Institut
Supérieur
d' **I**nformatique
de **M**odélisation
et de leurs
Applications
Campus des Cézeaux
BP 10125
63173 Aubière Cedex

Organisation
Européenne
pour la **R**echerche
Nucléaire
CH 1211
Genève 23 Suisse

Rapport d'ingénieur
Stage de 3ème année
Filière 1 : Informatique des systèmes embarqués

Réécriture du
Module réseau de Quattor
Pour l'expérience LHCb au CERN

Présenté par : **S. Cipière**
Responsable ISIMA : **E. Mesnard**
Responsable CERN : **L. Brarda**

Durée : 5 mois
avril à septembre
2010

Institut
Supérieur
d' **I**nformatique
de **M**odélisation
et de leurs
Applications
Campus des Cézeaux
BP 10125
63173 Aubière Cedex

Organisation
Européenne
pour la **R**echerche
Nucléaire
CH 1211
Genève 23 Suisse

Rapport d'ingénieur
Stage de 3ème année
Filière 1 : Informatique des systèmes embarqués

Réécriture du
Module réseau de Quattor
Pour l'expérience LHCb au CERN

Présenté par : **S. Cipière**
Responsable ISIMA : **E. Mesnard**
Responsable CERN : **L. Brarda**

Durée : 5 mois
avril à septembre
2010

Remerciements

Je tiens avant de commencer ce rapport à remercier toute l'équipe de l'expérience LHCb au CERN qui m'a accueilli chaleureusement et m'a accompagné durant les cinq mois de mon stage.

Je voudrais remercier particulièrement mon maître de stage Loïc Brarda qui m'a fait confiance et n'a pas hésité à me confier la responsabilité de réécrire le module réseau de Quattor. Je me dois aussi de remercier Enrico Bonaccorsi pour son aide précieuse et l'enthousiasme avec lequel il partage son expérience des réseaux avec ses collègues.

Pour mon dernier rapport en tant qu'étudiants à l'ISIMA je voudrais saluer tous mes professeurs qui ont su me faire partager leurs passions pour les systèmes d'information et les univers qui les entourent.

Acknowledgement

Before starting this report I wish to thanks the whole LHCb experiment's staff from the CERN that welcomed me warmly and accompanied me during the five months of my internship.

I wish to especially thank my internship's tutor Mr. L. Brarda who grants me all his confidence to rewrite the Quattor's network module. And I must thanks Mr. E. Bonaccorsi for his precious help and the enthusiasm with which he shares his experience with colleagues.

For my last report as a student I would like to thanks all my teachers whose shared with me their addiction for computer science and the universes around it.

Table des illustrations

Résumé

Quattor est une solution logicielle qui permet l'installation, la configuration et la maintenance de milliers d'ordinateurs à distance. En centralisant ainsi ces tâches les gains en termes de temps et de simplicité sont énormes. Pour atteindre ses objectifs, Quattor utilise, selon ce qui lui est demandé, différents modules, dont un dédié à la configuration des réseaux. Durant les 5 mois qu'ont durés mon stage au **CERN** et plus précisément dans l'expérience **LHCb**, il m'a été attribué la tâche de réécrire totalement depuis le départ, ce composant réseau, en y ajoutant des fonctionnalités. Il m'a été demandé de conserver les fonctionnalités existantes et de faire en sorte que le nouveau composant soit compatible avec les anciennes configurations. De plus, je devais ajouter une gestion de la configuration qui puisse se réaliser sans qu'il soit nécessaire de devoir redémarrer le service réseau du nœud à configurer. J'ai nommé cette configuration, « configuration dynamique » ou « configuration à la volée », car elle s'oppose pour moi à la configuration statique qui s'appuie sur les données écrites dans des fichiers et qui seront utilisées lors d'un redémarrage de la machine ou du service réseau.

Le nouveau **composant réseau** de Quattor sera capable de conformer des nœuds ne possédant pas de disques, sans les déconnecter du réseau. Il aura été entièrement repensé, et codé en suivant les principes de robustesse, d'efficacité, de maintenabilité et d'évolution.

Abstract

Quattor is a system administration toolkit providing a powerful, portable, and modular set of tools for the automated installation, configuration, and management of clusters and farms. It is developed as a community effort and provided as open-source software. This centralized remote tool allows you to win a lot of time and effort. To reach its goal Quattor is a sum of sub components, one of them is the network module. During my 5 months internship, in **CERN** and most sharply in the **LHCb experiment**, my supervisor Mr. Brarda asked me to fully rewrite this network module from scratch. My objectives were in the first time to conserve compatibility between the new module and the old configurations, and in a second time to add a possibility to apply the settings to the node “on fly”. In my view the “on fly” configuration is the opposite of the “static” configuration which use files writing and need to restart the service network of the node.

The new Quattor’s **network component** will be able to setup diskless node without disconnect them from the network. It will be fully rebuild and think from scratch, the new code will be written by keeping in mind the principles of sturdiness, efficiency, maintainability, and next features.

Table des matières

1 Introduction

J'ai effectué mon stage de troisième année au CERN Organisation Européenne de Recherche Nucléaire, sous la tutelle de L. BRARDA (CERN) et E. MESNARD (ISIMA).

Le CERN fournit aux scientifiques les moyens de réaliser des expériences très poussées dans le domaine de la physique des particules.

Aujourd'hui en août 2010, quatre expériences s'articulent autour de l'accélérateur de particules LHC et étudient ce qu'il se passe lors de collisions de particules. Chaque expérience repose en premier lieu sur un détecteur qui collecte une grande quantité de données qu'il faut analyser et enregistrer. Une partie de l'analyse est effectuée alors que les données transitent sur le système d'acquisition pour réduire le volume à enregistrer.

Mon objectif était de réécrire depuis zéro le module réseau de Quattor, en y ajoutant une fonctionnalité qui permet de configurer les différents paramètres liés au réseau sans redémarrer le service réseau.

2 Contexte

2.1 Le CERN

Fondé en 1954, le CERN (Organisation Européenne pour la Recherche Nucléaire) est devenu un exemple éclatant de collaboration internationale, comptant aujourd'hui 20 états membres. Situé de part et d'autre de la frontière franco-suisse, près de Genève, c'est le plus grand laboratoire de physique des particules du monde.

Au CERN, le grand collisionneur de hadrons accélère des faisceaux de protons à une vitesse proche de celle de la lumière. Les protons entrent en collision, recréant au centre de l'expérience LHCb les conditions qui prévalaient juste après le Bing Bang.

Quand de la matière et de l'antimatière entrent en contact, les conséquences sont dramatiques. En un clin d'œil, toutes deux disparaissent, se détruisant mutuellement dans un flash d'énergie. Cette relation explosive soulève de fascinantes questions. Par exemple, si la matière et l'antimatière ont été créées en quantités égales au moment du Big Bang, pourquoi vivons-nous dans un Univers uniquement formé de matière ? Un mécanisme inconnu serait-il intervenu pour empêcher matière et antimatière de se détruire complètement ?

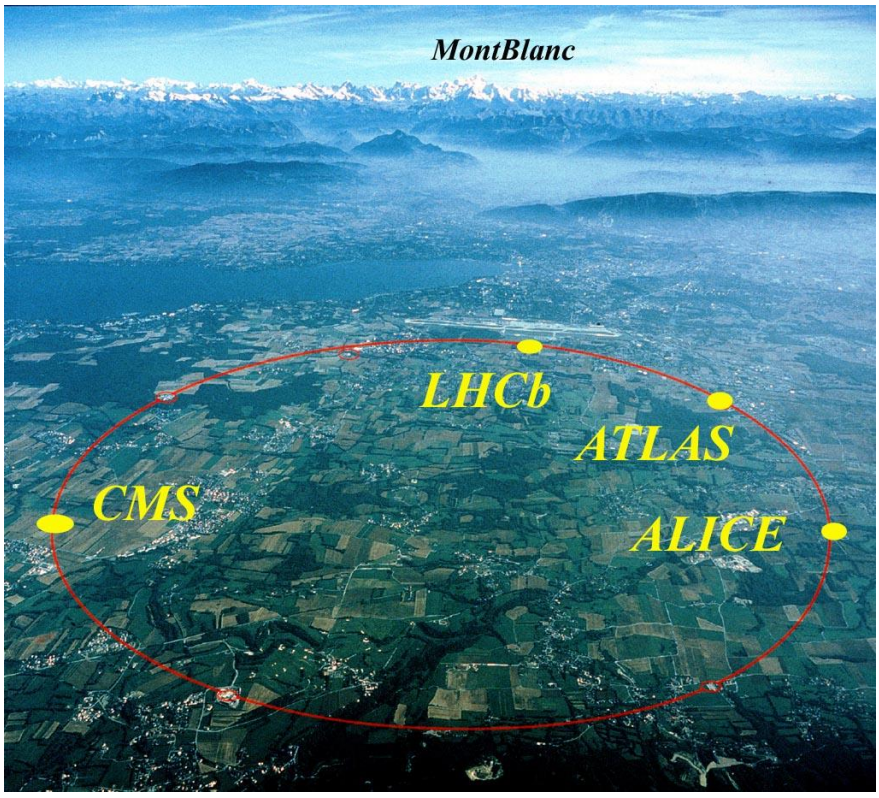

Illustration 1 : **Superposition du LHC sur une vue aérienne**

2.2 L'expérience LHCb

Le LHCb *Large Hadron Collider beauty*, est une collaboration internationale comptant 700 scientifiques de 50 universités et laboratoires dans 15 pays, aidés par plusieurs centaines de techniciens et d'ingénieurs. La plupart des composants du détecteur ont été conçus et fabriqués loin du site du CERN. Situé dans une grande caverne à 100 mètres sous terre, LHCb est un alignement de couches de détection, chacune conçue pour identifier et mesurer une propriété différente des particules issues des collisions.

Au lieu de fuser dans toutes les directions, les quarks de beautés générés par les collisions de protons filent dans des directions proches de la ligne du faisceau. Cette topologie caractéristique a déterminé l'architecture du détecteur. Les sous détecteurs sont alignés les uns derrière les autres sur 20 mètres, comme des livres rangés sur une étagère.

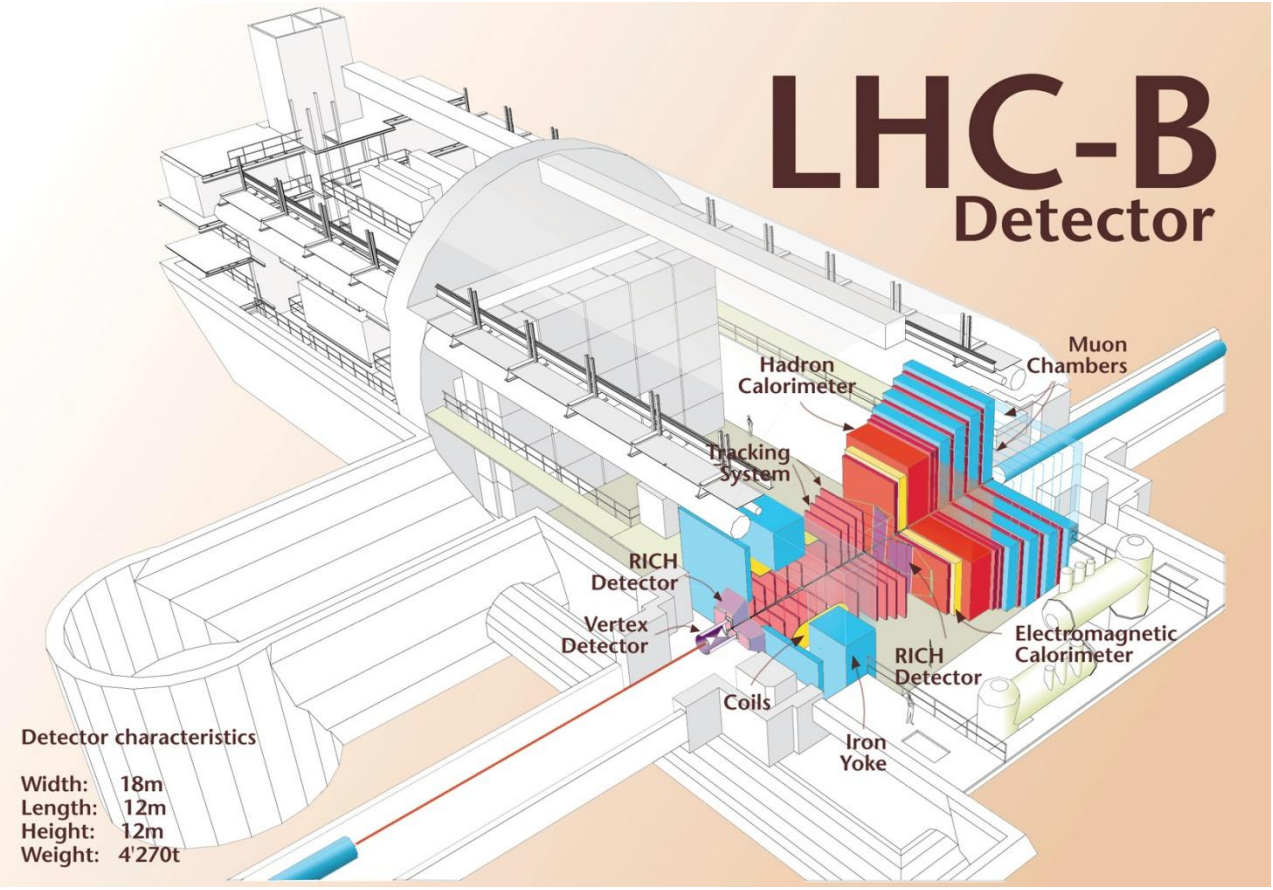

Illustration 2 : **Les différents capteurs constituants l'expérience LHCb**

LHCb a été conçu pour étudier les légères asymétries entres matière et antimatière à partir de particules connues sous le nom de quarks de beauté. Bien qu'absent de l'Univers actuel, les quarks de beautés existaient en grandes quantités après le Big Bang. Les collisions au cœur de l'expérience LHCb en produiront des milliards tout comme elles produiront des milliards d'antiquarks de beauté.

En étudiant avec une précision inégalée la légère différence entre la désintégration du quark et de l'antiquark de beauté, LHCb contribue à dissiper l'un des plus grands mystères de l'Univers.

