Architectures et mécanismes de fédération dans les environnements cloud computing et cloud networking

Houssem Medhioub

THÈSE DE DOCTORAT CONJOINT TELECOM SUDPARIS ET L'UNIVERSITÉ PIERRE ET MARIE CURIE

SPÉCIALITÉ

Informatique

École doctorale Informatique, Télécommunications et Électronique de Paris

Présentée par

Houssem MEDHIOUB

Pour obtenir le grade de
DOCTEUR

Architectures et mécanismes de fédération dans les environnements Cloud Computing et Cloud Networking

Soutenue le 28 Avril 2015 devant le jury composé de :

M. Olivier PERRIN	INRIA Lorraine, Université Nancy 2	Rapporteur
M. Pascal LORENZ	Université de Haute Alsace	Rapporteur
M. Adrien LEBRE	Mines de Nantes, INRIA	Examinateur
M. Stefano SECCI	UPMC	Examinateur
M. Djamal ZEGHLACHE	Télécom SudParis	Directeur de thèse

Thèse n°2015TELE0009

Dédicaces

À ma chère famille...

Remerciements

C'est avec un grand plaisir que je tiens tout d'abord à exprimer toutes mes profondes reconnaissances et ma gratitude à tous ceux qui ont participé de près ou de loin à l'élaboration de ce travail dans les meilleures conditions.

Je tiens également à remercier Monsieur Olivier PERRIN, Professeur à l'Université de Lorraine, Monsieur Pascal LORENZ, Professeur à l'Université de Haute Alsace qui m'ont fait l'honneur d'être les rapporteurs de mon manuscrit de doctorat. J'associe à ces remerciements Monsieur Adrien LEBRE, chercheur à l'INRIA et Maître de Conférences à l'Ecole des Mines de Nantes et Monsieur Stefano SECCI Maître de Conférences à l'Universite Pierre et Marie Curie pour avoir accepté de faire partie des membres de jury et d'assister en qualité d'examinateurs à la soutenance de cette thèse. Je tiens à vous exprimer mes profonds respects, mes vifs remerciements et toute ma gratitude d'avoir consacré une partie de votre temps à lire et juger ce travail.

Je voudrais remercier tout particulièrement mon Directeur de thèse et encadrant Monsieur Djamal ZEGHLACHE qui n'a pas épargné le moindre effort dans l'encadrement de ce doctorat afin de me permettre de défier les entraves rencontrées et de travailler avec volonté. Il a été toujours disponible pour m'orienter à entreprendre les bonnes décisions. Durant toute l'aventure professionnelle dans le monde du Cloud que nous avons traversée ensemble, j'ai beaucoup apprécié ses grandes qualités morales et son extrême modestie. Qu'il trouve dans ce travail le fruit de son effort et l'expression de ma profonde gratitude.

Je remercie aussi tous les membres du département RS2M au sein de Télécom SudParis pour l'ambiance chaleureuse et avec qui j'ai développé mon sens de collaboration, d'échange et de partage.

Je profite de cette occasion pour exprimer mes reconnaissances à tous mes amis qui ont toujours été à mes côtés pour me supporter.

Résumé

Présenté dans la littérature comme une nouvelle technologie, le Cloud Computing est devenu incontournable dans la mise en place et la fourniture des services informatiques. Cette thèse s'inscrit dans le contexte de cette nouvelle technologie qui est en mesure de transformer la mise en place, la gestion et l'utilisation des systèmes d'information. L'adoption et la vulgarisation du Cloud ont été ralenties par la jeunesse même des concepts et l'hétérogénéité des solutions existantes. Cette difficulté d'adoption se manifeste par l'absence de standard, l'hétérogénéité des architectures et des API, le Vendor Lock-In imposé par les leaders du marché et des manques qui ralentissent la fédération.

La motivation principale de la thèse est de simplifier l'adoption du cloud et la migration vers ses environnements et technologies. Notre objectif est de proposer des solutions d'interopérabilité et de fédération dans le Cloud. Le travail de recherche s'est aussi articulé autour de deux grands axes. Le premier concerne le rapprochement des réseaux du futur et des Clouds. Le deuxième axe concerne l'interopérabilité et la fédération entre solutions et services cloud.

Une analyse de l'état de l'art sur le Cloud Computing et le Cloud Networking, a permis de confirmer des manques pressentis et de proposer deux architectures de fédération Cloud. La première architecture permet le rapprochement entre le Cloud Computing et le Cloud Networking. La seconde architecture facilite l'interopérabilité et le courtage de services Cloud. L'étude des deux architectures a fait ressortir deux composants primordiaux et essentiels pour assurer la fédération : une interface générique et un système d'échange de messages. Ces deux composants correspondent à deux contributions centrales de la thèse et reflètent l'ensemble des contributions (quatre au total) du travail de recherche.

Mots clés : Cloud Computing, Cloud Networking, Cloud Hybride, IaaS, PaaS, Fédération, Interopérabilité, Courtage, Open Cloud Interface, OCNI, Cloud Messaging, CMBS

Abstract

Presented in the literature as a new technology, Cloud Computing has become essential in the development and delivery of IT services. Given the innovative potential of Cloud, our thesis was conducted in the context of this promising technology. It was clear that the Cloud would change the way we develop, manage and use information systems. However, the adoption and popularization of Cloud were slow and difficult given the youth of the concepts and heterogeneity of the existing solutions. This difficulty in adoption is reflected by the lack of standard, the presence of heterogeneous architectures and APIs, the introduction of *Vendor Lock-In* imposed by the market leaders and the lack of cloud federation principles and facilitators.

The main motivation for our PhD is to simplify the adoption of the cloud paradigm and the migration to cloud environments and technologies. Our goal has consequently been to improve interoperability and enable federation in the cloud. The thesis focused on two main areas. The first concerns the convergence of future networks and clouds and the second the improvement of federation and interoperability between heterogeneous cloud solutions and services.

Based on our work in state of the art about Cloud Computing and Cloud Networking, we defined in this thesis two architectures for Cloud federation. The first architecture enables the merging (convergence) of Cloud Computing and Cloud Networking. The second architecture addresses interoperability between services and proposes cloud-brokering solutions. The study enabled the identification of two essential components for cloud federation, namely : a generic interface and a message exchange system. These two components have been two contributions of our thesis. The proposed federation architectures and these two components summarize the four major contributions of our work.

Keywords

Cloud Computing, Cloud Networking, Hybrid Cloud, IaaS, PaaS, Federation, Interoperability, Brokering, Open Cloud Interface, OCNI, Cloud Messaging, CMBS

Table des matières

Glossaire

Amazon EC2	Amazon Elastic Compute Cloud EC2.
AMQP	Advanced Message Queuing Protocol.
APaaS	Application Platform as a Service.
API	Application Programming Interface.
AWS	Amazon Web Services.
BaaS	Backup as a Service.
BaaS	Backend as a Service.
BGP	Border Gateway Protocol.
CaaS	Communication as a Service.
CC	Cloud Computing.
CDMI	Cloud Data Management Interface.
CF	Cloud Federation.
CI	Cloud Interface.
CIMI	Cloud Infrastructure Management Interface.
CloNe	Cloud Networking.
CM	Cloud Messaging.
CMBS	Cloud Message Brokering Service.
CN	Cloud Networking.
CNG	Cloud Networking Gateway.
cNIS	Common Network Information Service.
CSA	Cloud Security Alliance.
CSCC	Cloud Standards Customer Council.
DaaS	Data as a Service.
DaaS	Desktop as a Service.
DBaaS	Database as a Service.
DCaaS	Data Center as a Service.

DCP	Distributed Control Plane.
DMTF	Distributed Management Task Force.
DPaaS	Database Platform as a Service.
DRaaS	Disaster Recovery as a Service.
E-VPN	Ethernet Virtual Private Network - Ethernet VPN.
EDA	Event-Driven Architecture.
EoR	End-of-Row.
ETSI	European Telecommunications Standards Institute.
EVB	Edge Virtual Bridging.
FNS	Flash Network Slice.
GRE	Generic Routing Encapsulation.
HaaS	Hardware as a Service.
HPCaaS	High Performance Computing as a Service.
HTTP	HyperText Transfer Protocol.
IaaS	Infrastructure as a Service.
IDaaS	Identity as a Service.
IEC	Internation Electrotechnical Commission.
IETF	Internet Engineering Task Force.
INDL	Infrastructure and Network Description Language.
IPaaS	Integration Platform as a Service.
IRF	Intelligent Resilient Framework.
ISO	International Organization for Standardization.
JMS	Java Message Service.
JSON	JavaScript Object Notation.
LAN	Local Area Network.
LISP	The Locator/ID Separation Protocol.

MaaS	Management as a Service.
MaaS	Metal as a Service.
MaaS	Monitoring as a Service.
MC-LAG	Multi-Chassis Link Aggregation.
MEF	Metro Ethernet Forum.
MOM	Message Oriented Middleware.
MPLS	Multi-Protocol Label Switching.
MQ-B	Message Queuing Broker.
MQ-LB	Message Queuing Load Balancer.
MQ-S	Message Queuing Service.
MQaaS	Message Queues as a Service.
MQTT	Message Queuing Telemetry Transport.
NaaS	Network as a Service.
NCF	Network and Cloud Federation.
NDL	Network Description Language.
NIST	National Institute of Standards and Technology.
NM/perfSONAR	Network Measurement/perfSONAR.
NML	Network Mark-up Language.
NNDL	Network Node Description Language.
NVGRE	Network Virtualization using Generic Routing Encapsulation.
NVP	Nicira Network Virtualization Platform.
OASIS	Organization for the Advancement of Structured Information Standards.
OCC	Open Cloud Consortium.
OCCI	Open Cloud Computing Interface.
OCI	Open Cloud Initiative.
OCNI	Open Cloud Networking Interface.
ODCA	Open Data Center Alliance.
OGF	Open Grid Forum.
ONF	Open Networking Foundation.
PaaS	Platform as a Service.

PBB	Provider Backbone Bridging.
PBB-EVPN	Provider Backbone Bridging E-VPN.
PRaaS	Process as a Service.
pyCMBS	Python Cloud Message Brokering Service.
pyOCNI	Python Open Cloud Networking Interface.
RaaS	Resource as a Service.
RBridges	Routing Bridges.
RDF	Resource Description Framework.
REST	REpresentational State Transfer.
RPC	Remote Procedure Call.
SaaS	Software as a Service.
SDL	System Description language.
SDN	Software-defined Networking.
SECaaS	Security as a Service.
SLA	Service Level Agreement.
SNIA	Storage Networking Industry Association.
SOA	Service Oriented Architecture.
SoA	State Of the Art.
SPB	Shortest Path Bridging.
SPOF	Single Point Of Failure.
SQS	Amazon Simple Queue Service.
STaaS	Storage as a Service.
STOMP	Streaming Text Oriented Message Protocol.
STP	Spanning Tree Protocol.
STT	Stateless Transport Tunneling Protocol.
ToR	Top-of-Rack.
TOSCA	Topology and Orchestration Specification for Cloud Applications.
TRILL	Transparent Interconnection of Lots of Links.
UML	Unified Modeling Language.
vCDNI	vCloud Director Networking Infrastructure.
VCS	Virtual Cluster Switching.

VLAN	Virtual Local Area Network.
VM	Virtual Machine.
VPLS	Virtual Private LAN Service.
VPN	Virtual Private Network.
VXDL	Virtual eXecution Infrastructures Description Language.
VXLAN	Virtual Extensible LAN.
W3C	World Wide Web Consortium.
WAN	Wide Area Network.
WSGI	Web Server Gateway Interface.
XaaS	Everything as a Service.
XML	Extensible Markup Language.
XMPP	Extensible Messaging and Presence Protocol.
ZMTP	ZeroMQ Message Transport Protocol.

Table des figures

Liste des tableaux

Chapitre 1

Introduction Générale

1.1 Contexte

Depuis plusieurs années et selon différentes entreprises de conseils types Gartner et IDC, le Cloud Computing fait partie des dix premières technologies qui ont le potentiel d'affecter les individus, les entreprises et les systèmes d'information. Vu le potentiel d'innovation du Cloud, notre thèse s'inscrit dans le contexte de cette technologie prometteuse. Afin de bien situer et illustrer nos travaux par rapport à l'état du marché et la maturité des concepts et solutions existantes, nous considérons l'évolution de deux schémas publiés par *Gartner* [35] : le *Hype Cycle* et le *Magic Quadrant*. Ces schémas nous permettent d'avoir une vision sur l'état et l'évolution du Cloud. De ce fait, nos travaux de recherches ont commencé alors que les notions basiques du Cloud en général et les notions de fédération en particulier n'étaient pas encore définies.

L'intérêt du schéma Gartner *Hype Cycle* est de représenter l'évolution des technologies par rapport à leurs maturités, leurs visibilités et leurs adoptions. Publié une fois par an, ce schéma permet d'illustrer l'évolution des technologies dans une courbe de progression de type *Hype Cycle*.

Comme toute technologie introduite dans le *Hype Cycle*, le Cloud suit cinq principales phases. Ces phases, qui sont illustrées dans la figure 1.1, sont :

- **Le déclencheur technologique** : C'est la première phase qui illustre l'arrivée sur le marché d'une nouvelle technologie. Dans le cas du Cloud, il a fait son apparition en 2008.
- **Le pic des attentes exagérées** : Dans cette phase, on assiste à un emballement sur les technologies avec des attentes exagérées et non réalistes. Cette situation est due à la jeunesse voire l'absence des modèles fiables et reconnus des technologies et des solutions. Le Cloud a perduré dans cette phase pendant trois ans entre 2009 et 2011.
- **Le creux du désillusionnement** : Durant cette phase, les technologies n'arrivent pas à répondre aux attentes des utilisateurs. Par conséquent, elles sont rapidement

remises en question, voire délaissées. Le Cloud n'a pas fait exception et il a perduré dans cette phase depuis 2012. De plus, dans la version 2014 du Hype Cycle, la notion du Cloud Hybride a fait son apparition dans cette phase.

- **La pente de l'éclaircissement** : Dans cette phase, la vision et les architectures de la technologie commencent à bien se définir. De plus, les solutions deviennent plus fiables et leurs nombres d'utilisateurs grandissent. L'adoption des utilisateurs se fait lentement, mais sûrement. Le Cloud n'est toujours pas dans cette phase, mais il ne devrait pas tarder à y entrer.
- **Le plateau de la productivité** : Durant cette phase, les technologies sont bien reconnues et les solutions sont bien stables. Dans sa publication de 2014, Gartner a prédit que le Cloud atteindra cette phase entre 2016 et 2019.

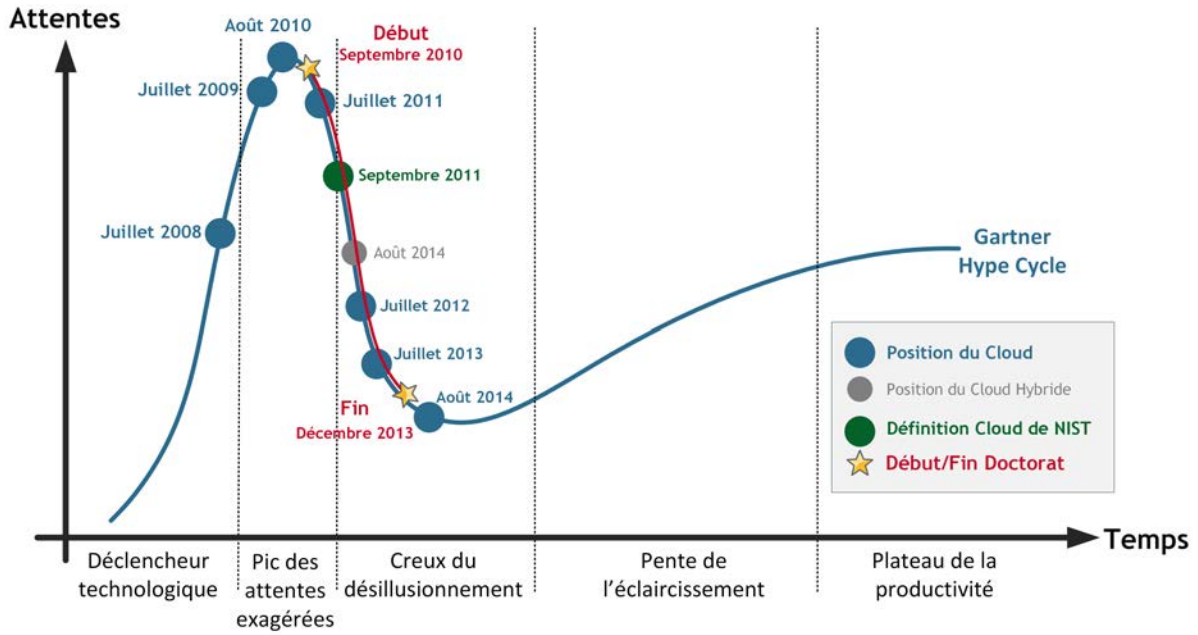

Figure 1.1 – *Début et fin du doctorat par rapport à l'évolution du Cloud dans le Gartner Hype Cycle et l'apparition de la définition NIST du Cloud*

Afin de bien positionner le contexte de notre doctorat, nous avons ajouté la date de début et de fin de notre doctorat par rapport à l'évolution du Cloud dans la figure 1.1 du *Hype Cycle*. Le début officiel, qui était en septembre 2010, se positionne en pleine phase des pics des attentes exagérées alors que les notions du Cloud n'étaient pas encore définies. Quant à la date de fin, qui était en décembre 2013, elle coïncide avec la limite inférieure de la courbe décroissante de la phase du creux de désillusionnement. À base du *Gartner Hype Cycle*, il est bien clair que les travaux de ce doctorat ont été réalisés pendant une période là où les concepts et les technologies ne sont pas encore matures. D'ailleurs, la définition NIST du Cloud, communément reconnue de nos jours, a été établie en septembre 2011, une année après le début officiel de nos travaux de doctorat.

Le deuxième schéma Gartner qui met en évidence le contexte de nos travaux est le

Magic Quadrant. Ce schéma classifie les fournisseurs de Cloud IaaS en quatre quadrants selon l'aptitude à exécuter et l'exhaustivité de la vision. Les quatre quadrants sont :

- **Les leaders** qui ont une bonne aptitude à exécuter et une bonne vision ;
- **Les challengers** qui ont une aptitude à exécuter ;
- **Les visionnaires** qui ont une bonne vision ;
- **Les marchés de niches** qui n'ont ni une grande capacité à exécuter ni une excellente vision.

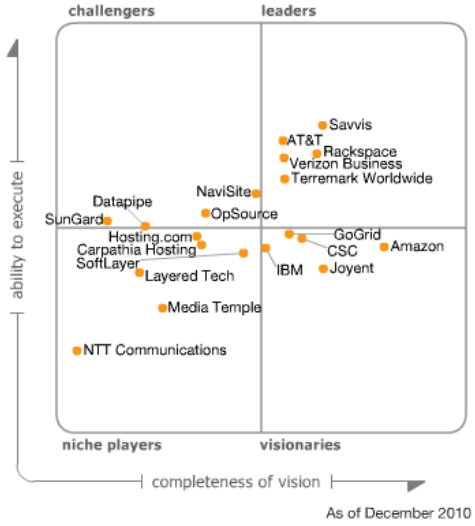

Figure 1.2 – *Gartner Magic Quadrant Cloud IaaS - 2010 [33]*

Figure 1.3 – *Gartner Magic Quadrant Cloud IaaS - 2013 [34]*

En analysant et comparant les deux schémas Gartner *Magic Quadrant* de 2010 (figure 1.2) et de 2013 (figure 1.3), nous pouvons émettre plusieurs constatations qui peuvent cadrer le contexte des travaux de cette thèse. Comme premier constat, nous citons le fait qu'en 2010, il existe un grand nombre de fournisseurs Cloud positionnés majoritairement au centre du schéma *Magic Quadrant*. Ce grand nombre reflète d'une part la divergence des approches et des visions et d'autre part l'immaturité des technologies. De ce fait, au début de nos travaux, il n'y avait aucun standard de fait et aucune vision établie de ce que c'est réellement le Cloud ou les technologies à utiliser. Un deuxième constat est lié au changement de la liste des fournisseurs entre 2010 et 2013. Cette liste a vu la disparition de plusieurs fournisseurs (exemple OpSource et NaviSite) et l'apparition de nouveaux fournisseurs (exemple Microsoft et HP). Ces changements ont bien été ressentis tout au long de nos travaux et nous avons toujours été contraints de tenir compte de cette instabilité des normes, des standards, des interfaces, des architectures, des visions, etc. Comme troisième constat, nous citons le fait qu'en 2013, le nombre de leaders des fournisseurs Cloud reste assez limité. Excepté Amazon AWS qui a pu imposer sa vision du Cloud et devenir le leader incontesté, les autres fournisseurs peinent à trouver la bonne

stratégie pour devenir un leader confirmé. Cette situation instable des fournisseurs reflète encore une fois l'incertitude du marché Cloud. Cette incertitude est toujours réelle et elle a duré jusqu'à 2013 (date de fin de notre doctorat).

1.2 Constats et objectifs

Il est clair que le Cloud allait changer notre façon de mettre en place, gérer et utiliser les systèmes d'information. Cependant, l'adoption et la vulgarisation du Cloud étaient lentes et difficiles vu la jeunesse des concepts et l'immaturité des solutions existantes. Cette difficulté se manifeste à partir de plusieurs constatations comme :

- L'absence de standard ;
- Un grand nombre de fournisseurs et de technologies ;
- L'hétérogénéité des architectures et des API ;
- La difficulté de migrer entre les fournisseurs ;
- L'absence des notions d'interopérabilité entre les technologies ;
- Le *Vendor Lock-In* imposé par les leaders du marché du Cloud ;
- Le manque, voire l'absence, des notions et des concepts de fédération dans le Cloud ;
- La focalisation sur les ressources de *Compute* et *Storage* en négligeant le *Network* ;
- L'émergence des technologies réseau sans une réelle convergence avec le Cloud Computing.

La motivation principale de nos travaux de recherche est de simplifier l'adoption et la migration vers les environnements et les technologies Cloud. C'est pourquoi nous avons défini comme cible la résolution des constats cités préalablement. Pour y arriver, nous avons établi la liste suivante des principaux objectifs :

- Maîtriser et mener une étude détaillée de l'état de l'art ;
- Traiter le rapprochement des réseaux du futur et des Clouds ;
- Traiter les problèmes d'interopérabilité entre les différents fournisseurs et technologies de Cloud ;
- Proposer des architectures et des solutions pour améliorer l'aspect de fédération et interopérabilité des Cloud ;
- Concevoir un plan de contrôle distribué aussi bien en intra qu'en inter domaine.

1.3 Contributions

En considérant les constatations et les objectifs cernés dans la section précédente, nous avons défini quatre contributions majeures :

1. **Modéliser une architecture de convergence Cloud Computing et Cloud Networking**

Le but de cette contribution est de définir une architecture qui rapproche le Cloud Networking et le Cloud Computing. Cette architecture comporte la conception d'un plan de contrôle distribué au niveau des DataCenter (LAN) et au niveau des opérateurs réseau (WAN).

2. **Modéliser une architecture de Courtage Cloud**
   Le but de cette contribution est de définir une architecture de courtage Cloud. Cette architecture résout les problèmes d'interopérabilité et fédération des Clouds.
3. **Définir une interface Cloud Networking**
   Le but de cette contribution est de spécifier une interface, que nous trouvons manquante, qui permet d'étendre le Cloud Computing avec tout type de service Cloud Networking. Cette interface est l'un des deux composants qui assurent la fédération des Clouds.
4. **Définir un système de courtage Cloud à base de messages**
   Le but de cette contribution est de définir le deuxième composant pour assurer la fédération des Clouds. Ce composant est un service d'échange et de courtage Cloud à base de messages entre plusieurs domaines, fournisseurs ou entités dans le Cloud.

1.4 Structure du document

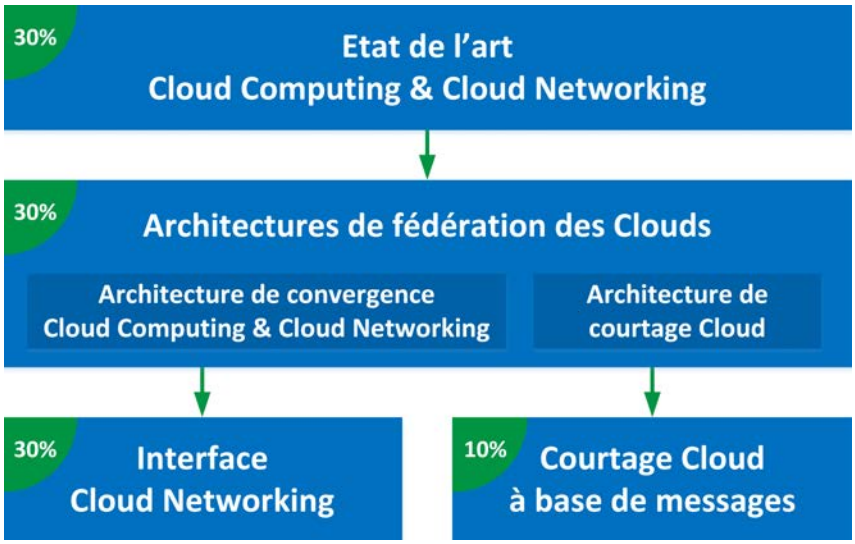

Figure 1.4 – *Structure du document*

Hormis l'introduction et la conclusion générale, cette thèse comporte quatre principaux chapitres. Comme illustré dans la figure 1.4, les quatre principaux chapitres de cette thèse sont structurés d'une façon qui reflète l'avancement des travaux de ce doctorat au niveau des contributions. Ces quatre chapitres sont :

- **Chapitre 2 : Cloud Computing et Cloud Networking**
  Nous présentons dans ce chapitre l'état de l'art des deux principaux sujets de notre thèse qui sont le Cloud Computing et le Cloud Networking.

- **Chapitre 3 : Architectures de fédération des Clouds**
  Nous détaillons dans ce chapitre les deux principales contributions au niveau des architectures de fédération Cloud. La première contribution concerne l'architecture de convergence entre le Cloud Computing et le Cloud Networking. Quant à la deuxième, nous proposons une nouvelle architecture d'interopérabilité et de courtage Cloud.
- **Chapitre 4 : Interface Cloud Networking**
  Nous présentons dans ce chapitre notre contribution d'une interface Cloud générique et spécialisée dans le Cloud Networking. Cette interface est l'un des deux composants essentiels pour assurer la fédération Cloud.
- **Chapitre 5 : Courtage Cloud à base de messages**
  Nous présentons dans ce chapitre notre contribution du service d'échange de messages. Ce système est le deuxième compostant qui assure la fédération et le courtage Cloud.

Chapitre 2

Cloud Computing et Cloud Networking

Sommaire

2.1 Introduction

Le Cloud Computing, ou Informatique en nuages, est un paradigme qui a attiré une attention toute particulière ces dernières années dans le monde de l'informatique. Malgré que la première énonciation de ce concept date des années 1960 par John McCarthy qui a introduit la notion du *Computer Utility*, sa réelle utilisation et sa définition moderne, date des années 2000. Durant ces années, le concept du Cloud Computing a été défini par l'émergence de nouveaux types de services fournis par de grands acteurs tels que *Google*, *Salesforce* ou *Amazon AWS*.

En parallèle, dans les années 1990, un concept similaire au Cloud Computing a été introduit par l'industrie des réseaux. Ce concept, orienté réseaux, se base sur une une couche d'abstraction. Par la suite, afin de s'aligner par rapport aux caractéristiques du Cloud Computing, le concept du Cloud Networking est apparu. Ce concept s'occupe principalement des problématiques de communications dans les environnements Cloud. Il est d'une grande importance dans l'évolution des services de types réseau fournis en mode Cloud.

