ServiceTranslator

Un sous-projet de VeriNeC

Département d'Informatique
Université de Fribourg, Suisse

Responsable: Prof. Dr. Ulrich Ultes-Nitsche
Assistants: David Buchmann et Dominik Jungo

Etudiant:
Robert A. Awesso
Route des Arsenaux 39
1700 Fribourg
akehlelou.awesso@unifr.ch

Robert A. Awesso

9 septembre 2005

Résumé

Le but du projet VeriNeC est de simplifier la configuration des réseaux. Il est basé sur la définition d'un réseau abstrait et peut simuler différents états d'un réseau tout comme générer des données de configuration pour le réseau.

Le sous-projet ServiceTranslator s'occupe de la génération de fichiers de configuration pour deux services dans le monde linux. Le service DNS et le service DHCP. Sachant que les fichiers de configuration sont des fichiers de texte et que le couple XML et XSLT peut donner n'importe quel type de document en sortie, nous voyons aisément toutes les possibilités qui nous sont données.

Mots-clés : XML, XSLT, XML Schema, DNS, DHCP

Table des matières

Chapitre 1

Introduction

1.1 Description du projet

Le projet ServiceTranslator, dont nous allons parler, s'insère dans un plus vaste projet appelé Verified Network Configuration[1] (VeriNeC) dont le but est de simplifier la configuration des réseaux en se basant sur le fonctionnement d'un réseau virtuel représentant le réseau réel. VeriNeC se soucie également en particulier des problèmes de sécurité. Dans VeriNeC, le réseau concret est donc représenté, grâce aux langages Java et XML [2] par un réseau virtuel qui peut être visualisé graphiquement avec ses noeuds et ses câbles. En plus de cette visualisation graphique, un document XML décrit toujours le réseau de manière à ce qu'on puisse utiliser la force du langage XML pour automatiser les configurations en utilisant le langage eXtensible Stylesheet Language for Transformation (XSLT). Un document XML décrit également chaque noeud du réseau et les services qu'il offre. Ainsi, il est possible de pouvoir extraire du document XML des informations relatives à un service de manière à pouvoir le configurer. Une fois qu'on a l'information sur un service, on a plusieurs moyens de le configurer : panneau de configuration dans Windows, base des registres dans Windows, fichiers de configurations dans le monde linux, etc.

Dans notre projet, il s'agira, dans un premier temps, de choisir une implémentation d'un service défini dans VeriNeC et qui n'a pas encore de traducteur et d'en écrire un. Dans un deuxième temps, il s'agira de penser à à l'implémentation d'un service qui n'existe pas encore dans VeriNeC et de l'y ajouter alors en écrivant le schema, un exemple de document XML et le traducteur qui va avec.

1.2 Services choisis

Le premier service sur lequel nous travaillerons est le Domain Name System (DNS). Le DNS est déjà défini dans le schema de VeriNeC et il existe déjà un traducteur pour la solution Berkeley Internet Name Daemon (BIND)[1] or il existe plusieurs autres solutions DNS dont la solution Dan J. Bernstein Domain Name System (DJBDNS) pour laquelle nous avons choisi d'écrire un traducteur. Nous allons d'abord nous employer à écrire un traducteur pour DJBDNS, ce qui suppose que nous commençons par chercher à comprendre comment cette solution DNS marche en la comparant à BIND.

Après DJBDNS, nous nous tournerons vers un service qui n'existe pas encore dans VeriNeC. Nous nous emploierons dans ce cas à adjoindre au schema une section pour le service Dynamic Host Configuration Protocol (DHCP) que nous aurons d'abord cherché à comprendre au mieux. Après l'élargissement du schema, nous écrirons également un traducteur pour l'implémentation DHCPD.

[1] Certains disent Berkeley Internet Name Domain mais nous avons retenu "Daemon" car ça nous semble mieux désigner BIND étant donné qu'il tourne en arrière fond

Chapitre 2

Le service DNS

2.1 Fonctionnement et généralités

DNS signifie Domain Name System. C'est grâce à ce système qu'on peut saisir dans un navigateur un nom de domaine en toutes lettres et pouvoir atteindre l'endroit que ce nom désigne, bien que les noeuds du réseau se connaissent entre eux par leurs adresses IP. Pour nous fixer les idées et comprendre les bases du fonctionnement du DNS, considérons la figure 2.1 que nous offre le site ZoneEdit [3].

- Le client saisit un nom de domaine (www.domaine.com) dans son navigateur.
- Le navigateur contacte le Fournisseur d'Accès à Internet (FAI) du client pour connaître l'adresse IP liée à ce domaine.
- Le FAI cherche la réponse dans son cache et s'il l'a, il la retourne avec la marque " non-authoritative " car le FAI n'est pas responsable pour le nom de domaine demandé par le client.
- Si le FAI n'a pas la réponse dans son cache, il questionne les serveurs responsables du domaine pour une réponse et s'il ne connaît pas ces serveurs DNS, il questionne directement les serveurs racine.

Le DNS fonctionne donc sur le modèle classique client-serveur où le client est un programme qui demande à quelle adresse IP est associé un certain nom de domaine ou l'inverse. Le serveur à son tour essaye de fournir une réponse s'il l'a ; et s'il ne l'a pas, il peut soit transférer la demande à un autre serveur qu'il pense être capable de fournir une réponse (mode récursif), soit répondre au client demandeur qu'il ne sait pas, tout en indiquant quel serveur serait susceptible d'avoir une réponse (mode itératif).

Les serveurs jouent deux rôles :