Fugaces, les quarks de beauté ne subsistent qu'un millionième de millionième de seconde dans LHCb avant de se décomposer en particules plus légères. C'est cependant suffisant pour qu'ils soient affectés par de subtils effets quantiques, signalant leur présence.

LHCb est également à l'affût d'indices de l'existence d'une nouvelle famille de particules qui pourrait former une partie de la matière sombre de notre Univers. Cette matière inconnue se manifeste en faisant tourner des galaxies plus vite que prévu et en déviant la lumière des étoiles.

LHCb n'enregistre que 2000 des 10 millions de collisions de protons se produisant à chaque seconde. Cette sélection des collisions les plus intéressantes est l'œuvre du système de déclenchement électronique qui mène une analyse extrêmement rapide à partir des quelques sous détecteurs clés. Le système de déclenchement prend une décision préliminaire en seulement quatre millionièmes de seconde, sans quoi les données sont perdues. Ce n'est là qu'un des défis informatiques de LHCb. La quantité d'informations est suffisante pour remplir 300 CD par heure. Les données de l'expérience sont ensuite analysées à l'aide de plus de 100 centres de calcul répartis dans le monde entier. Des dizaines de milliers d'ordinateurs peuvent ainsi traiter simultanément les données sur ce nouveau système global appelé la Grille. Et c'est à ce niveau que Quattor intervient, car face à un si grand nombre de machines distantes, une solution de configuration centralisée apparaît être une nécessité.

Le parc informatique de l'expérience LHCb, peut se schématiser comme nous le montre l'illustration 3, en trois sous domaine, un dédié au control, un à l'acquisition et le troisième à la surveillance. Il comporte actuellement plus de 800 nœuds dont 500 sont des systèmes sans disques. Et il est prévu de nouveaux investissements qui doubleront le nombre de ces nœuds dans le courant de l'année 2010.

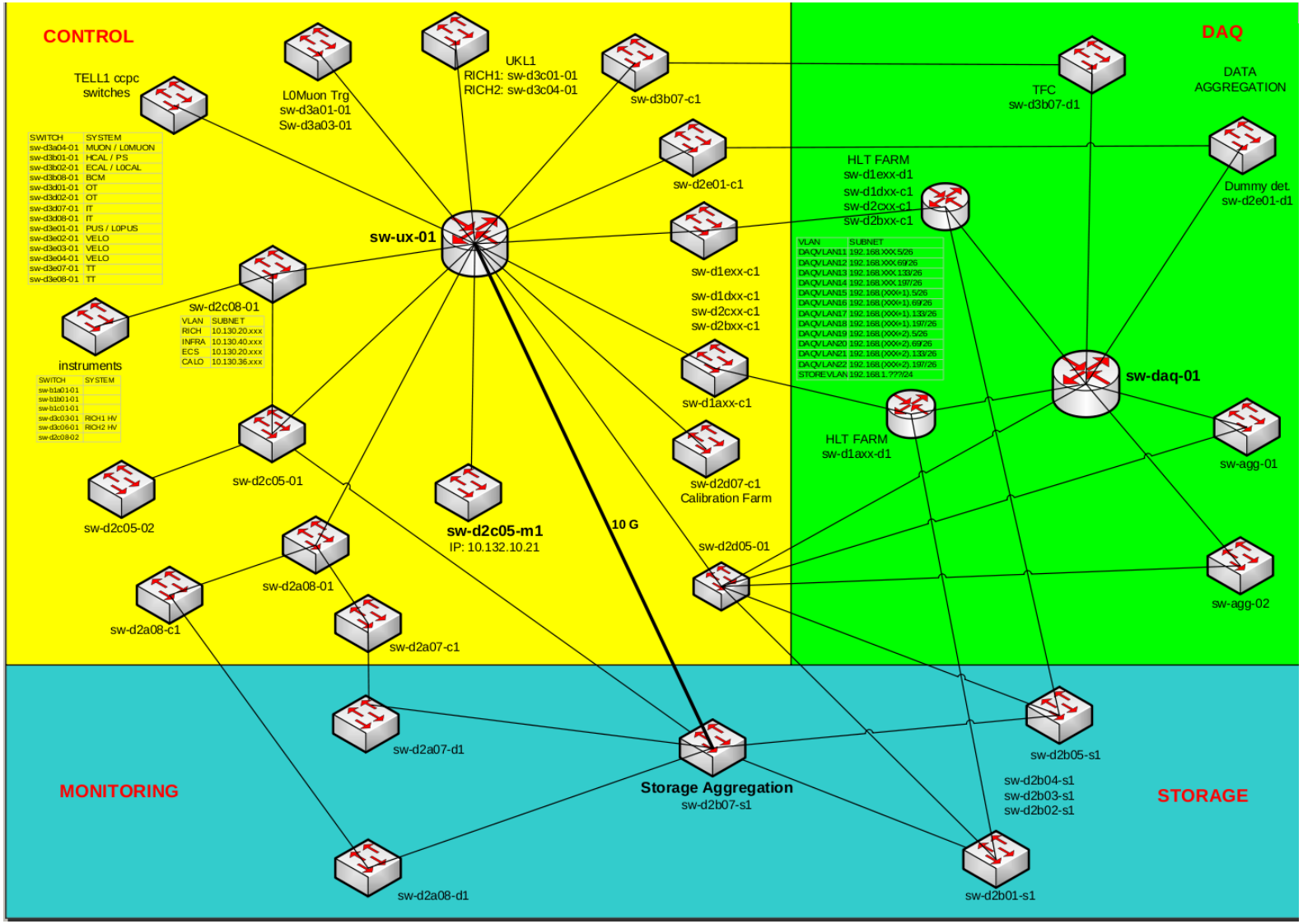

Illustration 3 : **Représentation de la structure du réseau LHCb**

2.3 Quattor

Quattor est un système d'administration, qui permet l'installation, la configuration et la gestion de grappe et de grille. Il est développé par une communauté qui permet de fournir un code open source et de distribuer le logiciel sous licence libre (*EU DataGrid Software License*, *Apache License V2.0*). Aujourd'hui Quattor est présent sur une dizaine de sites en Europe dont celui du CERN où 7 000 nœuds sont déployés sous son contrôle.

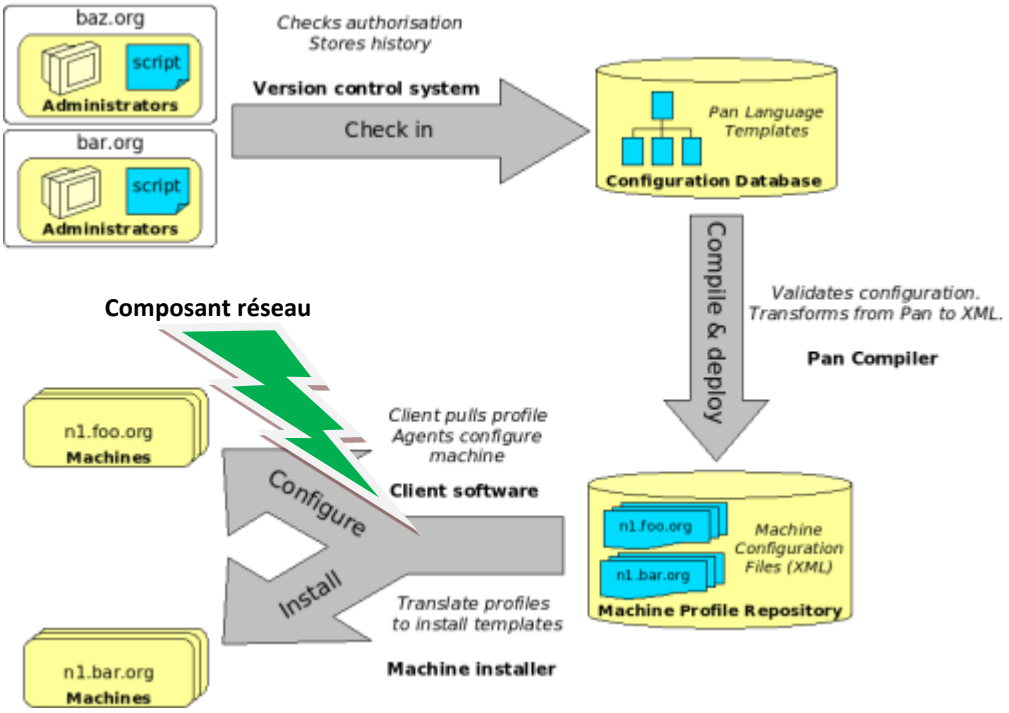

Illustration 4 : **Représentation du fonctionnement de Quattor**

L'intérêt d'une solution telle que celle proposée par Quattor est que lorsque vous souhaitez modifier la configuration d'une machine vous n'avez pas à vous déplacer, ou même accéder à la machine à distance pour rentrer manuellement chaque commande. La configuration de chaque machine est décrite par un ensemble de fichiers textes en cascade (nommés templates), le but étant de définir les paramètres communs à plusieurs machines dans un seul fichier qui sera inclus dans d'autre templates (pour simplifier, la template correspondant à une machine contient uniquement les informations qui la concerne elle uniquement, et inclus une template avec les paramètres concernant un type de machine, et ainsi de suite jusqu'à inclure une template où son définis les paramètres communs à toutes les machines). Ces templates sont écrites dans un language spécifique (PAN) et sont ensuite compilées pour produire un fichier XML par machine. Les machines dont la configuration a changé sont ensuite notifiées et viennent récupérer leur fichier XML. Celui-ci est alors analysé et les modifications de configuration sont appliquées par Quattor et les différents composants qui le composent. Ces composants sont écris en PERL et sont chargés de configurer un sous-ensemble du système : par exemple, l'authentification, le système d'impression ou pour ce qui nous concerne ici, le réseau. Une API spécifique, décrite au chapitre 3.3 nous permets d'accéder au contenu du fichier xml et d'interagir avec le système. Pour l'instant, Quattor est compatible uniquement avec les environnements de type RedHat, le CERN et LHCb utilisant *Scientific Linux CERN* (SLC), un dérivé de RedHat Enterprise Linux (RHEL), dans les versions 4 et 5. C'est donc ce système qui était mon système cible.

2.4 L'équipe d'administration LHCb online

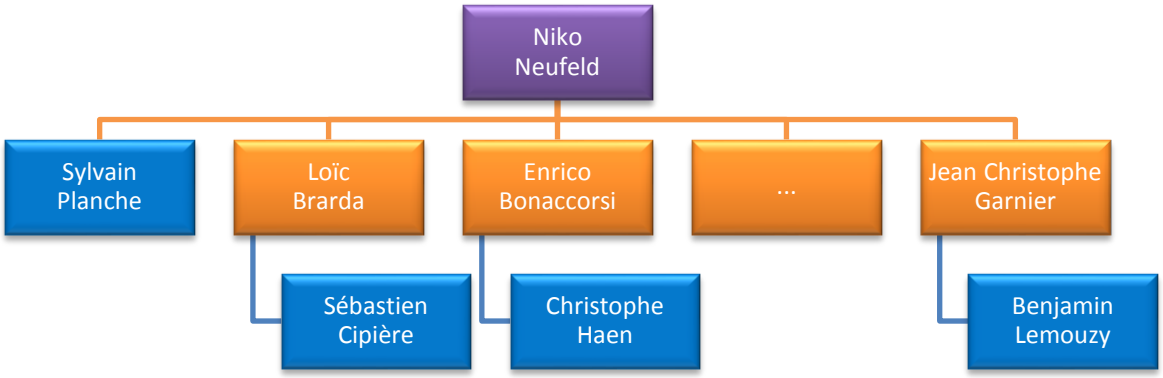

Illustration 5 : **Organigramme de l'équipe d'administration LHCb online**

Mon poste se situait au sein de l'équipe d'administration LHCb online, qui est l'une des composantes du LHCb, inclus dans le département PH (physique), qui est l'un des nombreux départements du CERN. Mon maître de stage était M. Loïc Brarda, qui est l'un des administrateurs système de l'expérience LHCb, il est de plus le spécialiste Quattor du LHCb. La structure réelle est trop complexe pour être représentée simplement de façon schématique, c'est pourquoi j'ai pris le parti de ne vous présenter qu'un organigramme simplifié du LHCb online, et de mes interlocuteurs privilégiés.

2.5 Environnement de développement et logiciels

Illustration 6 : **Logo du système d'exploitation Windows Vista**

Durant mon stage j'ai été amené à développer dans le monde Windows et Linux, sur ma machine personnelle, un ordinateur portable Packard Bell équipé d'un processeur Intel Core Duo T2250 cadencé à 1.73GHz, de 2Go de RAM, sous Windows Vista SP2. Cette machine disposée de deux systèmes d'exploitation et j'ai pu travailler sous Ubuntu 10.04 LTS *Lucid Lynx.*

Illustration 7 : **Logo du système d'exploitation Ubuntu**

Pour toutes les tâches bureautiques, rédactionnelles et organisationnelles j'ai utilisé Microsoft Office 2007 que j'ai complété grâce au partenariat entre l'ISIMA et *MSDN Academic Alliance*, qui m'a notamment donné accès à Project 2007 avec lequel j'ai pu établir le diagramme de Gantt de mon projet, avant de m'essayer à Planner une solution libre.

Illustration 8 : **Logo de la suite bureautique Microsoft Office 2007**

En ce qui concerne l'accès aux scripts en perl et l'édition rapide de code, encore une fois je me suis tourné vers une solution gratuite sous la forme de Notepad++. La version que j'utilise actuellement est la V5.4.5.

Illustration 9 : **Logo de l'éditeur de texte Notepad++**

Sous Ubuntu où la majorité de mon code a été développé je suis resté fidèle à Kate.

Illustration 10 : **Logo de l'éditeur de texte Kate**

Les diagrammes que je présente et que j'ai réalisés l'ont été à l'aide de DIA dans sa version 0.96.1, qui est un logiciel lui aussi libre. Il est inspiré de Microsoft Visio mais je le trouve plus précis et simple d'utilisation que ce dernier.

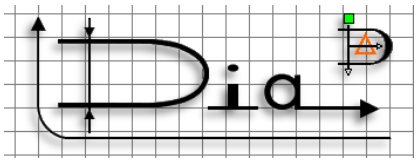

Illustration 11 : **Logo de l'éditeur de diagramme Dia**

La capture d'écran est assurée par Capturino dans sa version 2.0 beta écrit par Jean-Paul BELLENGER et distribué gratuitement. Ce logiciel dispose de toutes les fonctionnalités nécessaires à un excellent logiciel de capture d'écran et m'a permit de réaliser celles qui illustrent ce rapport. Pour la partie des captures réalisées sous Ubuntu j'ai retenu Ksnaphot.