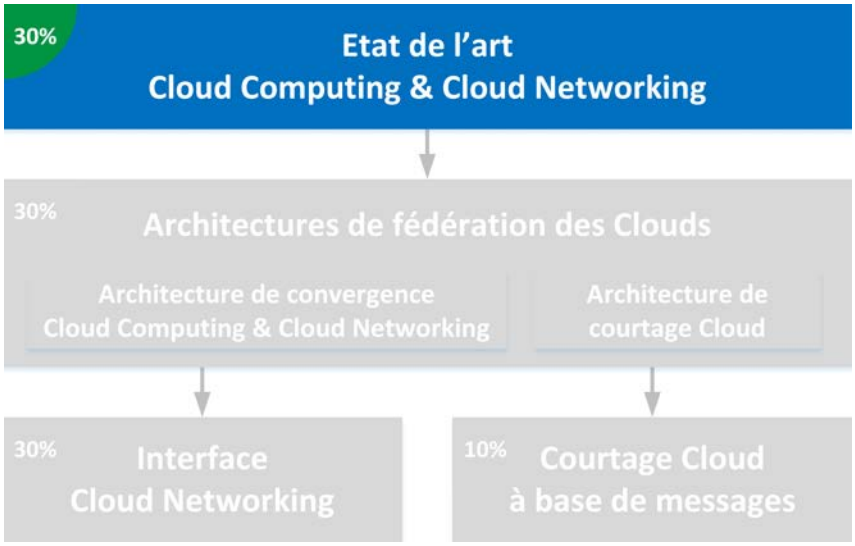

Figure 2.1 – *Structure du document - Chapitre état de l'art*

La problématique est qu'au début des travaux de ce doctorat, les notions basiques du Cloud ne sont pas encore établies vu qu'il existe plusieurs définitions et des visions divergentes des concepts du Cloud Computing et du Cloud Networking. Pour ces raisons, une grande importance est accordée, dans ce doctorat, à la maîtrise et la définition de ces deux concepts. Comme illustré dans la figure 2.1, 30 % des travaux de cette thèse portent sur la bonne compréhension du Cloud et sur la définition de l'état de l'art. Grâce à ces travaux et à la convenable illustration de l'état de l'art, plusieurs projets de recherches ont pu être déposés et financés durant ce doctorat (voir la liste de ces projets dans l'annexe C).

Dans la suite de ce premier chapitre, nous présentons les travaux d'état de l'art. Ces

travaux concernent les deux principaux sujets de ce doctorat : le Cloud Computing et le Cloud Networking. Cet état de l'art se vaut d'être simple et concis. Nous présentons essentiellement la définition, les caractéristiques, les avantages et les technologies des deux concepts. Nous décrivons tout d'abord le Cloud Computing dans la section 2.2 ensuite le Cloud Networking dans la section 2.3. Enfin, nous concluons dans la section 2.4.

2.2 Cloud Computing

2.2.1 Définition

Il existe de nombreuses définitions du terme *Cloud Computing* (CC) et il y a peu de consensus sur une seule et universelle définition. Cette multitude de définitions reflète en soi la diversité et la richesse technologique du *Cloud Computing*. Dans ce qui suit, nous citons quelques une des plus pertinentes. Selon [91, 76, 100, 149, 126], qui se basent sur une vision rapprochée des grilles de calcul *'Grid Computing'*, le *Cloud Computing* se base principalement sur le paradigme de l'informatique distribuée à grande échelle afin d'assurer un service à la demande accessible à travers Internet. Une deuxième définition, proposée dans [106, 58] et qui est plus abstraite, définit le *Cloud Computing* par l'utilisation des ressources informatiques (matériels et logiciels) qui sont offertes en tant que service à travers un réseau (typiquement Internet). Une troisième définition, élaborée par un groupe de travail de la commission européenne[137], considère le *Cloud Computing* comme un environnement d'exécution élastique de ressources impliquant de multiples acteurs pour offrir un service tarifié avec un certain niveau de qualité de service. Cette définition a été étendue dans [136] en prenant en considération les perspectives des différents acteurs de l'écosystème *Cloud Computing* (fournisseur, développeur, utilisateur).

Cependant, la définition proposée par la *National Institute of Standards and Technology (NIST)* dans [116], qui a été reprise par plusieurs travaux [68, 59, 94, 92, 130, 120, 87], est devenue quasi la référence et communément acceptée par le public. NIST définit le *Cloud Computing* comme étant un modèle qui permet l'accès via un réseau d'une façon simple et à la demande à un ensemble de ressources informatiques mutualisées et configurables (ex réseaux, serveurs, stockage, applications et services). Ces ressources informatiques peuvent être allouées et libérées rapidement avec le minimum d'effort de gestion ou d'interaction avec les fournisseurs de services. De plus, NIST précise que le *Cloud Computing* est composé de cinq caractéristiques essentielles, trois modèles de services et quatre modèles de déploiement. Ces éléments sont énumérés par la suite.

2.2.2 Caractéristiques

Généralement, un service, une solution ou un environnement d'exécution devrait satisfaire une liste de caractéristiques pour qu'il soit considéré comme étant du *Cloud Computing*. Parmi ces caractéristiques, il y a celles qui sont reconnues comme fondamentales. Par exemple, NIST définit cinq caractéristiques essentielles qui sont [116, 81, 120] :

- **Ressources à la demande**
  Un utilisateur peut allouer unilatéralement des ressources informatiques (serveurs, réseau, stockage, environnement d'exécution, application) au besoin, de façon automatique et sans nécessité d'interaction humaine avec chaque fournisseur de services ;
- **Large accès réseau**
  Les ressources *Cloud Computing* sont disponibles à travers le réseau et accessibles via des mécanismes standards qui favorisent leurs utilisations à partir des appareils clients hétérogènes, voire légères (ex ordinateurs portables, téléphones, tablettes) ;
- **Mutualisation des ressources**
  Les ressources informatiques du fournisseur *Cloud Computing* sont mutualisées pour servir plusieurs clients en utilisant un modèle multi-tenant. Ces ressources, physiques ou virtuelles, sont allouées et libérées dynamiquement selon la demande du consommateur. Généralement, l'utilisateur n'a ni le contrôle ni la connaissance de l'emplacement exact des ressources allouées. Dans certains cas, il peut choisir l'emplacement géographique à un niveau haut (ex par pays, continent ou data-center) ;
- **Élasticité rapide**
  Les ressources sont allouées et libérées d'une façon élastique, idéalement d'une façon automatiquement, pour s'adapter rapidement à la demande qu'elle soit croissante ou décroissante. Pour le consommateur, les ressources disponibles à l'allocation apparaissent comme illimitées et peuvent s'allouer à tout moment ;
- **Services mesurés**
  Toutes les ressources allouées peuvent être surveillées et contrôlées afin de mesurer leurs consommations avec un niveau d'abstraction approprié selon le type du service (ex stockage, temps de calcul, bande passante).

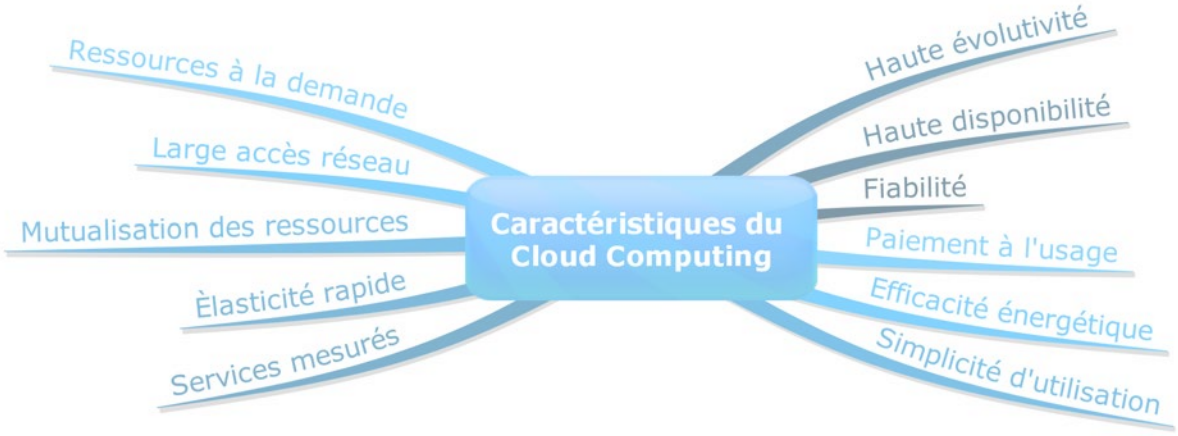

Figure 2.2 – *Caractéristiques du Cloud Computing*

En plus des cinq caractéristiques définies par NIST, il y a d'autres caractéristiques dont nous citons les plus pertinentes [128, 136, 149, 141] :

- **Haute évolutivité**
  Les services de type *Cloud Computing* devraient être évolutifs et doivent satisfaire toute demande de croissance de la part des utilisateurs selon le besoin des services et des ressources allouées. Cette évolutivité doit se faire d'une façon automatique et en cours d'exécution ;
- **Haute disponibilité**
  La haute disponibilité des ressources *Cloud Computing* est primordiale. Par exemple, l'accès réseau aux ressources doit être assuré à plein temps sans aucune interruption ;
- **Haute fiabilité**
  Toutes les ressources *Cloud Computing* doivent être d'une grande fiabilité. Par exemple, la probabilité de perte de donnée doit être quasi nulle (de l'ordre de 10^{-10}) ;
- **Paiement à l'usage** *'Pay as you Go'*
  La philosophie de facturation des ressources *Cloud Computing* se base sur l'usage. L'idée est que l'utilisateur ne paye que ce qui a consommé ;
- **Efficacité énergétique**
  Malgré que cette caractéristique n'a pas été unanimement adoptée, mais la diminution de l'énergie consommée par les ressources *Cloud Computing* est d'une grande importance et sa prise en considération ne cesse de croître. En effet, l'efficacité énergétique contribue à diminuer les coûts ;
- **Simplicité d'utilisation**
  L'allocation, la gestion et l'utilisation des ressources *Cloud Computing* doivent être simples. Idéalement, elles doivent se faire à travers des interfaces et des *Application Programming Interfaces(APIs)* efficaces et génériques.

La figure 2.2 schématise et résume les caractéristiques du *Cloud Computing*.

2.2.3 Acteurs

Comme mentionné dans [110, 59], l'écosystème du *Cloud Computing* est composé principalement par cinq acteurs majeurs (*Cloud Provider*, *Cloud Consumer*, *Cloud Carrier*, *Cloud Broker*, *Cloud Auditor*) :

- ***Cloud Provider*** : Le fournisseur des ressources *Cloud Computing*. Il est responsable de fournir un service *Cloud Computing* qui satisfait les caractéristiques définies dans la section 2.2.2, tout en respectant les *Service Level Agreements(SLAs)* établies avec les autres acteurs (en particulier le *Cloud Consumer*). Le *Cloud Provider* a comme activité l'allocation, l'orchestration et la gestion des ressources qu'il offre tout en assurant le bon niveau de sécurité ;
- ***Cloud Consumer*** : L'utilisateur des ressources *Cloud Computing*. Cet utilisateur peut être un utilisateur final ou un développeur selon le type du service Cloud alloué. Cet utilisateur peut être une personne, un groupe de personnes, les petites et moyennes entreprises, les multinationales ou les gouvernements ;

- ***Cloud Carrier*** ou ***Network Provider*** : Le fournisseur de réseau est l'intermédiaire qui assure principalement la connectivité entre les ressources *Cloud Computing* et la liaison entre les acteurs de l'écosystème *Cloud Computing* (en particulier entre le *Cloud Provider* et le *Cloud Consumer*). Cet utilisateur peut jouer un simple rôle d'acheminements des paquets, comme il peut jouer un rôle plus important en offrant des fonctionnalités avancées dans le réseau. Ces fonctionnalités sont basées sur des SLAs établies avec les autres acteurs de l'écosystème ;
- ***Cloud Broker*** : Le courtier Cloud est un intermédiaire qui négocie la relation entre les *Cloud Providers* et les *Cloud Consumers*. Il peut offrir de nouveaux services qui simplifient les tâches de gestion du *Cloud Consumer*. Ce dernier peut demander les ressources *Cloud Computing* auprès du *Cloud Broker* au lieu du *Cloud Provider* directement. Pour récapituler, le *Cloud Broker* peut assurer l'orchestration, l'agrégation et l'arbitrage des services *Cloud Computing* ;
- ***Cloud Auditor*** : L'auditeur Cloud s'occupe de la vérification et l'audition des services *Cloud Computing*. Il évalue les services offerts par les *Cloud Providers*, *Cloud Carriers* et *Cloud Brokers* du point de vue performances et sécuritaires. Le but principal est de vérifier que les fournisseurs respectent bien les SLAs qu'ils proposent.

2.2.4 Classification

Le concept *Cloud Computing* englobe plusieurs notions et se base sur une large panoplie de technologies comme celle de la virtualisation. Même les utilisateurs des ressources Cloud sont hétérogènes (exemple : simple utilisateur, un développeur, une grande entreprise, etc.). Du fait de cette multitude de concepts, de technologies et des utilisateurs, il existe plusieurs classifications du *Cloud Computing*. Dans ce qui suit, nous citons les classifications selon le type du service, le modèle de déploiement et les abréviations.

Type de service *Cloud Computing*

Afin de mieux définir la classification selon le type du service *Cloud Computing* et comme illustré dans la figure 2.3, un environnement informatique standard peut être composé par plusieurs couches qui partent du bas niveau (le matériel physique) vers le haut niveau (l'application à utiliser). Ces couches sont : Calcul, Réseau, Stockage, Virtualisation, Système d'exploitation, Intergiciel, Environnement de développement, Environnement d'exécution, Données et Applications. La classification selon le type du service correspond au niveau de responsabilité dans la gestion de ces couches que ce soit par les fournisseurs ou par les utilisateurs. Traditionnellement, toutes les couches sont gérées par l'utilisateur lui-même. Avec le *Cloud Computing*, l'utilisateur n'a plus en charge la totalité des couches et en fonction du niveau des sous-ensembles de couche nous distinguons le type de service. Selon NIST[116, 68] et d'autres travaux [136, 99, 59, 137, 100, 72, 74] et comme illustré dans la figure 2.3, il y a principalement trois types de services *Cloud Computing* qui sont *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)* et *Software as a Service (SaaS)* :

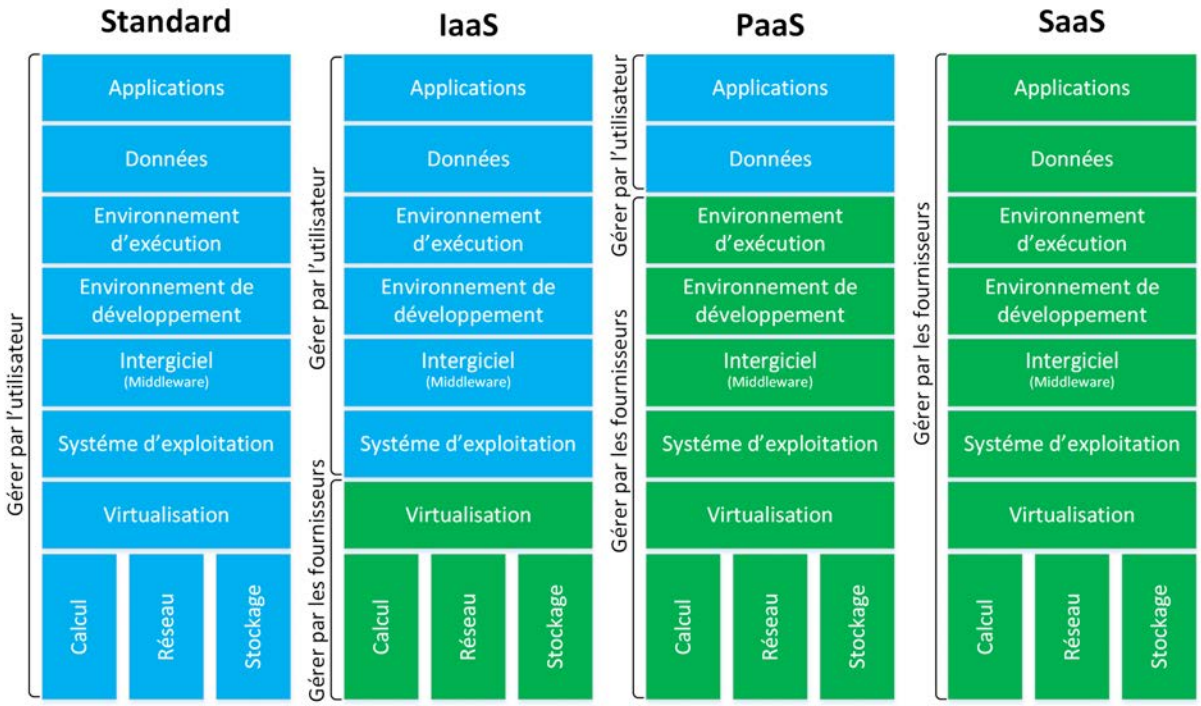

FIGURE 2.3 – *Types de service Cloud Computing*

- ***IaaS***
  Les services *Cloud Computing* de type IaaS correspondent à des ressources infrastructures offertes à la demande. Ces ressources sont des ressources de calculs, de stockage ou de réseau et peuvent être soit virtuelles, soit physiques. Le fournisseur a la gestion des couches Calcul, Stockage, Réseau et Virtulisation. L'utilisateur des ressources IaaS est responsable de la gestion de toutes les couches à partir et au-dessus du système d'exploitation. L'utilisateur n'a ni le contrôle, ni la gestion, ni la visibilité de l'infrastructure sous-jacente ;

- ***PaaS***
  Les services *Cloud Computing* de type PaaS correspondent principalement à des environnements de développement offerts à la demande. L'utilisateur n'a plus en charge que les couches de données et d'applications. Pour ce faire, il y a utilisation des librairies, langages et outils offerts par le fournisseur pour structurer ses données et développer ses applications. Une fois développé, le fournisseur doit déployer et maintenir le bon fonctionnement de l'application et cela en gérant toutes les couches basses allant de l'infrastructure jusqu'aux environnements d'exécution ;

- ***SaaS***
  Les services *Cloud Computing* de type SaaS correspondent tout simplement à des applications prêtes à l'utilisation offertes à la demande. L'utilisateur n'a qu'à utiliser le service *Cloud Computing* offert. Il n'a rien à gérer et c'est le fournisseur qui a toute la responsabilité de maintenir le service en gérant toutes les couches.

Dans [104], il y a une correspondance entre le type de déploiement et le niveau de l'utilisateur. Le type SaaS correspond au niveau utilisateur, le PaaS correspond au niveau développeur et le IaaS correspond au niveau utilisateur d'infrastructure.

Les abréviations à base du style *'*aaS'*

Bien que les trois types de services *Cloud Computing* (IaaS, PaaS et SaaS) soient la base de distinction du type de service, le style de nomenclature *'* as a Service'* a été utilisé ailleurs pour caractériser des services et ressources *Cloud Computing*. Chacune des nouvelles abréviations peut-être vues comme un sous-ensemble d'un ou plusieurs des trois types de base. Parmi ces abréviations, nous citons quelqu'une d'entre elles dans le tableau 2.1.

Les modèles de déploiement *Cloud Computing*

Pour classifier les modèles de déploiement *Cloud Computing*, il faut juste répondre à deux questions[93] : qui utilise le Cloud ? Et qui gère et possède l'infrastructure hébergeant le service Cloud ? Dans [116, 68], NIST a défini quatre modèles de déploiement. Ces modèles ont été aussi mentionnés dans plusieurs autres travaux [119, 93, 136, 58, 87]. Comme nous l'avons illustré dans la figure 2.4, les quatre modèles sont :

- **Cloud Public**
  Dans un Cloud Public, le fournisseur gère l'infrastructure et offre ses services aux utilisateurs Cloud grand public d'une façon complètement ouverte. Les ressources informatiques sont partagées entre les utilisateurs. Ces derniers n'ont aucun contrôle ou visibilité sur l'infrastructure qui est gérée par un tiers (le fournisseur de Cloud) ;
- **Cloud Privé**
  Dans un Cloud Privé, l'utilisateur des ressources *Cloud Computing* contrôle, voire possède, l'infrastructure d'hébergement. Les ressources disponibles ne sont pas destinées au grand public, mais pour une utilisation privée. Dans ce type de déploiement, l'infrastructure est gérée avec des solutions (Open Source ou propriétaire) de type Cloud pour offrir les ressources et services par le biais d'interface Cloud ;
- **Cloud Hybride**
  Dans un Cloud Hybride, les ressources peuvent être allouées à partir d'un Cloud Privé et d'un Cloud Public. C'est un environnement qui combine les deux modèles Public et Privé. Comme utilisation de ce type de Cloud Hybride, il est possible de stocker et gérer les données confidentielles dans l'environnement privé et celles qui sont moins confidentielles dans un Cloud Public. Une autre utilisation est d'avoir recours aux ressources Cloud publiques d'une façon ponctuelle, lors des pics d'activité ;
- **Cloud Communautaire**
  Dans un Cloud Communautaire, l'infrastructure de Cloud est provisionnée à l'usage exclusif d'une communauté d'utilisateurs, par exemple les organismes gouvernementaux. L'infrastructure peut être détenue, gérée et exploitée par un ou plusieurs des organismes de la communauté, un tiers, ou une combinaison d'entre eux.

Abréviation	Nom complet de l'abréviation
APaaS	Application Platform as a Service
BaaS	Backup as a Service
BaaS	Backend as a Service
CaaS	Communication as a Service
DaaS	Data as a Service
DaaS	Desktop as a Service
DBaaS	Database as a Service
DCaaS	Data Center as a Service
DPaaS	Database Platform as a Service
DRaaS	Disaster Recovery as a Service
HaaS	Hardware as a Service
HPCaaS	High Performance Computing as a Service
IDaaS	Identity as a Service
IPaaS	Integration Platform as a Service
MaaS	Monitoring as a Service
MaaS	Management as a Service
MaaS	Metal as a Service
MQaaS	Message Queues as a Service
NaaS	Network as a Service
PRaaS	Process as a Service
RaaS	Resource as a Service
SECaaS	Security as a Service
STaaS	Storage as a Service
XaaS	Everything as a Service

Table 2.1 – *Les abréviations à base de '*aaS'*

2.2.5 Avantages et obstacles

Le *Cloud Computing* offre de nombreux avantages que plusieurs travaux ont listés [119, 137, 127, 94, 130, 72]. Parmi ces avantages, il y a :

- Réduction des coûts d'infrastructure ;
- Réduction des coûts de développement ;
- Réduction des coûts des logiciels ;

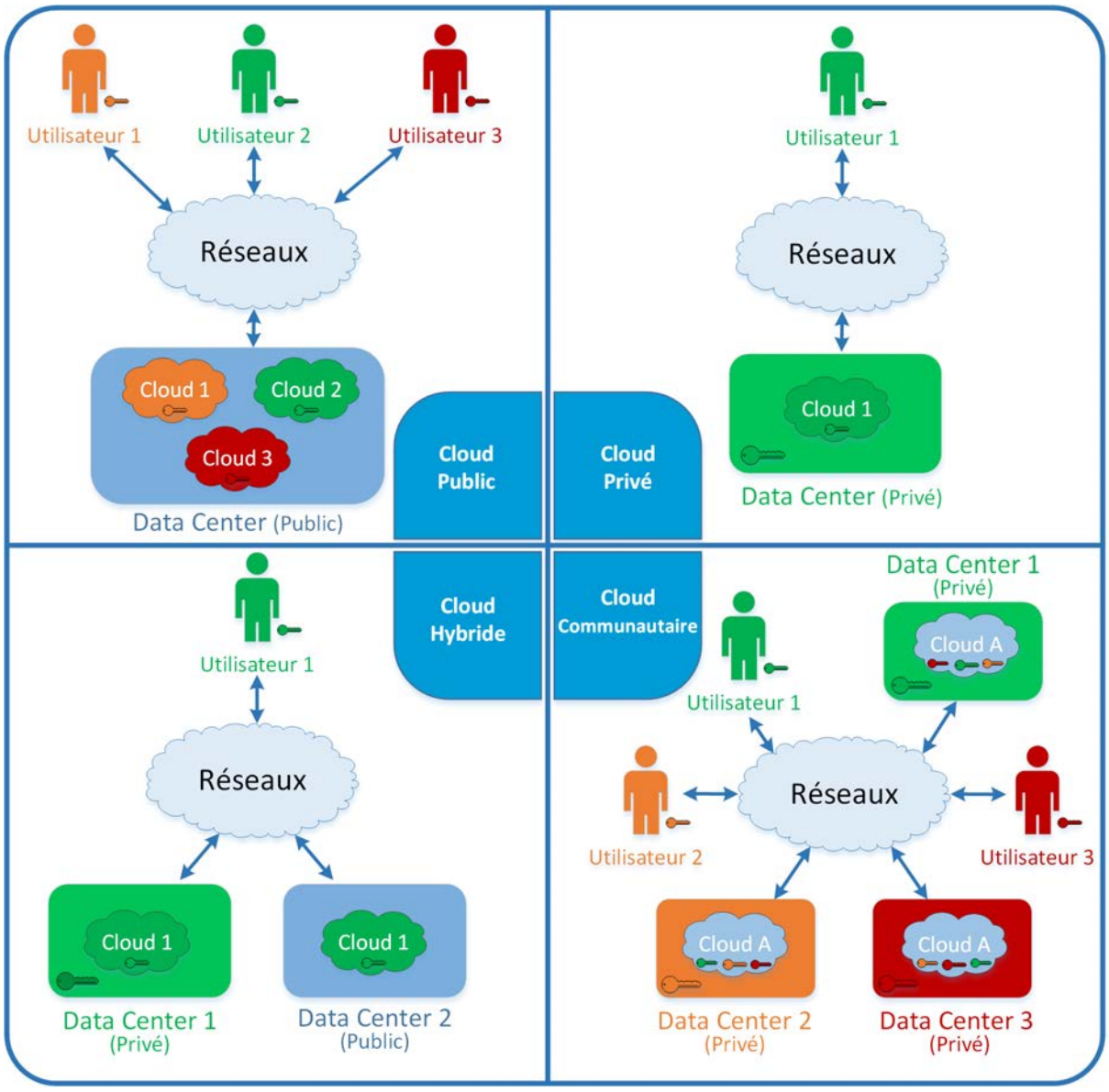

FIGURE 2.4 – *Modèles de déploiement Cloud Computing*

- Des ressources et services plus rapide à allouer et plus simple à utiliser ;
- Accès aux ressources plus flexible ;
- Meilleure utilisation, plus efficace, des ressources ;
- Augmentation de la puissance de calcul ;
- Grande capacité de stockage (quasi illimitée) ;
- Moins de problèmes d'entretien ;
- Gestion des mises à jour plus simple et rapide ;
- Pas de perte de données ;
- Tout est considéré comme un service défini par un SLA ;
- Infrastructure allouée et disponible juste à temps ;

- Réduction du temps de mise sur le marché.

Le *Cloud Computing* n'a pas que des avantages, il possède quelques obstacles et désavantages qui sont abordés dans [64, 94, 65, 119]. Parmi ces obstacles, il y a :

- Données *Lock-in* ;
- Confidentialité des données ;
- Chiffrement des données ;
- Nécessité d'un accès réseau constant ;
- Mauvais fonctionnement avec les connexions à basse vitesse ;
- Faible niveau de la qualité de service dans le réseau ;
- Risque d'engorgements lors des transferts de données ;
- Problème d'interopérabilité ;
- Problème de portabilité ;
- Faible contrôlabilité ;
- Manque de fonctionnalités d'audit ;
- Des contrats de service SLAs non normalisés.

2.2.6 Organismes de standardisation

National Institute of Standards and Technology (NIST) : NIST[42] est une agence de normalisation gouvernementale des États-Unis. Avec la collaboration des industriels, NIST a proposé l'une des définitions la plus populaire du *Cloud Computing* [116].