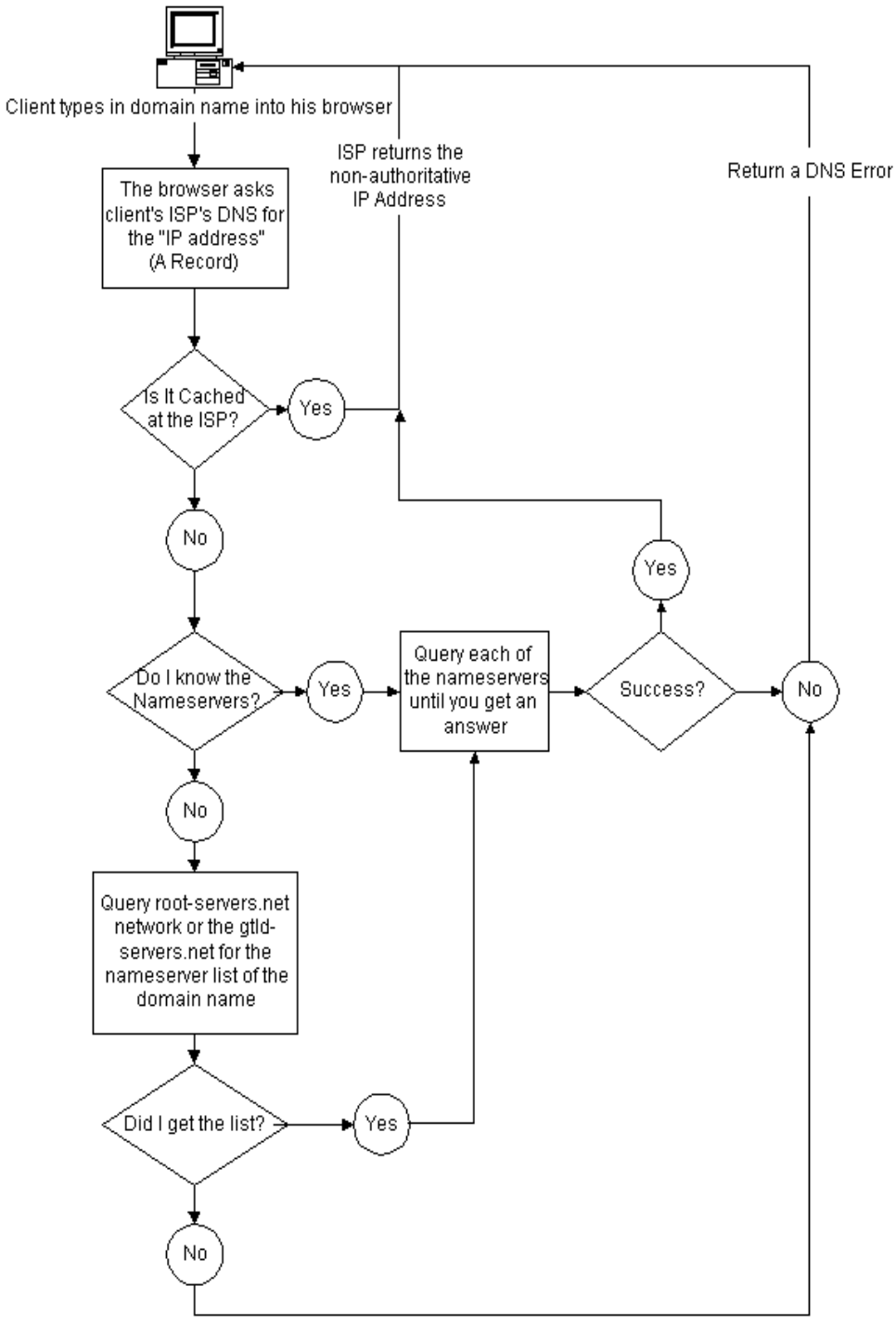

FIG. 2.1 – Les bases du fonctionnement du DNS

- un rôle de " responsable de zone " (on dit alors que le serveur est autoritaire sur la zone) qui signifie que pour une partie du réseau, c'est ce serveur qui a l'information officielle en ce qui concerne les noms de domaine et les autres serveurs doivent s'adresser à lui pour les obtenir s'ils ne les ont pas dans leur cache ou pour actualiser les informations qu'ils ont à leur disposition. La première des données que le serveur aura dans sa base de données est le SOA (Start Of Authority) qui institue son autorité sur la zone.
- un rôle de cache qui signifie que quand les informations des parties du réseau dont il n'est pas responsable passent par lui, il les stocke pour un besoin ultérieur.

On dit qu'un serveur est primaire quand il reçoit directement les informations relatives à la zone dont il est responsable dans un fichier de configuration, informations introduites par un administrateur. On dit qu'il est secondaire quand il récupère les informations d'un autre serveur et devient autoritaire sur la zone.

La RFC 1034 [4] sur le DNS nous apprend qu' " un nom de domaine identifie un noeud. A chaque noeud est attribué un ensemble d'informations sur des ressources, lequel peut être vide. L'ensemble d'informations de ressources associé à un nom particulier est composé de quatre enregistrements de ressources en anglais, Ressource Record (RR) séparés . " Les RR contiennent les éléments suivants :

- Owner qui est le nom de domaine où le RR est trouvé,
- Type qui est une valeur encodée sur 16 bits spécifiant le type de ressource décrit par cet enregistrement. Les types se réfèrent à une définition abstraite des ressources,
- Class qui est une valeur encodée sur 16 bits identifiant une famille de protocoles ou une instance d'un protocole,
- TTL qui est la durée de vie du RR. Cette valeur est représentée sous forme d'un entier sur 32 bits, est exprimée en secondes, et est principalement utilisée par les résolveurs lorsqu'ils mémorisent temporairement les RR. Le champ TTL définit combien de temps un RR peut être gardé localement avant de devoir être considéré comme obsolète,
- RDATA qui est le type et parfois les données dépendantes de la classe décrivant la ressource.

Précisons que les types principaux sont :

- A qui est une adresse IP
- CNAME qui est l'alias d'un nom d'hôte
- MX qui est une adresse serveur de courrier électronique
- NS qui est le serveur de nom autorisé pour le domaine
- PTR qui est un pointeur et

– SOA qui est le Start Of Authority

Sur le marché, une solution DNS est largement utilisée : il s'agit de Berkeley Internet Name Daemon (BIND). Il existe plusieurs autres solutions DNS dont DJBDNS de Dan J. Berstein sur laquelle nous nous attarderons après avoir vu BIND.

2.2 BIND

BIND [5] est donc la solution la plus commune utilisée dans le monde Unix et Linux. Il fonctionne grâce au fichier principal named.conf et aux fichiers de zones correspondant aux entrées zone du fichier named.conf. Ce dernier en lui-même est une suite de déclarations du style :

```
<declaration> ["nom_declaration"] [classe_declaration] {
   <option-1>;
   <option-2>;
   <option-N>;
};
```

On peut trouver un exemple assez complet de fichier named.conf sur le site Linux-sottises [6]

Pour comprendre l'interaction entre le fichier named.conf et les fichiers de zones, considérons les exemples ci-dessous.

Soit le fichier named.conf minimal suivant :

```
zone "domain.com" IN {
  type master;
  file "domain.com.zone";

};
```

Le fichier de zone nommé domain.com.zone et stocké dans le même dossier que named.conf se présentera par exemple sous la forme :

```
$ORIGIN domain.com $TTL 86400 @     IN     SOA    dns1.domain.com.
hostmaster.domain.com. (
                    2001062501 ; serial
                    21600      ; refresh after 6 hours
                    3600       ; retry after 1 hour
                    604800     ; expires after 1 week
                    86400 )    ; minimum TTL of 1 day
```

```
        IN      NS      dns1.domain.com.
        IN      NS      dns2.domain.com.

        IN      MX      10      mail.domain.com.
        IN      MX      20      mail2.domain.com.

                IN      A        10.0.1.5
server1         IN      A        10.0.1.5
server2         IN      A        10.0.1.7
dns1            IN      A        10.0.1.2
dns2            IN      A        10.0.1.3

ftp             IN      CNAME    server1
mail            IN      CNAME    server1
mail2           IN      CNAME    server2
www             IN      CNAME    server2
```

où les lignes :

```
    IN      NS      dns1.domain.com.
    IN      NS      dns2.domain.com.
```

d'une part et les lignes :

```
    dns1            IN      A        10.0.1.2
    dns2            IN      A        10.0.1.3
```

d'autre part, associent au domaine domain.com, les adresse IP 10.0.1.2 et 10.0.1.3 comme serveurs DNS.

Après avoir vu comment BIND arrive à lier un nom de domaine à une adresse IP, tournons-nous vers DJBDNS pour comprendre comment lui, il fait ce pointage.

2.3 DJBDNS

Dan J. Bernstein a développé DJBDNS [7] car il ne supportait plus les problèmes répétés de sécurité et les bugs qui étaient nombreux avec BIND. BIND est également un programme assez monolithique, ce que D. Berstein

ne supporte pas non plus, lui pour qui la modularité des programmes est très importante.