Illustration 12 : **Icône du logiciel de capture d'écran Capturino v2**

Pour l'accès distant aux différentes machines sur lesquelles j'ai eu à intervenir sous Ubuntu j'ai opté pour la console Konsole qui a la particularité de personnaliser rapidement le style graphique de la console. Solution que j'utilisais pour pouvoir distinguer facilement mes machines distantes les unes des autres en attribuant une couleur de console à chacune.

Illustration 13 : **Icône du logiciel de console Konsole**

Sous l'univers Windows j'ai choisi les deux désormais célèbre putty et WinSCP pour récupérer mes fichiers sur les serveurs et mes connexions en SSH aux serveurs du LHCb online. Ces deux logiciels utilisables sans licence sont très facilement téléchargeables sur Internet.

Illustration 14 : **Logo du logiciel WinSCP**

Illustration 15 : **Icône du logiciel PuTTy**

3 Le langage Perl

Le composant réseau de Quattor doit être écrit en perl, c'est un langage que je ne connaissais pas et que j'ai appris à maîtriser durant mon stage.

3.1 Présentation

Le langage Perl est un langage de programmation informatique qui s'appuie sur différents autres langages. Les personnes habituées au langage C et aux scripts en Shell ne seront pas dépaysées lorsqu'elles écriront leurs premières lignes en Perl.

C'est un langage qui est beaucoup usité dans l'administration système et également dans le monde de la bioinformatique.

L'une des grandes forces du Perl est sa capacité à gérer les expressions régulières ou expressions rationnelles (ER). Les ER sont des suites de caractères qui peuvent sembler barbares et incompréhensibles aux profanes. Vous pouvez vous forger votre propre idée grâce à cette ER qui a pour but de vérifier si une chaîne de caractères contient une adresse MAC.

```
'^(\S+)\s+.*?HWaddr\s+([\dA-Fa-f]{2}([:-])[\dA-Fa-f]{2}
(\3[\dA-Fa-f]{2}){4})\s+'
```

Illustration 16 : **Expression régulière identifiant une adresse MAC**

Pour mémoire une adresse MAC valide se présente sous la forme de huit octets séparés par ':' ou par '-' ce qui nous donne une série de caractères du type: `5A:BB:F3:18:76:CE` .

Ainsi, pour les initiés, les ER permettent de manipuler et de vérifier du texte de façon très aisée. De plus le perl est un langage assez permissif puisqu'il propose un typage dynamique, ce qui permet au programmeur de ne pas se soucier des conversions, qui dans d'autres langages sont fastidieuses. Cela correspond bien à la philosophie que revendique le Perl, et qui est souvent résumée par l'acronyme ***TimTowTdi*** (qui signifie *There Is More Than One Way To Do It*) et qui signifie « il y a plusieurs façons d'atteindre son but », comme il en est question dans l'ouvrage Learning Perl de Schwartz et Wall.

3.2 Utilisation

Le perl est un langage pratique et facile d'accès, il utilise les passages de variables par copies et par paramètres. La syntaxe peut vite devenir chargée comme nous l'avons vu avec les expressions régulières, mais si le code est bien documenté et que les variables sont nommées intelligemment, cela reste très compréhensible. C'est dans cette optique que j'ai développé mon composant. Pour créer la documentation de mes codes j'utilise généralement Doxygen, qui permet de générer des documents au format HMTL très pratique à communiquer aux partenaires. En avril 2010 le langage perl n'était pas supporté nativement par Doxygen, mais heureusement une solution existe pour pouvoir utiliser Doxygen avec le langage perl. Elle est proposée par Bart Schuller et Tom Aeby, et accessible depuis http://www.bigsister.ch/doxygenfilter/ . Elle rend accessible la majeure partie des fonctionnalités de Doxygen. Pour plus de détails je vous invite à vous référer à l'annexe 4 Intitulée « Doxyfilter » (Doxygen pour le langage perl), qui vous permettra de vite pouvoir documenter vos codes en perl.

Le perl m'a réservé quelques surprises dont une m'a fait perdre environs deux jours de travail. Elle est liée à la façon dont sont transmis les paramètres à une fonction, ceux-ci sont véhiculés dans un tableau que l'on peut récupérer grâce à « @_ » , au travers d' une ligne du type :

```
my (arg1, arg2, …, argN) = @_ ;
```

Illustration 17 : **Récupération des paramètres dans une fonction**

Le problème se produit lorsque l'on passe en paramètre à une fonction, un objet de type table de hachage, qui à une clef associe une valeur. La façon dont est stockée en mémoire une table de hachage fait que lorsqu'une est passée en paramètre tous les éléments suivants seront considérés comme étant soit une clef, soit une valeur selon leur parité dans le tableau. Cette problématique a été d'autant plus ténue à mettre au jour qu'elle répondait parfaitement à l'objectif de la première fonction dans laquelle je l'ai rencontrée et qui avait pour but de joindre deux tables de hachages. Pour le reste, tout est très standard, à quelques détails de syntaxe près nous sommes très proches du C avec le point-virgule à la fin des lignes, et les structures conditionnelles. La différence majeure se trouvant au niveau des variables non typées.

3.3 Intégration dans Quattor

Mon module a pour but d'être intégré dans Quattor, qui est une somme de différents composants. Ainsi j'avais à ma disposition une API dans laquelle j'ai pu puiser des fonctions telles que celles que j'utilise pour exécuter les commandes à l'intérieur de mon code. Cependant, une fois encore, la documentation de cette API bien qu'elle semble exister, est très difficilement accessible. Ce qui m'a obligé à m'inspirer de composants déjà existants pour découvrir comment utiliser les fonctions. De plus, j'ai codé des fonctions en pensant qu'elles existaient déjà par ailleurs dans cette API, mais comme je n'avais pas accès à un index qui m'aurait permis de trouver la fonction que je recherchais, la réécriture m'est apparue comme étant plus rapide. Ce fut notamment le cas des deux fonctions récursives que j'ai créées pour dupliquer un arbre complet pouvant contenir indifféremment des tableaux et des tables de hachage. J'ai d'ailleurs soumis l'idée de faire remonter ces deux fonctions au-dessus de mon composant, car elles ont une visée globale qui n'est pas spécifique à ce dernier, mais l'idée n'a rencontrée aucun écho.

Un autre aspect très pratique de l'API fournit avec Quattor se situe au niveau de l'affichage dans la console. En effet, il existe plusieurs niveaux d'appel qui vont de la simple information, à l'erreur, en passant par l'alerte et la recherche de problème. A ces appels auxquels l'on peut accéder, par exemple, au travers d'une ligne telles que celle présentée à l'illustration 18 qui introduit un message de recherche d'erreur, le X pouvant être compris entre 0 et 9.

```
$self->debug( X , ``mon message de recherche d'erreur'') ;
```

Illustration 18 : **Exemple de message de recherche d'erreurs**

Le niveau zéro étant relativement inutile, vu que l'on perd tout l'intérêt de la solution, le message est alors affiché de façon obligatoire comme un message de type information. Alors que l'intérêt majeur de ce message type « debug » réside dans le fait que l'on peut choisir la densité des informations à afficher, le niveau 9 étant le plus prolixe et affichant tous les niveaux inférieurs. Il suffit pour cela, lors de l'appel de la commande qui lance la configuration réseau, de passer la valeur de X désirée en paramètre comme suit :

```
lcm --profile file://dir/to/xml_file --debug X --verbose --co network ;
```

Illustration 19 : **Commande lançant le module réseau avec un niveau d'information X**

A l'intérieur du module réseau et lors de son utilisation par Quattor, la fonction configure() est appelée automatiquement. Il m'a donc suffit de me servir de cette fonction comme d'un « *main* » pour appliquer les différentes fonctions que j'ai envisagées pour ce composant. Le schéma synoptique de l'illustration 25 vous présente les différentes étapes que j'ai voulu faire suivre au composant lors d'une configuration du réseau. Et que je détaillerai plus avant dans la section de ce rapport consacrée à ma solution.

4 Le réseau

Le composant dont j'avais la charge ayant pour nom « composant réseau de Quattor », il y avait de grandes chances qu'à un moment se présentent à moi des problématiques liées aux réseaux.

4.1 Présentation

Je ne vous ferai pas l'affront de vous rappeler qu'un réseau sert à interconnecter des machines entre elles, et que, comme des personnes dans une ville, si elles veulent s'écrire par voie postale, il est nécessaire que chacune possède une adresse précise et unique. D'un point de vue très trivial, l'on peut dire que le module réseau de Quattor sert à distribuer les adresses aux machines.

Comme chaque fois qu'il est question de réseau, il est de bon ton de commencer par la représentation par couche, afin de positionner notre travail ; je ne dérogerai pas ici à cette règle.

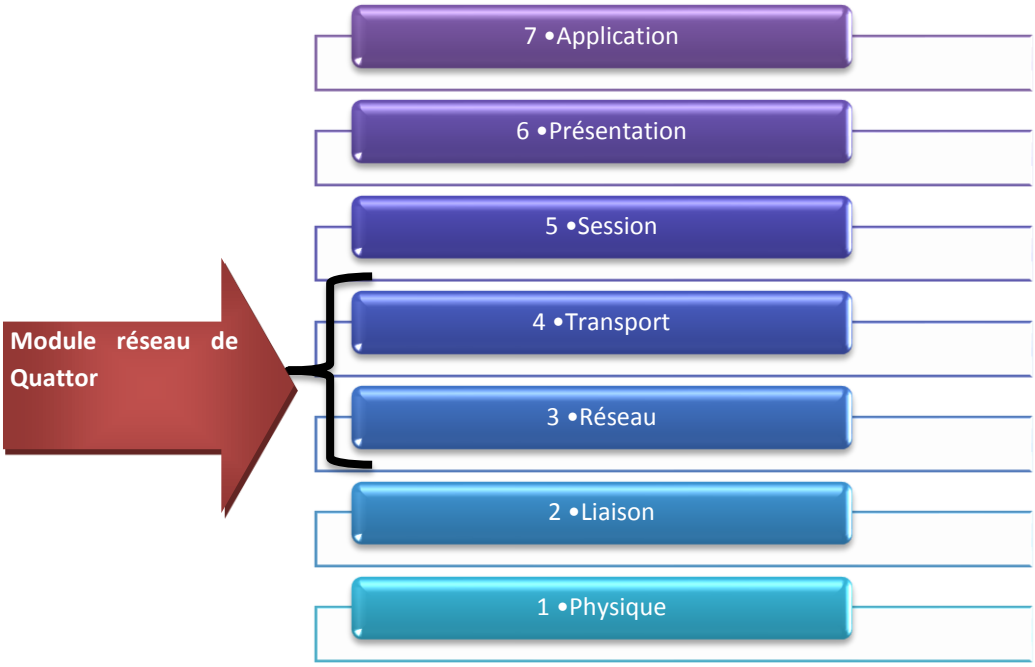

Illustration 20 : **Le modèle OSI**

Le composant réseau de Quattor agit principalement au niveau des couches 3 et 4 du modèle OSI, c'est lui qui va appliquer les directives de l'administrateur, sur la machine, Ainsi selon ce qui lui a été demandé au travers d'un fichier XML (cf annexe 3). Mon rôle, et plus précisément celui de mon composant, est d'appliquer au nœud à configurer toutes commandes qui rendront le réseau fonctionnel et le conformeront comme désiré. Il m'est ici difficile de ne pas tomber dans l'énoncé trop technique de toutes les commandes mises en œuvre ; si le détail technique vous importe, référez-vous à l'annexe 2 dans laquelle je fournis mon document de spécification où je listais à mon commanditaire M. Brarda toutes les fonctionnalités que je comptais déployer dans le composant, en réponse au cahier des charges qui m'avait été confié. Une approche plus visuelle est proposée aux lecteurs dans l'une des sections suivantes, au niveau de l'illustration 22 qui représente graphiquement le fichier « core-schema.tpl » qui fut l'un des points de départ et pilier de mon travail au LHCb.

4.2 Mise à niveau

Ma spécialité au sein de l'ISIMA est l'informatique des systèmes embarqués. Nous disposons d'une filière qui a pour but de former des ingénieurs particulièrement sensibilisés aux problématiques des réseaux. Cependant j'ai eu la chance de pouvoir obtenir ce stage qui m'a permis d'élargir encore plus l'éventail de mes compétences en informatique. Si les connaissances de bases sur les réseaux font parties du tronc commun de chaque ingénieur diplômé de l'ISIMA. Je ne disposais pas de toutes les commandes nécessaires pour mener à bien ce projet à mon arrivé au CERN. Il a donc fallu que je me documente pour pouvoir réaliser les spécifications détaillées de l'annexe 2 pour se faire j'ai eu la chance de profiter d'un cours de mise à niveau que M. Bonaccorsi a organiser pour l'équipe du LHCb online et qui sur six heures a rafraichit les généralités sur les réseaux. Ce qui m'a permis de rechercher plus sereinement les commandes dont j'avais besoin sur internet. J'ai listé les sites que j'ai retenus comme étant les plus pertinents dans la section webographie de ce rapport. Comme mes machines cibles, durant ce stage, fonctionnent sous Red Hat Entreprise Linux 5, j'ai pu utiliser les documentations des distributions fedora 6, centOS 5 et Scientific Linux 5, dont les noyaux sont très similaires.

J'ai effectué cette mise à niveau à la suite de mon apprentissage du langage de programmation perl comme vous pouvez le constater dans le diagramme de Gantt qui constitue l'illustration 24 j'ai ainsi dévolu quatre semaines à ces mises à niveau, pour pouvoir être à l'aise, avec l'environnement dans lequel j'ai évolué durant cinq mois. Je peux cependant faire remarquer que durant toute la durée de mon projet il m'est arrivé de revenir sur ces documents qui m'ont servis de base, et aussi que comme toujours la commande,

```
man [commande]
```

Illustration 21 : **La commande « man »**

C'est avérée très utile, lorsque la syntaxe pour configurer un paramètre n'était pas évidente. La syntaxe des commandes a d'ailleurs été le plus gros défis à relever sur ce projet. Car toute la question a été de savoir si pour chaque paramètre à configurer il existait une commande capable de configurer un nœud de façon dynamique. Et s'il est facile de trouver de la documentation sur les configurations nécessitant un redémarrage du service réseau, il est beaucoup plus ténu de trouver les commandes et les paramètres qui m'ont étés nécessaire.