Open Grid Forum (OGF) : OGF[47] est une organisation de développement de normes spécialisée dans le domaine des *Grid Computing*, de *Cloud Computing* et toutes les technologies liées aux systèmes distribués. La communauté Open Grid Forum (OGF) poursuit ces sujets à travers un processus ouvert pour le développement, la création et la promotion de spécifications et des cas d'utilisation.

Distributed Management Task Force (DMTF) : L'initiative *Cloud Management* de DMTF[29] est axée sur l'élaboration de normes de gestion des infrastructures de *Cloud Computing* interopérables et de promouvoir l'adoption de ces normes dans l'industrie.

Organization for the Advancement of Structured Information Standards (OASIS) : OASIS[43] conduit l'élaboration, la convergence et l'adoption de normes ouvertes pour le domaine de l'information. En tant que source de plusieurs normes fondamentales en usage aujourd'hui, OASIS voit le *Cloud Computing* comme une extension naturelle du Service Oriented Architecture (SOA) et des modèles de gestion de réseau.

Storage Networking Industry Association (SNIA) : SNIA[53] est une association de producteurs et consommateurs de produits de stockage. Son objectif est de promouvoir les normes et activités relatives au stockage et à la gestion de l'information. Au sein de la SNIA, l'initiative *Cloud Storage Iniative* encourage l'adoption du *Cloud Storage* comme un nouveau modèle de prestation.

European Telecommunications Standards Institute (ETSI) : La commuté technique *TC Cloud/Grid*[31] d'ETSI s'occupe des problématiques liées à la convergence entre les technologies de l'information et des télécommunications. ETSI a un intérêt particulier pour les solutions d'interopérabilité pour des environnements qui impliquent des contributions des deux industries de l'informatique et Télécom. Le *TC Cloud/Grid* collabore avec d'autres organisations actives dans les domaines du *Cloud Computing*.

Open Cloud Consortium (OCC) : OCC[44] supporte l'élaboration de normes pour l'interopérabilité entre les solutions *Cloud Computing*. Elle appuie les implémentations de référence, de préférence open source. En outre, il développe des analyses pour le *Cloud Computing*.

Cloud Standards Customer Council (CSCC) : CSCC[28] est un groupe qui défend les droits de l'utilisateur final. Il a une vocation d'accélérer l'adoption du *Cloud Computing* et de surmonter les questions des normes, de la sécurité et de l'interopérabilité concernant la transition vers le Cloud. Cloud Standards Customer Council (CSCC) fournit des matériaux tels que les meilleures pratiques et les cas d'utilisation pour aider les entreprises et les utilisateurs.

Cloud Security Alliance (CSA) : CSA[27] a été créé pour promouvoir l'utilisation des meilleures pratiques pour fournir l'assurance de la sécurité dans le *Cloud Computing*. Il définit les modèles et les règles d'utilisation du *Cloud Computing* pour aider à sécuriser toutes les autres formes de l'informatique.

Internet Engineering Task Force (IETF) : IETF[37] est une communauté internationale des opérateurs de réseaux, les fournisseurs et les chercheurs qui développent des normes Internet. IETF coopère avec plusieurs organisations de standardisations comme le World Wide Web Consortium (W3C), International Organization for Standardization (ISO) ou Internation Electrotechnical Commission (IEC).

Metro Ethernet Forum (MEF) : Le MEF[40] est un consortium d'industriel international à but non lucratif. Elle se consacre à l'adoption dans le monde des réseaux et services Ethernet. Le forum est composé de plusieurs organisations, y compris des opérateurs de réseaux, les vendeurs et les fournisseurs des équipements. Depuis 2011, le MEF travaille sur le rôle et l'intérêt des réseaux et services Ethernet dans le domaine du *Cloud Computing*, plus précisément le Cloud Privé.

2.2.7 Cloud Computing ouvert

Comme mentionné précédemment, le *Cloud Computing* a plusieurs avantages, mais aussi des obstacles. Pour surmonter ces obstacles, le concept du *Cloud Computing Ouvert* est d'une grande importance. Comme illustré dans [10, 100], le Cloud Computing Ouvert a des bénéfices et des principes. Grâce au Cloud Computing Ouvert, l'utilisateur a la **liberté de choisir** entre les différents fournisseurs et les services *Cloud Computing* selon son besoin qui peut changer au cours du temps. De plus, la liberté de choix concerne aussi la possibilité de changer facilement de fournisseurs ou de type de services. Un deuxième avantage du Cloud Ouvert concerne la **flexibilité** qui permet aux utilisateurs de travailler et collaborer entre eux malgré qu'ils utilisent des fournisseurs et des architectures différentes. Un troisième avantage concerne la **rapidité** et la **simplicité** d'allocation et de gestion des ressources Cloud allouées dans un modèle de déploiement hybride. Cette simplicité passe par l'utilisation d'interfaces ouvertes et standardisées (*Open Interface*).

Pour s'assurer que le Cloud soit ouvert, quelques principes doivent être suivis (en particulier par les fournisseurs de Cloud). Parmi ces principes, nous trouvons la nécessité d'une collaboration ouverte entre les fournisseurs Cloud. Cette collaboration est importante, d'une part pour satisfaire les challenges communs de tous les fournisseurs (exemple la sécurité, la portabilité, l'interopérabilité, la supervision) et d'autre part, pour la bonne utilisation et l'implémentation des standards ouverts (en particulier au niveau des interfaces). Un deuxième principe concerne les fournisseurs Cloud qui dominent le marché. Ces derniers ne doivent pas profiter de leur dominance pour bloquer leurs utilisateurs dans leurs environnements (le problème du *Lock-In*). Par ailleurs, les utilisateurs ne devraient pas avoir des barrières pour changer de fournisseur Cloud. Un troisième principe concerne les organismes de standardisations, les groupes de travail et la communauté du Cloud Computing Ouvert qui doivent collaborer et synchroniser leurs travaux pour ne pas multiplier les standards. Ces standards doivent être synchronisés pour fonctionner sans conflits ou chevauchements. Parmi les consortiums autour du Cloud Computing ouvert, nous citons *Open Data Center Alliance (ODCA)* [14] et *Open Cloud Initiative (OCI)* [18].

2.3 Cloud Networking

2.3.1 Définition

Le *Cloud Networking* est le service d'interconnexion entre les ressources de types *Cloud Computing*. Ce service est lié aux opérations nécessaires au niveau des réseaux locaux Local Area Network (LAN), réseaux étendus Wide Area Network (WAN) et aux fonctions de gestion qui doivent se mettre en place pour l'adoption du *Cloud Computing* [118]. Le *Cloud Networking* inclut plusieurs acteurs, mais l'acteur principal reste le fournisseur de réseau *'Cloud Carrier'*. Il offre de plus en plus de nouveaux services de type Cloud avec des fonctions d'interconnexion et de communications qui sont plus riches, plus flexibles et surtout en ligne avec les caractéristiques et les exigences du *Cloud Computing*. Comme définie dans [56, 62], le *Cloud Networking* introduit une nouvelle façon de déployer, configurer et gérer les réseaux et les services de communications. Cette nouvelle philosophie se base sur l'automatisation des opérations complexes du réseau, pour offrir un service à la demande qui soit simple à utiliser et rapide à déployer. L'émergence du *Cloud Networking* est le résultat de plusieurs facteurs. Parmi ces facteurs, il y a la croissance continue de la demande des ressources d'interconnexion, surtout avec l'augmentation du trafic entre les Clouds, l'apparition des courtiers Cloud *'Cloud Broker'* et l'explosion de l'utilisation des appareils mobiles. Un deuxième facteur est la limite des réseaux traditionnels qui ne sont plus suffisants avec leurs fonctionnalités basiques et leurs approches *'best effort'*. Un troisième facteur est lié à l'apparition de nouvelles technologies d'interconnexion qui ont concrétisé le *Cloud Networking*. Parmi ces technologies, il y a les réseaux virtuels et le Software-defined Networking (SDN).

Au sujet des services d'interconnexion dans le domaine du *Cloud Computing*, il y a un autre terme utilisé qui est le Network as a Service (NaaS) (utilisé par exemple dans [70, 69]). Le *Cloud Networking* et le *NaaS* sont équivalents. Malgré leurs similitudes, nous adoptons le nom *Cloud Networking* que nous considérons plus général que le nom NaaS. Le NaaS est plus orienté pour les technologies réseau, alors que le *Cloud Networking* englobe toute notion de services d'interconnexion entre les ressources *Cloud Computing* qui ne sont pas nécessairement à base de technologies réseau.

2.3.2 Caractéristiques

Le *Cloud Networking* a plusieurs caractéristiques qui sont similaires à celles du *Cloud Computing*. De plus, il y a aussi d'autres caractéristiques qui sont plus orientées réseau[61, 141]. Parmi ces caractéristiques, nous citons :

- **Ressources à la demande**
  Les ressources du *Cloud Networking* peuvent être allouées à la demande et d'une façon automatique ;
- **Élasticité rapide**
  Les ressources d'interconnexion sont allouées et libérées d'une façon élastique selon

le besoin des utilisateurs ;

- **Haute évolutivité**
  Le service d'interconnexion doit satisfaire les besoins des ressources Cloud et évoluer automatiquement pour ne pas être le goulot d'étranglement du système ;
- **Faible latence**
  Pour le bon fonctionnement des applications et des services *Cloud Computing*, le temps de latence au niveau réseau doit être le plus faible que possible ;
- **Haute fiabilité**
  En respectant les SLAs établies entre les acteurs, les performances des services *Cloud Networking* doivent être garanties et d'une haute fiabilité ;
- **Réparation automatique**
  Le *Cloud Networking*, au niveau architectural et fonctionnel, doit offrir des fonctions de réparation automatiques en cas de problèmes ;
- **Services mesurés**
  Tous les services et ressources *Cloud Networking* doivent être mesurables. Ces mesures peuvent être, par exemple, liées à la durée et à la quantité de consommation ;
- **Simplicité d'utilisation**
  Comme pour les services Cloud, l'utilisation des ressources et services *Cloud Networking* doit être simple.

2.3.3 Avantages

Les avantages du *Cloud Networking* sont nombreux et plusieurs travaux les décrivent [118, 56]. Parmi ces avantages, nous citons :

- **Réduction des coûts**
  Grâce à l'allocation des ressources et services d'interconnexion au même niveau que les autres ressources Cloud (calcul et stockage), les utilisateurs ne payent plus les équipements réseau. Ce gain est bien notable surtout avec un déploiement de types Cloud Public. Pour les autres types de déploiement, il y a aussi une réduction des coûts due à la flexibilité et à la simplicité de gestion des services d'interconnexion ;
- **Déploiement rapide**
  Les services d'interconnexions, dont leurs complexités sont gérées automatiquement soit par les fournisseurs soit par les solutions Cloud, sont offerts à la demande aux utilisateurs. Grâce à ce mécanisme, les services *Cloud Networking* arrivent à se déployer rapidement, tout en respectant les exigences des environnements *Cloud Computing* ;
- **Grande liberté de mobilité**
  Les services *Cloud Networking* garantissent les deux caractéristiques 'Large accès réseau' et 'Haute évolutivité'. Grâce à ces deux caractéristiques, l'accès aux services *Cloud Computing*, voire les ressources *Cloud Computing* elles-mêmes, gagne une

grande liberté de mobilité. Par ailleurs, il y a eu l'apparition d'un nouveau concept, nommé *Mobile-Cloud*[69], qui traite l'aspect de mobilité dans le Cloud ;

- **Amélioration de la productivité**
  La maintenance et l'évolution des services *Cloud Networking* sont assurées par les fournisseurs (pour le Cloud Public) et les développeurs de solutions Cloud (pour le Cloud Privé). Vu que ces tâches ne sont plus gérées par l'utilisateur, la productivité s'améliore naturellement ;
- **Sécurité accrue**
  La caractéristique de la 'Haute fiabilité' des ressources *Cloud Computing* et *Cloud Networking* passe par une sécurité accrue au niveau des services de communications. Les fournisseurs et les applications Cloud ne cessent d'enrichir et d'améliorer leurs offres avec des fonctionnalités de sécurité au niveau de l'interconnexion réseau. Parmi ces fonctionnalités, il y a les services de cryptages, d'autorisation, de filtrage, de protection contre les logiciels malveillants et la protection des dénis de services ;
- **Enrichissement des applications Cloud**
  Grâce à la simplicité d'allocation et gestion des ressources *Cloud Networking*, les applications peuvent prendre davantage de contrôle automatique sur les services d'interconnexion. Ce gain enrichit les applications Cloud avec des nouvelles fonctionnalités. Par exemple, nous trouvons l'adaptation des règles de routage et des fonctions de filtrage selon le besoin applicatif.

2.3.4 Exigences

Les solutions du *Cloud Networking* et les réseaux modernes devraient répondre à une liste d'exigences afin de satisfaire les caractéristiques et offrir les avantages cités dans les deux sections précédentes. Ces exigences sont décrites dans plusieurs travaux [95, 57, 67], parmi elles nous trouvons :

- **Adaptabilité**
  Les réseaux devraient s'adapter dynamiquement et à la demande selon les changements liés aux applications, aux règles de sécurité et à l'état du réseau lui-même ;
- **Automatisation**
  Avec le *Cloud Networking*, les opérations manuelles devraient diminuer au maximum et la plupart des tâches de reconfiguration ou réparation doivent s'exécuter automatiquement. L'automatisation devrait suivre les exigences liées à l'éventuelle mobilité des ressources *Cloud Computing* ;
- **Maintenabilité**
  Les services du *Cloud Networking* devraient être mis à jour et corrigés rapidement avec le minimum de perturbation au niveau du fonctionnement des applications. Ces opérations devraient même être faites d'une façon complètement transparente ;
- **Sécurité intégrée**
  Les services et solutions *Cloud Networking* devraient intégrer, au niveau architectural

et technique, l'aspect de sécurité. Cette intégration devrait être au niveau cœur des solutions et non pas en tant que simple fonctionnalité à rajouter telle qu'une extension.

2.3.5 Technologies

Le *Cloud Networking* ne se base pas sur une seule technologie bien particulière. D'ailleurs aucune règle n'existe à ce niveau. Cependant, il y a quelques approches et technologies bien connues qui ont accéléré l'émergence du *Cloud Networking*. Parmi ces approches, nous citons la virtualisation des réseaux et les SDN :

Virtualisation des réseaux

La philosophie des réseaux virtuels consiste à avoir plusieurs réseaux hétérogènes qui coexistent et partagent la même infrastructure physique [133, 71, 122, 144, 75, 78]. Plusieurs projets de recherche ont formalisé cette approche. Parmi ces projets, nous citons GENI[36] et 4WARD[21]. La figure 2.5 schématise, d'une façon abstraite, le concept de la virtualisation des réseaux. Suite à l'opération de virtualisation, plusieurs réseaux virtuels cohabitent et partagent les mêmes ressources physiques. Généralement, un réseau virtuel est vu comme une instance logique d'un réseau constitué par un groupe de nœuds virtuels interconnectés via des liens virtuels.

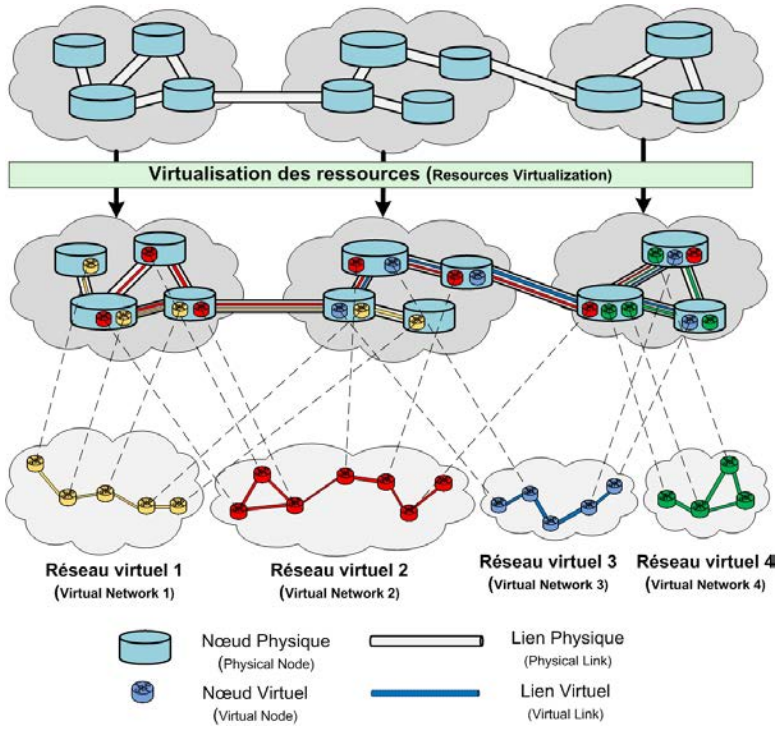

Figure 2.5 – *La virtualisation des réseaux*

Les réseaux virtuels ne sont pas une technologie, mais plutôt une approche dans lesquelles il est exigé d'assurer l'isolation entre les services d'interconnexion. En effet, la mise en œuvre des réseaux virtuels se base sur une panoplie de protocoles et de technologie dont les plus connus sont listés dans le tableau 2.2. Plusieurs travaux, comme ceux dans

[67, 73, 77, 118, 85], décrivent l'utilisation de ces protocoles. Les protocoles peuvent être classifiés selon plusieurs critères. Parmi ces critères, nous citons :

- niveau du protocole : couche 2, couche 3, etc ;
- ancienneté : ancien protocole (avec une modification dans leurs utilisations), nouveau protocole ;
- nature du protocole : standardisé, propriétaire, OpenSource ;
- localité : LAN, WAN, IntraDatacenter, InterDatacenter, etc ;
- niveau du plan de communication : plan de données, plan de contrôle, plan de gestion ;
- type de ressources : commutateur Ethernet, commutateur Fiber Channel, routeur, Firewall, Load Balancer[79], etc ;
- emplacement de déploiement : Hyperviseur, Top-of-Rack (ToR), End-of-Row (EoR), Réseau cœur, routeur Edge, etc.

Un des exemples de réseau virtuel, abordé dans [67], correspond à l'utilisation de la virtualisation au niveau des serveurs. Les fonctions de communications virtualisées s'exécutent dans les machines physiques. Ces fonctions permettent de relier les machines virtuelles (VMs) et agréger le trafic de la machine physique pour le transmettre au commutateur physique. Ces fonctions de communications, qui sont virtualisées, peuvent même être intégrées au niveau des hyperviseurs. Le trafic des Datacenters comprend la Communication entre l'utilisateur et les serveurs et la communication entre les serveurs eux-mêmes (cela peut être entre VM/serveurs au sein du même DataCenter ou entre plusieurs DataCenter distants)[77]. Pour la migration des machines virtuelles entre les hyperviseurs et aussi entre les datacenter, les réseaux virtuels se basent sur des protocoles bien connus et performants comme le Virtual Local Area Network (VLAN), Provider Backbone Bridging (PBB)[2], Virtual Private Network (VPN)[131] et le Virtual Private LAN Service (VPLS)[109]. Afin de satisfaire les exigences du *Cloud Computing*, la gestion de ces protocoles peut être adaptée et automatisée.

SDN

Le SDN (Software-defined Networking) est une nouvelle architecture émergente des réseaux. Cette approche se base principalement sur trois principes[95] : *(1)* le premier principe, le plus important, est le découplage entre les couches réseau (surtout entre la couche d'acheminement des données et la couche de contrôle - voir figure 2.6) ; *(2)* le deuxième principe est la centralisation des opérations de contrôle et de gestion (cette centralisation est logique et ne devrait pas affecter les performances et l'évolutivité des systèmes) ; *(3)* le troisième principe est la standardisation des protocoles entre toutes les couches réseau et la définition des interfaces d'administration.

L'émergence de l'SDN est liée à la nécessité d'avoir une nouvelle architecture qui résout les limitations des réseaux actuels[20]. Parmi ces limitations, nous citons :

Abréviation	Nom complet du protocole
BGP	Border Gateway Protocol [129]
EVB	Edge Virtual Bridging [3]
E-VPN	Ethernet Virtual Private Network - Ethernet VPN[134]
FabricPath	FabricPath (Cisco) [80]
GRE	Generic Routing Encapsulation [89]
IRF	Intelligent Resilient Framework (HP) [101]
LISP	The Locator/ID Separation Protocol[88]
MC-LAG	Multi-Chassis Link Aggregation [118]
MPLS	Multi-Protocol Label Switching [132]
NVGRE	Network Virtualization using Generic Routing Encapsulation [139]
NVP	Nicira Network Virtualization Platform (Nicira)
PBB	Provider Backbone Bridging [2]
PBB-EVPN	Provider Backbone Bridging E-VPN [135]
QFabric	QFabric (Juniper) [103]
RBridges	Routing Bridges [123]
SPB	Shortest Path Bridging [19]
STP	Spanning Tree Protocol [1]
STT	Stateless Transport Tunneling Protocol [82]
TRILL	Transparent Interconnection of Lots of Links [142]
vCDNI	vCloud Director Networking Infrastructure (VMware)
VCS	Virtual Cluster Switching (Brocade)
VLAN	Virtual Local Area Network [115]
VPLS	Virtual Private LAN Service [109]
VPN	Virtual Private Network [131]
VXLAN	Virtual Extensible LAN [113]

Table 2.2 – *Des protocoles utilisés dans le Cloud Networking*

- **Grande complexité**
  Face à l'explosion du nombre des nœuds connectés et l'augmentation de leurs trafics (suite à l'utilisation de la virtualisation, BigData et du *Cloud Computing*), les technologies de communication classiques ont montré une limite considérable : leur grande complexité. Plusieurs protocoles ont été définis et chacun d'entre eux essaye de ramener une solution bien particulière à un problème bien identifié dans un

contexte bien précis. Avec ces protocoles, aucune approche simpliste, avec un bon niveau d'abstraction, n'a été définie. La grande complexité réside dans la diversité et l'hétérogénéité des protocoles qui peuvent être implémentés différemment dans les équipements, selon les fournisseurs du matériel et les versions des protocoles. Par exemple, un simple ajout ou migration d'un nœud peut entraîner plusieurs interventions et modifications dans différents appareils telles que les commutateurs, les routeurs ou les pare-feu. Ces interventions de configuration sont généralement séparées pour chaque appareil. En raison de sa nature statique, le réseau ne peut pas s'adapter dynamiquement et faire face à l'évolution des exigences du trafic, des applications et des demandes des utilisateurs ;

- **Incapacité de passer à l'échelle**
  Avec la popularité des solutions *Cloud Computing*, nous avons, d'une part, une augmentation des demandes de ressources dans les DataCenter et d'autre part, une grande complexité dans la gestion des équipements réseau, dont le nombre ne cesse de croître. Vu que la plupart des opérations de configuration des équipements réseau sont statiques et manuelles, il est devenu impossible de suivre les exigences des solutions *Cloud Computing*. Par conséquent, il est impossible de faire passer à l'échelle les services de communication à base de technologie classique ;
- **Dépendance des équipements**
  Malgré leurs standardisations, les implémentations des protocoles par les fournisseurs d'équipement réseau ne sont pas forcément les mêmes. De plus, les fournisseurs ne cessent de modifier les protocoles ou de proposer des solutions propriétaires pour répondre aux exigences des applications. De ce fait, les tâches de gestion réseau sont dépendantes des types et des fournisseurs des équipements réseau.

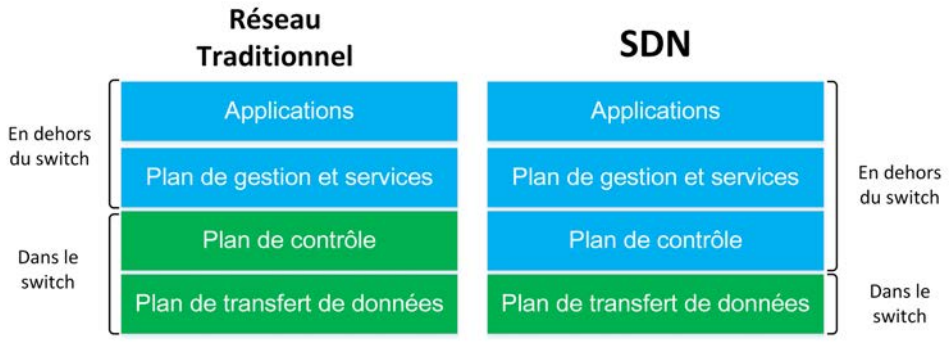

Figure 2.6 – *Différence entre réseau traditionnel et SDN*

Pour la mise en œuvre des SDNs, des technologies comme SoftRouter[107, 112], ForCES[84, 66] ou OpenFlow[114, 102] ont fait leurs apparitions. Mais, OpenFlow est devenu la référence et le protocole le plus avancé et le plus adopté. Au point que les deux termes SDN et OpenFlow sont devenus synonymes. Malgré ça, il faut toujours noter la différence entre les deux. Le SDN est une architecture qui consiste à séparer la couche de

contrôle de la couche d'acheminement des données. Quant à OpenFlow, c'est un protocole qui décrit les contrôleurs SDN et leurs communications avec les dispositifs de routage réseau[108]. OpenFlow permet, d'une part, un accès direct aux appareils réseau comme les commutateurs et les routeurs (qu'ils sont physiques ou virtuels) pour manipuler leurs plans d'acheminement. D'autre part, il permet l'externalisation du plan de contrôle en dehors des appareils réseau pour centraliser, d'une façon logique, les opérations de gestion. Parmi les contrôleurs SDN, nous citons NOX, Beacon, Maestro, POX, Floodlight, ovs-controlle, FlowVisor et OpenDaylight. Coté fournisseurs d'équipements compatibles OpenFlow, nous trouvons HP, Arista, Brocade, IBM, NEC Juniper et Cisco et parmi les commutateurs virtuels compatibles OpenFlow, nous citons Open vSwitch, OpenWRT et Indigo.

OpenFlow a été initialement déployé dans le réseau du campus de l'université de Stanford[114, 90]. Normalisé en 2009, la responsabilité d'OpenFlow a été déplacée en 2011 de l'Université de Stanford, où il a été inventé, à Open Networking Foundation (ONF). ONF[48] est une organisation à but non lucratif qui a été fondée par Deutshe Telecom, Facebook, Google, Microsoft, Verizon et Yahou. La liste des membres de cette organisation ne cesse d'augmenter. Elle regroupe déjà une centaine d'acteurs (exemple Cisco, HP, Juniper, Citrix, Oracle, VMware).

Grâce à OpenFlow, les services de communications peuvent suivre la nature dynamique des applications. Le réseau peut s'adapter rapidement et aisément, selon le besoin. De ce fait, les entreprises peuvent bénéficier de plusieurs avantages[20, 63]. Parmi ces avantages, nous citons :

- Simplification de la gestion des réseaux grâce à la centralisation logique du plan de contrôle. Avec OpenFlow, la gestion automatique de plusieurs ressources réseau hétérogènes est devenue une réalité. Ces ressources peuvent être de plusieurs fabricants et de type virtuel ou physique. Une simple modification dans les règles au niveau de contrôle se suit par une mise à jour du réseau entier. De plus, des opérations de gestion des réseaux qui se faisaient manuellement peuvent être automatisées et orchestrées intelligemment au niveau du plan de contrôle centralisé. Cette automatisation réduit considérablement la complexité de gestion des réseaux ;
- L'élimination du *Vendor Lock-in* au niveau des équipements et des services réseau en raison des contrôleurs SDN ouverts et des protocoles standardisés de gestion des équipements ;
- Une réduction des coûts opérationnels en raison d'une approche simple et automatisée pour le déploiement ;
- Nouvelle opportunité d'innovation en permettant les applications de programmer et contrôler, en temps réel, le réseau selon les demandes.

2.4 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art des deux principaux sujets à laquelle on s'intéresse dans cette thèse : le Cloud Computing et le Cloud Networking. Au début de ce doctorat et en nous basant sur la bonne compréhension des notions et concepts du Cloud, nous avons constaté l'absence des notions de fédération et d'interopérabilité dans le Cloud ce qui favorise le *Vendor Lock-In*. Pour palier à cette limite et comme décrit dans la suite de ce manuscrit, nos travaux consistent à proposer des solutions pour introduire le concept de fédération au niveau des Clouds. De ce fait, l'objectif principal et essentiel de notre doctorat est de proposer une architecture et des solutions de fédération des Clouds.