DJBDNS a quatre composantes principales : Tinydns, Dnscache, Axfrdns et Walldns. Tinydns est le module qui fournit l'information sur la configuration du domaine. Il fonctionne uniquement avec le protocole UDP sur le port 53. Dnscache est le module traitant les requêtes des clients et jouant le rôle de cache pour les requêtes déjà traitées. Axfrdns est le module qui permet les transferts de zones. Pour rappel, notons que le transfert de zone permet la réplication des données et ainsi favorise la disponibilité et la tolérance aux pannes. Walldns enfin est le module chargé de la résolution inverse adresse IP vers nom de machine.

Plusieurs questions se posent à nous à ce niveau : le module dnscache semble déjà jouer le rôle de serveur de nom, quel est alors le rôle de tinydns ? De plus les deux modules écoutent sur le port 53 et utilisent le protocole UDP, comment cela se passe-t-il ? Le site Itworld [8], nous apprenons qu'il faut prévoir (pour le cas où tinydns et dnscache sont installés sur la même machine) que les processus des deux modules n'écoutent pas sur la même adresse IP ; par exemple, tinydns peut écouter sur 127.0.0.4 et dnscache sur 127.0.0.3, le mieux étant que les deux modules soient sur des machines différentes. De plus, précisons que tandis que tinydns est le serveur qui a les enregistrements sur le domaine auquel appartient la machine sur laquelle est installé DJBDNS et renseigne aussi dnscache en cas de nécessité, dnscache se contente, quand il est consulté par un client de consulté d'autres serveurs pour avoir la réponse à la requête d'un client, s'il ne l'a pas en cache ou si le TTL est dépassé. Bref, par rapport à la descritpion d'un serveur DNS faite à la section 2.1, disons que tinydns a l'autorité sur la zone tandis que dnscache joue le rôle de cache. Remarquons que ceci est fait de manière monolithique dans BIND mais une particularité des programmes de Bernstein est la modularité et l'effort qu'un processus accomplisse une seule tâche à la fois.

Etant donné que nous voulons procéder à une configuration des noeuds par les fichiers de configuration. Nous allons voir les différents fichiers présents dans les dossiers des différents modules à l'installation de DJBDNS de manière à déterminer quels seront les fichiers qui devront être modifiés par notre traducteur.

En général, le dossier tinydns pour le serveur est installé dans le dossier /etc (/etc/tinydns) après exécution du programme tinydns-conf. Tinydns-conf crée en outre les sous-dossiers root et log du dossier tinydns. Le dosssier root contient un fichier Makefile qui permet d'exécuter tinydns-data. Le fichier data est également placé dans le dossier root et c'est le lancement de la commande make qui exécute le programme tinydns-data, générant le fichier

data.cdb. Le fichier data de DJBDNS rassemble les données qu'on trouve dans le fichier named.conf et dans les fichiers de zones dans BIND.

De même, pour le cache, un programme dnscache-conf génère le dossier dnscache dans etc (/etc/dnscache) et crée également des sous-dossiers root et log. Notons la présence d'un troisième sous-dossier de /etc/dnscache appelé seed. Un sous-dossier de root nommé ip est également créé et il contient la liste des adresses IP desquelles le serveur dns peut accepter des requêtes.

Ici également comme dans BIND, il y a un fichier de configuration principal : il s'agit du fichier data [9] dont nous avons parlé plus haut.

Syntaxe du fichier data :

- La ligne de A commence par " + ",
- la ligne jumelée de A et de PTR commence par " = ",
- la ligne de PTR commence par l'accent circonflexe ,
- la ligne de MX commence par " @ ",
- la ligne jumelée de NS, A et SOA commence par " . ",
- la ligne jumelée de NS et A commence par " & ",
- la ligne de CNAME commence par " C ",
- la ligne de SOA commence par " Z ",
- la ligne de TXT commence par " ' ",
- la ligne commençant par " : " est une ligne générique[1]

Exemple de fichier data pour un nom de domaine domain.com

```
+domain.com:10.0.1.5:86400
.domain.com:10.0.1.2:ns1.mydomain.com
@domain.com:10.0.1.5:mail.domain.com:10:86400
+www.domain.com:10.0.1.5:86400
=*.domain.com: domain.com :86400
```

Explications :

La première ligne crée un enregistrement de type A appelé domain.com et qui pointe vers l'adresse IP 10.0.1.5" avec un TTL of 86400.

La deuxième ligne commence avec un ".", ce qui renseigne le nom du serveur DNS qui va contrôler ce domaine.

La troisième ligne définit le MX du domaine. Cela crée un enregistrement MX pour domain.com appelé mail.domain.com et qui pointe sur 10.0.1.5, avec un poids de 10 (c'est-à-dire qu'il est en priorité haute dans le cas ou plusieurs MX sont inscrits) et un TTL de 86400.

[1]exemple : :dom :type :dest :ttl :temps :lo où type est un entier entre 1 et 65535. Cet entier doit être différent de 2, 5, 6, 12, 15, 252 qui désignent respectivement NS, CNAME, SOA, PTR, MX et AXFR

La quatrième ligne est de nouveau un enregistrement de type A appelé www.domain.com et pointant vers l'adresse IP 10.0.1.5 La dernière ligne crée un pointeur d'un premier nom de domaine vers un second. Le signe * permet de faire un "catch all", c'est à dire que tous les sous domaines non explicités dans la zone seront redirigés vers l'adresse indiquée, ici example.com.

Ainsi, par opposition à BIND où le fichier named.conf a des fichiers de zone externes stockés dans /var/named/, le fichier data de DJBDNS est le fichier qui contient toutes les zones.

Du point de vue de l'implémentation, rappelons que le schema d'un noeud de VeriNeC est déjà intégré au projet et il s'agissait juste d'adapter ce schema de manière a pouvoir écrire un traducteur. Traducteur que nous mettrons en annexe. Nous y avons utilisé l'élément `<xsl:for-each>` pour extraire chaque élément qui nous intéressait et nous avons utilisé l'élément `<xsl:text>` pour y adjoindre des caractères comme ceux du début de chaque ligne du fichier data. Etant donné que que les implémentations de nos deux traducteurs pour DJBDNS et DHCPD se ressemblent assez, nous parlerons plus en détail de l'implémentation du traducteur pour DHCPD.

Chapitre 3

DHCP

3.1 Définition et fonctionnement

L'acronyme DHCP signifie Dynamic Host Configuration Protocol [10]. DHCP comme la définition l'indique, est un protocole qui permet de configurer dynamiquement et automatiquement un hôte d'un réseau.

DHCP fonctionne sur le modèle client-serveur et en général tous les autres hôtes d'un réseau sont des clients du serveur DHCP. Celui-ci a une adresse IP fixe et alloue dynamiquement des adresses IP appartenant aux autres membres du réseau. Bien noter qu'allocation dynamique ne signifie pas qu'un hôte ne peut pas recevoir une adresse fixe.