5 Spécification

L'objectif du projet est de mettre à jour et revoir complètement le module de connexion réseau du projet Quattor.

Quattor est un système qui permet de centraliser les installations de machines Linux à distance. Ainsi, on peut mettre à jour ou modifier certains paramètres ou configurations, sur des milliers de machines, juste en éditant un fichier. L'intérêt en termes de gain de temps et de praticité est évident.

Le module en charge des connexions réseau existant est un module âgé, qui date de 2005. Ce module n'est pas très compréhensible au premier abord et la communauté utilisant Quattor a accueillie avec enthousiasme le projet qui m'a été confié.

Le point de départ des spécifications fut une présentation qui avait pour objectif d'introduire Quattor de façon généraliste.

Le projet était clair, faire un composant maintenable, et capable de configurer dynamiquement les nœuds du réseau. Le langage perl était dicté par l'intégration dans Quattor, quant-aux latitudes techniques elles n'étaient pas très importantes car s'il existe deux commandes pour appliquer le même paramètre, il n'y a aucun coût qui puisse nous faire préférer l'une à l'autre. De plus, le temps d'exécution, ou la consommation CPU ou mémoire ne faisaient absolument pas partie des préoccupations de mon commanditaire.

Ainsi je n'ai pas eu de choix technique à effectuer, la tâche la plus coûteuse en temps lors de la rédaction des spécifications fut la recherche des commandes qui permettent de passer une commande en dynamique. Recherche durant laquelle j'ai parcourue toute la documentation RHEL5 relative aux configurations réseaux et où à chaque action désirée je recherchais la commande dynamique et/ou les fichiers dans lesquels écrire pour que la modification soit appliquée au prochain redémarrage du service réseau ou du nœud.

6 Ma solution

Ma solution a pour point de départ le code du composant réseau existant, ce code était donné pour être fonctionnel. Il était aussi réputé pour être très difficilement maintenable de par son manque de commentaire, ses noms de variables et de fonctions obscures, qui le rendaient imperméable à la compréhension au premier abord. Mes principaux objectifs étaient donc de réaliser un composant meilleur sur tous ces aspects. C'est pour cela que j'ai décidé d'utiliser Doxygen que je vous ai déjà présenté précédemment. J'ai de plus apporté un intérêt particulier au nommage de mes variables, de façon à ce que la lecture de leurs noms renseigne le lecteur sur leurs fonctions. Comme pour tout bon code, j'ai créé de nombreuses fonctions pour que ce dernier soit modulaire, en essayant d'avoir une tâche unique pour chaque fonction. Dans certains rares cas, une fonction peut effectuer plusieurs types d'action car les coûts en termes de passage de paramètres et de clarté du code n'en valent pas les bénéfices.

Deux fichiers furent mes points de départ. Il s'agit du « core-schema.tpl » et du « network.pm.cin », qui sont accessibles depuis l'adresse hébergée par source forge :

http://quattor.svn.sourceforge.net/viewvc/quattor/trunk/ncm-components/core/ncm-network/

Ils contiennent les différentes structures utilisées par le composant et le code du module réseau. Si je n'ai pas obtenu l'autorisation de modifier la structure du « core-schema.tpl » pour préserver la compatibilité ascendante, j'ai cependant réussi à ajouter trois paramètres qui sont l'ip_forwarding, le userctl et le srcaddr. Les deux premiers permettent d'autoriser ou non le nœud à transmettre les requêtes ARP, et à l'utilisateur de configurer lui-même ses cartes réseaux. Quant au srcaddr, il permet d'assigner une adresse source spécifique aux paquets transmis. L'illustration suivante vous permet d'avoir une vision d'ensemble des différents paramètres que gère le composant réseau de Quattor. C'est l'une des premières illustrations que j'ai réalisées car elle permet de conceptualiser ce que fait le composant et les structures qu'il met en œuvre, de façon beaucoup plus aisée qu'en lisant le fichier core-schema.tpl. Ainsi nous voyons qu'une première couche est constituée par la structure « structure-network » qui contient des informations généralistes qui s'appliqueront à toutes les cartes réseau du nœud. Puis, au travers du paramètre « interfaces », les paramètres seront spécifiques à une seule carte réseau (*NIC* en anglais pour *Network Interface Card*) dont l'identifiant sera la clef de la table de hachage « interfaces », la valeur correspondante à cette clef étant elle-même une table de hachage qui associera les paramètres et leurs valeurs dans la majorité des cas. Dans les autres cas cette clef sera encore un niveau intermédiaire qui ouvrira sur un tableau comme pour le paramètre « route », ou sur une nouvelle table de hachage si l'on considère « aliases ». Ici, c'est le type de flèches utilisé qui traduit le fait que nous ayons affaire à un tableau ou à une table de hachage.

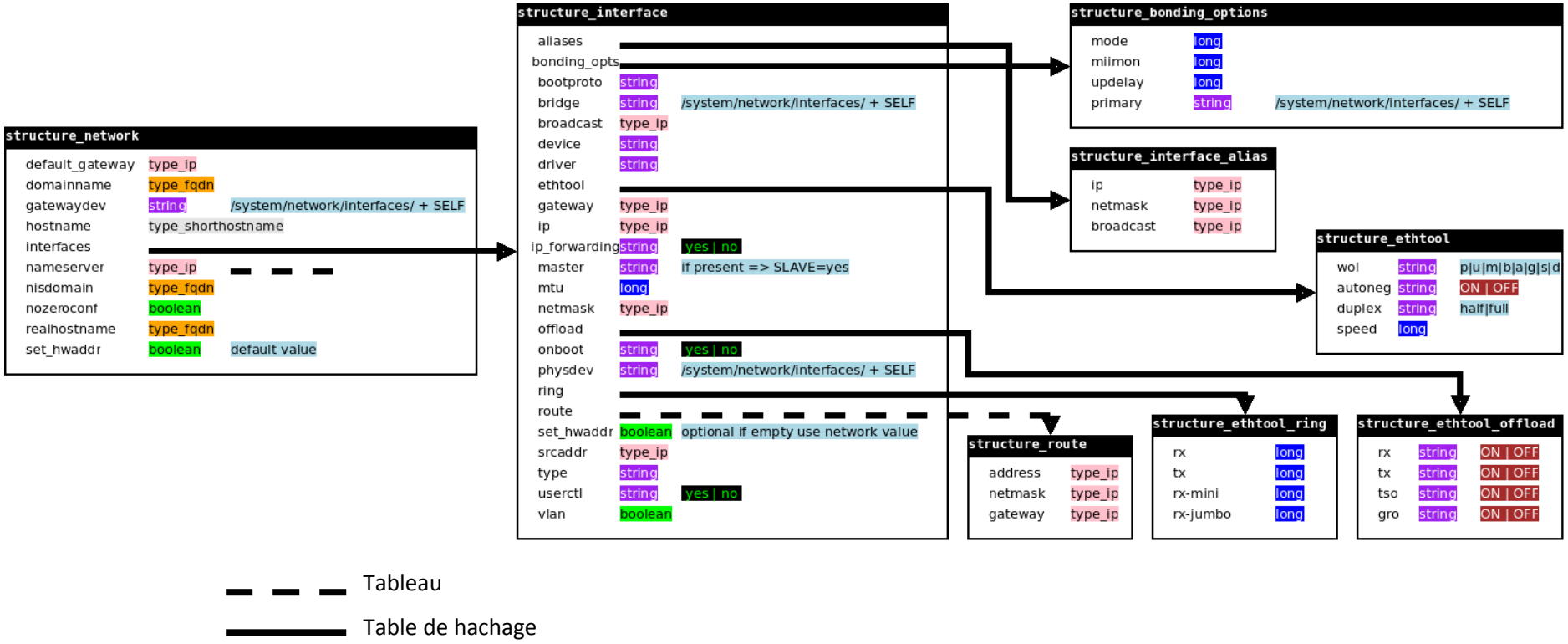

Illustration 22 : **Représentation des structures contenues dans le fichier « core-schema.tpl »**

La période de développement de ce module réseau s'est étalée sur toute la durée de mon stage, c'est-à-dire cinq mois. Dans le but d'organiser mon travail, et pour donner de la visibilité à mes responsables, j'ai utilisé une solution de planification. J'utilise généralement pour ce faire l'outil de Microsoft MS-Project 2007. Cette année je me suis essayé à la solution libre planner, solution qui s'est avérée tout-à-fait fonctionnelle et efficace. Elle permet de générer des rapports d'activité au format HTML, encore une fois très pratique pour communiquer avec des tiers distants. Les illustrations 23 et 24 montrent la progression entre le planning tel qu'il était au début du projet en avril 2010 et ce qui s'est vraiment réalisé. Ainsi, nous pouvons nous rendre compte que dans le planning prévisionnel la rédaction des spécifications était la première tâche à réaliser une fois le cahier des charges en ma possession. Dans les faits, avant de pouvoir rédiger ces spécifications, il m'a fallu être capable de lire et analyser le code existant. Pour ce faire il a donc fallu que j'apprenne le langage de programmation perl, que je rafraîchisse mes bases sur les réseaux, et que je prenne le temps de lire le code précédent. Ces recherches préliminaires ont ainsi engendré un délai de trois semaines supplémentaires avant la première version des spécifications.

Nous pouvons aussi constater que de nouvelles tâches qui n'avaient pas été identifiées au préalable sont apparues, telle que la soumission des spécifications à la communauté des utilisateurs de Quattor, pour leurs permettre de pouvoir exprimer leurs opinions sur ce nouveau projet qui impactera tous les sites où Quattor est déployé. Cependant l'impact ne sera pas trop important car la nouvelle version de Quattor sera totalement compatible avec les anciennes versions des configurations qui fonctionnaient avec la version précédente du module réseau de Quattor.

De plus, comme souvent dans les projets, des aléas surviennent, comme des maintenances ou des mises à jour qui rendent les machines indisponibles pendant des journées entières, ce qui a pour conséquences de rallonger les délais initialement prévus ; de fait il sera judicieux d'augmenter les marges d'erreurs temporelles pour les prochains projets dont j'aurai la charge et dans lesquels il me sera demandé d'estimer les différents coûts.

Ainsi, bien que je garde espoir d'arriver jusqu'à la mise en production de mon composant durant mon stage, je n'en suis plus aussi sûr qu'à mes débuts ici. J'espère toutefois, lors de ma soutenance le 14 septembre 2010 à 14h à l'ISIMA, pouvoir vous confirmer que mon composant est bel et bien déployé et fonctionnel.

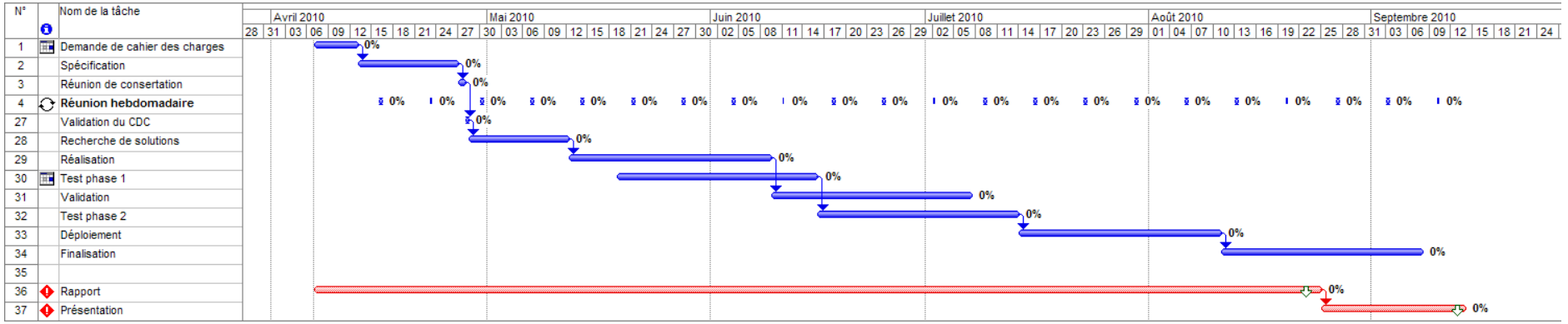

Illustration 23 : **Planning prévisionnel (MS project 2007)**

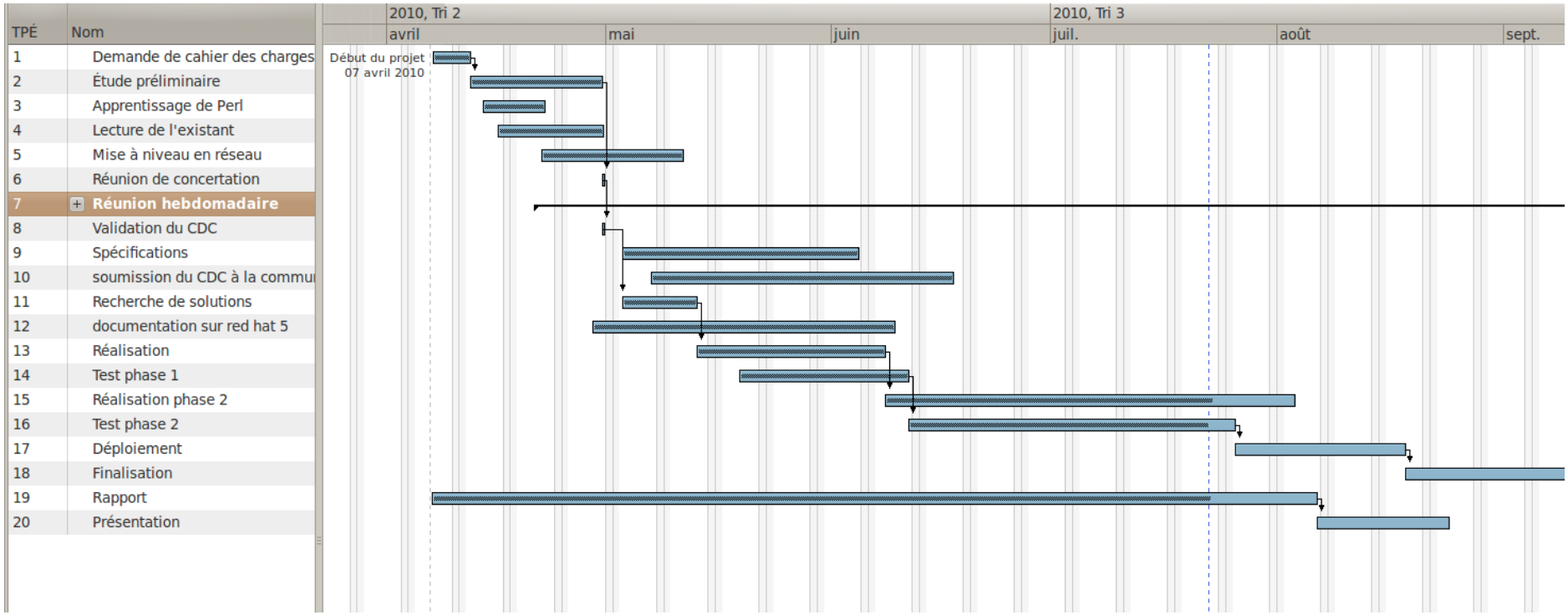

Illustration 24 : **Planning effectif (planner)**

Voyons quelles furent les problématiques auxquelles j'ai eu à faire face et les solutions que j'ai utilisées pour les contrer ou les contourner avant d'en arriver là. Je vous propose de commencer par une vue d'ensemble de la logique, suivie par le composant réseau de Quattor tel que je l'ai développé, et ce, résumé dans l'illustration 25 suivante.