Nous proposons dans le chapitre suivant deux architectures orientées à la fédération des Clouds. La première architecture cible la fédération et la convergence entre le Cloud Computing et le Cloud Networking. Quant à la deuxième architecture, elle établit la notion de courtage Cloud (*Cloud Brokering*).

Chapitre 3

Architectures de fédération des Clouds

Sommaire

3.1 Introduction

Dans le domaine du Cloud, le nombre de fournisseurs de services ne cesse d'augmenter. De plus, les solutions et technologies utilisées sont nombreuses et hétérogènes. Cette hétérogénéité est bénéfique pour l'utilisateur afin de trouver la solution optimale qui lui correspond. Cependant, les besoins des utilisateurs peuvent changer dans le temps. De même, les offres des fournisseurs peuvent évoluer. De ce fait, la multitude et la divergence des solutions deviennent d'un coté un frein à l'adoption du Cloud et d'un autre coté pénalise l'optimisation continue des environnements et services de type Cloud.

Le concept de fédération est souvent la solution pour les problèmes d'hétérogénéité. Cependant, dans le domaine du Cloud, les approches de fédération ne sont pas évoluées (voire même inexistantes au début des travaux de ce doctorat). Face à cette situation et vu la grande importance de la fédération, nous avons visé comme objectif principal de proposer des architectures orientées à la fédération des Clouds. Comme illustré dans la figure 3.1, 30 % des travaux de cette thèse portent sur la modélisation des architectures de fédération des Clouds.

La première architecture proposée se focalise sur le rapprochement et la convergence entre le Cloud Computing et le Cloud Networking. Cette architecture a été validée et adoptée durant les travaux du projet européen SAIL [52]. Quant à la deuxième architecture proposée, elle se focalise sur l'aspect de courtage du Cloud Computing. Cette architecture a été validée et adoptée durant les travaux du projet français CompatibleOne [26].

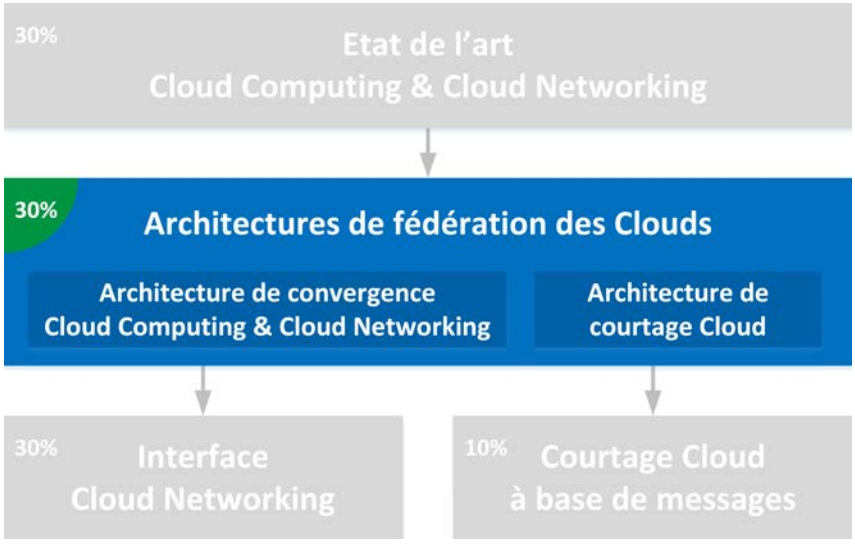

Figure 3.1 – *Structure du document - Chapitre Architectures de fédération des Clouds*

Dans la suite de ce premier chapitre, nous présentons tout d'abord dans la section 3.2 l'architecture de convergence entre le Cloud Computing et le Cloud Networking. Ensuite, l'architecture de courtage Cloud sera décrite dans la section 3.3. Enfin, nous concluons dans la section 3.4.

3.2 Architecture de convergence entre Cloud Computing et Cloud Networking

3.2.1 Contexte

Le Cloud Computing s'appuie sur les progrès de la virtualisation et l'informatique distribuée pour soutenir une utilisation efficiente des ressources, en mettant l'accent sur l'élasticité des ressources et des services à la demande. Les plates-formes et outils actuels du Cloud Computing se concentrent principalement sur l'allocation et la fourniture des ressources de calcul et de stockage. Les utilisateurs doivent cependant combiner leurs propres infrastructures de réseaux avec d'autres Clouds et déployer leurs propres applications pour créer, modifier et personnaliser à la volée le Cloud global résultant de type Cloud Hybride.

Le Cloud Computing dans des environnements fermés, comme Amazon AWS, est utile principalement pour les petites et moyennes entreprises afin d'accroître et diminuer, à la demande, leur infrastructure informatique sans une grande exigence au niveau de l'interconnexion. Les grandes entreprises ne peuvent exploiter que les avantages de scalabilité du Cloud Computing, sous condition de pouvoir maître en place avec succès leurs infrastructures complexes sur des topologies spécifiques des fournisseurs de Cloud. Cette condition est difficile à satisfaire avec les approches classiques vu la complexité de l'aspect de Networking. La première étape, dans le sens du Cloud Networking, est la définition et la mise en œuvre d'une architecture de contrôle distribuée orientée Networking. Cette architecture offre des interfaces aux ressources du réseau et du Cloud Computing. Ces interfaces permettent la mise en place des systèmes selon les besoins des utilisateurs tout en offrant la possibilité de négocier des accords de niveau de service (SLA). Travailler à travers de multiples Clouds nécessite des APIs ouvertes qui devraient être standardisées par des groupes de travail, comme le groupe OGF OCCI [45].

La combinaison de Cloud Computing et Cloud Networking offre des performances réseau optimales, flexibles, sécurisées et améliore les services et les applications. Un certain nombre d'architectures et de frameworks de virtualisation réseau ont été proposés dans la littérature pour offrir des réseaux virtuels personnalisés avec un contrôle de bout en bout. Ces réseaux sont composés par des ressources informatiques virtuelles et reliées entre elles par des routeurs virtuels. Le partage des ressources et des mécanismes de contrôle de l'utilisateur au niveau des routeurs virtuels offre une personnalisation complète du paradigme de la communication dans les infrastructures virtuelles. Pour concevoir des réseaux de Cloud Networking optimisés, il est important d'avoir des mécanismes qui utilisent une architecture de virtualisation de réseau, avec des éléments de transmission distribués et un contrôle centralisé.

La complexité du Cloud Networking augmente fortement, lorsque la gestion dynamique des ressources entre en jeux. La question reste ouverte concernant la création, la modi-

fication, la migration ou la suppression des services virtuels dynamiques sur l'ensemble du système, tout en respectant les SLA. Les fournisseurs traditionnels de Cloud Computing déplacent juste des images entre ordinateurs physiques dans un même domaine d'administration. Les futures infrastructures de Cloud doivent également mettre en place de nouvelles infrastructures de réseaux privés virtuels sur plusieurs domaines. Ce processus prend aujourd'hui plusieurs jours, alors que le temps de mise en service doit être réduit à quelques minutes avec le Cloud Networking. Par ailleurs, l'adaptation automatique selon les besoins applicatifs est une autre facette du défi du Cloud Networking.

Assurer la sécurité est déjà une tâche complexe dans les centres de données fermées. Avec l'introduction de la virtualisation et en particulier la transition vers le Cloud, les frontières du système deviennent mal définies. Par l'introduction des services virtualisés, différentes sociétés se partageront le même matériel physique et le maintien de l'isolement devient un aspect principal de la sécurité. De plus, l'introduction de la virtualisation rend difficiles la détection, la localisation et l'analyse des causes des attaques et vulnérabilités. La cause racine et les effets peuvent être séparés par plusieurs niveaux de virtualisation. Des perturbations mineures à des niveaux inférieurs de virtualisation peuvent conspirer pour provoquer une interruption de service à un niveau plus élevé de virtualisation. Inversement, un défaut au niveau de la virtualisation inférieure peut être entièrement inaperçu au niveau supérieur. Par conséquent, un contrôle constant des défauts et des perturbations doit être maintenu à tous les niveaux d'abstraction et la recherche des causes racines doit être faite à travers les différentes couches d'abstraction. Tout cela se fait en gardant les barrières d'abstraction de virtualisation aussi intacte que possible. Cela pose un certain nombre de défis, tels que la conception d'interfaces à travers les frontières de virtualisation pour l'analyse des causes racines et des procédés de corrélation des perturbations dans les différentes couches virtuelles.

En général, le Cloud Networking et Cloud Computing offrent de nombreuses possibilités d'optimisation. L'une d'elles concerne le placement des applications et des processus ainsi que la migration dynamique et automatique des applications vers l'infrastructure optimale. Cette opération d'optimisation par migration nécessite l'extension des ressources réseau via les mécanismes et fonctions du Cloud Networking.

3.2.2 Motivation

Le Cloud Networking est une approche innovante qui permet de contrôler, déployer et provisionner les ressources réseau d'une manière beaucoup plus dynamique, flexible et évolutive par rapport à ce qui est possible traditionnellement. En effet, la gestion actuelle des réseaux virtuels est statique dans la nature, et s'adresse principalement au problème de subdiviser une infrastructure physique en partitions qui peuvent être gérées indépendamment. En revanche, le Cloud Networking répond aux besoins des applications classiques et nouvelles en soutenant l'utilisation extensible et flexible des ressources, l'agilité dans la fourniture de ressources de réseau et la différenciation des niveaux de service et de

performance. Le Cloud Networking est basé sur la vision d'un framework de gestion unifié pour l'informatique et la communication. Le Cloud Networking permettra d'optimiser l'ensemble des allocations de ressources en tenant compte des ressources du réseau et de l'informatique d'une façon unifiée. Cette unification de gestion éliminera les problèmes liés à un contrôle disjoint et séparé entre la partie de réseau et la partie de calcul. Parmi les facteurs importants qui alimentent l'intérêt du Cloud Networking, il y a la possibilité de réduire les coûts en minimisant la nécessité d'investir dans les ressources informatiques locales et l'emploi de personnel à gérer, ainsi que la capacité d'allouer rapidement et à la demande les ressources pour répondre à l'augmentation des tendances d'utilisation.

L'un des nouveaux concepts qui peuvent traiter le Cloud Networking, introduit par le projet SAIL, est le FNS - *Flash Network Slice*. Le FNS est l'ensemble des connexions réseau dynamiques, flexibles et à haute vitesse qui permettent aux ressources Cloud de servir les applications nécessitant une connectivité réseau garantie ainsi que des ressources de calcul et de stockage importantes (le FNS sera détaillé dans la section 3.2.4). L'ensemble des nouvelles applications du Cloud Computing ne peut pas être approvisionné avec les réseaux existants en raison de la nature statique des réseaux traditionnels et le couplage inexistant entre le réseau et le niveau applicatif. Il est plus judicieux d'avoir des SLAs uniques qui couvrent à la fois la mise en réseau et les ressources informatiques et applicatives. De ce fait, un plan de contrôle commun est nécessaire pour gérer les ressources réseau au plus près des ressources de stockage et calcul. Ce plan de contrôle est géré par des interfaces bien définies qui doivent être établies entre les opérateurs de réseau et de Cloud Computing pour la configuration des ressources du réseau et de l'informatique. Cependant, ce n'est pas la situation actuelle. En effet, le réseau et les ressources de calcul sont traités séparément que ce soit pour l'allocation ou pour l'optimisation.

Les services de Cloud Computing mettent en valeur les avantages fournis par l'approche centrée au tour des Data Center comme la mutualisation des ressources et la réduction des coûts. Cependant, cette approche n'est pas la meilleure solution pour chaque problème, en particulier :

- Si le nombre de requêtes émises d'un lieu géographique pour récupérer un objet particulier grandit, il est plus optimal d'utiliser le système de *caching* ;
- Si un accès à faible latence est nécessaire, il est approprié de localiser les données (avec traitement associé) près de ses utilisateurs ;
- Si les organisations peuvent avoir des exigences légales pour exécuter des applications et stocker des données dans des juridictions particulières (ou à éviter certaines juridictions).

Tous ces problèmes créent aussi une nouvelle opportunité pour les opérateurs de télécommunications, qui sont les seuls capables de fonctionner à grande échelle "plus près de l'utilisateur" pour joindre dans le business du Cloud.

Le domaine de recherche du Cloud Networking est très intéressant d'un point de vue technique et très prometteur d'un point de vue commercial. En effet grâce au Cloud

Networking, les solutions applicatives peuvent devenir évolutives en s'appuyant sur des technologies et des protocoles existants ou nouveaux au niveau des fournisseurs réseau. Dans ce cadre, la virtualisation de réseau est utilisée comme une technologie fournissant des réseaux abstraits et isolés pour les utilisateurs. De plus, le niveau d'abstraction des réseaux doit être ajustable selon les besoins applicatifs des utilisateurs. Vue que la combinaison de deux systèmes sécurisés n'induit pas à un système sécurisé combiné, les interdépendances spécifiques de sécurité entre le Cloud Computing et l'abstraction du réseau doivent être étudiées. Cela signifie que les méthodes et les mécanismes mis en place peuvent être déployés et mis en production plus rapidement. De même, les opérateurs de réseaux traditionnels peuvent tirer profit en combinant leurs expériences et les avantages de l'exploitation du Cloud Networking avec les nouvelles possibilités de services Cloud.

3.2.3 Objectives

La figure 3.2 représente les deux états (actuel et évolué) du *Cloud Computing* et du *Cloud Networking*. Les services *Cloud Computing* classiques (en haut à gauche) sont orientés aux services de calcul et de stockage avec un simple réseau intra-domaine (approvisionnement à l'intérieur du même centre de données). En parallèle, les réseaux ont également évolué en introduisant la virtualisation des réseaux (en bas à gauche) pour offrir une grande flexibilité et une allocation dynamique, automatique et à la demande de réseaux dédiés et isolés y compris les réseaux virtuels tels que présentés dans [78, 75]. L'évolution peut aussi se constater avec l'émergence des nouvelles architectures comme les SDNs (concrétisée avec la popularité du protocole OpenFlow [114]). Les deux approches, de *Cloud Computing* classique et l'évolution des réseaux, ne se sont pas encore fusionnées. Cette fusion permet l'allocation et la gestion, à la demande, des ressources Cloud distribuées dans un déploiement hybride en incorporant tout type de service d'interconnexion.

L'évolution ciblée est illustrée dans la partie droite de la figure 3.2. L'objectif est l'unification et la convergence entre le *Cloud Computing* et les nouvelles technologies de communications (exemple la virtualisation du réseau et SDN) pour avoir une architecture cohérente de Cloud distribué.

D'une façon simple et précise, les objectives sont :

- Développer une architecture de Cloud Networking :
  Une architecture pour définir d'une part les rôles, les responsabilités, les interfaces et d'autre part un modèle de référence pour le déploiement d'applications complexes sur multiples fournisseurs de ressources Cloud. Ce déploiement se base sur plusieurs opérateurs et des technologies de communications hétérogènes.
- Concevoir les fonctions de commande et des protocoles pour cette architecture :
  Les fonctions de contrôle agiront sur les ressources de Cloud Computing et de communications.
- Concevoir le monitoring et le système de détection, localisation et réparation des incidents :

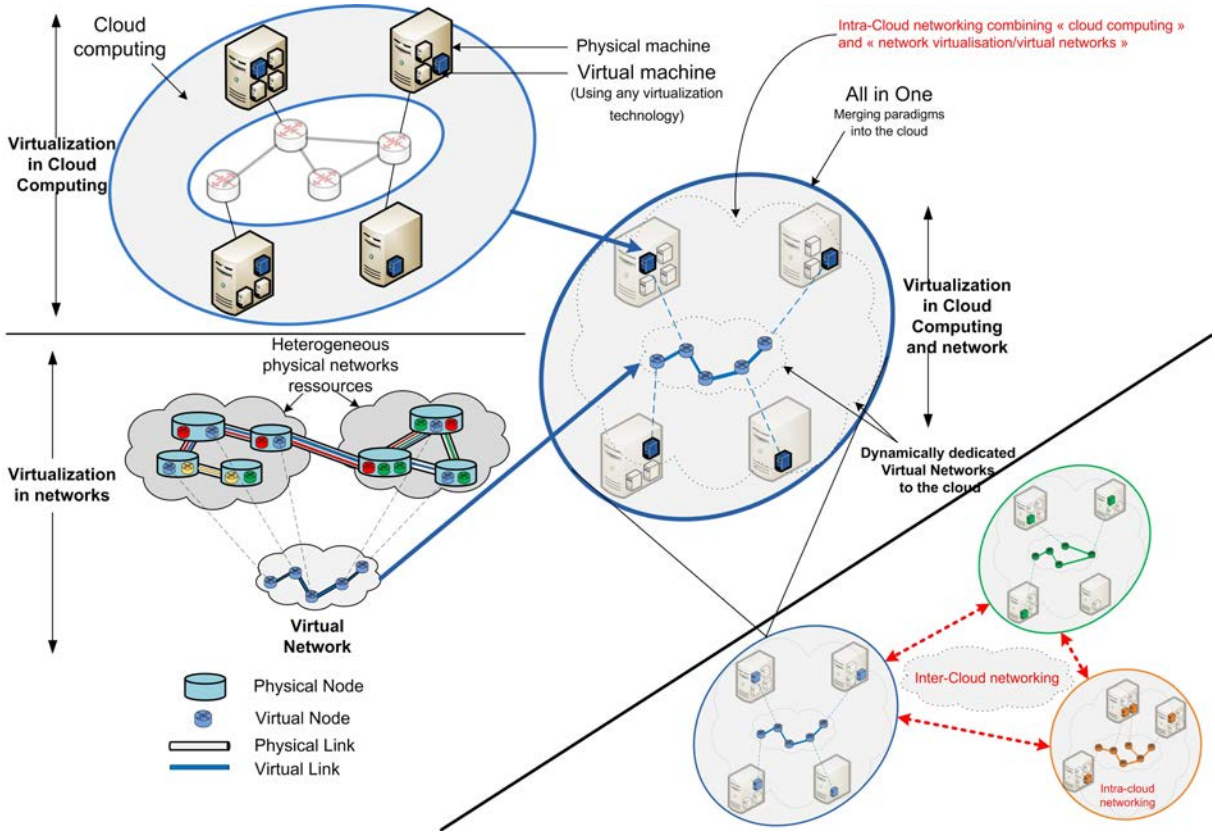

FIGURE 3.2 – *Convergence entre Cloud Computing et Cloud Networking*

La réparation se base sur des fonctions qui ont un rôle d'identification et réparation des défauts, indépendamment des couches de virtualisation.

- Concevoir des solutions de sécurité pour le Cloud Networking :
  Les solutions doivent assurer la sécurité contre les attaques résultantes de la nature du système virtualisé tout en respectant les mécanismes de sécurité générale et classique.
- Évaluer toutes les solutions dans un prototype :
  Ce prototype doit prouver la faisabilité de l'architecture proposée.

L'architecture du Cloud Networking est fondée sur quelques contraintes de base issues de la nature de la communication réseau et la tendance actuelle dans le Cloud Computing. Ces contraintes architecturales sont décrites comme suit :

- compatibilité avec les approches actuelles du Cloud Computing ;
- réalisation sur les nouvelles technologies de réseau émergentes ;
- réalisation sur les technologies de réseau actuellement déployées ;
- réalisation à travers de multiples domaines administratifs et localisation.

3.2.4 CONCEPT FNS

Le but du Cloud Networking est de compléter le Cloud Computing en abordant l'aspect réseau. Dans le projet SAIL, un nouveau concept nommé FNS *Fash Network Slice* a été

introduit. Un FNS est une ressource de réseau qui peut être provisionnée et dimensionnée sur une échelle de temps comparable aux ressources de calcul et de stockage existantes. Un FNS peut être utilisé pour construire et relier les infrastructures virtuelles qui couvrent plusieurs fournisseurs. Grâce à ce concept, il est possible de créer des environnements dans lesquels les ressources et services se déploient dans plusieurs centres de données à travers le monde. Le but d'un FNS est d'adresser pleinement les capacités du réseau dans le paradigme IaaS selon les exigences du Cloud Networking. Un FNS a les propriétés suivantes :

- Ressource du réseau : un FNS est une ressource offrant une capacité de communication ;
- Liens : un FNS peut être attaché à d'autres ressources par des liens. Par exemple, une machine virtuelle peut être attachée à un FNS ou deux FNS peuvent être attachés les uns aux autres. Un lien peut relier des domaines administratifs ;
- Qualité du service : un FNS a la propriété de qualité de service associé à sa capacité de communication ;
- Domaine administratif unique : un FNS est construit et géré dans un seul domaine administratif ;
- Temps de mise en place : la mise en place d'un FNS est établie automatiquement et en un laps de temps comparable avec les ressources d'infrastructure virtuelle existantes, telles que les machines virtuelles.

Le FNS fournit une capacité de communication entre les ressources qui lui sont attachées. Lorsque deux FNS sont liés les uns aux autres, ils fournissent une capacité de communication commune. Comme les liens entre les FNS sont autorisés à dépasser les domaines, cela donne une capacité de communication qui est essentielle à la mise en place d'infrastructures entre plusieurs domaines. Un FNS peut être mis en œuvre par une variété de technologies de réseau sous-jacentes comme MPLS, VPN, OpenFlow, VLAN. Les protocoles et les technologies utilisées pour connecter ces FNS peuvent également varier.

3.2.5 Architecture haut niveau

L'architecture de haut niveau du Cloud Networking se compose d'un modèle à trois couches. Le modèle à trois couches est un framework qui caractérise l'infrastructure virtuelle par rapport à trois points de vue différents.

La figure 3.3 est une représentation visuelle pour caractériser les composants de l'architecture du Cloud Networking. Un domaine administratif est un ensemble d'équipements physiques ou virtuels qui est sous la direction d'une autorité administrative. Comme illustré dans la figure 3.3, les domaines administratifs dans le contexte du Cloud Networking peuvent être des réseaux au niveau WAN ou les centres de données (ils sont représentés en bas de la figure 3.3). Une infrastructure virtuelle peut s'étendre sur plusieurs domaines

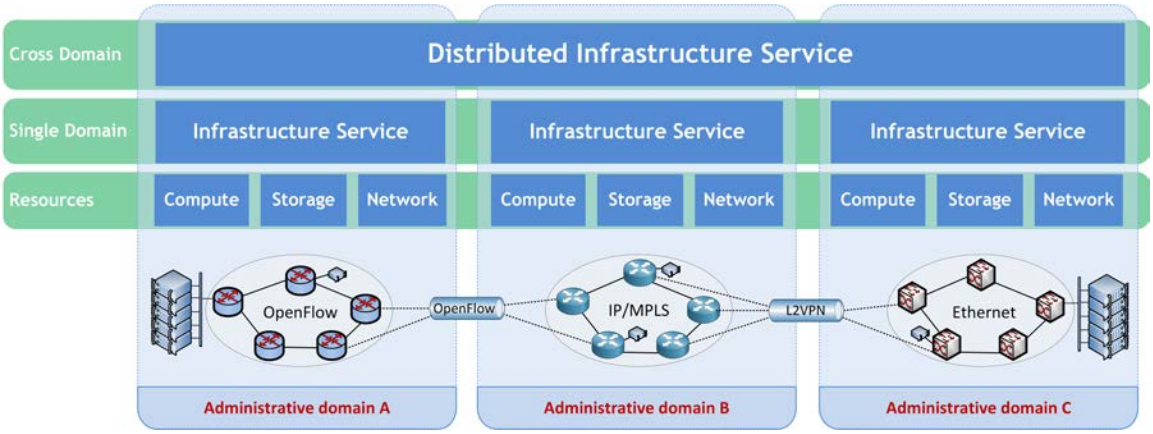

FIGURE 3.3 – *Modèle à trois couches*

administratifs. Les trois couches au-dessus des domaines administratifs représentent trois vues différentes de l'infrastructure virtuelle. Les trois couches sont *Resources*, *Single-domain infrastructure* et *Cross-domain infrastructure*.

RESSOURCES

Une ressource est une représentation abstraite d'un composant d'une infrastructure virtuelle telle qu'une machine physique ou virtuelle ou un volume de stockage. Cette ressource appartient à un domaine administratif unique. La couche des ressources comprend des ressources de calcul, les ressources de stockage et les ressources réseau qui sont gérées par différents sous-systèmes. Les ressources ont certaines propriétés et peuvent avoir des liens vers d'autres ressources.

Il est intéressant de remarquer que le Cloud Networking propose des ressources au niveau des opérateurs réseau et aussi au niveau des centres de données. L'architecture du Cloud Networking propose aussi des ressources de calcul et de stockage qui peuvent être déployées dans les réseaux des opérateurs. Ces ressources peuvent avoir un ensemble de propriétés différentes selon le domaine administratif où elles résident. Par exemple, l'emplacement réseau d'une machine virtuelle est une propriété pertinente dans le réseau WAN, mais pas autant dans le centre de données.

Une ressource virtuelle peut être créée, gérée et détruite. Ces actions de commande sont généralement effectuées par un sous-système tel que l'interface de gestion sur un hyperviseur de machine virtuelle ou un gestionnaire de périphérique de stockage. L'acte de création d'une ressource virtuelle se base sur une ressource physique (tel que l'espace disque) ou une ressource logique (telles qu'une adresse IP ou bande passante). Quant à la destruction d'une ressource virtuelle, elle permet de libérer ces ressources physiques ou logiques.

Les ressources virtuelles sont souvent reliées à d'autres ressources virtuelles : une machine virtuelle peut être connectée à un réseau virtuel ou un dispositif de stockage. Elle peut être aussi gérée dans un seul domaine administratif, mais peut avoir des liens avec

des ressources virtuelles dans d'autres domaines administratifs. Cela est particulièrement raisonnable côté connectivité réseau, car il s'agit d'un moyen d'interagir au-delà de la limite d'un fournisseur.

Infrastructure Uni-Domaine - *Single-Domain Infrastructure*

Nous considérons un *Single-Domain Infrastructure* comme étant un nombre de ressources gérées collectivement dans un seul domaine administratif. Les liens entre ces ressources déterminent la topologie de l'infrastructure et limitent leur comportement de gestion. De plus, un *Single-Domain Infrastructure* peut être créé, mis à jour, géré et détruit.

Un *Single-Domain Infrastructure* est géré par une autorité administrative unique qui dispose des droits de gestion sur le matériel sous-jacent et de la technologie de virtualisation. Par conséquent, dans un seul domaine, l'autorité administrative a une connaissance sur les ressources disponibles et des capacités de virtualisation à tout moment.

Le *mapping* entre un *Single-Domain Infrastructure* et l'infrastructure physique peut être exécuté et connu. Ce mapping prend en considération plusieurs critères d'optimisations. Par exemple, une machine virtuelle peut être placée dans un emplacement optimal du réseau selon le temps de latence du lien réseau avec l'utilisateur final.

Certaines sélections de technologie peuvent être faites à cette couche. En effet, un volume disque pourrait être placé sur un stockage local ou un stockage réseau et une liaison de réseau peut être mise en correspondance avec un tunnel VPN isolé ou un réseau partagé ouvert.

Le placement optimal et la sélection de la technologie dépendent en grande partie des politiques de l'autorité administrative du domaine. Cependant, les propriétés des ressources virtuelles, leurs liens et les propriétés de l'infrastructure virtuelle peuvent influencer les choix d'allocation.

Infrastructure Inter-Domaine - *Cross-Domain Infrastructure*

Nous considérons un *Cross-Domain Infrastructure* comme étant un ensemble de ressources virtuelles gérées collectivement à travers de multiples domaines administratifs. Il peut être créé, mis à jour, géré et détruit. De plus, un *Cross-Domain Infrastructure* peut être partitionné en plusieurs *Single-Domain Infrastructure*. Dans ce cas, un *Single-Domain Infrastructure* peut contenir des ressources virtuelles qui ont des liens avec les ressources virtuelles dans d'autres *Single-Domain Infrastructure*. Cela permet alors de relier les infrastructures virtuelles et de déterminer la topologie du *Cross-Domain Infrastructure*.