Le mécanisme par lequel une machine, qui s'insère dans un réseau, obtient sa configuration réseau est le suivant : le client procède à un broadcast sur le réseau pour s'annoncer et le serveur DHCP qui reçoit le broadcast répond et ainsi une communication peut s'établir entre le nouveau membre du réseau et le serveur de manière à ce que le client puisse obtenir sa configuration réseau. Le broadcast du client se fait sur l'adresse IP 255.255.255.255 et sur le réseau local. Il est donc reçu par toutes les machines du réseau, y compris le serveur DHCP. Evidemment, les autres machines ignorent le broadcast du client DHCP sauf le serveur DHCP. Celui-ci renvoi un message broadcast contenant toutes les informations requises pour la configuration. Si le client accepte la configuration, il renvoie un paquet pour informer le serveur qu'il garde les paramètres, sinon, il fait une nouvelle demande. Si le client a déjà une adresse IP, les échanges ne se font plus par broadcast. On peut se demander ici quels échanges sont encore nécessaires, le client ayant déjà une adresse IP. C'est là que nous devons savoir que le serveur DHCP accorde les paramètres de configuration pour une période limitée appelée bail de l'anglais " lease ". Un client qui voit son bail arriver à terme peut demander au

serveur un renouvellement du bail. C'est de là qu'on peut avoir des messages entre le client et le serveur, le client ayant déjà une adresse IP. De même, lorsque le serveur verra un bail arrivé à terme, il émettra un paquet pour demander au client s'il veut prolonger son bail. Si le serveur ne reçoit pas de réponse valide, il rend disponible l'adresse IP. C'est toute la subtilité du DHCP : on peut optimiser l'attribution des adresses IP en jouant sur la durée des baux. Le problème est là : si toutes les adresses sont allouées et si aucune n'est libérée au bout d'un certain temps, plus aucune requête ne pourra être satisfaite. Sur un réseau où beaucoup d'ordinateurs se connectent et se déconnectent souvent (réseau d'école ou de locaux commerciaux par exemple), il est intéressant de proposer des baux de courte durée. A l'inverse, sur un réseau constitué en majorité de machines fixes, très peu souvent rebootées, des baux de longues durées suffisent. N'oublions pas que le DHCP marche principalement par broadcast, et que cela peut bloquer de la bande passante sur des petits réseaux fortement sollicités.

On pourrait croire qu'un seul aller-retour peut suffire à la bonne marche du protocole. En fait, il existe plusieurs messages DHCP qui permettent de compléter une configuration, la renouveler... Ces messages sont susceptibles d'être émis soit par le client pour le ou les serveurs, soit par le serveur vers un client. Regardons deux diagrammes qui montrent la séquence temporelle de ces messages et par la suite un tableau qui les récapitule avec leur description.

Sur la figure 3.1, il s'agit d'un client qui demande sa première configuration. Sur la figure 3.2, c'est un renouvellement de bail

Et maintenant le récapitulatif des différents messages que le client et le serveur DHCP s'envoient avec leur descriptions :

- DHCPDISCOVER (1) pour localiser les serveurs DHCP disponibles et demander une première configuration
- DHCPOFFER (2) réponse du serveur à un message DHCPDISCOVER, qui contient les premiers paramètres
- DHCPREQUEST (3) requête diverse du client pour par exemple prolonger son bail
- DHCPDECLINE (4) le client annonce au serveur que l'adresse est déjà utilisée
- DHCPACK (5) réponse du serveur qui contient des paramètres et l'adresse IP du client
- DHCPNAK (6) réponse du serveur pour signaler au le client que son bail est échu ou si le client annonce une mauvaise configuration réseau
- DHCPRELEASE (7) le client libère son adresse IP
- DHCPINFORM (8) le client demande des paramètres locaux, il a déjà son adresse IP

Le nombre entre parenthèses est utilisé pour identifier ces requêtes dans

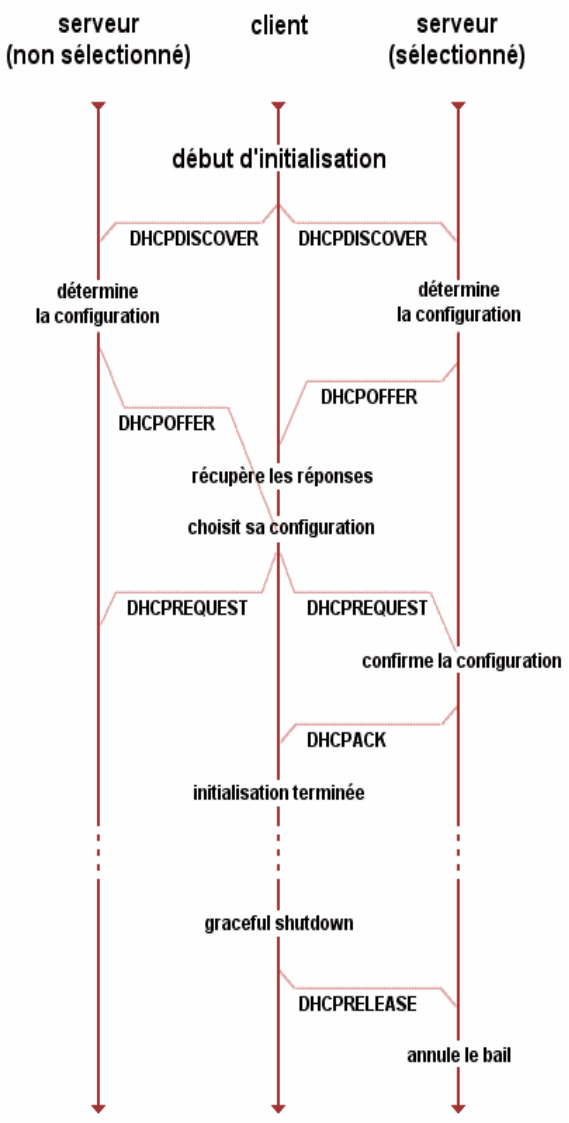

FIG. 3.1 – Messages de première configuration

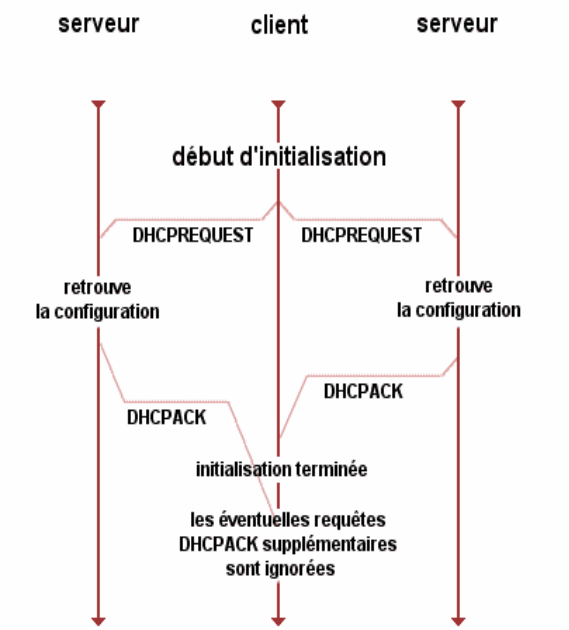

FIG. 3.2 – Messages de renouvellement de bail

les messages DHCP. Et pour mieux comprendre comment cela se passe, regardons de plus près le format des trames DHCP. La trame DHCP est en fait la même que BOOTP, et a le format de la figure 3.3 (les chiffres entre parenthèses indiquent la taille du champ en octets) dont l'explication des champs suit :

- op : vaut 1 pour BOOTREQUEST (requête client), 2 pour BOOTREPLY (réponse serveur)
- htype : type de l'adresse hardware (adresse MAC, par exemple) hlen : longueur de l'adresse hardware (en octet). C'est 6 pour une adresse MAC
- hops : peut être utilisé par des relais DHCP
- xid : nombre aléatoire choisi par le client et qui est utilisé pour reconnaître le client
- secs : le temps écoulé (en secondes) depuis que le client a commencé sa requête
- flags : flags divers
- ciaddr : adresse IP du client, lorsqu'il en a déjà une
- yiaddr : la (future ?) adresse IP du client
- siaddr : adresse IP du (prochain) serveur à utiliser
- giaddr : adresse IP du relais (passerelle par exemple) lorsque la connexion

directe client/serveur n'est pas possible
- chaddr : adresse hardware du client
- sname : champ optionnel. Nom du serveur
- file : nom du fichier à utiliser pour le boot
- options : Champs réservé pour les options. La taille de ce champ était limitée à 64 octets pour la première version de BOOTP ; maintenant, il n'y a plus de limitation. Dans tous les cas, un client DHCP doit être prêt à recevoir au minimum 576 octets, mais la possibilité lui est offerte de demander au serveur de restreindre la taille de ses messages.