Le composant entre en action lorsqu'il est appelé par Quattor et retourne à ce dernier une valeur ?? selon que la configuration se sera bien passée ou non. Il y a en fait trois types de sortie possible, la meilleure étant lorsque le nœud est connecté au réseau, et que la configuration a bien été appliquée. Une seconde sortie laisse le nœud connecté au réseau, mais fait que les modifications n'ont pas été appliquées correctement et que ce sont les valeurs originelles, c'est-à-dire d'avant l'appel du module réseau, qui ont été restaurées, avec succès. La dernière, beaucoup plus problématique, signifie que même après avoir essayé de restaurer les valeurs originelles, le nœud ne réussit pas à atteindre correctement le réseau au travers d'une commande `ccm-fetch`, ce qui veut dire que le nœud est incapable d'atteindre la base de données de Quattor pour pouvoir être à nouveau configuré. Ce dernier cas, s'il n'est pas dû à une coupure du réseau, nécessite la configuration manuelle du nœud, ce qui est synonyme d'échec critique.

Avant d'en arriver aux types de sortie, le composant exécute un cycle très « standard » durant lequel les valeurs actuelles sont sauvegardées dans le but d'être restaurées en cas de problèmes. Il récupère les informations qui sont nécessaires à son fonctionnement et les ordonne pour pouvoir les traiter facilement. Lors de ce traitement, il se débarrasse des fichiers obsolètes pour ne pas encombrer le nœud cible et il remplace ou ajoute les fichiers de configuration nécessaires à la nouvelle conformation du nœud. Une fois cette étape passée, nous arrivons au seul choix de l'administrateur qui impactera la façon dont le composant fonctionne. Il s'agit du paramètre « restart », un booléen qui autorise à redémarrer ou non le service réseau. Ce paramètre est essentiel dans le cas de la gestion de nœuds sans disque car si l'on coupe la connectivité réseau de ces nœuds, ils se retrouvent bloqués. Ensuite, le composant exécute une phase de tests dans le but de valider le cycle qu'il vient de réaliser. Cette phase a été décomposée en deux tests distincts qui se comportent *in fine* comme un seul. Le premier s'appuie sur les requêtes ICMP pour effectuer un `ping` sur une machine distante tandis que le second implique directement Quattor en envoyant une requête `ccm-fetch` qui permet de vérifier si le nœud pourra avoir accès aux prochaines configurations demandées par Quattor. Ce qui nous ramène aux différentes valeurs retournées par le composant et conclut le cycle de fonctionnement du composant réseau qui rend alors la main à Quattor. A ce stade, nous avons vu quel était le contexte technique qui me servait de point de départ, la façon dont mon temps fut organisé, et la logique que suivra le composant

Passons désormais à un aperçu plus technique du code.

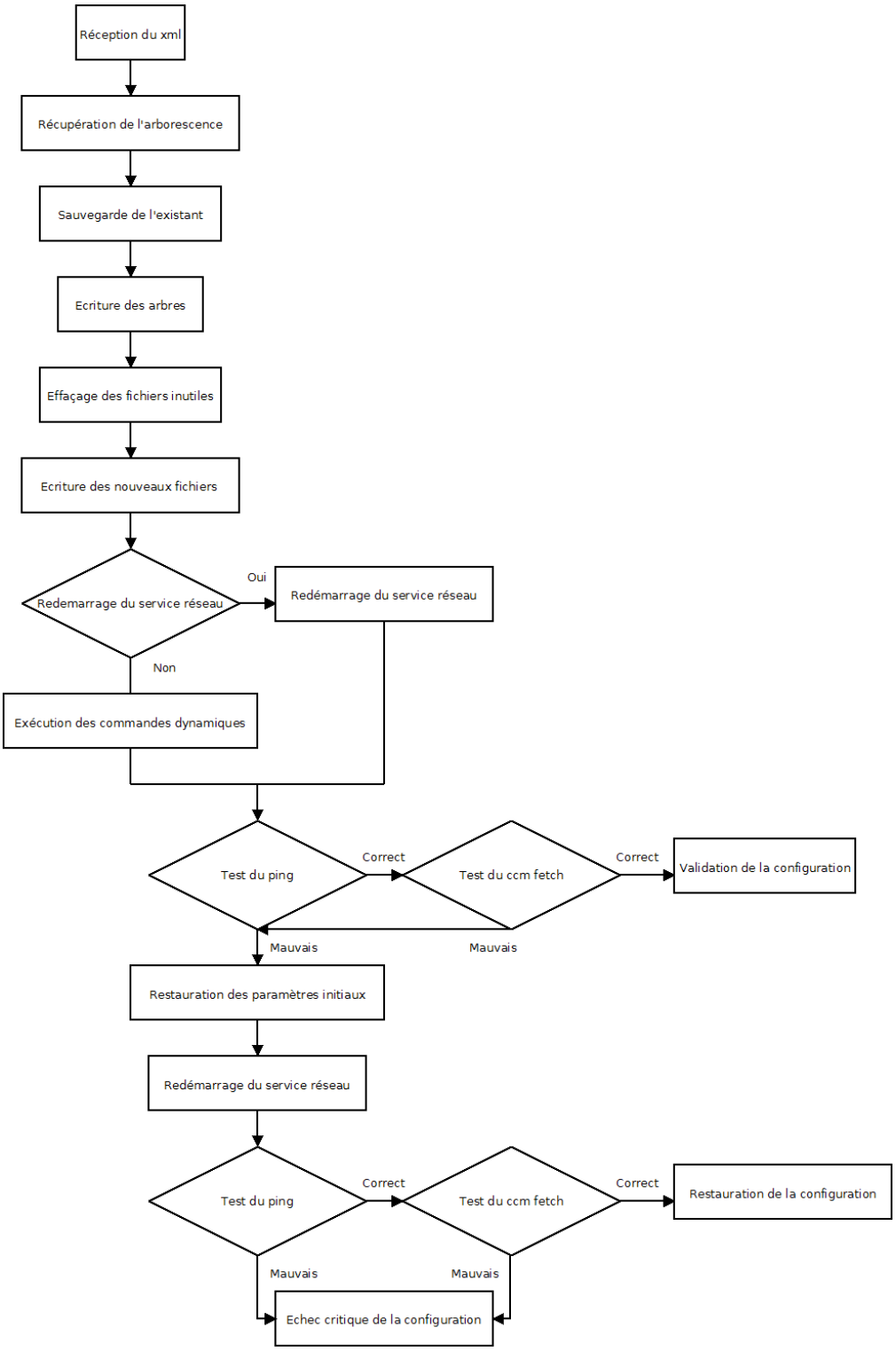

Illustration 25 : **Schéma synoptique du fonctionnement du composant réseau**

Nous avons vu que le composant est partie intégrante de Quattor et qu'il profite de l'API environnante de ce dernier. Pour ce faire, il est nécessaire d'intégrer au code quelques inclusions, dont voici ci-dessous toutes celles utilisées par le nouveau composant réseau :

```
use strict;
use warnings;

use CAF::FileEditor;
use CAF::FileWriter;
use CAF::Process;

use Data::Dumper;

use EDG::WP4::CCM::Element;

use File::Basename;
use File::Compare;
use File::Copy;

use LC::Exception;
use LC::File;
use LC::Find;
use LC::Process;

use Net::Ping;

use NCM::Check;
use NCM::Component;

use Switch;
```

Illustration 26 : **Inclusions utilisées par le nouveau composant réseau**

Les inclusions spécifiques à Quattor ont été surlignées en jaune pour une meilleure lisibilité. Leur utilisation a été inspirée par le composant Quattor qui gère l'accès aux comptes des utilisateurs et qui se situe dans sourceforge au niveau pointé par l'illustration suivante.

SourceForge.net > Findsoftware > Quattor > SCM Repositories >
Quattor > trunk > ncm-components > core > ncm-accounts > account.pm.cin

Illustration 27 : **Emplacement du fichier account.pm.cin dans sourceforge**

Les différentes fonctions qu'il est possible d'appeler au travers de cette API, et dont je me suis servi, avaient pour but de sécuriser les écritures dans des fichiers, ainsi que la façon dont étaient exécutées les commandes invoquées par mon composant. Ces solutions n'étaient pas intégrées dans la version précédente, et augmentent la robustesse du composant.

7 Les tests

Les tests furent une partie sensible du développement. En effet, le module réseau a la possibilité, s'il n'est pas fonctionnel, de déconnecter des milliers de machines d'un réseau, rendant ainsi tout contrôle à distance impossible. Ce qui imposerait d'effectuer une intervention sur chaque machine pour pouvoir rétablir sa connectivité. Il apparaît donc ici clairement que le composant final doit être exempt de tout défaut. Pour ce faire, il faudra donc le soumettre à des batteries de tests ayant pour but de mettre à l'épreuve toutes ses fonctionnalités. De plus, pour ne pas qu'un cas critique comme celui que je viens d'évoquer se produise, j'ai attaché, dès le début du développement, une importance majeure à la fonction qui permet de rétablir les configurations d'origine, au cas où le nœud ne trouve pas le réseau avec sa nouvelle conformation.

Voyons à présent quels furent les tests qu'a subit le composant avant d'en arriver à sa version finale.

Les premiers essais furent effectués sur le composant lors de son développement, ils avaient pour objectif de tester les capacités d'une fonction particulière. Pour les réaliser, le perl dispose d'un outil puissant et très pratique qui permet d'afficher dans la console le contenu d'une variable. Cette fonction peut être invoquée par une commande du type de celle présentée dans l'illustration 28.

```
$self->debug( 8 , 'texte de recherche d'erreur :'.Dumper($ma_variable);
```

Illustration 28 : **Exemple d'appel permettant l'affichage du contenu d'une variable**

Comme vous le constatez dans l'illustration précédente, cette fonction se conjugue très facilement avec les différents niveaux d'affichage que j'ai présentés dans la partie 3.3 relative à l'API de Quattor.

Pour pouvoir utiliser la fonction Dumper(), il est nécessaire que vous rajoutiez au début de votre fichier l'inclusion suivante :

```
use Data::Dumper;
```

Illustration 29 : **inclusion permettant d'utiliser la fonction Dumper()**

L'illustration 30 vous montre le résultat de la commande Dumper() appliquée à un arbre. Nous pouvons constater que cette dernière est en effet très pratique, détaillant la totalité du contenu de la variable, qui dans ce cas était une table de hachage passée en paramètre. J'attire également votre attention sur la syntaxe utilisée. Ainsi, les accolades « {,} » traduisent une table de hachage, tandis que les crochets « [,] » signifient que nous sommes en présence d'un tableau. Nous pouvons aussi aisément nous rendre compte que les différentes structures peuvent s'imbriquer les unes à l'intérieur des autres. De fait, il apparaît que la clef « interfaces » est une table de hachage qui contient deux clefs, elles aussi associées à des tables de hachage qui contiennent elles-mêmes une table de hachage simple où une clef est associée à une valeur avec la syntaxe `'clef' => 'valeur'`.