Un *Cross-Domain Infrastructure* est géré par de multiples autorités administratives. Contrairement à un *Single-Domain Infrastructure*, l'état du matériel sous-jacent et des fonctionnalités de virtualisation ne sont probablement pas entièrement partagés au-delà des limites d'un domaine. Dans ce type d'infrastructure, des interfaces spécifiques sont nécessaires pour négocier le placement des ressources et leurs liens d'interconnexion. Via ces interfaces, les autorités de domaines administratifs peuvent échanger des informations limitées de leur domaine qu'ils sont prêts à partager afin de faciliter l'optimisation de l'allocation entre les domaines.

La décomposition d'une requête d'infrastructure virtuelle demandée entre les domaines administratifs peut être effectuée en fonction des capacités des domaines administratifs et de leur interconnexion. Évidemment, les propriétés et les liens entre les ressources virtuelles et les propriétés de l'infrastructure virtuelle influencent le processus de décomposition.

3.2.6 Concept DCP

Le plan de contrôle distribué, DCP - *Distributed Control Plane*, décrit une catégorie de protocoles, des interfaces et des opérations de contrôle qui permettent à deux ou plusieurs fournisseurs de services d'infrastructures d'interagir et d'échanger des informations inter administratif. Voici une description de certaines interactions qui se produisent entre les fournisseurs via le DCP :

- **Résolution des références (*Reference resolution*)** : c'est le processus de conversion d'une représentation abstraite de l'information distante vers l'information réelle. À titre d'exemple, si une liaison de réseau doit être établie, l'information sur l' extrémité distante de la liaison est nécessaire. Cette information peut être représentée par une référence abstraite qui se résout par la DCP pour obtenir l'information réelle. En général, le processus de résolution de référence permet la mise en ouvre de plusieurs politiques de camouflage des informations. Dans ce cas, les fournisseurs devraient être en mesure de mettre en œuvre leurs propres politiques en matière de divulgation d'information. Par exemple, un fournisseur peut souhaiter que certaines informations n'aient exposé qu'un ensemble de pairs de fournisseur bien sélectionnés ;
- **Notification** : c'est un échange d'informations asynchrone comprenant du *Publish-Subscribe*. Ce type d'échange d'informations est utilisé, d'une part, pour découpler la demande de renseignements de la réponse et d'autre part, pour assurer une coordination distribuée. À titre d'exemple , la création d'un lien réseau entre les domaines peut exiger la coopération entre les deux fournisseurs qui opèrent de manière asynchrone. On peut demander des informations à un fournisseur avant qu'il soit prêt à les fournir. Le service de notification fournit alors un moyen pour transférer les informations quand elles sont disponibles ;
- **Partage distribué d'information (*Distributed information sharing*)** : un service d'information distribuée peut fournir une vue globale sur l'état de l'infrastructure. La vue peut être maintenue par un protocole distribué entre les fournisseurs.

Le DCP fonctionne au niveau de la couche *Cross-domain infrastructure* et concerne, de manière générale, la coordination distribuée et l'accès à l'information. La communication entre les domaines via le DCP n'a pas besoin d'être synchrone. De plus, les protocoles et les interfaces spécifiques utilisés peuvent dépendre de la relation entre les domaines et les technologies utilisées. Cependant, les protocoles génériques peuvent être utilisés pour mettre en œuvre des services de coordination et de communication communs.

3.2.7 Infrastructure Uni-Domaine

L'élément principal d'une fédération Cloud dans une infrastructure Uni-Domaine est le service de l'infrastructure (*Infrastructure Service*). Il s'agit d'un élément fondamental de l'architecture du Cloud Networking. Ce service de l'infrastructure représente l'ensemble des interfaces qui permettent la création, le suivi et la gestion des infrastructures virtuelles fournies par le fournisseur de services et accessibles par l'utilisateur du service.

Nous proposons que les requêtes d'instanciation des utilisateurs au service de l'infrastructure soient effectuées à l'aide d'un langage de description de haut niveau (ce langage fait l'objet des travaux illustrés dans le chapitre 4). Ces requêtes doivent être traduites en actions de bas niveau à l'intérieur du plan de contrôle pour l'allocation des ressources sous-jacentes. La description de haut niveau des besoins de l'utilisateur doit être simple. Les besoins en ressources peuvent être largement distribués. Par exemple, l'interface client doit faciliter l'utilisation des serveurs hautement distribués sans avoir à se soucier de leur emplacement topologique.

L'objectif principal de l'interface est de permettre à l'utilisateur de spécifier des objectifs de haut niveau sous la forme d'accords de niveau de service (SLA). Ces SLAs seront automatiquement décomposés en actions de contrôle de bas niveau. Le langage proposé doit définir les objectifs de service qui dépassent les frontières entre les différents fournisseurs de Cloud. Il convient également de tenir compte des exigences fonctionnelles et non fonctionnelles des utilisateurs. Ces exigences peuvent être contraintes par les processus de l'utilisateur final qui peut imposer par exemple l'aspect de migration automatique.

Pour la configuration et la surveillance des services, des fonctions de gestion du Cloud Networking fournissent les fonctionnalités nécessaires pour gérer les ressources virtuelles et doivent faciliter la gestion des réseaux existants. Ces fonctions peuvent être considérées comme des entités de gestion de couche supérieure pour répondre aux besoins spécifiques du Cloud Networking.

3.2.8 Infrastructure Inter-Domaine

L'élément principal d'une fédération Cloud dans une infrastructure Inter-Domain est le DCP. Dans cette section, nous considérons que les interactions de contrôle passent à travers le DCP. En effet, le dispositif de contrôle dans l'architecture de Cloud Networking est basé sur un certain nombre de principes tels que la nécessité de la délégation et le besoin de coordination répartie entre les domaines administratifs.

Le modèle général pour le développement du plan de contrôle est constitué de plusieurs exigences dont nous citons les principaux :

- **Divulgation limitée des informations** : chaque fournisseur de services d'infrastructure n'est pas disposé à exposer tous les détails de son service, de la topologie physique ou matrice de trafic. Par conséquent, le fournisseur de services d'infrastructure devrait être en mesure d'entreprendre un approvisionnement de ressources

inter-domaine sans une connaissance détaillée des domaines administratifs sous-jacents ;

- **Provisionnement rapide des ressources** : l'utilisateur doit être en mesure d'approvisionner des ressources de communication dans plusieurs domaines dans des échelles de temps comparable à la fourniture de ressources de calcul existantes dans des offres de type IaaS ;
- **Mise à l'échelle indépendamment du nombre de fournisseurs de services d'infrastructure** : il devrait être possible d'utiliser les services de nombreux fournisseurs de services d'infrastructure sans impact majeur sur les performances du service ;
- **Mise à l'échelle selon le nombre de demandes de l'utilisateur** : il est important de donner des garanties concernant la mise à l'échelle par rapport au nombre de requêtes traitées parallèlement ;
- **Indépendance d'un fournisseur de services d'infrastructure** : le fournisseur de services d'infrastructure devrait être en mesure d'accepter ou de refuser toute demande d'un utilisateur de service d'infrastructure. Des politiques privées peuvent être appliquées localement si nécessaire ;
- **Filtrage** : un domaine peut appliquer n'importe quelle politique de filtrage. Il peut choisir de limiter les interactions externes avec son propre domaine, y compris l'interaction d'approvisionnement inter-domaines.

Un objectif clé est de rendre le plan de contrôle indépendant des technologies hétérogènes de réseaux sous-jacents en définissant et en précisant des interfaces et des cadriciels de communications avec le bon niveau d'abstraction.

Le contrôle de l'infrastructure peut être décomposé en deux grands aspects dans l'architecture du Cloud Networking : *delegation-based control* et *distributed coordination*. Le premier de ces deux aspects réfère aux demandes invoquées par l'utilisateur du service d'infrastructure qui se traduisent par des actions effectuées sur les ressources virtuelles. C'est une cascade à travers la hiérarchie de délégation de services d'infrastructure selon les fonctions de gestion des services. Le second aspect réfère à l'interaction entre les administrateurs dans le plan de contrôle distribué (DCP) pour partager ou négocier des informations de configuration. Ces aspects sont présentés dans la figure 3.4. Ils sont liés à ce que les demandes de contrôle basées sur la délégation peuvent déboucher sur des actions qui nécessitent une coordination distribuée.

Contrôle basé sur la délégation

Le contrôle basé sur la délégation est exercé par une interface de service de l'infrastructure (*Infrastructure Service* dans la figure 3.4). L'acte de délégation constitue une relation de confiance qui est la base de la délégation de responsabilités. De ce fait, l'élément clé de l'interface de service de l'infrastructure est la capacité d'authentification mutuelle entre l'utilisateur et le fournisseur.

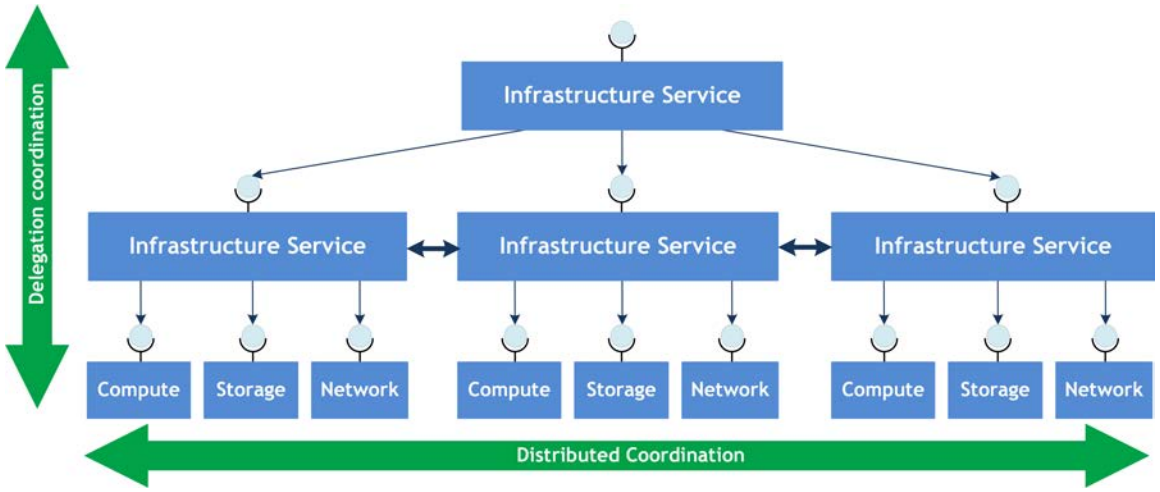

Figure 3.4 – *Les deux aspects du contrôle de l'infrastructure pour le Cloud Networking*

La délégation est concernée par le transfert des responsabilités de gestion. Il s'agit d'une relation à double sens dans lequel l'utilisateur du service d'infrastructure délègue la responsabilité de gestion au fournisseur de services d'infrastructure, et inversement les prestataires de services reportent l'état de l'infrastructure à l'utilisateur. Le fournisseur est libre de choisir la façon de la gestion et la mise en œuvre de l'infrastructure, y compris en utilisant une plus grande délégation à d'autres fournisseurs de services d'infrastructure. Cependant, il n'est pas obligé d'informer l'utilisateur de la façon de délégation (à moins qu'il y ait des règles dans les SLA qui l'obligent à le faire).

Comme la responsabilité de gestion est transmise d'une façon hiérarchique, le contrôle passe en bas de la hiérarchie et les rapports remontent vers le haut. Les niveaux supérieurs de la hiérarchie ont une vision plus globale et plus abstraite avec moins de détails de mise en œuvre. En bas de la hiérarchie se trouvent les services d'infrastructure avec leurs propres domaines administratifs qui contrôlent directement les ressources utilisées pour mettre en œuvre le service. Ceux-ci ont une connaissance complète des ressources sous leur contrôle, mais n'ont pas une vision globale de l'infrastructure.

Contrôle basé sur la coordination distribuée

La coordination distribuée est concernée par différentes opérations impliquant plusieurs agents. En particulier, nous considérons les opérations qui nécessitent une coordination entre plusieurs domaines administratifs au niveau bas d'une hiérarchie de délégation.

Une façon évidente de parvenir à une coordination distribuée est d'impliquer un point de contrôle commun pour coordonner les actions. Dans ce cas, les deux domaines administratifs auront toujours un utilisateur commun en dessus d'eux dans la hiérarchie de délégation. Toutefois, si les services d'infrastructure inférieurs ne sont pas disposés à communiquer les détails de mise en œuvre jusqu'à la chaîne de délégation, ils peuvent ne pas être en mesure de faire participer ce point de contrôle commun. Plus généralement, une fois la responsabilité de la gestion a été déléguée, elle ne fait plus partie des responsabilités

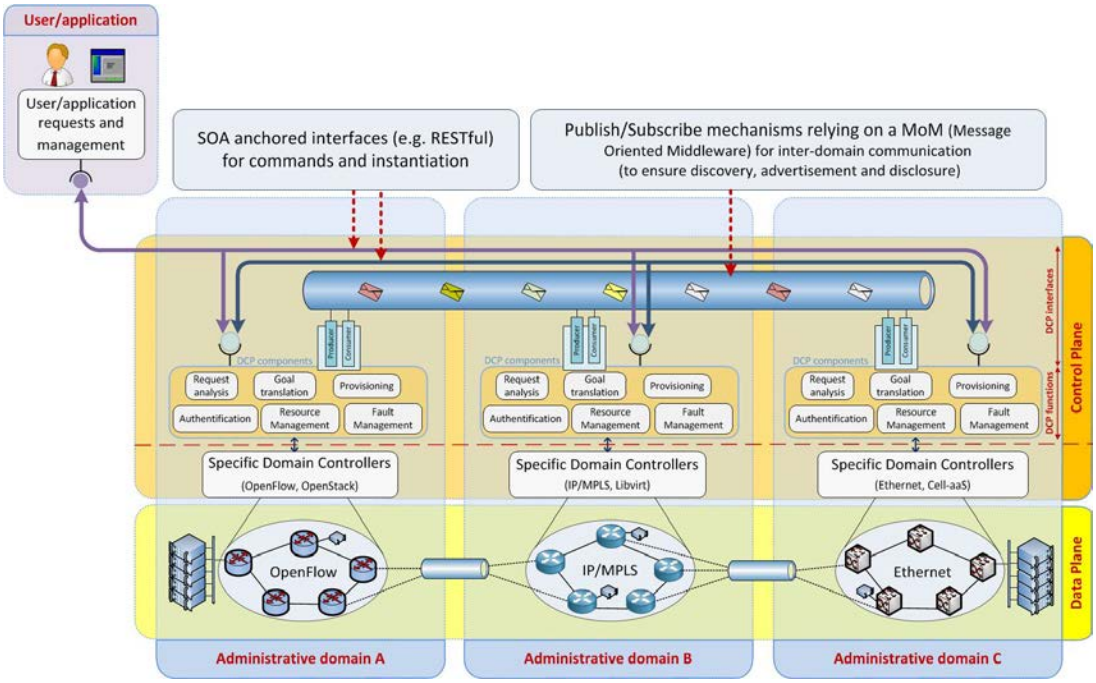

FIGURE 3.5 – *Les interactions de contrôle entre les domaines*

du délégant. Ainsi, la hiérarchie de délégation ne peut pas être exploitée pour réaliser la coordination distribuée.

En général, nous supposons que les domaines administratifs devront partager les informations nécessaires pour établir des liens avec leurs voisins immédiats, et donc il y aura un partage de l'information. Nous supposons aussi que la coordination distribuée suivra un modèle de communication point à point, en utilisant le partage des informations limitées, en dehors du champ d'application de la hiérarchie de la délégation. Cette forme de coordination distribuée se fait par l'intermédiaire du plan de contrôle distribué - DCP.

Afin d'interagir à travers le DCP, deux services d'infrastructure devront prouver qu'ils ont le pouvoir de délégation pour participer à la gestion d'une infrastructure ou d'une ressource donnée, et ils devront s'identifier en tant que voisins pour que l'information puisse être partagée.

3.2.9 CONCEPTION DU DCP

Les interactions de contrôle entre les interfaces de l'architecture du Cloud Networking prévoient un ensemble de fonctions qui sont responsables de l'attribution et le contrôle des ressources réseau distribuées au sein et entre les domaines administratifs. La forme de ces interfaces et le mode d'interaction entre eux soutiennent les principes de contrôle de l'infrastructure qui sont distribués et qui peuvent être sous le contrôle de différentes entités administratives.

Un objectif clé de la conception de l'architecture est de rendre l'infrastructure de contrôle indépendante des technologies hétérogènes de réseaux sous-jacents en définissant et précisant des interfaces et des communications middleware avec le bon niveau d'abstraction.

Une solution possible est de construire des interfaces inter-domaines comme la montre la figure 3.5 en s'appuyant sur :

- Interfaces SOA, par exemple Interfaces REST, pour les commandes et l'instanciation dans la partie commune et distribuée du contrôle ;
- Mécanismes Publish/Subscribe s'appuyant sur un MOM pour les communications inter-domaines.

Les interfaces SOA sont similaires à l'interface de service de l'infrastructure en termes d'actions de commandes et d'instanciation. Les mécanismes de publish-subscribe permettent la découverte de topologie entre et au sein des domaines avec une divulgation d'information en vertu d'accords de sécurité et de services entre les domaines. Lors de cette opération, l'information est cachée ou communiquer en toute sécurité et de manière sélective en fonction de SLA établi entre les fournisseurs. Le système de publish-subscribe est généralement conçu pour contrôler également la divulgation de l'information et intègre une découverte et un cadre de notification pour faciliter la coopération inter-domaine, l'interaction et l'échange d'informations (y compris la résolution de référence).

L'objectif des interactions via le DCP est de réaliser la coopération entre deux agents indépendants. L'utilisation d'un MOM découple l'interaction dans le temps et l'espace et fournit les agents à la liberté d'agir de façon indépendante et mettre en œuvre leurs fonctions comme ils ont choisi.

La conception du système prend en charge les deux types d'arrangement et d'interaction : hiérarchique (*Hierarchical*) pour le contrôle basé sur la délégation et pair à pair (*Peering*) pour le contrôle basé sur la coordination distribuée. Dans un arrangement de type *Peering*, chaque fournisseur de services d'infrastructure doit conclure des accords de partage des ressources avec les fournisseurs adjacents (pas entre les domaines qui sont plus d'un saut). Un service distribué est créé par l'interaction entre ces relations.

Dans un agencement hiérarchique, un fournisseur de services d'infrastructure compose un service d'infrastructure distribuée basée sur les ressources fournies par plusieurs fournisseurs de services d'infrastructure. Le premier fournisseur agit comme un utilisateur de ces fournisseurs et interagit avec chacun indépendamment. Ce fournisseur de services d'infrastructure n'a pas besoin de posséder l'infrastructure utilisée pour mettre en œuvre le service.

Même si les fournisseurs de communications préfèrent offrir des services qui reflètent la structure actuelle d'Internet à faible couplage, la nature des interactions de contrôle est susceptible d'être influencée autant par des relations métiers et business que par la technologie. Le modèle hiérarchique permet la création d'une nouvelle entité d'affaires qui peut composer les services et choisit les différentes sources d'infrastructure en fonction de plusieurs paramètres (par exemple le prix, la fiabilité, la sécurité, etc.). En outre, un fournisseur existant de services d'infrastructure (par exemple, un fournisseur de réseau, les fournisseurs de Cloud Computing) peut implémenter cette fonctionnalité de délégation.

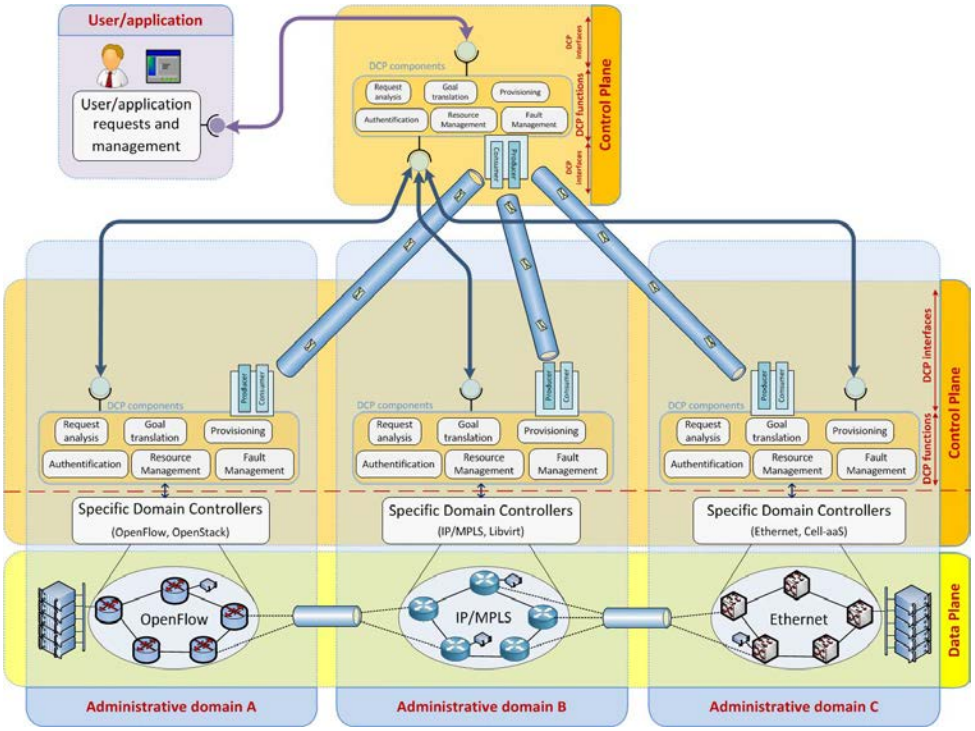

FIGURE 3.6 – *Contrôle d'interaction hiérarchique*

La délégation à travers l'interface de services d'infrastructure est un acte synchrone dans lequel l'utilisateur passe la responsabilité de la gestion au fournisseur. L'invocation des actions de gestion, en réponse à l'adoption de la responsabilité, peut être asynchrone. La livraison des rapports d'état à l'utilisateur peut être synchrone en réponse à une requête ou sous la forme de notifications.

En plus des interactions de commande au niveau de la couche *cross-domain infrastructure service*, il peut y avoir une interaction à la couche de ressources à établir et à gérer des liens entre les ressources dans différents domaines d'administration. Cela représente la réalisation de l'interconnexion inter ressources convenu à la couche supérieure. Les deux formes de coordination sont atteintes grâce à l'utilisation du DCP en coopération des ressources de plusieurs domaines en réponse aux demandes reçues par l'interface de service de l'infrastructure. La portée de cette gestion dépend des accords de niveau de service établis entre les domaines, car elles déterminent le partage de l'information et à la disposition du plan de contrôle commun.

Les deux approches hiérarchiques et de *peering* peuvent être adoptées pour la conception de la DCP pour la coordination et la gestion inter-domaine. Ces deux configurations possibles pour le DCP et ses interactions avec les contrôleurs de domaine spécifiques sont illustrées à la figure 3.6 et la figure 3.7 pour les solutions hiérarchiques et de peering respectivement.

Dans la solution hiérarchique, un dispositif de commande commun agit comme une racine du système et gère toutes les communications entre les différents domaines et est le

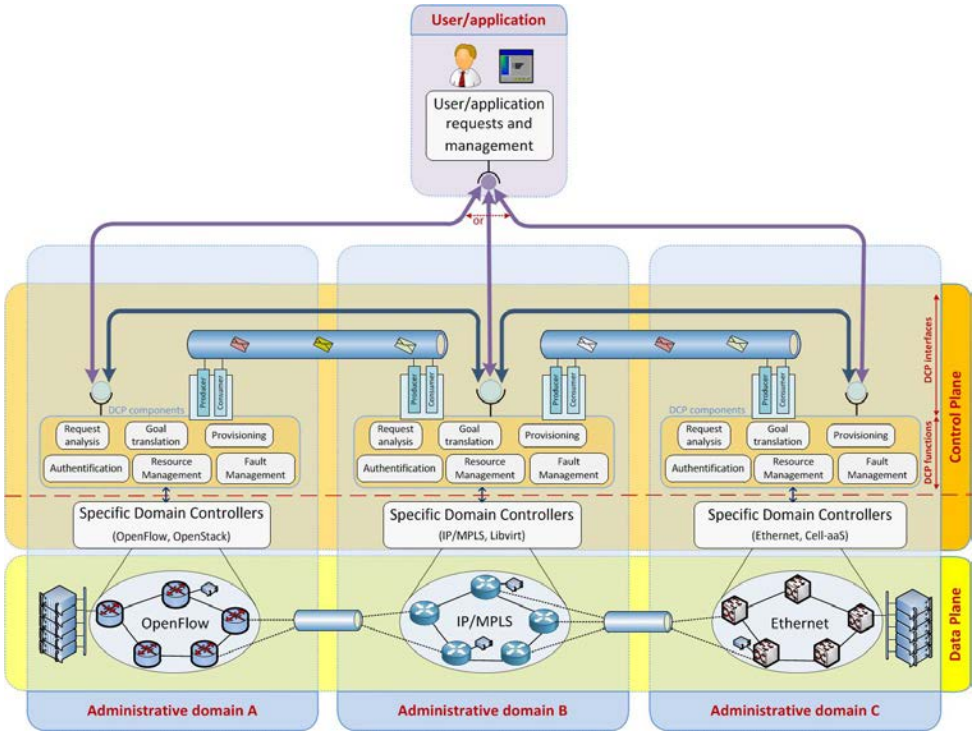

FIGURE 3.7 – *Contrôle d'interaction pair-à-pair*

point d'accès de service pour l'interface de service. Dans l'approche de peering, le DCP s'appuie sur le paradigme de *peering* où tous les contrôleurs sont des pairs logiques qui coopèrent pour parvenir à une gestion conjointe. L'utilisateur peut accéder à l'interface de service fourni par n'importe quel domaine participant.

3.3 Architecture de courtage Cloud

3.3.1 Contexte

Les travaux de courtage Cloud s'inscrivent dans le cadre d'un projet national français nommé CompatibleOne. Le projet s'est inscrit dans un programme et une vision structurée en deux phases. La première phase, nommée Compatible0.5, effectuée sur ressources propres au sein de l'Institut Mines-Télécom, prépare la plateforme d'expérimentation et de validation qui servira de socle à CompatibleOne. Cette première phase installe des outils et librairies essentiels aux études de portabilité, de migration et de compatibilité dans les Clouds. Quant à la deuxième phase, le projet CompatibleOne, il se repose sur les travaux de Compatible0.5 et propose des développements en logiciel libre pour faciliter la création de Clouds privés, publics et hybrides. Ces développements permettent d'éliminer les contraintes du Cloud qui mettent l'entreprise en forte dépendance des technologies spécifiques des fournisseurs (Amazon AWS, Google Cloud et Microsoft Azure étant les plus dominants). C'est le problème connu par le *Vendor Lock-In*.

Grâce au courtage Cloud, chaque acteur peut acquérir des ressources et des services clouds et peut les combiner avec l'existant à des coûts réduits. Tout entreprise ou acteur peuvent obtenir des ressources (machines virtuelles, stockage, réseau et puissance de calcul), y déployer ses applications et services, les porter ou migrer d'un environnement à un autre sans avoir à les concevoir à nouveau.

L'objectif du courtage Cloud est de s'abstraire des dépendances, d'assurer la portabilité des interfaces et des images et surtout de libérer les acteurs et entreprises des fournisseurs de services Cloud qu'ils soient de type IaaS, PaaS ou SaaS. Dans notre travail, on se focalise sur les couches IaaS et PaaS.