Et pour ce dernier champ des options, le détail est sur la figure 3.4.

Les options portent toutes un numéro qui les identifie. Par exemple, l'option 15 est celle qui permet de donner au client le nom de domaine du réseau. Il est bien entendu possible d'envoyer plusieurs options dans le même message DHCP ; dans tous les cas, que l'on ne transmette qu'une seule option utile ou plusieurs, on doit toujours finir la zone d'options par une option 255 (end). Le numéro de l'option n'est codé que sur 1 octet, donc il ne peut y avoir que 256 options possibles. L'octet 2 code la longueur du champ de données qui suit ; il ne tient donc pas compte des 2 octets de code d'option et de longueur. Certaines options ne comportent pas de données complémentaires, comme l'option 255. Dans ce cas, il n'y a ni champ de longueur ni champ de données. Les messages DHCP pour la négociation de la configuration du client (DHCPACK...) sont tout simplement une option ! Il s'agit de l'option 53 qui comporte un champ de données de longueur 1 contenant le numéro identifiant la requête (1 pour DHCPDISCOVER...). Les 4 premiers octets du champ d'options doivent être initialisés respectivement avec les valeurs 99, 130, 83 et 99 (valeurs en décimal). Cette séquence est appelée le "magic cookie" (gâteau magique en français). Le client DHCP a la possibilité d'imposer au serveur DHCP une taille maxi pour le champ d'options (option 57). La conséquence d'une telle limitation est que le serveur peut manquer de place pour envoyer toutes les options souhaitées. Pour répondre à ce problème, le serveur est autorisé à utiliser les champs sname et file pour finir son envoi. Le client est averti de cet usage par une option 52 dans la zone d'options.

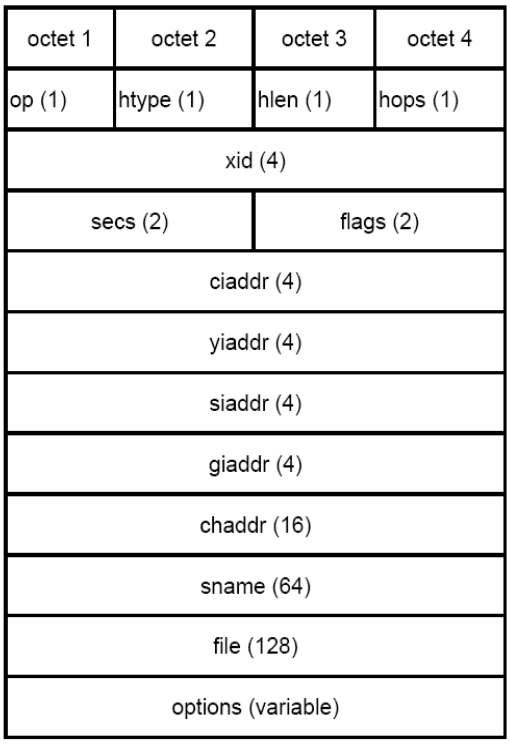

FIG. 3.3 – La trame DHCP

FIG. 3.4 – le champ option de la trame DHCP

3.2 Le serveur DHCPD dans VeriNeC

Pour pouvoir intégrer le DHCP dans VeriNeC, il nous a fallu trouver une solution DHCP qui, à l'instar des services DNS vus plus haut, permet une configuration de la machine à partir d'un fichier de configuration principal. Notre choix s'est porté sur le serveur DHCPD [11] qu'on retrouve dans des distributions Linux importantes comme RedHat, Suse, Mandrake, Gentoo, etc.

La configuration du serveur DHCPD se fait à travers le fichier dhcpd.conf qu'on retrouve dans le dossier /etc/. Le fichier dhcpd.conf est essentiellement une suite de paramètres et de déclarations. Tandis qu'il y a plusieurs dizaines d'options, le nombre des déclarations est plus limité : on a essentiellement shared-network, subnet, group, pool et host. Nous avons ci-dessous un exemple de fichier dhcpd.conf pour nous fixer les idées :

```
max-lease-time 240;
default-lease-time 240;
deny unknown-clients; option
domain-name "bar.com";
option domain-name-servers foo1.bar.com, foo2.bar.com;

subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.2 192.168.1.100;
    range 192.168.1.110 192.168.1.254;
    option broadcast-address 192.168.1.255;
}

group {
    option routers 192.168.2.101;

    host foo3 {
        hardware ethernet 00:c0:c3:11:90:23;
        option host-name pc3;
    }

    host foo4 {
        hardware ethernet 00:c0:c3:cc:0a:8f;
        fixed-address 192.168.1.105;
    }
}
```

```
host foo5 {
    hardware ethernet 00:c0:c3:2a:34:f5;
    server-name "bootp.bar.com";
    filename "boot";
}
```

3.3 Implémentation

Du point de vue de l'implémentation, nous avons ajouté une section au schema du noeud pour qu'il prenne en compte le DHCP, nous avons écrit un exemple de noeud XML et enfin nous avons écrit un traducteur de manière à obtenir en output un fichier de configuration tel que celui ci-dessus. Tous ces éléments sont à retrouver dans l'annexe à cette documentation.

Dans le Schema : A cause du grand nombre des options, nous n'avons pas prévu une définition d'élément pour chacune d'elles, à la différence des déclarations dont le nombre est plus limité. Elles sont au nombre de six :

- La déclaration shared-network est utilisée pour informer le serveur DHCP que certains sous-réseau IP partagent en réalité le même réseau physique.
- La déclaration subnet est utilisée pour que DHCPD puisse déterminer si une adresse IP appartient à un sous-réseau. Elle peut aussi être utilisée pour fournir des paramètres spécifiques au sous-réseau et pour spécifier quelles adresses peuvent être dynamiquement allouées aux clients démarrant sur ce sous-réseau. De telles adresses sont spécifiées en utilisant la déclaration range.
- La déclaration range définit la plage d'adresses quand les adresses IP sont alloués dynamiquement sur un sous-réseau.
- La déclaration host est utilisé dans le cas d'allocation d'une adresse IP fixe à un ou des clients connus.
- La déclaration pool est utilisée pour regrouper dans un même sous-réseau des plages d'adresses. Par exemple, un pool pour les clients connus et un pool pour les clients inconnus.
- La déclaration group est utilisée tout simplement pour appliquer un ou plusieurs paramètres à un groupe de déclaration. Elle peut être utilisée pour grouper hôtes, réseaux partagés, sous-réseaux, ou même d'autres groupes.