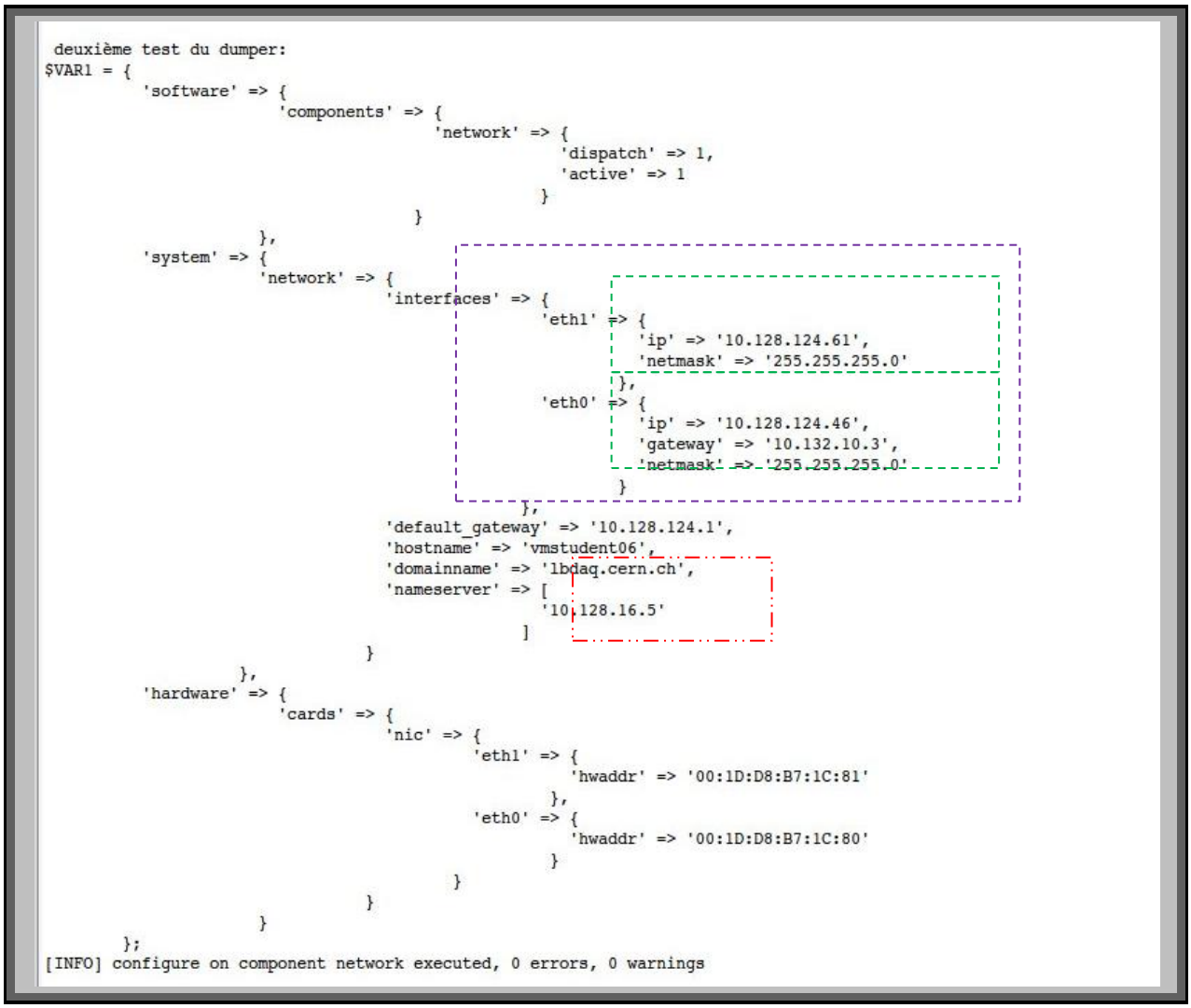

Illustration 30 : **Résultat de la commande Dumper() appliquée à un arbre**

Ici, pour accéder à l'adresse IP de eth1, si l'arbre est passé en paramètre au travers de $tree, il faut utiliser :

```
$tree->{"system"}->{"network"}->{"interfaces"}->{"eth1"}->{"ip"}
```

Illustration 31 : **Exemple d'accès au contenu d'un arbre**

C'est durant cette phase de tests que j'ai pu notamment mettre en lumière la problématique du passage d'une table de hachage en paramètre. Cf. partie 3.2. Une fois la première phase de développement effectuée, j'ai proposé une version alpha du module. J'ai continué les tests sur des machines virtuelles hébergées sur une solution Windows server 2003. Ces tests sur machines virtuelles m'ont permis de valider certaines des fonctionnalités basiques du composant comme le changement d'adresse IP, avec et sans redémarrage du service réseau. Les résultats détaillés des tests sont accessibles dans l'annexe 5 qui contient le tableau récapitulatif des différents tests effectués par mes soins sur le nouveau module réseau de Quattor. Cette tranche de tests a permis de mettre en évidence des lacunes du premier jet de code, en particulier une, liée au changement d'adresse MAC de la carte et à l'écriture dans les fichiers.

En effet, il s'est avéré que si l'on écrit ou modifie dans le fichier de configuration « `/etc/sysconfig/network-scripts/ifcfg-[périphérique]` » la ligne relative à l'adresse MAC qui ressemble à « `HWADDR = CA :FE :02 :08 :30 :FF` », avant d'avoir déconnecté la carte du réseau avec la commande :

```
sudo sbin/ifdown [périphérique]
```

Illustration 32 : **Commande permettant de déconnecter une carte réseau**

on se retrouve alors dans l'impossibilité de pouvoir déconnecter ultérieurement la carte du réseau car lors de l'appel à ifdown, le système vérifie la concordance entre l'adresse MAC physique de la carte et celle contenue dans ce fichier, à cette ligne. La seule solution permettant de retrouver un comportement normal étant de rechercher l'adresse MAC actuelle de la carte réseau à l'aide de la commande :

```
ifconfig
```

Illustration 33 : **Commande retournant des informations sur les cartes réseaux**

Il faut ensuite éditer le fichier incriminé en modifiant les informations erronées par celles retournées par ifconfig.

Puis avec l'aide de M. Bonaccorsi, qui m'a permis de disposer de plus de cartes réseau par machine virtuelle, j'ai pu effectuer les tests relatifs au *bonding,* qui consiste, comme son nom l'indique, à lier des cartes ensemble. A la suite de ces différents tests, mon module est passé dans sa phase béta. Actuellement, je souhaite pouvoir continuer mes tests sur des machines physiques avec des configurations cohérentes, avant de soumettre une version « *Release Candidate* » à la communauté.

Deux machines physiques ont été mises à ma disposition afin de me permettre de tester en conditions réelles mon composant et les modifications qu'il doit apporter à une machine pour qu'elle puisse changer de conformation; il s'agit de ha01 et plus19. Mais, avec les machines physiques sont arrivés les problèmes physiques. Ainsi, bon nombre des tests qui m'apparaissaient comme étant nécessaires pour valider la robustesse de mon code n'ont pas pu être réalisés, par faute de place, et seuls les tests basiques ont permis de valider le module réseau que je livre. En effet, il serait intéressant de vérifier comment se comporte la configuration quand l'on passe d'un nœud possédant une seule carte réseau active, à un nœud relié aux réseaux par plusieurs cartes réseau, et inversement. Pour ce test, deux cartes réseau sont un minimum, et un nombre plus conséquent permettrait d'assurer aux futurs utilisateurs une configuration fiable, jusqu'au nombre de cartes déjà testé, au minimum. Mais de la façon dont est architecturé le réseau du LHCb et de par son utilisation optimale, je n'ai pas pu réaliser ce test. Je vous invite à vous référer à la section « problèmes rencontrés », qui explicitera plus en détails certains des points qui m'ont mis dans l'incapacité de mener mes tests comme je l'entendais. Ceci a pour conséquence, de mon point de vue, que le composant que je livre n'a pas été suffisamment testé pour prétendre à une robustesse sérieuse, bien qu'il satisfasse les requêtes basiques qui lui sont demandées.

7.1 Tests restant à réaliser

Configuration initiale	Configuration finale
1 carte réseau	Plusieurs cartes sur plusieurs réseaux
Plusieurs cartes sur plusieurs réseaux	1 carte réseau
Plusieurs cartes sur plusieurs réseaux	Plusieurs cartes en bonding
Plusieurs cartes en bonding	Plusieurs cartes sur plusieurs réseaux

Comme je vous le présenterai par la suite dans la section dédiée aux problèmes rencontrés, de nombreux paramètres m'ont empêché de mener tous les tests que j'aurais souhaité réaliser. Je laisse donc ici, à l'attention de mes successeurs, une liste des tests qu'il serait judicieux de réaliser pour s'assurer du bon fonctionnement du composant, dans des situations plus exotiques que dans les cas standards. Je garde un espoir de pouvoir en réaliser certains avant la fin de mon stage, mais pour ce faire il me faudra l'appui d'un des administrateurs réseau.

Une fois ces tests réalisés, nous pourrons certifier que le composant configure correctement les nœuds dans les situations les plus courantes, et pour tous les paramètres. Ensuite, il faut toujours garder à l'esprit que pour certaines des configurations les plus exotiques, des effets de bord peuvent toujours se manifester, tant que le cas précis n'a pas été essayé et validé.

J'invite d'ailleurs ici la communauté des utilisateurs de Quattor à ajouter dans le tableau récapitulatif des tests effectués, toutes les configurations, les plus pertinentes et utiles à leurs yeux, pour les futurs utilisateurs de Quattor et de son module réseau.

8 Cas remarquables

Je prends ici le parti de fournir des résultats de tests très détaillés. En effet, certains des cas remarquables que je mets en exergue peuvent s'avérer très piégeurs, et la simple lecture de la documentation n'en permet pas l'anticipation. C'est pourquoi je recommande vivement la lecture complète de ce rapport, cela pourra apporter beaucoup aux administrateurs qui s'en donneront la peine. Ainsi, voici les cas que j'ai jugés dignes d'y apparaitre.

8.1 une carte répondant sur deux adresses IP distinctes sans mise en place d'alias.

Après avoir passé la commande :

```
ifconfig eth0 x.y.z.b
```

Illustration 34 : **Attribution de l'adresse IP x.y.z.b au périphérique eth0**

et alors que la carte était préalablement configurée avec `ifconfig eth0 x.y.z.a`, je me suis rendu compte que deux de mes consoles, l'une dédiée à x.y.z.a et l'autre à x.y.z.b, accédaient toutes les deux à la machine plus19 au même instant. Cette action n'est normalement pas possible pour une seule interface, sans mise en place de solutions spécifiques telles que le sont les alias par exemple. Des tests préliminaires avaient été effectués pour s'assurer que x.y.z.b n'était pas accessible avant la configuration et les autres interfaces de la machine avaient été soit désactivées, soit configurées à l'adresse x.y.z.c . Face à ce mystère que j'ai fait remonter à M. Brarda, et après investigation, la meilleure hypothèse que nous avons formulée est que la liaison est gérée au niveau des adresses MAC, précisément ce serait le switch force10, qui associe aux deux adresses IP la même adresse MAC dans sa table dédiée aux requêtes ARP.

8.2 une carte sans connexion physique mais accessible.

Dans la même veine, sur ma machine de test plus19, lorsque la carte eth0 est accessible, cela rend également accessible la machine par l'adresse allouée à la carte eth1, située dans le même subnet, malgré le fait que cette dernière ne soit pas câblée.

8.3 les alias montés à l'unité et descendus en groupe.

Voici un autre comportement étrange qui mérite d'être souligné, et qui est lui relatif aux alias et particulièrement à la commande :

```
ifdown eth0:aliasx
```

Illustration 35 : **Commande devant déconnecter l'alias eth0:aliasx**

Cette commande devrait uniquement descendre l'alias visé, c'est-à-dire eth0 :aliasx, cependant il s'avère qu'elle déconnecte du réseau tous les alias liés au périphérique eth0. En revanche, la commande présentée ci-dessous va bien, quand à elle, connecter uniquement l'alias concerné et précisé dans la commande:

```
ifup eth0:aliasy
```

Illustration 36 : **Commande permettant de connecter eth0 :aliasy**

9 Problèmes rencontrés

Comme pour tous les projets informatiques, les problèmes que j'ai eus à surmonter durant mon stage furent nombreux et variés. Dans cette section, j'aborderai les problèmes que je trouve judicieux d'évoquer et qui n'ont pas trouvé leur place dans une autre partie de ce rapport.

Au CERN, la sécurité est omniprésente tant pour protéger les personnels que pour protéger les systèmes d'information. Nous nous étions heurtés à cette problématique, avec M. Delord, durant notre projet de troisième année. Il en a été de même durant ma première journée de stage, journée pendant laquelle je n'ai pas pu me connecter à une machine du CERN, bien que mes login et mot de passe étaient corrects.

L'un des problèmes que j'ai rencontrés durant la progression de mon code était lié au manque de cohérence des structures. C'est, je pense, l'une des conséquences du fait que le composant n'a a priori pas été réfléchi puis développé, mais qu'il a grandi par couches successives, au fur et à mesure des besoins des différents utilisateurs. Pour illustrer mes propos, prenons l'exemple d'un paramètre réseau qui peut être soit allumé soit éteint. A la question « le paramètre est il activé ? » la réponse est binaire « oui » ou « non », ce qui en informatique se traduit par un booléen « vrai » ou « faux ». Dans le core-schema, nous trouvons trois façons de répondre à cette question ; la première, la plus logique, est l'utilisation d'un booléen, nous nous y attendions, mais ensuite nous trouvons des chaînes de caractères qui contiennent soit « yes » ou « no », soit « ON » ou « OFF », ce qui complique la tâche du programmeur car pour chaque paramètre, bien qu'il sache que la réponse sera de type « vrai » ou « faux », il doit vérifier lequel des trois types est employé pour ce paramètre précis, et adapter ses jeux de tests conditionnels en conséquence.

L'accès à des machines de test fut aussi l'un des problèmes qui ont beaucoup ralenti mon travail, car si j'ai réussi assez facilement à avoir accès à des machines virtuelles comportant deux cartes réseaux, il fut beaucoup plus difficile d'obtenir des machines comportant plus de cartes réseau, notamment lorsque j'ai voulu effectuer les tests de bonding avec plusieurs cartes. Et il en fut de même pour les machines réelles, dont les protocoles de restauration étaient plus compliqués à mettre en œuvre. L'un de ces protocoles m'obligeait même à changer de site (site géographiquement établi à 10Km de mon bureau habituel) pour avoir accès directement à la console de la machine. De plus, lors de mes tests, je me suis retrouvé confronté à de nombreux problèmes liés à cette console, des problèmes de gel de l'affichage, de disponibilité de licences pour accéder à cette ressource, d'où mon impossibilité de résoudre ces problèmes rapidement et seul. En effet, mes droits d'accès étaient limités à ceux d'un simple utilisateur lambda, ce qui m'interdisait de pouvoir rebooter le KVM01, opération qui s'avérait être la solution la plus rapide pour solutionner les problèmes liés à ce composant. Il fallait donc, à chaque incident, que j'en réfère par courrier électronique à une personne suffisamment autorisée pour effectuer ce redémarrage du KVM01, ce qui au début du mois d'août, période de vacances pour la majorité des titulaires, n'était pas une tâche facile.

Très proche de l'utilisation déjà optimale du réseau, l'un des éléments qui ont limité mes tests dans leurs envergures est le switch force10, sur lequel est connecté le serveur ha01 qui avait été dédié à mes tests de bonding sur plusieurs interfaces. Ici aussi, bien que le serveur dispose de nombreuses cartes réseau, je n'ai pu avoir accès qu'à deux d'entre elles, par manque de place sur le commutateur force10. N'étant pas habilité à intervenir sur ce matériel, je ne pouvais là encore effectuer de modifications sur sa configuration que par l'intermédiaire d'un tiers, difficilement joignable. Tout ceci est incompatible avec des modifications qu'il aurait été nécessaire d'effectuer jusqu'à plusieurs fois par heure. Car ce commutateur particulièrement intelligent analyse les flux qui transitent par lui et s'il détecte une incohérence entre la configuration qu'il attend du serveur et les paquets d'informations qu'il reçoit, il coupe les communications. De là, mon couple commutateur-serveur, configuré sur un type de bonding particulier, ne pouvait pas répondre favorablement à mes requêtes de reconfiguration du serveur en un simple serveur avec une seule interface.