CompatibleOne est un projet R&D dont l'ambition est de participer au positionnement des acteurs français dans le marché du Cloud. Ce projet développe les composants technologiques qui sont nécessaires à des acteurs économiques pour accroître leur activité dans l'exploitation de solutions de Cloud Computing, le service autour du Cloud Computing et la fourniture d'applications en mode SaaS (*Software as a Service*). Pour obtenir un logiciel évolutif et offrant une interopérabilité, le projet élabore un méta-modèle, des méta-langages et des APIs (*Application Programming Interface*) qui serviront comme référence pour la phase d'implémentation. Les principaux différenciateurs du logiciel par rapport à l'existant sont la portabilité des applications (possibilité de création de Cloud hybride en s'appuyant sur des ressources privées ou publiques) et l'intégration de la sécurité et de la qualité de service dans l'ensemble des couches du Cloud (infrastructure, plateforme, application, portail d'accès, etc.).

3.3.2 Courtage Cloud Open Source

En parallèle à l'adoption du Cloud, il est intéressant d'observer la forte pénétration de l'Open Source sur le marché. À force de constater l'engouement des utilisateurs de

l'informatique pour l'Open Source, les solutions Cloud se retrouvent de même dans une philosophie Open Source. En effet, les utilisateurs se basent de plus en plus sur des logiciels Open Source dans leurs offres. En effet, ces solutions leur permettent de minimiser significativement les coûts tout en assurant la maîtrise de leurs infrastructures et en éliminant le *Vendor Lock-In*. Les solutions Open Source et les standards ouverts ne sont pas seulement des technologies de choix, mais aussi des éléments stratégiques. Nous assistons à l'émergence de nombreuses entreprises qui utilisent l'Open Source pour pénétrer plus rapidement les marchés tout en transformant leurs solutions en standards de fait. Dans le monde ouvert que représente le Cloud aujourd'hui, ces stratégies sont très efficaces pour s'imposer sur le marché. De plus, le modèle collaboratif de l'Open Source lié à de nouveaux usages favorise le foisonnement d'innovations en termes de logiciels, mais surtout en termes de services.

La diversification des offres Cloud ainsi que leur spécialisation pour répondre aux besoins spécifiques des secteurs du marché, font des technologies Open Source des technologies stratégiques au développement des entreprises offrant des services Clouds. En évitant de réinventer la roue et en bénéficiant de la possibilité d'utiliser librement des technologies innovantes ayant fait leurs preuves, les entrepreneurs peuvent enfin se consacrer à leur cœur de métier. De plus, ils ont accès à ces technologies sans avoir à investir dans des contrats onéreux les privant de marges bénéficiaires de plus en plus réduites du fait de la concurrence. Si la tentation peut être grande d'acheter une technologie bien particulière de façon à arriver rapidement sur le marché avec une offre, le bénéfice de cette entrée rapide risque fortement de compromettre l'avenir et l'évolution de l'entreprise. Ce risque est lié à la forte dépendance technologique d'un seul fournisseur.

De nombreuses entreprises, bien qu'attirées par le concept et la promesse de diminution des coûts de l'informatique, se refusent à l'idée d'adopter des Clouds publics principalement pour des raisons de sécurité et de souveraineté. Par contre, elles envisagent très fortement la migration de leurs systèmes d'information vers des Clouds privés avec d'éventuelles passerelles vers le public (Clouds Hybrides). Ici, l'existence de Cloud Open Source orienté courtage prend tout son sens, car il garantira aux utilisateurs une grande indépendance de choix tant des composants que des fournisseurs. De plus, le Cloud Open Source assurera l'interopérabilité et l'évolutivité des systèmes, ainsi que la sécurisation des données critiques. Par la même occasion, il supportera le développement de tout un secteur d'activité hautement spécialisé et à haute valeur ajoutée autour de l'intégration, l'administration, la migration et la fédération de Clouds privés, mais aussi le développement de nouveaux services.

3.3.3 Objectives

Les principaux objectifs du Courtage Cloud sont :

- Assurer l'évolutivité : Il est plus que probable que de nouvelles techniques, logiciels et outils verront rapidement le jour. Pour rester dans la compétition, les acteurs

devront s'adapter en modifiant leur Cloud, en intégrant ou en substituant un nouveau composant. L'ouverture et la capacité à évoluer d'un Cloud seront déterminantes pour garder la compétitivité des entreprises. L'émergence des standards est également un élément important pour garder une évolution en continu et fluide ;

- Assurer l'interopérabilité et la compatibilité : L'interopérabilité doit permettre, aux développeurs de nouveaux services ainsi qu'aux utilisateurs, de changer de fournisseurs de cloud en fonction de leurs besoins et des offres. Ceci sans risque d'avoir à redévelopper leurs services ou de ne pas pouvoir réutiliser leurs données personnelles et critiques. Les offres de Cloud vont se multiplier et le besoin d'interopérabilité va devenir un élément majeur pour les acteurs du domaine, car l'absence d'interopérabilité représente un frein à la compétition et à l'innovation ;
- Assurer la qualité de service : La raison première de l'émergence du Cloud Computing réside dans la mutualisation des ressources informatiques. Pour réaliser cette mutualisation, on crée de nombreux objets virtuels ce qui rend la gestion de la qualité de service particulièrement complexe. Il est essentiel d'avoir une stratégie globale de gestion de la qualité de service qui prend en compte l'ensemble des acteurs : fournisseurs et utilisateur des services Cloud ;
- Assurer la sécurité et la confidentialité : La virtualisation et les couches d'abstraction introduisent des risques supplémentaires allant du code malicieux capable de différencier le natif du virtuel à un code caché qui peut contrôler l'environnement de virtualisation. D'un point de vue utilisateur, la sécurité actuelle n'est pas synonyme de confiance. Il faut alors pouvoir prouver et démontrer le niveau de sécurité et de confidentialité annoncées. Des outils d'audit et de traces sont indispensables. Les aspects de vérification et validation sont des éléments supplémentaires pour obtenir cette confiance.

Outre sa dimension Open Source et interopérable, la solution de Courtage Cloud doit avoir comme différenciateur majeur, par rapport aux autres plates-formes de Cloud existantes, sa capacité à orchestrer dynamiquement des services combinés de type IaaS, PaaS et SaaS. Le but est d'adresser de façon optimale et garantie des contraintes de fonctionnement exprimées par les utilisateurs dans un contexte intra-Cloud ou inter-Cloud. Ces innovations clefs adressent en particulier les verrous suivants : modélisation des ressources d'un Cloud et des contraintes de fonctionnement de services Cloud, administration et contrôle unifiés des ressources dans un environnement hybride, gestion de la dynamicité de l'infrastructure et mise au point d'algorithmes d'optimisation de l'orchestration des services. L'exploitation commerciale de ces innovations implique par ailleurs d'être capable de facturer les services consommés et donc d'éliminer le problème d'existence d'une plate-forme de facturation de ressources Cloud.

Modélisation

L'écosystème Cloud est un système complexe dont l'administration et l'optimisation

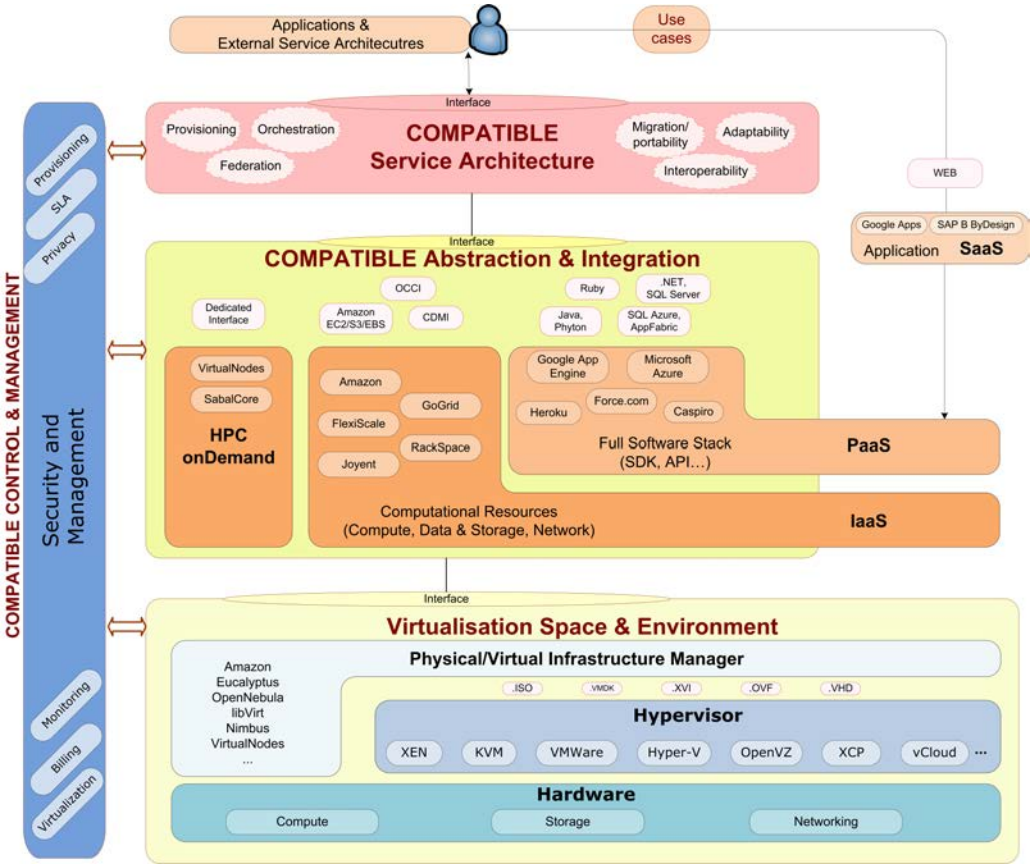

Figure 3.8 – *Première version de l'architecture de Courtage Cloud*

nécessitent la mise au point de modèles statiques et dynamiques représentant l'état des ressources et leur évolution dans le temps. Le courtier Cloud adressera ce verrou en définissant une syntaxe, une sémantique et une grammaire de description unifiée des ressources et des services d'un Cloud en termes de qualité de service et en termes de sécurité. La difficulté de la tâche réside, d'une part, dans l'étendue des ressources et des services à modéliser (ressources matérielles, réseau, machines et réseaux virtuels). D'autre part, la difficulté réside dans la nécessité d'aboutir à un modèle d'administration commun à l'ensemble des couches d'un Cloud (Iaas, PaaS, SaaS) qui soit compatible avec les infrastructures et outils de référence. Un méta-modèle facilitera l'adaptation des applications ainsi que la standardisation des méthodes et outils de garantie de la qualité de service dans un Cloud.

Interopérabilité

L'interopérabilité entre Clouds de natures hétérogènes et de fournisseurs différents est un enjeu clé pour le développement économique du Cloud dans un monde résolument ouvert à la concurrence. Aujourd'hui, les offres existantes ne peuvent pas interopérer que partiellement par l'intermédiaire de librairies capables d'abstraire une partie du modèle sous-jacent (libvirt dans le cas des machines virtuelles par exemple). Avec un courtage Cloud, l'interopérabilité sera étendue à l'ensemble des briques du cloud, grâce à :

- l'implémentation du modèle de représentation et d'administration commun aux

ressources et services ;

- la mise au point de mécanismes de migration de machines virtuelles dans une fédération de clouds ;
- la définition d'interfaces, de protocoles de routage et de plan de contrôle communs pour l'inter-Cloud Networking ;
- la capacité d'allouer dynamiquement des réseaux virtuels inter-Clouds.

3.3.4 Architecture de Courtage Cloud proposée

L'architecture de Courtage Cloud proposée a passé par plusieurs versions évolutives. La première version, figure 3.8, était orientée principalement pour un positionnement du courtage Cloud par rapport à l'écosystème Cloud. Cette architecture initiale de référence a servi comme un point de départ et a permis par la suite de définir une deuxième version de l'architecture de courtage Cloud (figure 3.9). À base de la deuxième version de l'architecture de courtage Cloud, une troisième version intermédiaire a été proposée avec une vision plus détaillée (figure 3.10).

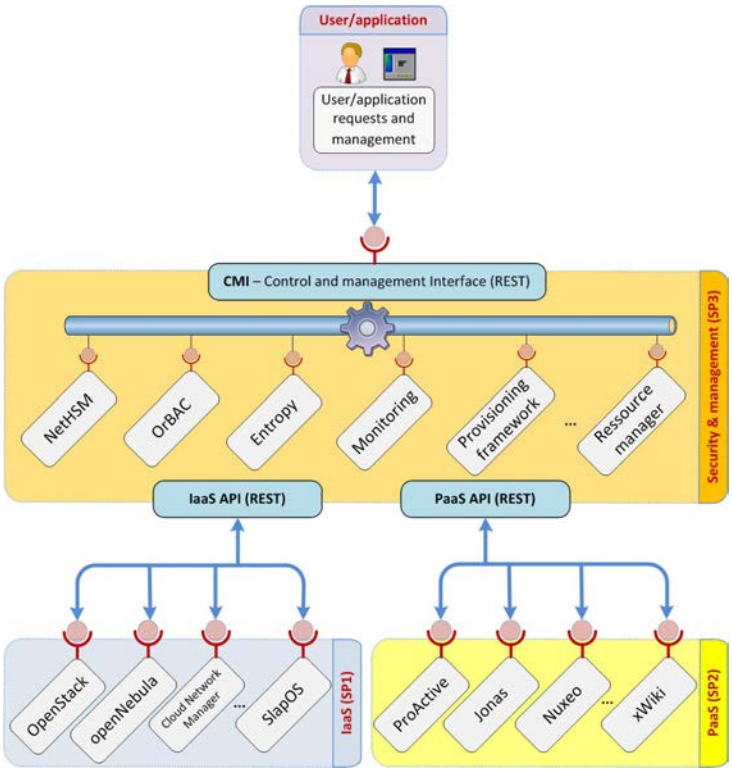

Figure 3.9 – *Deuxième version de l'architecture de Courtage Cloud*

Enfin, dans le cadre d'une coopération au sein d'un groupe de travail d'architecte, ces travaux ont finalement mené à la production de l'architecture finale du Courage Cloud adopté dans le projet CompatibleOne. Cette architecture finale est illustrée dans la figure 3.11.

La fourniture des ressources et services Cloud en utilisant l'architecture du courtage Cloud est réalisée en quatre étapes. Ces étapes sont schématisées par quatre quadrants dans

la figure 3.11. Ces quadrants représentent les quatre étapes du cycle de fonctionnement :

- Première étape : Concepteur (*Designer*) ;
- Deuxième étape : Planificateur (*Scheduler*) ;
- Troisième étape : Services ;
- Quatrième étape : Opérateur (*Operator*).

Chacune de ces étapes est nécessaire à la fourniture des ressources IaaS et PaaS. Ces étapes sont détaillées ci-après :

Designer

La première étape porte sur l'interactivité avec l'utilisateur (premier quadrant de la figure 3.11). Les utilisateurs spécifient dans cette étape leurs besoins en termes de ressources IaaS et PaaS. A base de cette spécification, un manifeste décrivant la requête de l'utilisateur sera créé. Ce manifeste décrit les détails des services qui doivent être offerts par les fournisseurs de Cloud, les critères techniques et économiques, les caractéristiques et les contraintes qui doivent être pris en considération.

Scheduler

Dans la deuxième étape, le manifeste généré par la première étape est transféré à un parseur (deuxième quadrant de la figure 3.11), pour analyser et valider ces documents. L'analyseur construit un plan d'approvisionnement précis pour la description des services requis.

Services

Les services du Courtier Cloud sont responsables de l'exécution du plan d'approvisionnement (troisième quadrant de la figure 3.11). Le courtier fournit des services d'arbitrage et de négociation pour sélectionner les fournisseurs appropriés afin de satisfaire toutes les exigences et les contraintes exprimées dans le plan d'approvisionnement (par exemple, des zones géographiques, la sécurité, les règles d'accès, de performance, de facturation, etc). Plus précisément, les services du courtier sont responsables du traitement de la condition d'accord de niveau de service (SLA) pour créer une instance du service correspondant à la description fournie par le plan. Selon le besoin et le contenu du plan d'approvisionnement, un ou plusieurs services de courtage Cloud sont sollicités. Le choix des services se fait via un composant qui s'appelle *Broker*. Une liste non exhaustive de ces services est détaillée dans ce qui suit :

- Service de Production d'Image ;
- Service de Placement ;
- Service de Networking ;
- Service de Monitoring ;
- Service de Sécurité ;

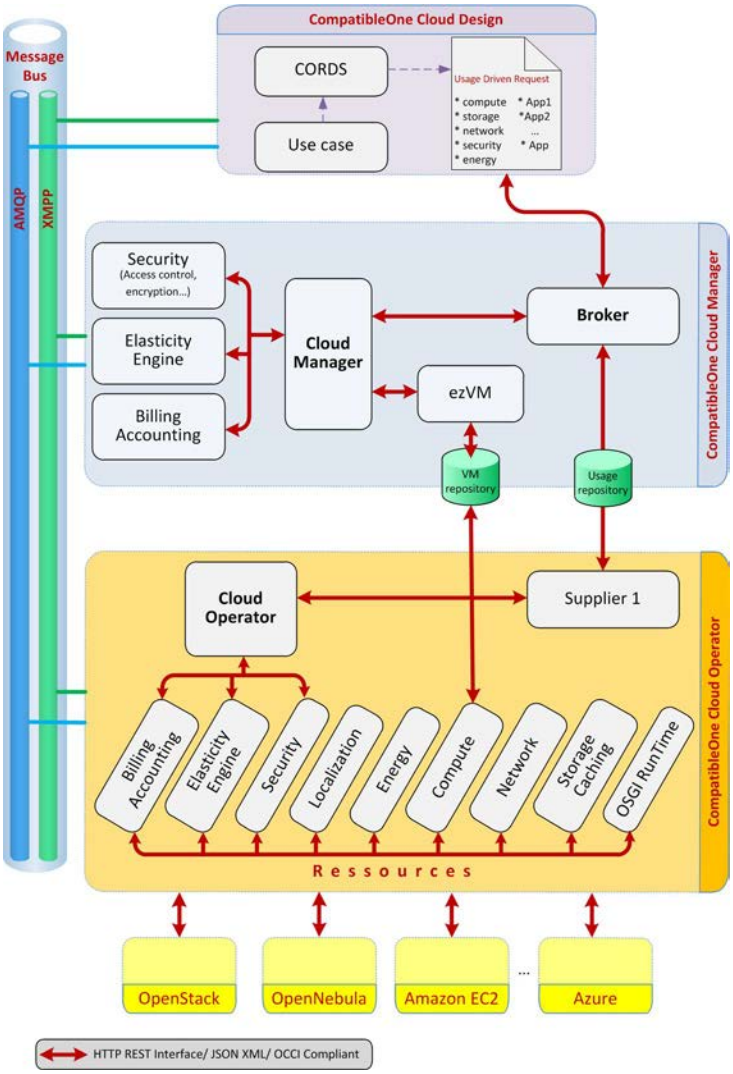

FIGURE 3.10 – *Troisième version de l'architecture de Courtage Cloud*

- Service d'Efficacité Énergétique ;
- Service de Comptabilité & Facturation ;
- Service d'Élasticité.

Le service de courtage est sollicité selon la demande de l'utilisateur, dans le but d'agréger les plans d'approvisionnement pour un ensemble de contrats avec des fournisseurs. En effet, un contrat est un document provenant du plan, qui décrit la transaction à opérer entre l'utilisateur qui fournit les premiers manifestes et le fournisseur de Cloud qui gère l'action d'approvisionnement réel des ressources.

Operator

Dans la quatrième étape, les contrats obtenus sont envoyés à un ou plusieurs fournisseurs de ressources Cloud. Ces fournisseurs peuvent être des fournisseurs publics (Amazon AWS, Microsoft Azure ou HP Cloud) ou fournisseurs privés (OpenStack, OpenNebula ou VMware vCloud). Par exemple, les contrats d'approvisionnement IaaS seront redirigés vers Windows Azure si le choix du courtier a été réalisé sur un fournisseur Windows Azure dans la deuxième étape. En effet, le choix des fournisseurs de ressources se base sur une entité qui s'appelle *Procci*, dont le rôle est de faire l'aiguillage pour la sélection du bon fournisseur.

En plus des quatre étapes, il existe des services qui assurent le fonctionnement interne de la solution globale de courtage. Ces services se placent au milieu de la figure 3.11. Le premier service qui s'appelle *Publisher* gère une base de connaissance des services existants dans le framework de courtage. C'est le premier service qui se démarre lors de l'initialisation de la plateforme. Un deuxième service concerne la sécurité interne. Ce service permet d'assurer et garantir le bon niveau sécuritaire du fonctionnement de tout le framework de courtage. La gestion de l'identité peut être intégrée dans ce service. De plus, il est également responsable de la sécurité de l'échange de données pour permettre le transfert sécurisé en mode pair-à-pair. Le troisième service est lié au monitoring afin de collecter toutes les informations nécessaires par les autres composants de la plateforme de courtage.

3.3.5 Services de l'architecture proposée

Les services énumérés au tour du Courtier Cloud dans le troisième quadrant de la figure 3.11 sont détaillés dans cette section. Les services ci-dessus aident le courtier dans les opérations d'approvisionnement nécessaires lors de l'utilisation du plan.

Service de production d'Image

Le service de production d'image couvre la totalité de la production, livraison et personnalisation des images de machine virtuelle. Grâce à ce service, il est possible de produire n'importe quel format d'image requise par le fournisseur de Cloud qui est choisi à son tour par le service de placement du Courtier Cloud.

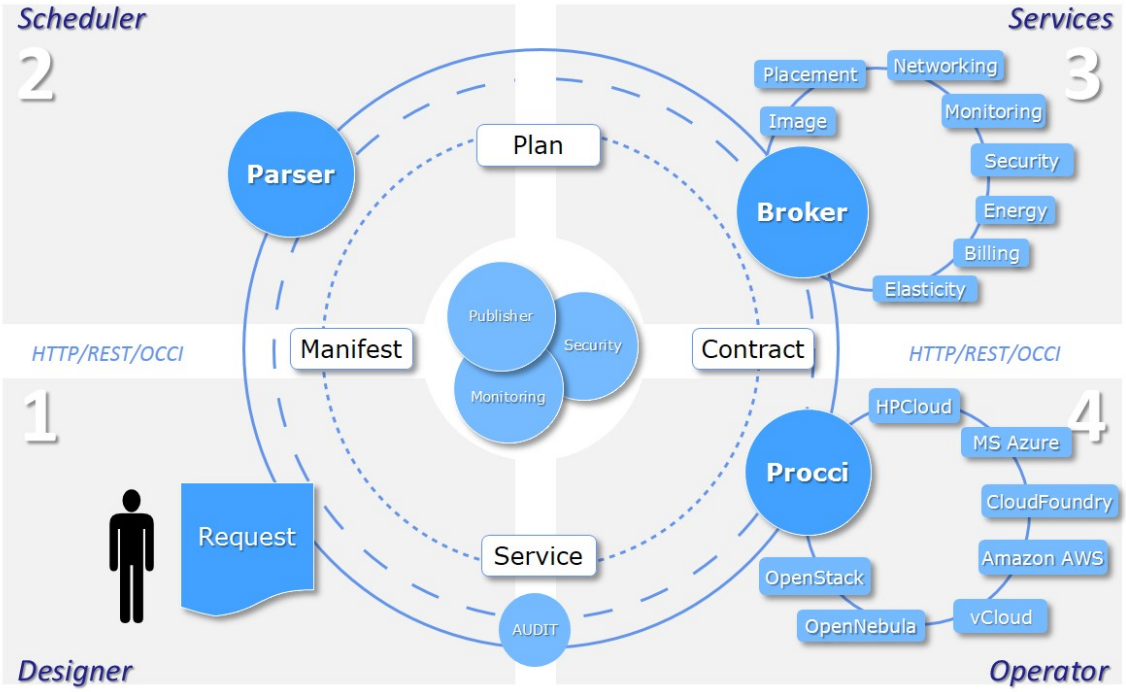

FIGURE 3.11 – *Quatrième version de l'architecture de Courtage Cloud*

SERVICE DE PLACEMENT

Le service de placement permet au courtier de déterminer l'emplacement optimal d'une ressource. Il se base sur les informations contenues dans les plans d'approvisionnement et aussi selon les SLAs et les descriptions des fournisseurs de ressources Cloud. Les algorithmes de placement peuvent être personnalisés par les utilisateurs en fonction de leurs besoins et de leurs modèles d'affaires.

SERVICE DE NETWORKING

Le service de Networking assure l'interconnexion sécurisée entre les ressources et les services Cloud. Il permet d'allouer des adresses dynamiques pour connecter les consommateurs et les fournisseurs, équilibrer la charge entre les fournisseurs via un réseau virtuel, migrer le service d'un fournisseur à un autre, et enfin offrir des adresses IP publiques élastiques à la demande.

SERVICE DE MONITORING

Le service de Monitoring gère les agents, les sondes et les consommateurs permettant à la plate-forme de surveiller tous les services requis et les ressources approvisionnées. Ce service fournit un support d'information afin de prendre des décisions pour d'autres services comme celui de placement, de sécurité ou de facturation.

Service de Sécurité

Le service de sécurité vérifie, assure et garantit la sécurité de ressources à allouer par les fournisseurs Cloud. Il interagit avec plusieurs autres services pour fournir la bonne décision et configuration liée à l'aspect de sécurité.

Service d'efficacité énergétique

Le service de l'efficacité énergétique assure la collecte et le stockage des données de la consommation des ressources afin d'optimiser la consommation d'énergie. Un ensemble d'outils sont déployés et instrumentés pour mesurer la consommation à travers différentes solutions existante sur le marché. Avec ce service, un consommateur peut exiger que ces traitements soient effectués sur des Cloud spécifiques avec les normes de *Green Cloud*.

Service de comptabilité et de facturation

Le service de comptabilité et de facturation permet une exploitation commerciale (calcul des coûts, la facturation, etc.) et fournit l'information comptable. Ce service gère non seulement les aspects techniques, mais aussi les aspects économiques démontrant la pertinence de l'utilisation du Cloud. Cela induit non seulement à de nouvelles technologies, mais aussi à de nouveaux modèles d'affaires.

Service d'élasticité

Le service d'élasticité met en œuvre un ensemble d'algorithmes de gestion, de contrôle et d'optimisation des ressources Cloud. Ce service fonctionne selon les besoins des utilisateurs et les données de surveillance fournies. Ces algorithmes sont capables d'ajuster et d'optimiser les ressources de Cloud (ressources de calcul, de réseau et de stockage) selon les exigences des utilisateurs, des applications et des environnements Cloud.

3.4 Conclusion

Dans ce chapitre, nous avons présenté deux architectures de fédération Cloud. La première architecture se focalise sur la convergence entre le Cloud Computing et le Cloud Networking. Cette architecture prend en considération les deux types d'infrastructure (Uni-domaine et Inter-Domaine). De plus, elle se base sur le nouveau concept de DCP. Ce concept a été proposé pour assurer la convergence souhaitée. Quant à la deuxième architecture, elle traite le Courtage Cloud. Cette architecture prend en considération tous types de services Cloud existants. De plus, elle a un niveau de flexibilité important pour pouvoir évoluer et décrire tout nouveau service Cloud. Dans les deux architectures proposées, deux éléments essentiels ont été définis pour assurer la fédération des Clouds. Le premier élément concerne les interfaces d'allocation et de contrôle. Le deuxième élément concerne le système d'échange de messages.

Nous proposons dans le chapitre suivant une contribution majeure de ce doctorat. Cette contribution concerne la description détaillée du premier élément essentiel des architectures de fédération qui est l'interface générique Cloud. Cette interface inclut une importante

extension liée à la gestion des ressources de type Cloud Networking.

Chapitre 4

Interface Cloud Networking

Sommaire

4.1 Introduction

La fédération Cloud passe par une utilisation d'interfaces ouvertes et interopérables pour les opérations d'allocation et de gestion des ressources Cloud. Ce genre d'interfaces devrait gérer les trois principaux types de ressources : *Compute*, *Storage* et *Network*. Cependant, les interfaces existantes se focalisent principalement sur les deux premiers types de ressources (*Compute* et *Storage*). La négligence de la composante *Network* se manifeste encore plus quand il s'agit de ressources de Cloud Networking situées au niveau des fournisseurs de réseaux.

