foo=`expr $foo + 1`
        if [ $test $foo -gt $debut ]; then
            echo -n $ThisLine "= "
            lecture= catattr $ThisLine "$1" |cut -f3- -d:
        fi
    done
} < attribute.txt > champs.txt
```

Encore une fois, comme BeOS est similaire à Unix, les commandes sont associées par des « pipes » ( | ) de façon à les enchaîner.

La première partie du shell script (listattr "\$1" |grep :|cut -c30- |tr -d ' ' > attribute.txt) demande la liste des attributs (listattr) du fichier passé en paramètre (\$1), recherche ensuite par « grep » le mot clé : et copie la chaîne à partir du 30<sup>ème</sup> caractère ( cut -c30-) dans le fichier « attribute.txt » en supprimant les espaces (tr -d ' ').

```
META:name  
META:nickname  
META:company  
META:address  
META:city  
META:state  
META:zip  
META:country  
META:hphone  
META:wphone  
META:fax  
META:email  
META:url  
META:group
```

Ce fichier attribute.txt qui va être ensuite passé en paramètre à la seconde partie du script de façon à construire le fichier « champs.txt » qui contiendra pour chaque attribut les informations suivantes : le nom du champ et son contenu

```
META:name = Dominischnikov, Alfred  
META:nickname = AD  
META:company = Strange Attractors  
META:address = 411 Gravity Well Dr.  
META:city = Nucleus  
META:state = CA  
META:zip = 94633  
META:country = USA  
META:hphone = 510.555.8401  
META:wphone = 510.555.1048  
META:fax = 510.555.6543  
META:email = quark@strangeness.com  
META:url = http://tenured.crackpot.edu/  
META:group = Bowling team
```

### *François Jouen Rebol et BeOS*

Après l'envoi de ce fichier au client, les fichiers temporaires sont effacés du disque du serveur.

La demande 4 (suppression de la fiche) est très simple: un appel à la fonction `rm` supprime le fichier.

La mise à jour des attributs est également très facile (codes 5) puisqu'on appelle directement la commande `addattr`. L'argument passé en paramètre est le contenu du champ qui a été modifié par le client.

Il n'y a pas de difficulté majeure pour ajouter une fiche (code 6). En premier lieu, on crée un fichier vide `"call/wait join "echo > " nom_fichier"`. Une fois cette opération réalisée, il suffit pour le client de mettre à jour le contenu des attributs en envoyant le codes 5.

Enfin, le code 6 permet de visualiser le fichier associé aux attributs

#### **En guise de conclusion**

Cet article avait pour objet d'illustrer comment Rebol est particulièrement bien adapté à l'informatique distribuée. En outre tout ce qui a été décrit peut s'étendre à d'autres types de fichiers. On pourrait ainsi concevoir un véritable gestionnaire de base de données en Rebol qui permettrait à l'utilisateur de définir lui même les "champs" de sa base de données, de la créer et ensuite de l'exploiter. L'association Rebol BeOS se révèle ainsi particulièrement fructueuse.

© François Jouen, Août 2002.