Dans le traducteur : Ici également, l'élément `<xsl:text>` a été d'une grande aide pour ajouter du texte à l'output. Notons également une utilisation de l'élément `<xsl:template>` qui est intéressante à détailler ici.

Etant donné que le même élément du document XML pouvait apparaître à différents niveaux de l'arbre, nous avons découvert la limite de l'élément `<xsl:for-each>` que nous avions utilisé dans le cas du traducteur de DJBDNS. Là, on pouvait utiliser `<xsl:for-each>`, car tous les éléments étaient au même niveau, le fichier data étant très peu arborescent. Au niveau du fichier dhcpd.conf, des difficultés ont surgit car c'est un fichier très arborescent, la déclaration subnet et la déclaration shared-network pouvant être au même niveau ou la déclaration shared-network pouvant contenir la déclaration subnet et la déclaration group pouvant contenir des déclarations shared-network ou subnet etc. Ainsi, il a fallu utilisé l'attribut select de l'élément `<xsl:apply-templates>` pour appliquer les définitions sur toutes les occurences de l'élément dans le document XML. De cette manière, le template est appliqué à toutes les occurences de l'élément sélectionné sans qu'on ait à utiliser l'élément `<xsl:for-each>`. Pour nous fixer les idées et mieux comprendre ce qui est dit ci-dessus, considérons les extraits suivant de notre traducteur de DHCPD :

Le template suivant est appliqué à toutes les occurences de l'élément `<vn:option-param>` grâce à la dernière ligne où `<xsl:apply-templates>` est utilisé avec son attribut select.

```
<xsl:template match="vn:option-param">
        <!--parameter with option begin-->
        <xsl:text>option </xsl:text>
        <xsl:value-of select="@param-name"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="@instance"/>
        <xsl:text>;</xsl:text>
        <xsl:value-of select="$Enter"/>
    </xsl:template>


<xsl:apply-templates select="vn:option-param"/>
```

Chapitre 4

Conclusion et perspectives

A la fin de ce travail, nous retenons les éléments positifs suivants : Nous avons pu mieux comprendre le DNS et le DHCP, nous avons pu améliorer notre connaissance des langages XML, XSLT et XML Schema. Nous nous sommes limités à DJBDNS et à DHCPD mais notons qu'il existe d'autres solutions DNS [12] comme ANS/CNS, PowerDNS, NSD, MaraDNS, Dents, etc. ainsi que d'autres solutions DHCP [13] tant commerciales que non commerciales répertoriées sur la page DHCP FAQ. Nous avons également mieux compris le fonctionnement des fichiers de configuration dans le monde linux. C'était également l'occasion de se familiariser avec LaTeX, étant donné que la documentation du projet devait obligatoirement se faire avec cet outil. Notons enfin comme remarque positive que grâce à ce projet, il nous est plus aisé d'ajouter d'autres services ayant la même architecture (surtout la présence d'un fichier de configuration principal) dans VeriNeC ou d'entreprendre un travail similiaire dans un autre projet.

Comme effort à faire, il s'agira de continuer à s'efforcer de pouvoir travailler en groupe. Comme nous avions un projet qui s'insère dans un plus large projet, il y avait le défi de pouvoir comprendre ce que d'autres ont fait de manière à pouvoir y intégrer sa pierre. Et comme dans les projets des années précédentes, cet exercice n'était pas simple, d'où la nécessité de continuer de faire des efforts, étant donné que le monde du travail est le monde par excellence où on travaille en groupe.

Chapitre 5

Annexes

5.1 dns.xml

Voici un exemple de configuration DNS dans VeriNeC :

```
<?xml version="1.0" encoding="UTF-8"?> <dns>
    <variable name="toonsdomain" value=""/>
    <Zone match="toons.foo.net." type="master" primaryns="ns.toons.foo.net."
        adminmail="root.toons.foo.net." serial="2004031700" refresh="10800"
        retry="3600" expire="604800" min_ttl="86400">
        <dnsNS match="toons.foo.net." target="gateway00.toons.foo.net."/>
        <dnsIPRange network="127.0.0">
            <description>local</description>
            <dnsA match="localhost" target="1"/>
        </dnsIPRange>
        <dnsIPRange network="192.168.15">
            <description>network</description>
            <dnsA match="toons.foo.net." target="95"/>
            <dnsA match="diufpc55" target="95"/>
            <dnsA match="sly" target="90"/>
            <dnsA match="ben" target="101"/>
            <dnsA match="roadrunner" target="254"/>
        </dnsIPRange>
        <dnsCNAME match="www" target="ben"/>
        <dnsCNAME match="ftp" target="sly"/>
        <dnsCNAME match="awesso" target="www"/>
        <dnsMX match="toons.foo.net." target="mail.toons.foo.net"
            priority="10"/>
    </Zone>
```

```
</dns>
```