Afin de remédier à ce manque de gestion des ressources du Cloud Networking et pour faciliter la fédération Cloud, nous proposons une interface ouverte et générique qui couvre les trois types de ressources Cloud. Cette proposition fait l'objet d'une contribution majeure de notre doctorat. Comme illustré dans la figure 4.1, 30 % des travaux de cette thèse portent sur la définition et la modélisation de cette interface. Cette nouvelle interface Cloud générique est nommée *Open Cloud Networking Interface* (OCNI).

Dans la suite de ce chapitre, la section 4.2 définit le contexte et les objectifs de l'interface OCNI. Ensuite, les sections 4.3 et 4.4 décrivent respectivement les principales interfaces Cloud existantes et les langages de description des ressources réseau classiques. Par la suite, l'interface OCNI sera définie dans la section 4.5 et sa spécification dans la section 4.6. Nous poursuivons ce chapitre par une description et une évaluation de notre implémentation Open Source d'OCNI dans la section 4.7. Après, nous décrivons dans la section 4.8 quelques frameworks qui ont utilisé l'interface OCNI. Enfin, nous concluons dans la section 4.9.

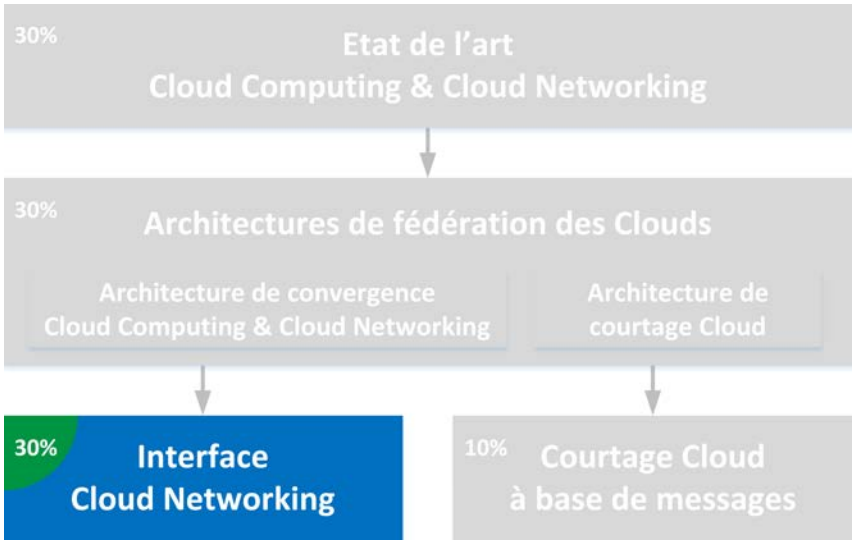

Figure 4.1 – *Structure du document - Chapitre du Cloud Interface OCNI*

4.2 Contexte et objectif

Le *Cloud Networking*, dans son sens le plus large, doit s'étendre au-delà de l'interconnexion existante des ressources *Cloud Computing*. Il doit aller plus loin que la mise en réseau spécifiée et disponible dans le Cloud d'aujourd'hui. Le *Cloud Networking* courant est généralement limité aux DataCenter et s'appuie exclusivement sur les technologies classiques de réseau pour interconnecter les différents sites et utilisateurs. Les fournisseurs de réseau et les services d'interconnexion au niveau WAN ne sont même pas pris en compte dans les spécifications actuelles dédiées au *Cloud Computing*.

Les technologies de communication, de type LAN pour les réseaux intra-Cloud et de type VPNs pour les réseaux inter-Cloud, constituent la majeure partie des offres d'interconnexion dans le domaine du *Cloud Computing*. En ce qui concerne les technologies de communications intra-Cloud, ceux qui existent actuellement peuvent satisfaire la plupart des exigences des DataCenters en isolant les réseaux de plusieurs utilisateurs et applications tout en partageant les mêmes ressources physiques (grâce à l'utilisation des VLAN, VXLAN, des technologies de virtualisation, etc.). Ce n'est pas le cas pour les réseaux inter-Cloud utilisés actuellement. En effet, ces derniers nécessitent d'utiliser et de respecter des spécifications et des standards ouverts ayant les mêmes caractéristiques des spécifications existantes du Cloud. En outre, les segments intra-Cloud et inter-Cloud sont traités séparément. Ils ont fait l'objet de solutions indépendantes, disjointes et hétérogènes.

Notre objectif est de combler l'écart entre les services d'interconnexion au niveau intra-Cloud et inter-Cloud en étendant la spécification OCCI qui se focalise sur les ressources de calculs et de stockage. Notre objectif est réalisé en ajoutant à OCCI la description et la gestion des ressources réseau ainsi que les services d'interconnexion. Cette stratégie permet d'une part l'inclusion de tout type de technologie de mise en réseau. D'autre part, elle facilite la participation des fournisseurs de réseaux *'Cloud Carrier'* et l'émergence des déploiements hybride du *Cloud Computing* via une interface Cloud standardisée.

Une approche réaliste et efficace pour atteindre cette convergence est d'étendre le standard Open Cloud Computing Interface (OCCI) d'OGF (*Open Grid Forum*) à toutes les technologies de réseau au lieu de se limiter à des VLAN et VPN. Notre solution préconisée se base sur la spécification OCCI pour avoir une spécification compatible nommée *Open Cloud Networking Interface* (OCNI). OCNI définit les ressources *Cloud Networking* (nœuds, liens du réseau et leurs interconnexions avec d'autres ressources de *Cloud Computing*), tout comme la définition des ressources *Cloud Computing* par OCCI. Le but est de faire évoluer le Cloud actuel à un déploiement hybride sur des infrastructures flexibles et extensibles fonctionnant dans de multiples domaines.

Notre but essentiel pour le *Cloud Networking* est de faciliter la mise en place de réseaux dédiés (peut-être un mélange de segments ou sous-graphes de réseaux virtuels et physiques) via les nœuds et les liens du réseau des fournisseurs. Les fournisseurs contribuent à l'interconnexion des machines virtuelles et des ressources des Datacenters distribuée en utilisant leurs propres ressources réseau et en coopérant avec d'autres fournisseurs afin

de composer l'infrastructure désignée et demandée par les utilisateurs. L'infrastructure souhaitée est un graphe composé de ressources Cloud et leurs liens d'interconnexion qui peuvent être considérés comme un réseau virtuel contenant des ressources de calcul, de stockage et des ressources de communication. Le *Cloud Networking* nécessaire pour établir les réseaux dans de multiples domaines et chez différents fournisseurs est nommé "*Flash Network Slice - FNS*". Le terme FNS a été défini dans [121, 138] par le projet européen SAIL [52] qui utilise OCNI pour les interfaces hautes vers les clients et les applications. Un FNS est une ressource Cloud de réseau complexe qui peut être approvisionnée rapidement via des interfaces HTTP *REpresentational State Transfer (REST)*. Le FNS est créé pour interconnecter des ressources distribuées, provenant de plusieurs infrastructures (par exemple, les centres de données), avec l'aide de fournisseurs de réseaux qui relient les ressources de *Cloud Computing* et de stockage distribué. Le FNS remplit les caractéristiques des solutions Clouds définies dans la section 2.2.2.

Un Flash Network Slice (FNS) peut être provisionné dans un domaine pour interconnecter des ressources qui lui sont liées. Le domaine dans ce cas peut être un centre de données ou un réseau d'opérateur. Dans le cas d'une infrastructure étendue sur plusieurs domaines, les FNS sont liés les uns aux autres pour fournir des capacités de communication entre les ressources des différents domaines. Un FNS est un composant logique qui peut être implémentée en utilisant différentes technologies de réseau sous-jacentes telles que VLAN, VXLAN, VPN-L2, L3-VPN ou OpenFlow.

Ce travail met l'accent sur l'un des deux mécanismes de fédération entre les Clouds : interface Cloud générique orientée *Cloud Networking*. Cette interface permet l'allocation et la gestion de tout type de ressource Cloud entre les utilisateurs et les fournisseurs dans plusieurs domaines hétérogènes. Le deuxième mécanisme de fédération qui assure une coopération inter-domaine (via un système d'échange de messages) sera traité dans le chapitre 5.

L'objectif de ce travail consiste essentiellement à étendre le modèle OCCI pour que le *Cloud Networking* soit également pris en compte. Cette extension permet aux fournisseurs de réseau de participer à la mise en place des infrastructures virtuelles et d'offrir des services de type Cloud orientés Networking à côté des fournisseurs de *Cloud Computing* classiques.

La figure 4.2 présente le contexte de l'allocation des ressources dans le Cloud lorsque plusieurs fournisseurs et domaines sont impliqués et lorsque les systèmes hétérogènes et leurs déploiements deviendront la norme plutôt que l'exception. Souvent, les fournisseurs de services Cloud utilisent des systèmes et des infrastructures assez homogènes pour faciliter le déploiement et la gestion. Le contexte considéré par notre travail est hétérogène avec des ressources allouées par les différents acteurs de *Cloud Computing* publics, privés et hybrides ainsi que les fournisseurs d'infrastructures réseau. Les fournisseurs de services réseau incluront, dans un proche avenir, des services de *Cloud Networking*. La figure 4.2 reflète cette situation, met en évidence les interfaces requises, révèle certains des défis

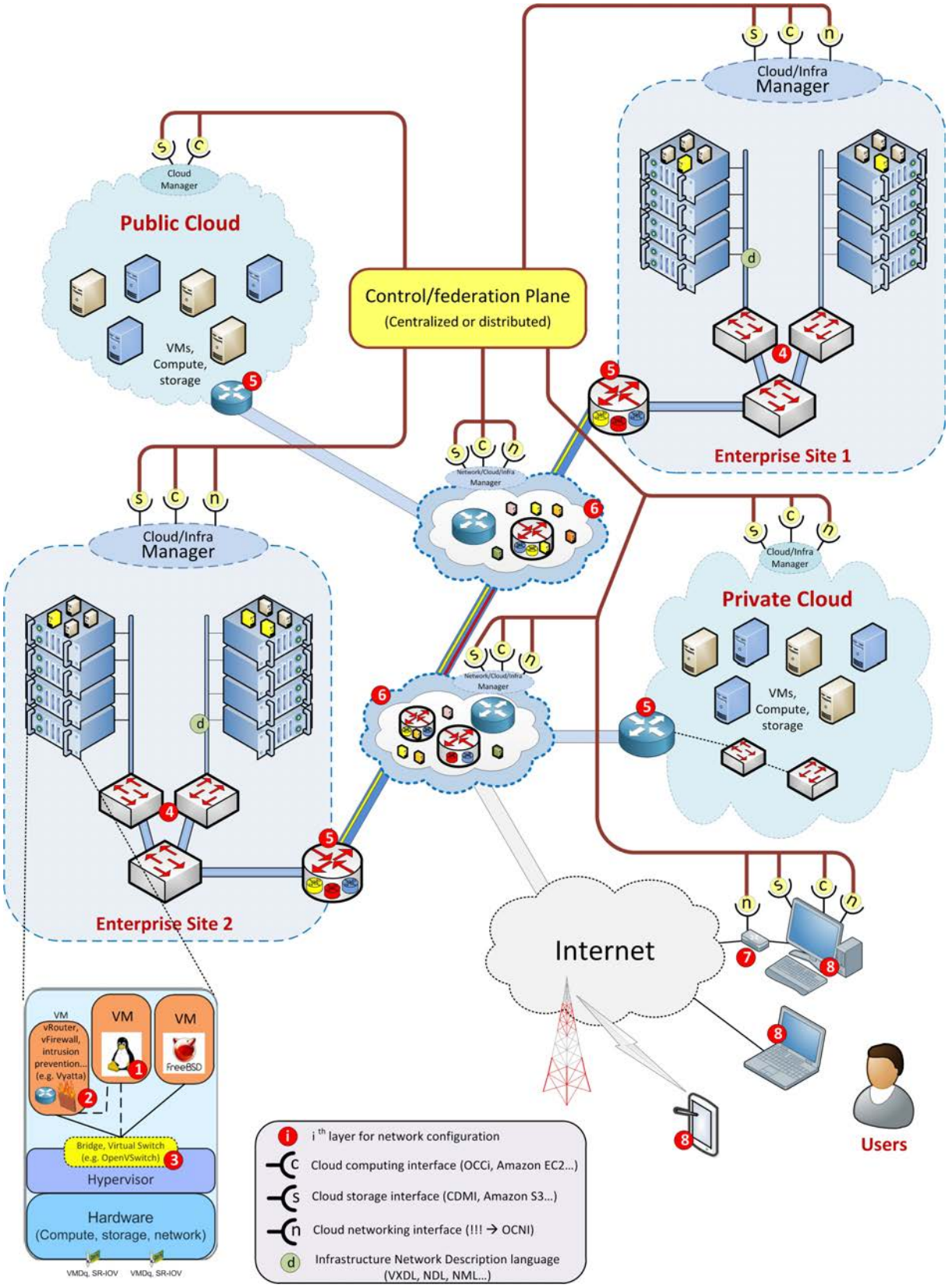

FIGURE 4.2 – *Interface Cloud Networking et contexte de configuration réseau*

de configuration *Cloud Networking* et illustre la fédération des Clouds hétérogènes d'une façon générale.

Aujourd'hui, un sous-ensemble d'interfaces et d'exigences de configuration est couvert par la communauté du *Cloud Computing*. Les ressources Clouds de calcul et de stockage peuvent être allouées et gérées via des interfaces et API Web bien définies. Parmi les interfaces de *Cloud Compute* nous citons Amazon EC2 Interface[4], OCCI[45] et CIMI[29] (voir section 4.3). Parmi les interfaces du *Cloud Storage* les plus populaires, nous citons Amazon S3 [5] et Cloud Data Management Interface (CDMI) [23]. Les interfaces manquantes sont les interfaces du *Cloud Networking* nécessaire à la connexion des ressources de calcul et de stockage aux nœuds du réseau et aux réseaux des fournisseurs *'Network Provider'*. L'objectif principal d'OCNI est de combler cette lacune et d'agir en tant que cadre intermédiaire clé entre OCCI et les langages de description des ressources réseau.

L'allocation des ressources *Cloud Networking* reste difficile à réaliser avec les frameworks existants vu qu'il n'y a pas de spécifications des interfaces dédiées au *Cloud Networking* pour faciliter l'allocation et l'interconnexion de tous types de ressources (ressources de calcul, de stockage et de réseau). Ces ressources peuvent appartenir à une ou plusieurs organisations dans un environnement de fédération (des sites de l'entreprise, les clients et leurs ressources de cloud allouées, des groupes d'utilisateurs ...). Le *Cloud Networking* devrait également venir avec un certain nombre de propriétés essentielles, de capacités, de services et de garanties. Ces fonctionnalités et services sont essentiels et permettent ainsi la configuration des éléments du réseau pour appliquer des règles régies par les politiques de placement et de déploiement convenues par les utilisateurs et les fournisseurs.

Pour atteindre tous les objectifs du *Cloud Networking* énoncés et pour fournir les fonctionnalités associées, huit niveaux ont été identifiés (où les ressources du réseau sont situées et ont besoin d'être configurées pour le *Cloud Networking*). Ces niveaux sont illustrés par les cercles rouges (numérotés de 1 à 8) dans les figures 4.2 et 4.3 et sont énumérés ci-dessous :

1. ***Configuration de VM*** : c'est la configuration traditionnelle d'une machine virtuelle (VM) ou même machine physique telle que l'adressage, l'identité et la configuration de la mise en réseau (par exemple la configuration du NIC) ;
2. ***Configuration de VM spécialisée*** : c'est la configuration des machines virtuelles spécialisées pour les services de communication (tels que vRouter, vFirewall, VPN, fonctions de prévention d'intrusion, contrôle d'accès ...). Cette VM spécialisée peut agir comme un élément de réseau prenant en charge d'autres machines virtuelles à l'intérieur ou à l'extérieur de la même machine physique. Avec ce genre de machine virtuelle, le routage ou les règles des pare-feu peuvent être définis par les utilisateurs ou les applications. Vyatta Core [140, 13] est un exemple de ces machines virtuelles spécialisées ;
3. ***Configuration d'hyperviseur*** : c'est la configuration du réseau qui reliera les NICs des VM à des NICs physiques (par exemple *'bridging'* ou *'bonding'*). Cette

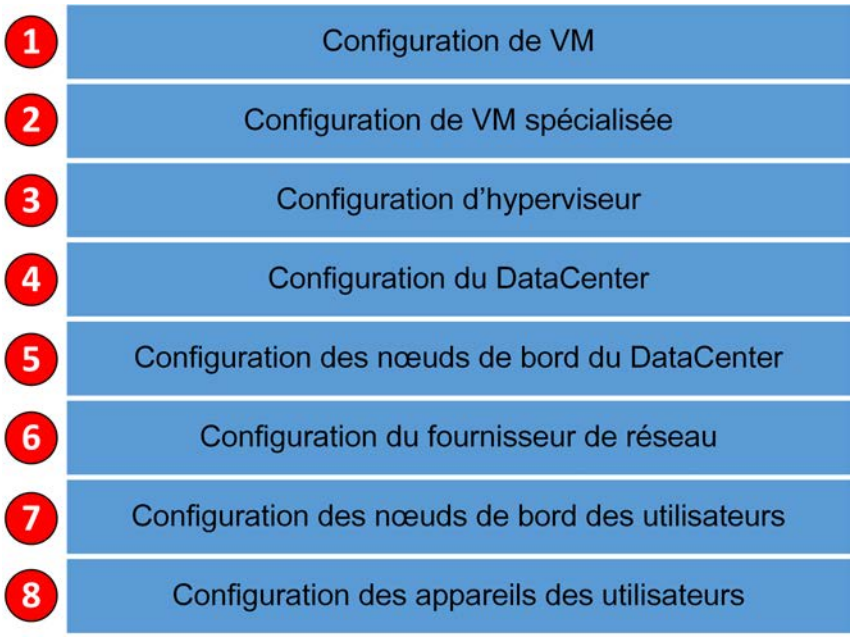

FIGURE 4.3 – *Les niveaux de configuration Cloud Networking*

configuration permet aussi le routage intelligent en utilisant par exemple OpenvSwitch [124, 15] et Crossbow [145, 6] ;

4. ***Configuration du DataCenter*** : c'est la configuration des commutateurs, des routeurs d'un DataCenter et des sites des entreprises ;
5. ***Configuration des nœuds de bord du DataCenter*** : c'est la configuration des éléments nœuds de bord du DataCenter qui seront connectés aux fournisseurs de réseau ;
6. ***Configuration du fournisseur de réseau*** : c'est la configuration des ressources des fournisseurs de réseau (*Acces point*, *Network-edge* et *Core network*) ;
7. ***Configuration des nœuds de bord des utilisateurs*** : c'est la configuration au niveau des nœuds de bord entre l'opérateur réseau et les utilisateurs. Ces nœuds comme ceux de niveau 6 peuvent inclure des fonctions et des services de transformation améliorés tels que la mise en cache ou la compression ;
8. ***Configuration des appareils des utilisateurs*** : c'est la configuration au niveau des appareils des utilisateurs comme les ordinateurs, les tablettes ou les téléphones intelligents.

Pour configurer les ressources associées à tous les niveaux identifiés, une spécification d'interface appropriée du *Cloud Networking* comme Open Cloud Networking Interface (OCNI) est nécessaire étant donné que ni la spécification OCCI ni les langages de description des ressources réseau ne peuvent répondre aux exigences du *Cloud Networking*. Il existe actuellement un grand nombre de langages et spécifications de description des ressources telles que cNIS [150], NDL [17, 148], NM/perfSONAR [9] et NML [8] (voir section 4.3). Si ces langages sont essentiels pour décrire les ressources du réseau et leurs propriétés, il ne

sont pas de type interface Cloud (exemple requête HTTP REST). En effet, ils doivent être combinés avec OCCI et OCNI afin d'automatiser la mise en place du *Cloud Networking* et les environnements de fédération *Cloud Computing*.

Globalement, OCNI se positionne entre les langages de descriptions des réseaux et les interfaces de type Cloud. Avec OCNI, il est possible de décrire toute composante ou infrastructure de réseau et aussi d'allouer et gérer ces ressources avec une interface de type Cloud. Les interfaces *Cloud Computing* et les langages de description de réseau les plus connus seront décrits dans les deux sections qui suivent.

4.3 Interfaces Cloud Computing

Il existe plusieurs spécifications et interfaces de gestion des ressources *Cloud Computing*. Cette section présente quelques-unes des plus connues :

Interface Amazon EC2 : Amazon EC2[4] est un standard de fait. Cette interface offre une grande stabilité et un grand nombre de fonctionnalités bien avancées. L'interface Amazon EC2 est la plus populaire. Sa popularité est due, d'une part du fait qu'il s'agit d'une des premières interfaces *Cloud Computing* et d'autre part à ces fonctionnalités innovantes (exemple d'utilisation des requêtes HTTP REST). Ellen Rubin, fondateur de CloudSwitch (*Cloud Gateway Appliance Vendor*), décrit l'interface Amazon EC2 parfaitement :

> "*If there were an industry standard, Amazon certainly has a strong claim for it. They're the clear leader, with technology second to none. They've made huge contributions to advance cloud computing. Their API is highly proven and widely used, their cloud is highly scalable, and they have by far the biggest traction of any cloud. So full credit to Amazon for leading the way in bringing cloud computing into the mainstream. But it's a big leap from there to saying that Amazon should be the basis for an industry standard.*[38] "

OCCI : Créée par OGF [47], OCCI [45] est une interface et un protocole pour toutes sortes de tâches de gestion. OCCI a été initiée pour créer une API de gestion à distance pour les services *Cloud Computing* de type IaaS. Elle permet le développement des outils d'interopérabilité pour tout type de tâche, y compris le déploiement, la configuration, la mise à l'échelle et la supervision. Elle a depuis évolué pour devenir une interface flexible avec un grand effort sur l'intégration, la portabilité, l'interopérabilité et l'innovation, tout en offrant un haut degré d'extensibilité.

Cloud Infrastructure Management Interface (CIMI) : est une spécification standardisée par Distributed Management Task Force (DMTF)[29]. Elle définit un modèle logique pour la gestion des ressources au sein d'un environnement *Cloud Computing* de type IaaS. La spécification CIMI fait partie des efforts entamés par DMTF dans la standardisation des interactions entre les environnements *Cloud Computing*. Le

but de CIMI est de développer une spécification qui simplifie l'interopérabilité dans la gestion des ressources *Cloud Computing*. Cette interopérabilité concerne plusieurs acteurs comme les fournisseurs de services, les consommateurs et les développeurs.

vCloud API : est l'interface et API Cloud de VMware. vCloud permet l'approvisionnement, la gestion et la supervision des services exécutés dans le *Cloud Computing*. VMware a soumis vCloud à la DMTF en septembre 2009.

CDMI : est une spécification proposée par Storage Networking Industry Association (SNIA)[53] qui spécifie un protocole d'approvisionnement automatique, de gestion et d'accès à un stockage Cloud[23]. CDMI peut aussi être utilisée par les développeurs qui utilisent les solutions de stockage Cloud. La spécification CDMI est devenue un Standard ISO sous le numéro ISO/IEC 17826 :2012.

Topology and Orchestration Specification for Cloud Applications (TOSCA) : est une spécification proposée par Organization for the Advancement of Structured Information Standards (OASIS)[43]. Le rôle principal de TOSCA est d'assurer la portabilité des applications et des services Cloud. TOSCA facilite cet objectif en permettant la description interopérable de services applicatifs et d'infrastructure Cloud, la définition des relations entre les composants d'un service et le comportement opérationnel de ces services.

4.4 Langages de description des réseaux

Il existe plusieurs spécifications et langages pour décrire les composantes et la topologie d'un réseau informatique. Dans cette section, nous présentons les spécifications et les langages les plus connus pour décrire un réseau (les modèles de ces langages sont disponibles dans l'annexe A) :

Network Description Language (NDL) : NDL [17] est un modèle d'information développé par des chercheurs de l'université d'Amsterdam. Le but principal de NDL est de pouvoir décrire des réseaux hétérogènes (en particulier les réseaux à base d'ordinateurs). Il a été utilisé principalement dans la communauté des universitaires, mais aussi adopté comme langage de description dans plusieurs réseaux réels comme celui du GLIF [16] (*Global Lambda Integrated Facility*).

Pour illustrer les relations entre les composants d'un réseau, il existe une modélisation UML détaillée de la spécification NDL (voir figure A.1). Côté langage de spécification, NDL comprend une série de schémas Resource Description Framework (RDF) (RDF décrit les relations sous forme de triplet : sujet, prédicat et objet). Ces schémas RDF décrivent les informations de la topologie du réseau, les technologies déployées, la configuration des ressources et leurs capacités. L'un des objectifs de NDL est d'avoir des ontologies comme description des réseaux. L'autre objectif de NDL est d'avoir la meilleure flexibilité possible dans la description (une sorte de modularité). Pour avoir cette flexibilité, cinq schémas ont été spécifiés :

1. schéma de la topologie *"Topology Schema"* : décris les ressources, les interfaces et leurs inter-connexions d'une façon abstraite.
2. schéma des couches *"Layer Schema"* : décris les propriétés des technologies du réseau.
3. schéma des capacités *"Capability Schema"* : décris les capacités des ressources.
4. schéma des domaines *"Domain Schema"* : décris les domaines administratifs et les services liés à chaque domaine.
5. schéma physique *"Physical Schema"* : décris l'aspect physique des éléments d'un réseau.

L'annexe A.1 illustre un simple exemple de code XML de description d'une ressource réseau selon la spécification NDL.

Common Network Information Service (cNIS) : La spécification cNIS [150] est le résultat d'un travail élaboré au sein du réseau GEANT2 [7]. Le but de cette spécification est d'offrir une interface unique d'une base de données qui contient toutes les informations du réseau. Cette interface peut être utilisée par différents clients du même réseau. Chaque utilisateur demande des informations spécifiques, certains ont besoin de la topologie physique du réseau, d'autres demandent des interfaces réseau ou bien tout simplement une vue abstraite de la connectivité. cNIS surmonte cette divergence et offre une spécification de description du réseau qui peut satisfaire toutes les demandes.

cNIS n'est pas un schéma XML mais plutôt un schéma de base de données relationnelle. Il est formulé sous forme d'un diagramme entité/relation. Les schémas cNIS peuvent être divisés en trois parties : le schéma commun, les schémas techniques et les extensions. Comme exemple, le diagramme de l'annexe A.2 représente le schéma commun de cNIS qui inclut les principales entités d'un réseau (noeuds, interfaces et liens). Une description détaillée de tous les schémas est disponible dans [150].

Network Measurement/perfSONAR (NM/perfSONAR) : perfSONAR [11] est une architecture orientée service destinée à échanger des informations de performances entre plusieurs réseaux. Par exemple, le service TS *"Topology Service"* accepte comme entrée une description de la topologie d'un réseau et retourne comme résultat les performances de ce dernier. Tous les services se communiquent entre eux par le biais de protocoles bien définis dans perfSONAR. Ces protocoles se basent sur une représentation XML des mesures d'un réseau et leurs éléments. Pour développer ces spécifications de description réseau, perfSONAR utilise les résultats des travaux du groupe de travail NM-WG[9]. Le but principal de ce groupe de travail est de fournir une spécification qui décrit les caractéristiques et les mesures que nous pouvons extraire d'un réseau. Selon la spécification NM, toute mesure est représentée par deux éléments :

1. la ressource réseau qui est la partie du réseau à mesurer (par exemple un nœud, un lien, un chemin...). Comme schématisées dans la figure A.3, les ressources réseau sont subdivisées en deux groupes : nœud *"Node"* et chemin *"Path"*.
2. la caractéristique à mesurer (par exemple le débit du trafic). Une structure hiérarchique des caractéristiques est schématisée dans la figure A.4. Certaines de ces caractéristiques s'appliquent sur tous types d'entités réseau et d'autres s'appliquent exclusivement soit sur des noeuds, soit sur des chemins. Une description détaillée de chaque caractéristique est fournie dans le document [111] et un exemple de code XML de la spécification NM est présenté dans l'annexe A.3.