10 Conclusion

Mes objectifs étaient de réécrire le module réseau de Quattor en y ajoutant de nouvelles fonctionnalités. C'est désormais chose faite, le nouveau composant a été repensé, codé, testé. Cependant Il n'est actuellement pas distribué comme étant le module réseau officiel de Quattor. Car il ne correspond pas aux attentes en termes de maintenabilité de M. Brarda.

La réécriture depuis zéro a permis de trouver de nouvelles solutions. Toutes les fonctionnalités, y compris les nouvelles, sont fonctionnelles et la compatibilité avec les configurations précédentes est complète. Le code du module est désormais bien documenté.

Le composant a été livré dans sa version v08_20_00_00 sur laquelle il peut notamment être envisagé de rajouter un paramètre « noaction » qui, comme son nom le traduit, permet de faire tourner le composant en mode verbeux uniquement, sans qu'aucune action ne soit effectuée. Les prochaines mesures à appliquer à ce composant devraient être de mener à bien les tests pour s'assurer de sa robustesse.

Je livre donc un composant fonctionnel, testé et répondant au cahier des charges que nous avions établi avec mon commanditaire et maître de stage M. Brarda. Le code que je fournis servira de base pour les développements futurs du module réseau de Quattor.

11 Glossaire

ACCÉLÉRATEUR : Un accélérateur est un tube sous vide dans lequel des particules sont accélérées par des champs électriques et dont la trajectoire est guidée par des champs magnétiques.

API : « *Application and Programming Interface* » que l'on peut traduire par « interface pour la programmation d'applications ». C'est un ensemble de bibliothèques permettant une programmation plus aisée car les fonctions deviennent indépendantes du matériel.

BONDING: Technique réseau qui permet de regrouper des interfaces réseau en une seule, dans le but d'augmenter le débit d'un serveur et aussi de le sécuriser. Ainsi, selon le mode choisi, si une carte rencontre un problème, le flux de données continue d'être assuré par les autres cartes du *bonding*.

BRIDGE: Peut se traduire littéralement en Français par « pont ». C'est un composant réseau qui sert à faire transiter des paquets entre des réseaux de même type en s'appuyant sur les adresses MAC.

BROADCAST: Correspond à l'envoie d'informations à toutes les machines disponibles de façon simultanée.

BUS: Ensemble de liaisons physiques qui peuvent être exploitées conjointement par plusieurs éléments pour communiquer.

CERN: Organisation Européenne de Recherche Nucléaire.

COMMUTATEUR: On emploie généralement le terme anglais *switch* pour désigner le commutateur réseau. C'est un composant qui sert à interconnecter des machines entre elles. Il est capable de segmenter les réseaux qu'il génère, contrairement au concentrateur (*hub)*.

ETHERNET: Protocole de réseau local à commutation de paquets. Bien qu'il implémente la couche physique et la sous-couche *Media Access Control (MAC)* du modèle OSI/ISO, le protocole Ethernet est classé dans la couche de liaison.

GATEWAY: « Passerelle » en Français, permet de relier deux systèmes informatiques entre eux.

IP: Protocole réseau, « ***I**nternet **P**rotocol* », qui correspond à un protocole de niveau 3 du modèle OSI et du modèle TCP/IP, permettant un service d'adressage unique pour l'ensemble des terminaux connectés.

KVM: « ***K**eyboard **V**ideo **M**ouse* », c'est un matériel qui permet de relier plusieurs ordinateurs aux mêmes clavier, écran(s) et souris, comme son nom l'indique. Il existe même des versions évoluées capables de commander à distance les serveurs en transférant par IP le contrôle du *KVM*.

LHC: « ***L**arge **H**adron **C**ollider* », c'est le nom de l'accélérateur de particules.

LHCB: « ***L**arge **H**adron **C**ollider **B**eauty* », c'est le nom de l'une des expériences du CERN, le b de « beauté » faisant référence aux quarks de beauté que les physiciens souhaitent étudier.

MAC (ADRESSE): « **M**edia **A**ccess **C**ontrol », c'est un numéro d'identifiant unique attribué à chaque carte réseau en fonction de son constructeur.

MTU: « ***M**aximum **T**ransmission **U**nit* », définit la taille maximale (en octets) du paquet pouvant être transmis en une seule fois (sans fragmentation).

NETMASK: « Masque de sous réseau » en français, comme son nom l'indique c'est un masque qui permet de différencier dans une adresse IP la partie relative au réseau et celle servant à identifier le nœud.

NISDOMAIN: Nom du domaine NIS servant à centraliser l'authentification des utilisateurs.

NOZEROCONF: Paramètre qui permet de désactiver la configuration automatique du réseau.

OSI (LE MODÈLE): « ***O****pen* ***S****ystems* ***I****nterconnection* », modèle de communications entre ordinateurs proposé par l'ISO (Organisation internationale de normalisation). Il décrit les fonctionnalités nécessaires à la communication et l'organisation de ces fonctions.

PING : Commande qui permet de savoir si un nœud du réseau est accessible et si oui de connaître sa réactivité.

QUATTOR: Quattor est un système d'administration qui permet l'installation, la configuration et la gestion de nombreux ordinateurs de façon centralisée.

ROUTE: Terme transparent, les routes se situent dans les tables de routage et indiquent quel chemin suivre pour accéder à une destination.

SSH : « *Secure Shell* », c'est un protocole sécurisé qui permet de se connecter à des serveurs distants. C'est le successeur de telnet.

SWITCH: Terme anglais correspondant à « commutateur réseau ».

VLAN: « ***V****irtual* ***L****ocal* ***A****rea* ***N****etwork* », comme son nom l'indique il sert à créer un réseau local virtuel, ce qui permet de générer des sous réseaux sans se heurter à des contraintes physiques.

XML: « *e****X****tensible* ***M****arkup* ***L****anguage* », que l'on peut traduire par « langage à balises étendues ». C'est un langage du même type que le HTML, mais pour lequel on peut définir de nouvelles balises personnelles.

12 Bibliographie & Webographie:

LANGUE FRANÇAISE SPÉCIFICITÉS LIÉES AU LANGAGE INFORMATIQUE :
http://franceterme.culture.fr/FranceTerme/
http://www.culture.gouv.fr/culture/dglf/

PERL:
http://www.med.univ-rennes1.fr/~poulique/cours/perl/perl_html/introperl.html
Cours d'introduction à Perl de l'université de Rennes.

http://www.perl.org/
Site internet officiel du langage Perl

http://perldoc.perl.org/
Site internet officiel de la documentation du langage Perl

Programming Perl
Wall/Christiansen/Orwant - Troisième edition, chez O'Reilly

Learning Perl
Schwartz, chez O'Reilly

QUATTOR :
www.quattor.org

RED HAT ET SYSTÈME D'EXPLOITATION SIMILAIRES :
www.redhat.com
www.centos.org
fedoraproject.org

12.1 Sources :

Les supports dont la source n'est pas explicitement mentionnée proviennent du CERN ou de la communauté Quattor.

Institut
Supérieur
d' **I**nformatique
de **M**odélisation
et de leurs
Applications
Campus des Cézeaux
BP 10125
63173 Aubière Cedex

Organisation
Européenne
pour la **R**echerche
Nucléaire
CH 1211
Genève 23 Suisse

Rapport d'ingénieur
Stage de 3ème année
Filière 1 : Informatique des systèmes embarqués

Réécriture du Module réseau de Quattor Pour l'expérience LHCb au CERN

ANNEXE

Présenté par : **S. Cipière**
Responsable ISIMA : **E. Mesnard**
Responsable CERN : **L. Brarda**

Durée : 5 mois
avril à septembre
2010

Annexe

Table des illustrations

1 Cahier des charges

Cahier des charges V0.1
Réécriture du module réseau Quattor

l. Brarda, e. Bonaccorsi

7 mai 2010

Réécrire le module réseau de Quattor "`network.pm.cin`", en Perl.

Implémenter :
- IP
- VLAN
- BONDING
- ROUTING
- ALIAS
- ETHTOOL

Implémentation optionnelles :
- l'IP POLICY ROUTING
- QoS
- GVRP
- IPv6

Le reboot des machines est à éviter au maximum.

Il faudra suivre le guide de style :
https ://trac.lal.in2p3.fr/Quattor/wiki/Web/CodingStyle.

Le stagiaire devra couvrir l'ensemble du cycle de développement de la rédaction du cahier des charges à la mise en production.

Illustration 1 : **Cahier des charges 1/2**

littéraire

L'objectif du projet est de mettre à jour et revoir complétement le module de connexion réseau du projet QUATTOR. Il s'agit de réécrire totalement et depuis zéro ce module qui est écrit en Perl. Actuellement le fichier se nomme `network.pm.cin` . la révision en usage est la 1858, c'est un module de mille lignes de code qui a pour but de configurer comme son nom l'indique la partie réseau de la machine cible. Le module en charge des connexions réseau existant est un module âgé, qui date de 2005. L'avis de la communauté des utilisteurs de Quattor aura un impact sur la rédaction du CDC final.

Les fonctionalités qui ont été identifiées comme principales sont la gestion des protocoles et services permettant d'utiliser IP, VLAN, BONDING, ROUTING, ALIAS, et ETHTOOL. Puis pourront être concidérées les implémenations de l'IP POLICY ROUTING, QOS, GVRP ainsi que la gestion des adresses en IPV6.

Le langage utilisé pour répondre à cette problématique est le PERL qui grâce aux expressions régulières permet d'appliquer les commandes désirées aux systèmes à configurer. L'un des points clefs à considérer sera de limiter au maximum d'avoir à redémarrer les machines à configurer. Ce qui permet de ne pas interrompre la production.

Une fois le module réalisé il faudra le tester dans le but de le valider. A cet effet les tests auront pour cible un échantillon représentatif des machines en usage dans les installations gérées à l'aide de QUATTOR. Les tests devront être variés et essayer de couvrir un éventail aussi large que possible des cas généraux et particuliers qui pourront se présenter durant l'exploitation.

Après la validation en simulation et en ayant pour cible des machines isolées, il faudra considérer le test à plus grande échelle. Pour se faire il faudra prévoir comment restaurer, rapidement et correctement les systèmes en cas de problèmes (ce qui ne devrait pas se produire). En effet le SYSTÈME D'INFORMATION (SI) du LHCB est stratégique et il est donc de mise d'être extrêmement prudent pour ne pas perturber son bon fonctionnement.

Ainsi la mise en production d'un module aussi capital que le module réseau dont un mauvais fonctionnement pourrait déconnecter du SI tout ou partie des machines, doit être une tâche approchant l'excellence.

Illustration 2 : **Cahier des charges 2/2**

2 Spécifications

Après avoir reçu le cahier des charges du projet qui m'était dévolu, j'ai rédigé les spécifications de ce dernier, en conservant deux objectifs principaux. Le premier était d'être sûr que ce que j'allai coder corresponde aux attentes de mon maître de stage. Une fois les spécifications validées, on disperse une grande partie des incompréhensions qui peuvent exister entre les clients et les programmeurs. Le deuxième objectif, pour lequel j'ai rédigé des spécifications aussi détaillées, était de résumer en un seul lieu mes buts et les moyens de les atteindre. Ainsi, le tableau de l'illustration 3 indique où se situe la source à l'intérieur de l'arbre transmis au travers du fichier xml, puis quels sont les fichiers à impacter et la ligne à ajouter ou modifier dedans ; ce tableau précise aussi la fonctionnalité visée.

fonctionnalités	sources	fichiers	variable	commandes
bonding	Ø	/etc/sysconfig/network-scripts/ifcfg-bondx	Ø	/sbin/ifenslave
broadcast address	/system/network/interfaces/[dev]/broadcast	/etc/sysconfig/network-scripts/ifcfg-[dev]	BROADCAST = 'addr'	/sbin/ifconfig [dev] broadcast 'adr'
default gateway address	/system/network/guess_default_gateway	/etc/sysconfig/network	Ø	/sbin/route add default gw 'addr'
domain name	/system/network/domainname	/etc/sysconfig/network	Ø	Ø
down	Ø	Ø	Ø	/sbin/ifdown [dev]
ethtool options	Ø	/etc/sysconfig/network-scripts/ifcfg-ethx	Ø	/sbin/ethtool
host name	/system/network/hostname	/etc/sysconfig/network	Ø	/bin/hostname 'name'
gateway address	/system/network/default_gateway	/etc/sysconfig/network	Ø	/sbin/route add gw 'address'
gateway address	/system/network/interfaces/[dev]/route/[j]/gateway	/etc/sysconfig/network-scripts/route-[dev]	Ø	/sbin/ip route add [net] via [addr] dev [dev]
gateway device	/system/network/gatewaydev	/etc/sysconfig/network	GATEWAY = [dev]	Ø
IP address	/system/network/interfaces/[dev]/route/[j]/address	/etc/sysconfig/network-scripts/route-[dev]	Ø	/sbin/ip route add [net] via [addr] dev [dev]
IP address	/system/network/interfaces/[dev]/ip	/etc/sysconfig/network-scripts/ifcfg-[dev]	IPADDR = 'addr'	/sbin/ifconfig [dev] 'addr'
IP forwarding	/system/network/interfaces/[dev]/ip_forwarding	/proc/sys/net/ipv4/ip_forward	echo [1 - 0] > ip_forward	Ø
IP forwarding	/system/network/interfaces/[dev]/ip_forwarding	/etc/sysconfig/network	FORWARD_IPV4 = [yes - no]	Ø
MTU	/system/network/interfaces/[dev]/mtu	/etc/sysconfig/network-scripts/ifcfg-[dev]	MTU = 'size'	/sbin/ifconfig [dev] mtu size
Name of the bond's master	/system/network/interfaces/[dev]/master	/etc/sysconfig/network-scripts/ifcfg-[dev]	MASTER = [dev] and SLAVE=yes	Ø
Name of the device	/system/network/interfaces/[dev]/device	/etc/sysconfig/network-scripts/ifcfg-[dev]	DEVICE = 'name'	Ø
Netmask address	/system/network/interfaces/[dev]/netmask	/etc/sysconfig/network-scripts/ifcfg-[dev]	NETMASK = 'addr'	/sbin/ifconfig [dev] netmask 'mask'
netmask address	/system/network/interfaces/[dev]/route/[j]/netmask	/etc/sysconfig/network-scripts/route-[dev]	Ø	/sbin/ip route add [net] via [addr] dev [dev]
NIS domain	/system/network/nisdomain	/etc/sysconfig/network	NISDOMAIN = 'name'	Ø
other broadcast address	/system/network/interfaces/[dev]/aliases/[name] /broadcast	/etc/sysconfig/network-scripts/ifcfg-[dev]:<alias-name>	BROADCAST = 'addr'	/sbin/ifconfig [dev] broadcast 'adr'
other IP address	/system/network/interfaces/[dev]/aliases/[name]/ip	/etc/sysconfig/network-scripts/ifcfg-[dev]:<alias-name>	IPADDR = 'addr'	/sbin/ifconfig [dev] 'addr'