5.2 djbdns.xsl

Ici, c'est un traducteur qui va permettre de générer le fichier data de DJBDNS :

```
<?xml version="1.0" encoding="utf-8"?> <xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:vn="http://diuf.unifr.ch/tns/projects/verinec/node"
xmlns:tr="http://diuf.unifr.ch/tns/projects/verinec/translation"
xmlns="http://diuf.unifr.ch/tns/projects/verinec/configuration" version="1.0">
    <xsl:variable name="Enter" select="'&#x00A;'"/>
    <xsl:variable name="data_dir" select="'/etc/tinydns/root'"/>
    <!-- <xsl:variable name="domain_name" select="//dnsNS/@match"/>-->
    <xsl:template match="/">
        <xsl:apply-templates select="vn:node/vn:services/vn:dns"/>
    </xsl:template>
    <xsl:template match="vn:dns">
        <service name="dns">

            <xsl:copy-of select="../../tr:nodetype/
            tr:service[@name='dns']/tr:target"/>
            <!-- <result-file filename="/etc/tinydns/root/data">
      </result-file>-->
            <xsl:apply-templates/>
        </service>
    </xsl:template>
    <xsl:template match="vn:Zone">
        <result-file>
            <xsl:attribute name="filename"><xsl:value-of select="$data_dir"/>
            <xsl:text>/data</xsl:text></xsl:attribute>
            <xsl:text>
# #How the data file looks # </xsl:text>
            <xsl:for-each select="vn:dnsIPRange">
                <xsl:call-template name="A-Entries"/>
            </xsl:for-each>
            <xsl:for-each select="vn:dnsNS">
                <xsl:text>=</xsl:text>
                <xsl:value-of select="@target"/>
```

```
            <xsl:text>:</xsl:text>
            <xsl:value-of select="@match"/>
            <xsl:value-of select="$Enter"/>
            <xsl:text>.</xsl:text>
            <xsl:value-of select="@match"/>
            <xsl:text>:</xsl:text>
            <xsl:value-of select="@target"/>
            <xsl:value-of select="$Enter"/>
        </xsl:for-each>
        <xsl:for-each select="vn:dnsMX">
            <xsl:text>@</xsl:text>
            <xsl:value-of select="@match"/>
            <xsl:text>:</xsl:text>
            <xsl:value-of select="@target"/>
            <xsl:text>:</xsl:text>
            <xsl:value-of select="../@min_ttl"/>
            <xsl:value-of select="$Enter"/>
        </xsl:for-each>
        <xsl:for-each select="vn:dnsTXT">
            <xsl:text>'</xsl:text>
            <xsl:value-of select="@match"/>
            <xsl:text>:</xsl:text>
            <xsl:value-of select="@target"/>
            <xsl:text>:</xsl:text>
            <xsl:value-of select="../@min_ttl"/>
            <xsl:value-of select="$Enter"/>
        </xsl:for-each>
        <xsl:for-each select="vn:dnsCNAME">
            <xsl:text>C</xsl:text>
            <xsl:value-of select="@match"/>
            <xsl:text>.</xsl:text>
            <xsl:value-of select="../@match"/>
            <xsl:text>:</xsl:text>
            <xsl:value-of select="@target"/>
            <xsl:text>.</xsl:text>
            <xsl:value-of select="../@match"/>
            <xsl:text>:</xsl:text>
            <xsl:value-of select="../@min_ttl"/>
            <xsl:value-of select="$Enter"/>
        </xsl:for-each>
    </result-file>
```

```
</xsl:template>
<xsl:template name="A-Entries">
    <xsl:for-each select="vn:dnsA">
        <xsl:choose>
            <xsl:when test="substring(@match, string-length(@match))='.'">
                <xsl:text>+</xsl:text>
                <xsl:value-of select="@match"/>
                <xsl:text>:</xsl:text>
                <xsl:value-of select="../@network"/>
                <xsl:text>.</xsl:text>
                <xsl:value-of select="@target"/>
                <xsl:text>:</xsl:text>
                <xsl:value-of select="../../@min_ttl"/>
                <xsl:value-of select="$Enter"/>
                <xsl:text>=</xsl:text>
                <xsl:value-of select="@match"/>
                <xsl:text>:</xsl:text>
                <xsl:value-of select="../@network"/>
                <xsl:text>.</xsl:text>
                <xsl:value-of select="@target"/>
                <xsl:text>:</xsl:text>
                <xsl:value-of select="../../@min_ttl"/>
                <xsl:value-of select="$Enter"/>
                <!--the idea is to link the domain name to the IP address
                so that when the NS links the domain name to
                the name server, this one finds its IP address-->
            </xsl:when>
            <xsl:otherwise>
                <xsl:text>+</xsl:text>
                <xsl:value-of select="@match"/>
                <xsl:text>.</xsl:text>
                <xsl:value-of select="../../@match"/>
                <xsl:text>:</xsl:text>
                <xsl:value-of select="../@network"/>
                <xsl:text>.</xsl:text>
                <xsl:value-of select="@target"/>
                <xsl:text>:</xsl:text>
                <xsl:value-of select="../../@min_ttl"/>
                <xsl:value-of select="$Enter"/>
            </xsl:otherwise>
        </xsl:choose>
```

```
        </xsl:for-each>
    </xsl:template>
</xsl:stylesheet>
```

5.3 dhcp.xsd

Partie que nous avons rajoutée au schema d'un noeud dans VeriNeC de manière à ce que le DHCP puisse être pris en charge :

```
<xs:element name="dhcp">
        <xs:annotation>
            <xs:documentation>Dynamic host configuration protocole
            aims to generate dhcpd.conf</xs:documentation>
        </xs:annotation>
        <xs:complexType>
         <xs:sequence>
          <xs:element ref="option-param" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="other-param" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="authority" minOccurs="0"/>
          <xs:element ref="shared-network" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="subnet" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="group" minOccurs="0" maxOccurs="unbounded"/>
         </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="option-param">
        <xs:annotation>
            <xs:documentation>parameters which line begins with"option".
            They are aroud 60. The instance can contain
            spaces and commas</xs:documentation>
        </xs:annotation>
        <xs:complexType>
            <xs:attribute name="param-name" type="xs:string" use="required"/>
            <xs:attribute name="instance" type="xs:string" use="optional"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="statement">
     <xs:annotation>
      <xs:documentation>parameters which line does not begin with "option".
```

```
  The value may contain spaces and commas. The server-identifier declar.
  is an example of the statement element. allow and deny parameters
  are also considered statements.</xs:documentation>
 </xs:annotation>
 <xs:complexType>
  <xs:attribute name="value" type="xs:string" use="required" />
 </xs:complexType>
</xs:element>
<xs:element name="host-param">
    <xs:annotation>
        <xs:documentation>parameters in the host element.
        card attribute can be ethernet or token-ring
</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attribute name="card" type="xs:string" use="required"/>
        <xs:attribute name="addr" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="range">
    <xs:annotation>
        <xs:documentation>define a range of IP addresses
</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:attribute name="begin" type="xs:string" use="required"/>
        <xs:attribute name="finish" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="shared-network">
    <xs:annotation>
        <xs:documentation>shared netork declaration</xs:documentation>
    </xs:annotation>
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="option-param" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="other-param" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="authority" minOccurs="0"/>
      <xs:element ref="subnet" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="pool" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
```

```
     <xs:attribute name="param-name" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="subnet">
    <xs:annotation>
        <xs:documentation>subnet declaration</xs:documentation>
    </xs:annotation>
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="option-param" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="other-param" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="authority" minOccurs="0"/>
      <xs:element ref="pool" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="range"/>
     </xs:sequence>
     <xs:attribute name="addr" type="xs:string" use="required"/>
     <xs:attribute name="netmask" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="group">
    <xs:annotation>
        <xs:documentation>groups all kinds of parameters and
        declarations including group declarations</xs:documentation>
    </xs:annotation>
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="option-param" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="other-param" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="shared-network" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="subnet" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="group" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="host" minOccurs="0" maxOccurs="unbounded"/>
     </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="host">
    <xs:annotation>
        <xs:documentation>host declaration</xs:documentation>
    </xs:annotation>
    <xs:complexType>
     <xs:sequence>
```

```
        <xs:element ref="option-param" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="other-param" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="host-param"/>
       </xs:sequence>
       <xs:attribute name="host-name" type="xs:string" use="required"/>
      </xs:complexType>
  </xs:element>
  <xs:element name="authority">
      <xs:annotation>
          <xs:documentation>authority declaration.
          authoritative attribute is true or false.
          When nothig is defined, default is authoritative</xs:documentation>
      </xs:annotation>
      <xs:complexType>
       <xs:attribute name="authoritative" type="xs:boolean" use="required"/>
      </xs:complexType>
  </xs:element>
  <xs:element name="pool">
      <xs:annotation>
          <xs:documentation>pool declaration</xs:documentation>
      </xs:annotation>
      <xs:complexType>
       <xs:sequence>
        <xs:element ref="option-param" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="other-param" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="range"/>
       </xs:sequence>
      </xs:complexType>
  </xs:element>
```