other name	/system/network/interfaces/[dev]/aliases/[name]/name	/etc/sysconfig/network-scripts/ifcfg-[dev]:<alias-name>	DEVICE = 'name'	/sbin/ifconfig [dev]
other netmask	/system/network/interfaces/[dev]/aliases/[name]/netmask	/etc/sysconfig/network-scripts/ifcfg-[dev]:<alias-name>	NETMASK = 'addr'	/sbin/ifconfig [dev] netmask 'mask'
protocol type	/system/network/interfaces/[dev]/bootproto	/etc/sysconfig/network-scripts/ifcfg-[dev]	BOOTPROTO = 'type'	Ø
TCP segmentation offloading	/system/network/interfaces/[dev]/offload/tso	Ø	Ø	ethtool -K [dev] tso [on - off]
Type (ex.Ethernet)	/system/network/interfaces/[dev]/type	/etc/sysconfig/network-scripts/ifcfg-[dev]	TYPE = 'type'	Ø
real host name	/system/network/realhostname	/etc/sysconfig/network	Ø	Ø
Start on boot	/system/network/interfaces/[dev]/onboot	/etc/sysconfig/network-scripts/ifcfg-[dev]	ONBOOT = [yes - no]	Ø
up	Ø	Ø	Ø	/sbin/ifup [dev]
user control	/system/network/interfaces/[dev]/usrctl	/etc/sysconfig/network-scripts/ifcfg-[dev]	USERCTL = [yes - no]	Ø
user control	/system/network/interfaces/[dev]/srcaddr	/etc/sysconfig/network-scripts/ifcfg-[dev]	SRCADDR = 'addr'	Ø
Vlan	Ø	/etc/sysconfig/network-scripts/ifcfg-ethx.y	Ø	/sbin/vconfig add [dev] Num.Vlan
wake up on line	/system/network/interfaces/[dev]/ethtool/wol	Ø	Ø	Ø
zeroconf	/system/network/nozeroconf	/etc/sysconfig/network	NOZEROCONF = [yes - no]	Ø

Illustration 3 : **Tableau récapitulatif des fichiers cibles et différentes commandes mises en œuvre**

3 Fichier XML de configuration réseau

```
<nlist name="network">
    <string name="default_gateway">10.128.16.254</string>
    <string name="domainname">lbdaq.cern.ch</string>
    <string name="hostname">admmon01</string>
    <nlist name="interfaces">
        <nlist name="bond0">
            <string name="broadcast">10.143.255.255</string>
            <string name="driver">bonding</string>
            <string name="ip">10.128.16.30</string>
            <string name="netmask">255.255.255.0</string>
            <list name="route">
                <nlist>
                    <string name="address">10.136.0.0</string>
                    <string name="gateway">10.128.16.254</string>
                    <string name="netmask">255.255.0.0</string>
                </nlist>
                <nlist>
                    <string name="address">10.137.0.0</string>
                    <string name="gateway">10.128.16.254</string>
                    <string name="netmask">255.255.0.0</string>
                </nlist>
            </list>
        </nlist>
        <nlist name="eth0">
            <string name="bootproto">none</string>
            <string name="broadcast">10.143.255.255</string>
            <string name="gateway">10.128.2.2</string>
            <string name="ip">10.128.16.30</string>
            <string name="master">bond0</string>
            <string name="netmask">255.255.255.0</string>
            <string name="onboot">yes</string>
        </nlist>
        <nlist name="eth1">
            <string name="bootproto">none</string>
            <string name="master">bond0</string>
        </nlist>
        <nlist name="eth2">
            <string name="broadcast">137.132.10.255</string>
            <string name="ip">10.132.10.13</string>
            <string name="netmask">255.255.255.0</string>
            <string name="onboot">yes</string>
        </nlist>
    </nlist>
    <list name="nameserver">
        <string>10.128.16.5</string>
        <string>10.130.16.5</string>
    </list>
    <boolean name="realhostname">true</boolean>
</nlist>
<string name="rootmail">lhcb-online-admins@cern.ch</string>
<string name="siterelease">Scientific Linux sl5</string>
</nlist>
```

Illustration 4 : **Partie réseau d'un fichier de configuration xml**

4 Doxyfilter

Doxygen fournit une syntaxe et des outils qui permettent de commenter le code informatique d'une façon unifiée, quel que soit le langage utilisé, en générant des documents au format HTML, très pratique à communiquer aux partenaires. Cependant, en avril 2010, le langage perl n'était pas supporté nativement par Doxygen. C'est grâce à la solution **Doxyfilter,** proposée par Bart Schuller et Tom Aeby, qu'il devient possible de commenter le perl à l'aide des solutions offertes par Doxygen.

Doxyfilter est accessible depuis http://www.bigsister.ch/doxygenfilter/ et est distribué sous licence libre (Larry Wall's "Artistic License").

Comme mentionné précédemment, le but est d'utiliser un système de commentaire unifié, comme vous pouvez le constater, le style des commentaires est très proche de ce que Doxygen propose en C.

```
## @fn ma_fonction($arg1, %arg2)
# Description de ma fonction
# @param    $arg1 présentation de mon premier argument
# @param    %arg2 présentation de mon second argument - ici …
# @test     commentaires sur les tests
# @return   type de retour
# @bug      bug connu
Sub ma_fonction
{… corps de ma function ; …
}
```

Illustration 5 : **Exemple de commentaire en Doxygen pour perl**

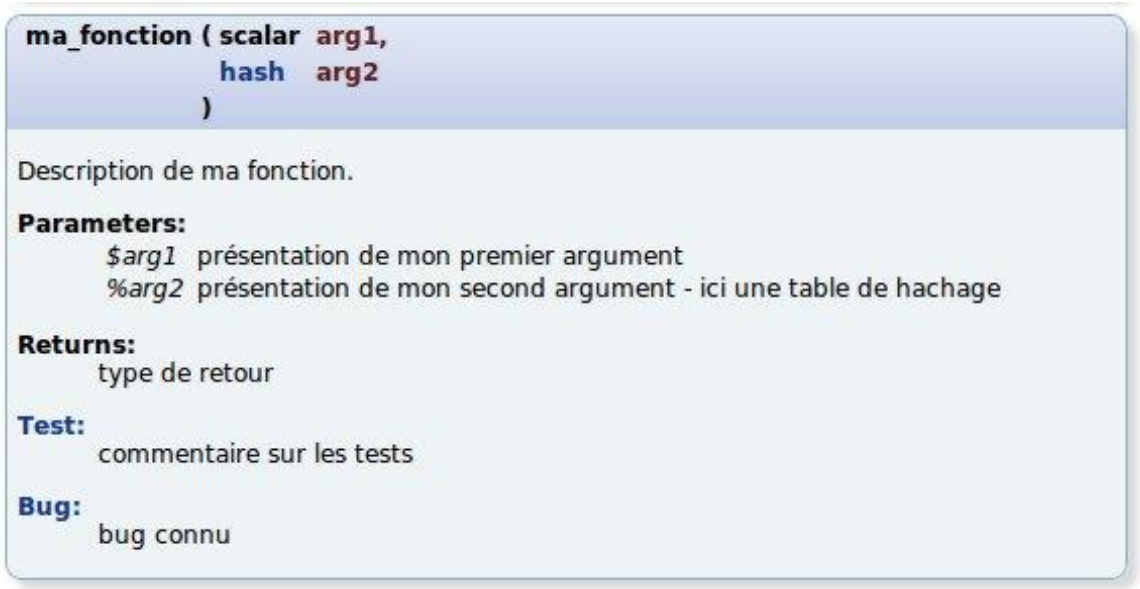

Illustration 6 : **Exemple de résultat dans un navigateur internet**

5 Récapitulatif des tests effectués

Paramètres	Avec redémarrage du réseau	Sans redémarrage du réseau	commentaires
Alias	✓	✓	Cf. “cas remarquables(8)”
Bonding	✓	/	
Bonding options	✓	/	
Bootproto	✓	/	
Broadcast	✓	✓	
Ethtool	✓	✓	
Ethtool Offload	/	/	
Ethtool Ring	/	/	
Gateway	✓	✓	
IP	✓	✓	
IP_forwarding	✓	✓	
MTU	✓	✓	
Network->restart	✓	/	
Netmask	✓	✓	
NISdomain	✓	✓	
On_boot	✓	✓	
Route	✓	✓	
Set_hwaddr	✓	✗	
Type	✓	/	
Usrctl	✓	✓	
VLAN	✓	✓	

Illustration 7 : **Tableau récapitulatif des tests effectués**

6 Les expressions régulières

Les expressions régulières (ER), que nous pouvons aussi nommer expressions rationnelles, sont des solutions très puissantes et malléables pour la recherche et l'identification de mots à l'intérieur de chaînes de caractères. Comme je vous l'ai exposé dans la section de présentation du langage perl, ces ER peuvent vite devenir difficiles à cerner au premier abord. C'est pour ça que j'ai rassemblé dans un seul tableau (cf. annexe6) toutes les différentes sortes de symboles et de composition de ces symboles que j'ai pu rencontrer au long de mes recherches. Cet aide-mémoire peut s'avérer très utile pour qui souhaite comprendre un code perl mal ou pas commenté, et usant des subtilités du langage.

On peut également sélectionner et rappeler les éléments qui auront été trouvés lors de l'analyse de l'expression régulière. De fait, voici une partie du code de mon prédécesseur qui me permettra d'illustrer mes propos.

```
my $ifconfig_mac_regexp='^(\S+)\s+.*?HWaddr\s+([\dA-Fa-f]{2}([:-])
  [\dA-Fa-f]{2}(\3[\dA-Fa-f]{2}){4})\s+';

my (%dev2mac,%mac2dev);
foreach my $tmp_dev (@ifconfig_devs) {
    $tmp_dev =~ s/\n/ /g;
    if ($tmp_dev =~ m/$ifconfig_mac_regexp/) {
        $dev2mac{$1}=$2;
        $mac2dev{$2}=$1;
    }
}
```

Illustration 8 : **Exemple du code précédent du composant réseau relatif aux ER**

Il me semble important de préciser que je n'ai retiré aucun commentaire de ces lignes de code. Ici, vous pouvez voir comment l'on peut récupérer ces éléments à l'aide des parenthèses puis de $x, où 'x' est un entier indiquant à quel contenu le programmeur souhaite accéder. Pour vous simplifier la lecture, j'ai mis en évidence ce mécanisme avec des couleurs. Le but de ces lignes étant d'associer dans deux tables de hachage le périphérique à son adresse MAC, et réciproquement l'adresse MAC à son périphérique.

.	Match any caracter
*	Match 0 or more times
*?	Match 0 or more times, not greedily
*+	Match 0 or more times and give nothing back
+	Match 1 or more times
++	Match 1 or more times and give nothing back
+?	Match 1 or more times, not greedily
?	Match 1 or 0 times
?+	Match 0 or 1 time and give nothing back
??	Match 0 or 1 time, not greedily
n	Match exactly n times
n?	Match exactly n times, not greedily
n,	Match at least n times
n,?	Match at least n times, not greedily
n+	Match exactly n times and give nothing back (redundant)
n,+	Match at least n times and give nothing back
n,m	Match at least n but not more than m times
n,m?	Match at least n but not more than m times, not greedily
n,m+	Match at least n but not more than m times and give nothing back
\033	octal char (example : ESC)
\a	alarm (bell) (BEL)
\cK	control char (example : VT)
\C	Match a single C char (octet) even under Unicode.
\d	Match a digit character
\D	Match a non-digit character
\e	escape (think troff) (ESC)
\E	end case modification (think vi)
\f	form feed (FF)
\h	Horizontal whitespace
\H	Not horizontal whitespace
\g1	Backreference to a specific or previous group,
\g-1	number may be negative indicating a previous buffer and may
\gname	Named backreference
\k<name>	Named backreference
\K	Keep the stuff left of the \K, don't include it in $&
\l	Backreference to a specific group.
\l	lowercase next char (think vi)
\L	lowercase till \E (think vi)

Illustration 9 : **Tableau récapitulatif des ER 1ère partie (1/2)**

\n	newline (LF, NL)
\N	Any character but \n (experimental)
\Nname	named Unicode character
\NU+263D	Unicode character (example : FIRST QUARTER MOON)
\pP	Match P, named property. Use \pProp for longer names.
\PP	Match non-P
\Q	quote (disable) pattern metacharacters till \E
\r	return (CR)
\R	Linebreak
\s	Match a whitespace character
\S	Match a non-whitespace character
\t	tab (HT, TAB)
\u	uppercase next char (think vi)
\U	uppercase till \E (think vi)
\v	Vertical whitespace
\V	Not vertical whitespace
\w	Match a “word” character (alphanumeric plus "_")
\W	Match a non-"word" character
\X	Match Unicode "eXtended grapheme cluster"
\x1B	hex char (example : ESC)
\x263a	long hex char (example : Unicode SMILEY)
[[:alpha:]]	Any alphabetical character.
[[:alnum:]]	Any alphanumerical character.
[[:ascii:]]	Any character in the ASCII character set.
[[:blank:]]	A GNU extension, equal to a space or a horizontal tab
[[:cntrl:]]	Any control character.
[[:digit:]]	Any decimal digit, equivalent to “\d”.
[[:graph:]]	Any printable character, excluding a space.
[[:lower:]]	Any lowercase character.
[[:print:]]	Any printable character, including a space.
[[:punct:]]	Any graphical character excluding "word" characters.
[[:space:]]	Any whitespace character. “\s” plus vertical tab (“\cK”).
[[:upper:]]	Any uppercase character.
[[:word:]]	A Perl extension, equivalent to “\w”.
[[:xdigit:]]	Any hexadecimal digit.

Illustration 10 : **Tableau récapitulatif des ER 2ème partie (2/2)**