5.4 dhcp.xml

Un exemple de configuration de noeud pour le DHCP :

```
<?xml version="1.0"?> <dhcp
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\VeriNeC\dhcp\dhcp.xsd">
    <!--parameters-->
    <option-param param-name="domain-name" instance="toons.foo.net"/>
    <option-param param-name="domain-name-servers"
```

```
    instance="ns1.toons.foo.net, ns2.toons.foo.net"/>
    <other-param value="server-name diuf-dhcp"/>
    <authority authoritative="true"/>
    <!--declarations-->
    <shared-network param-name="awesso-toons">
        <other-param value="filename boot"/>
        <subnet addr="204.254.239.32" netmask="255.255.255.224">
            <option-param param-name="domain-name" instance="toons.foo.net"/>
            <option-param param-name="domain-name-servers"
            instance="ns.toons.foo.net"/>
            <other-param value="server-name dhcp-server"/>
            <range begin="204.254.239.42" finish="204.254.239.62"/>
        </subnet>
    </shared-network>
    <subnet addr="204.254.239.64" netmask="255.255.255.224">
        <option-param param-name="domain-name" instance="toons.foo.net"/>
        <option-param param-name="domain-name-servers"
        instance="ns.toons.foo.net"/>
        <other-param value="server-name dhcp-server"/>
        <range begin="204.254.239.74" finish="204.254.239.94"/>
    </subnet>
    <group>
        <other-param value="routers 204.254.239.1"/>
        <host host-name="host1">
            <host-param card="ethernet" addr="00:c0:c3:cc:0a:8f"/>
        </host>
        <host host-name="host2">
            <host-param card="ethernet" addr="00:c0:c3:2a:34:f5"/>
        </host>
    </group>
</dhcp>
```

5.5 dhcp.xsl

Un traducteur pour générer le fichier dhcpd.conf :

```
<?xml version="1.0" encoding="utf-8"?> <xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:vn="http://diuf.unifr.ch/tns/projects/verinec/node"
```

```
xmlns:tr="http://diuf.unifr.ch/tns/projects/verinec/translation"
xmlns="http://diuf.unifr.ch/tns/projects/verinec/configuration" version="1.0">
    <xsl:variable name="Enter" select="'&#x00A;'"/>
    <xsl:variable name="Space" select="'&#x020;'"/>
    <xsl:variable name="Tab" select="'&#x009;'"/>
    <xsl:template match="/">
        <xsl:apply-templates select="vn:node/vn:services/vn:dhcp"/>
    </xsl:template>
    <xsl:template match="vn:dhcp">
     <service name="dhcp">
      <xsl:copy-of select="../../tr:nodetype/
        tr:service[@name='dhcp']/tr:target"/>
            <xsl:value-of select="$Enter"/>
            <xsl:value-of select="$Enter"/>
            <xsl:value-of select="$Enter"/>
            <result-file filename="/etc/dhcpd.conf">
                <xsl:value-of select="$Enter"/>
                <xsl:value-of select="$Enter"/>
                <xsl:text>#dhcpd.conf</xsl:text>
                <xsl:value-of select="$Enter"/>
                <xsl:value-of select="$Enter"/>
                <xsl:apply-templates select="vn:authority"/>
                <xsl:apply-templates select="vn:option-param"/>
                <xsl:apply-templates select="vn:statement"/>
                <xsl:apply-templates select="vn:shared-network"/>
                <xsl:apply-templates select="vn:subnet"/>
                <xsl:apply-templates select="vn:group"/>
                <xsl:apply-templates select="vn:range"/>
                <xsl:apply-templates select="vn:host-param"/>
                <xsl:apply-templates select="vn:pool"/>
                <xsl:value-of select="$Enter"/>
                <xsl:value-of select="$Enter"/>
            </result-file>
            <xsl:value-of select="$Enter"/>
      </service>
    </xsl:template>
    <xsl:template match="vn:authority">
        <xsl:if test="@authoritative='true'">
            <xsl:text>authoritative;</xsl:text>
            <xsl:value-of select="$Enter"/>
        </xsl:if>
```

```
    <xsl:if test="@authoritative='false'">
        <xsl:text>not authoritative;</xsl:text>
        <xsl:value-of select="$Enter"/>
    </xsl:if>
</xsl:template>
<xsl:template match="vn:option-param">
    <!--parameter with option begin-->
    <xsl:text>option </xsl:text>
    <xsl:value-of select="@param-name"/>
    <xsl:text> </xsl:text>
    <xsl:value-of select="@instance"/>
    <xsl:text>;</xsl:text>
    <xsl:value-of select="$Enter"/>
</xsl:template>
<xsl:template match="vn:statement">
    <!--parameter without option begin-->
    <xsl:value-of select="@value"/>
    <xsl:text>;</xsl:text>
    <xsl:value-of select="$Enter"/>
</xsl:template>
<xsl:template match="vn:shared-network">
    <xsl:text>shared-network </xsl:text>
    <xsl:value-of select="@param-name"/>
    <xsl:text>{</xsl:text>
    <xsl:value-of select="$Enter"/>
    <xsl:apply-templates/>
    <xsl:text>}</xsl:text>
    <xsl:value-of select="$Enter"/>
</xsl:template>
<xsl:template match="vn:subnet">
    <xsl:text>subnet </xsl:text>
    <xsl:value-of select="@addr "/>
    <xsl:text> netmask  </xsl:text>
    <xsl:value-of select="@netmask"/>
    <xsl:text>{</xsl:text>
    <xsl:value-of select="$Enter"/>
    <xsl:apply-templates/>
    <xsl:text>}</xsl:text>
    <xsl:value-of select="$Enter"/>
</xsl:template>
<xsl:template match="vn:group">
```

```
        <xsl:text>group </xsl:text>
        <xsl:text>{</xsl:text>
        <xsl:value-of select="$Enter"/>
        <xsl:apply-templates/>
        <xsl:text>}</xsl:text>
        <xsl:value-of select="$Enter"/>
    </xsl:template>
    <xsl:template match="vn:range">
        <xsl:text>range </xsl:text>
        <xsl:value-of select="@begin "/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="@finish"/>
        <xsl:text>;</xsl:text>
        <xsl:value-of select="$Enter"/>
    </xsl:template>
    <xsl:template match="vn:host-param">
        <xsl:text>host </xsl:text>
        <xsl:value-of select="../@host-name"/>
        <xsl:text>{</xsl:text>
        <xsl:value-of select="$Enter"/>
        <xsl:value-of select="@card "/>
        <xsl:value-of select="$Space"/>
        <xsl:value-of select="@addr"/>
        <xsl:text>;</xsl:text>
        <xsl:value-of select="$Enter"/>
        <xsl:text>}</xsl:text>
        <xsl:value-of select="$Enter"/>
    </xsl:template>
    <xsl:template match="vn:pool">
        <xsl:text>pool </xsl:text>
        <xsl:text>{</xsl:text>
        <xsl:apply-templates/>
        <xsl:text>}</xsl:text>
    </xsl:template>
</xsl:stylesheet>
```

Bibliographie

[1] VeriNeC, http ://diuf.unifr.ch/tns/projects/verinec/

[2] RAY Eric, Introduction à XML, Oreilly, 2004, 352 pages.

[3] Fonctionnement du DNS, http ://www.zonedit.com/doc/dns-basics.html

[4] RFC 1034, http ://www.ietf.org/rfc/rfc1034.txt

[5] BIND, http ://www.isc.org/index.pl ?/sw/bind/

[6] http ://www.linux-sottises.net/bind.php

[7] Le site officiel de DJBDNS, http ://cr.yp.to/djbdns.html

[8] Les ports de DJBDNS, http ://www.itworld.com/

[9] Syntaxe du fichier de data, http ://www.abcdns.org/djbdns.php

[10] Introduction DHCP, http ://www.themanualpage.org/dhcp/

[11] DHCPD, http ://www.daemon-systems.org/man/dhcpd.conf.5.html

[12] http ://www.dns.net/dnsrd/servers/

[13] http ://www.dhcp-handbook.com/