System Description language (SDL) : SDL est un langage XML développé dans le cadre du projet POSITIF (*Policy-based Security Tools and Framework Project*) pour décrire un réseau avec une orientation d'administration et de configuration. La figure A.5 présente le diagramme UML de la structure globale de SDL. D'une façon générale, avec SDL, toutes les ressources d'un réseau peuvent être réparties selon 4 catégories :

— Élément réseau *"NetworkElement"* : une abstraction de toute entité réseau (routeur, commutateur, imprimante, etc.) ;

— Interface : le point d'accès d'un élément réseau ;

— Lien *"Link"* : le lien physique entre deux ou plusieurs nœuds. Ce lien peut être filaire ou sans-fil ;

— Élément logique *"LogicalElement"* : Les caractéristiques et les capacités d'un noeud. Par exemple, avec cette catégorie, il est possible de décrire un service (FTP, HTTP...).

UML est le langage de modélisation adopté par SDL. La figure A.6 est un exemple de diagramme UML des noeuds et des interfaces. Côté spécifications, SDL utilise le "XML Schema". Dans l'annexe A.4 nous présentons un exemple de code XML du langage SDL.

Virtual eXecution Infrastructures Description Language (VXDL) : VXDL [105, 12] est un langage de description des réseaux. C'est une spécification utilisée initialement dans Grid'5000 et dédiée aux applications des grilles de calcul qui ont besoin d'informations concernant le réseau. VXDL a évolué et permet de décrire les ressources virtuelles en plus des ressources réseau classiques. De plus, il offre la possibilité de décrire la topologie d'un réseau virtuel [60].

L'objectif de cette spécification est d'offrir à l'utilisateur la possibilité de décrire :

1. Les ressources d'une façon individuelle ou regroupée ;
2. Les fonctions élémentaires liées aux ressources (par exemple la fonction de stockage des données) ;

3. La topologie d'un réseau ;
4. Les applications et les outils nécessaires pour le bon fonctionnement de chaque composant réseau ;
5. Le calendrier d'exécution de chaque application.

Conceptuellement, VXDL se base sur une modélisation en UML (voir le diagramme de la figure A.7). Côté implémentations, VXDL utilise la technologie XML. Dans l'annexe A.5, nous proposons un simple exemple de fichier XML qui représente une description d'une ressource et d'un lien selon la spécification VXDL.

Network Mark-up Language (NML) : La spécification NML[98] est le résultat du groupe de travail NML-WG (Network Mark-up Language Working Group) [8] au sein de l'organisation OGF. L'objectif de ce groupe de travail est de regrouper et fusionner l'effort de plusieurs travaux (cNIS, NDL, VXDL) afin d'avoir un langage unifié de description des réseaux. Un tel langage peut faciliter l'interopérabilité entre plusieurs projets et surtout entre des réseaux hétérogènes. Un modèle UML de NML est disponible (voir figure A.8) et plusieurs schémas XML ("XML Schema") ont été spécifiés. Avec ces schémas, il est possible de décrire :

1. La topologie d'un réseau d'une façon abstraite.
2. Les propriétés qui sont communes à de multiples technologies de réseau.
3. Un mécanisme d'interopérabilité, pour que d'autres projets puissent combiner leurs spécifications avec celles de NML.

Infrastructure and Network Description Language (INDL) : Le but de INDL [97, 96] est de capturer le concept de virtualisation des infrastructures informatiques et de décrire les capacités de stockage et de calcul des ressources. L'ontologie d'INDL se base sur l'ontologie de Network Mark-up Language (NML). Comme illustré dans la figure A.9, il y a une complémentarité entre INDL et NML dans la description des infrastructures.

Network Node Description Language (NNDL) : NNDL[83] est un langage XML générique conçu pour décrire une configuration complète de chaque nœud du réseau. La description d'un nœud avec NNDL a toutes les informations nécessaires pour les configurations des nœuds : adressage IPv4 et IPv6, paramètres de pare-feu, routage statique et dynamique et la description complète de la configuration des services réseau (voir figure A.10). NNDL fournit une base pour la description d'un grand nombre de scénarios de réseau virtuel.

4.5 Open Cloud Networking Interface

Notre travail consiste à définir une interface, que nous trouvons manquante, qui permet d'étendre le *Cloud Computing* avec tout type de service *Cloud Networking*. Comme illustré dans la figure 4.4, cette interface nommée OCNI satisfait les exigences et les rôles d'une part des interfaces *Cloud Computing* et d'autre part des langages de descriptions des réseaux. Grâce à cette interface, il est possible de décrire, allouer et gérer toute infrastructure ou service *Cloud Networking*, en plus de ceux du *Cloud Computing* classique, pour les offrir à la demande aux utilisateurs.

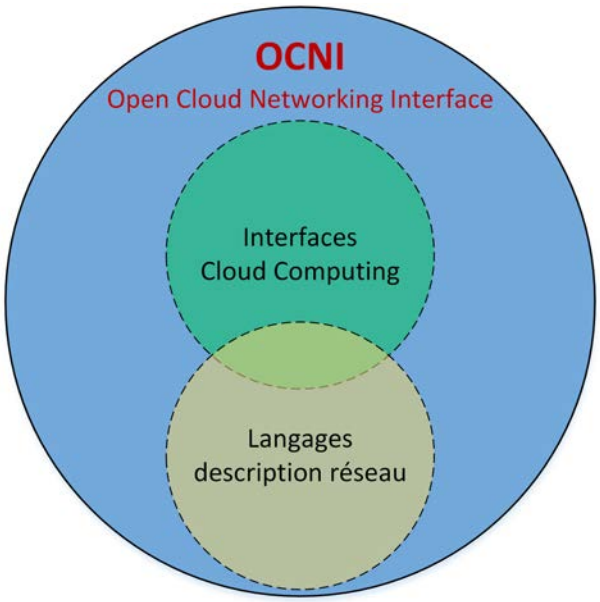

Figure 4.4 – *OCNI - solution intermédiaire entre les interfaces Cloud Computing et les langages de description des réseaux*

Notre objectif est d'introduire une extension du modèle et d'interface d'OCCI pour que la gestion du *Cloud Networking* soit réalisée exactement comme les ressources du *Cloud Computing* traditionnelles. La disponibilité d'une telle interface spécialisée du *Cloud Networking* facilitera le déploiement, la configuration, la gestion et la supervision des éléments de communications en particulier pour la création des Clouds privés et hybrides tout en associant les fournisseurs de réseaux et des services de communication (les *"Cloud Carriers"*). Le choix de la spécification OCCI est liée au fait qu'il s'agit de l'une des premières interfaces Cloud standardisées et surtout qu'elle a fait ses preuves via plusieurs implémentations[46].

L'interface du *Cloud Networking* proposée facilite ainsi la création des Clouds distribués par la configuration des éléments de réseau hétérogènes qui sont sous le contrôle de différents fournisseurs de réseaux. Cette interface et API générique doit être suffisamment souple et extensible pour gérer les multiples technologies de communication, telles que OpenFlow [114] et Open vSwitch [124] ainsi que les technologies traditionnelles comme les VLAN et VPN. En outre, cette interface doit être interopérable avec les interfaces de *Cloud*

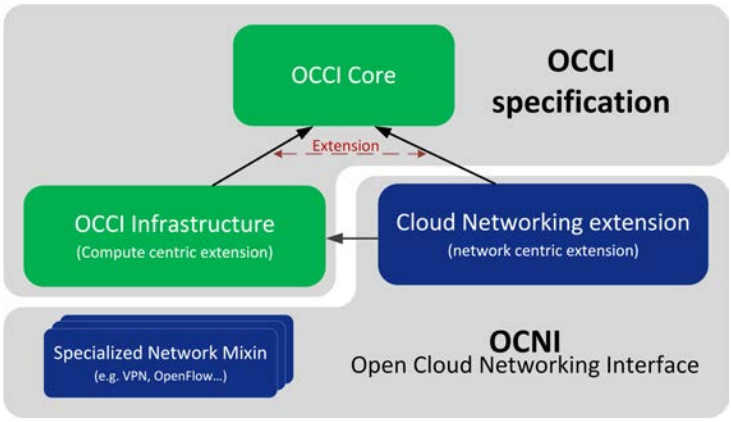

Figure 4.5 – *Architecture fonctionnelle de l'extension OCNI*

Computing existantes.

La principale contribution de notre travail est de définir cette interface générique appelée *Open Cloud Networking Interface* (*OCNI*). Cette interface est basée sur les technologies web standard (HTTP RESTful) pour rester en ligne avec les interfaces de *Cloud Computing* existantes et surtout avec OCCI avec laquelle elle doit être compatible. Par conséquent, nous nous basons sur la spécification d'OCCI [45] et nous l'étendons avec notre proposition OCNI.

Comme le montre la figure 4.5, OCNI est composée principalement de deux éléments. Le premier et majeur composant est une extension centrée sur le *Cloud Networking*. Il ajoute des fonctionnalités pour décrire les réseaux de la même manière que les langues de description des réseaux (présentés précédemment dans la section 4.4). Le deuxième élément est constitué d'un certain nombre de *"mixins"* orientés réseau. Un *mixin* est un mécanisme qui permet d'ajouter des attributs et fonctions, même à l'exécution, à un objet existant du modèle OCCI. Ceci permet l'introduction des objets spécialisés (qui ne sont pas embarqués à la base dans la spécification OCCI), et surtout la mise en réseau des protocoles autres que VLAN. La spécification OCNI vise à étendre le *Cloud Computing* avec des fonctionnalités du *Cloud Networking*. Comme exemple de *mixin* qui offre du *Cloud Networking* nous trouvons les *mixins* des VPNs et OpenFlow (plus de détails de ces *mixins* seront décrits dans la section 4.8).

OCNI spécifie un modèle de données abstrait des services de *Cloud Networking*. Le but principal d'OCNI est de fusionner la gestion des réseaux et des Clouds dans une spécification plus large pour parvenir à la convergence des réseaux et des services de *Cloud Computing*. Par conséquent, l'extension OCNI adhère donc à l'approche originale d'OCCI en adoptant le même cadre de modélisation, mais en spécialisant la spécification pour le domaine du *Cloud Networking*. Ceci se fait par l'ajout de classes axées sur les ressources et les relations des services de communication Cloud. Ces classes représentent des concepts abstraits qui peuvent être utilisés pour encapsuler les concepts de communication comme les réseaux virtuels, les SDNs et les technologies spécifiques (exemple VPN, BGP, OpenFlow, VXLAN, etc).

Les composants du modèle de données, présenté dans la figure 4.6, comprennent ceux liés aux spécifications OCCI et OCNI. Ces éléments sont décrits en termes de rôles, fonctions et relations. Les composants d'*OCCI Core* et *OCCI Infrastructure* sont introduits brièvement et ils sont décrits en détail respectivement dans [86] et [117]. Pour information, les éléments introduits par OCNI utilisent les mêmes concepts qu'OCCI et sont par conséquent conformes et compatibles avec la spécification OCCI originale. Cela permettra une intégration naturelle d'OCNI et les concepts du *Cloud Networking* afin de faciliter la composition dynamique des Clouds (y compris les ressources de *Cloud Networking*). La philosophie d'OCNI est de soutenir et de faciliter la mise en place, à la demande et d'une façon automatisée, des Clouds privés et hybrides dans des environnements de fédération.

Un modèle et diagramme UML de la spécification proposée est illustré dans la figure 4.6. Dans ce qui suit, nous présentons la description des classes d'OCCI et OCNI :

OCCI Core [86] :

- ***Category*** : La classe UML *Category* est la base du mécanisme d'identification de chaque entité.
- ***Kind*** : La classe UML *Kind*, ainsi que la classe *Mixin*, définissent le système de classification des entités.
- ***Mixin*** : La classe UML *Mixin* complète l'élément *Kind* en définissant le système de classification. Un *mixin* représente un mécanisme d'extension, permettant ainsi la définition de nouvelles capacités des ressources pour être ajoutée à la fois à la création et/ou en temps d'exécution.
- ***Entity*** : La classe UML *Entity* (entité) sert à définir le mécanisme de typage, d'une façon abstraite, de chaque composant instancié (les composants sont des ressources et des liens).
- ***Action*** : La classe UML *Action* définit une opération applicable à une ou plusieurs instances d'entité.
- ***Resource*** : La classe UML *Resource* (ressource) hérite de l'élément entité et décrit une ressource concrète qui peut être manipulée. Une ressource est adaptée pour représenter les ressources du monde réel, par exemple machines virtuelles, les réseaux, les services, etc.
- ***Link*** : Une instance de la classe UML *Link* (lien) définit l'association de base entre deux instances de ressources. Une instance de lien indique que deux instances de *Resource* sont liées.

OCCI infrastructure [117] :

* Resource :

- ***Compute*** : La classe UML *Compute* définit la ressource de calcul et traitement d'information (exemple machine physique ou virtuelle).
- ***Network*** : La classe UML *Network* définit la ressource d'interconnexion qui représente une ressource de réseau.

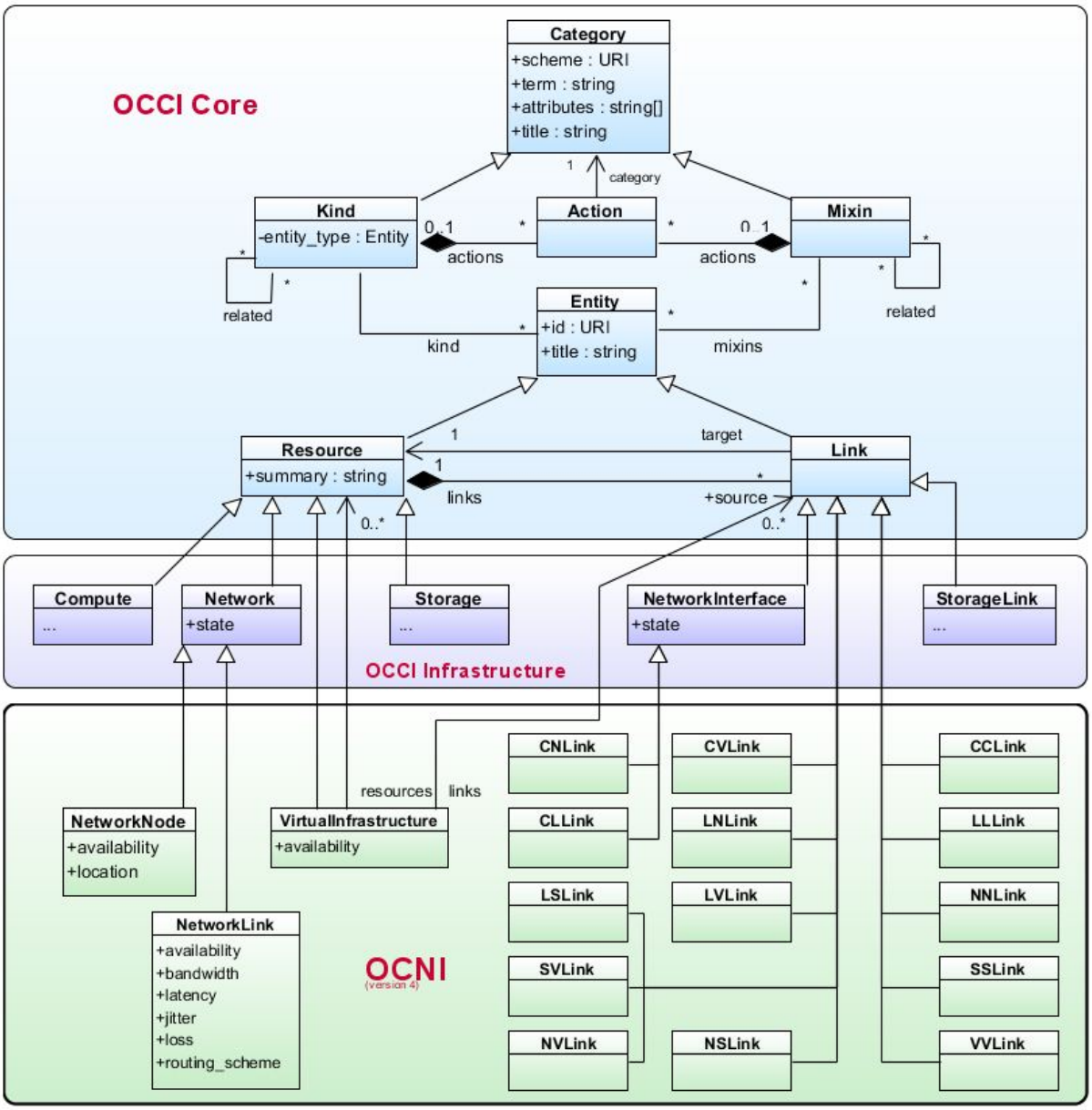

Figure 4.6 – *Diagramme UML d'OCNI*

- ***Storage*** : La classe UML *Storage* définit la ressource de stockage d'information.

* Link :

- ***NetworkInterface*** : La classe UML *NetworkInterface* relie deux instances de ressources l'une de type *'Compute'* et l'autre de type *'Network'*. Elle est complétée par le *mixin IPNetworkInterface* pour avoir l'adressage IP.
- ***StorageLink*** : La classe UML *StorageLink* relie deux instances de ressources l'une de type *'Compute'* et l'autre de type *'Storage'*.

OCNI :

* Resource :

- ***NetworkNode*** : La classe UML *NetworkNode* définit un nœud ou un service de communication du *Cloud Networking*.

- ***NetworkLink*** : La classe UML *NetworkLink* définit un lien ou un service de communication du *Cloud Networking*.
- ***VirtualInfrastructure*** : La classe UML *VirtualInfrastructure* définit une infrastructure virtuelle. Cette infrastructure peut être composée par plusieurs instances de ressources OCCI.

* Link :

- ***CNLink*** : La classe UML *CNLink* relie deux instances de ressources l'une de type *'Compute'* et l'autre de type *'NetworkNode'*.
- ***CLLink*** : La classe UML *CLLink* relie deux instances de ressources l'une de type *'Compute'* et l'autre de type *'NetworkLink'*.
- ***LSLink*** : La classe UML *LSLink* relie deux instances de ressources l'une de type *'NetworkLink'* et l'autre de type *'Storage'*.
- ***SVLink*** : La classe UML *SVLink* relie deux instances de ressources l'une de type *'Storage'* et l'autre de type *'VirtualInfrastructure'*.
- ***NVLink*** : La classe UML *NVLink* relie deux instances de ressources l'une de type *'NetworkNode'* et l'autre de type *'VirtualInfrastructure'*.
- ***CVLink*** : La classe UML *CVLink* relie deux instances de ressources l'une de type *'Compute'* et l'autre de type *'VirtualInfrastructure'*.
- ***LNLink*** : La classe UML *LNLink* relie deux instances de ressources l'une de type *'NetworkLink'* et l'autre de type *'NetworkNode'*.
- ***LVLink*** : La classe UML *LVLink* relie deux instances de ressources l'une de type *'NetworkLink'* et l'autre de type *'VirtualInfrastructure'*.
- ***NSLink*** : La classe UML *NSLink* relie deux instances de ressources l'une de type *'NetworkNode'* et l'autre de type *'Storage'*.
- ***AALink*** : La classe UML *AALink* relie deux instances de ressources de même type "*A*" ("*A*" peut être une instance de ressource de type *Compute*, *NetworkLink*, *NetworkNode*, *VirtualInfrastructure* ou *Storage*).

Dans la figure 4.7, nous décrivons la complémentarité entre OCNI et OCCI. Si les spécifications OCCI Core [86] et OCCI Infrastructure [117] peuvent décrire toutes les ressources de *Cloud Computing* en termes d'attributs et caractéristiques, ils ne couvrent pas suffisamment la connectivité entre les ressources vu qu'ils ne traitent qu'un sous-ensemble de technologie. En effet, la connectivité dans les réseaux et entre les réseaux et les centres de données ont été laissés de côté. Cela limite la description de la connectivité inter-Cloud dans un environnement de fédération *Cloud Computing* incluant plusieurs domaines et fournisseurs.

Grâce à OCNI, le modèle des liens abstrait d'OCCI s'est enrichi pour permettre la description de tout type de topologie de réseaux hétérogènes et aussi la description de toutes les possibilités de liaison entre les ressources. Les figures 4.8 et 4.9, soulignent la

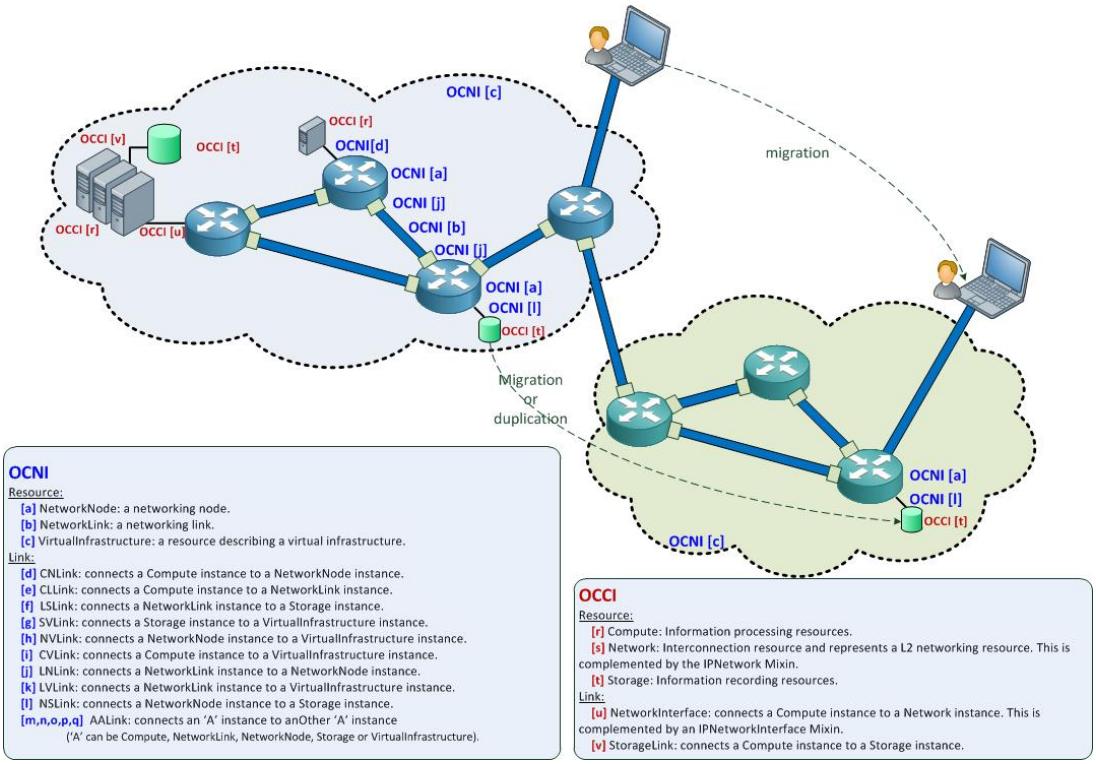

Figure 4.7 – *Complémentarité entre OCNI et OCCI*

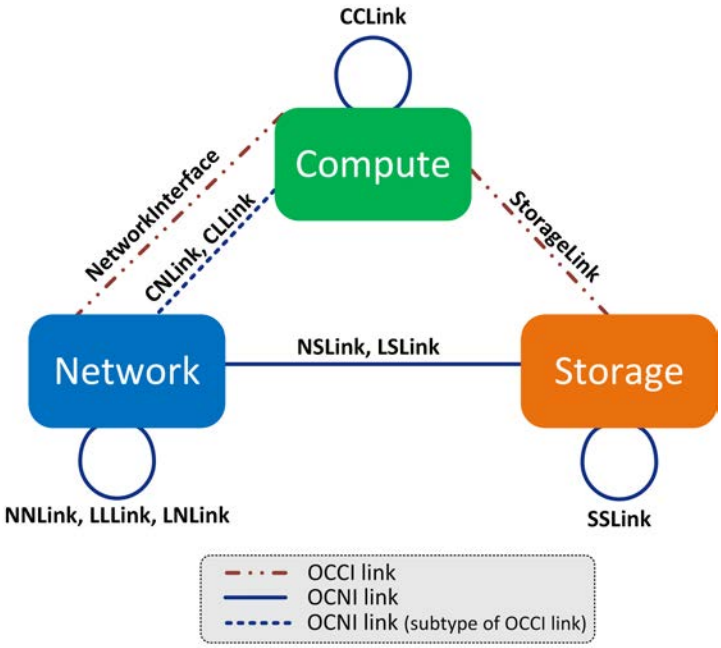

Figure 4.8 – *Liens OCNI vs liens OCCI*

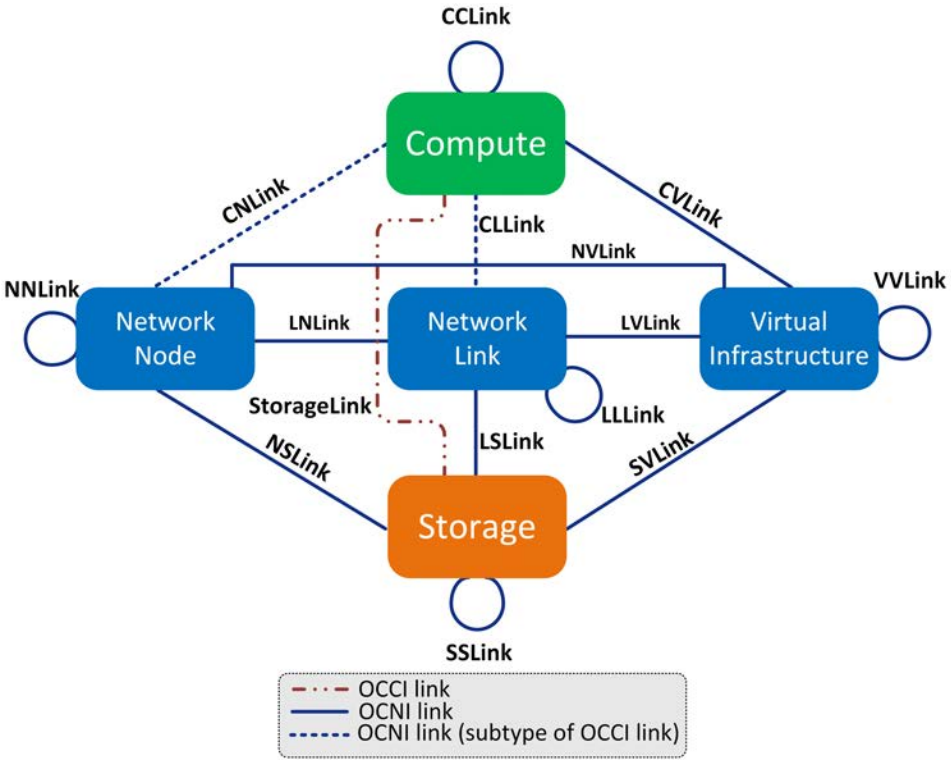

Figure 4.9 – *Liens OCNI vs liens OCCI (schéma détaillé)*

complémentarité des liens OCCI et OCNI et fournissent des détails sur la façon dont le *Cloud Networking* est ajouté par OCNI à la spécification à l'aide de nouveaux éléments de base (*NetworkNode* : nœud *Cloud Networking*, *NetworkLink* : lien *Cloud Networking* et *VirtualInfrastructure* : infrastructure comprenant les ressources *Cloud Networking* et *Cloud Computing*).

OCNI peut décrire n'importe quelle topologie de réseau afin de définir le graphe et service *Cloud Networking*. Il peut aussi agir comme une interface Cloud vers les utilisateurs, afin qu'ils puissent demander des services de connectivité dans un environnement de fédération à partir de multiples infrastructures et fournisseurs. OCNI permet également des interactions avec les contrôleurs du *Cloud Networking* qui gèrent l'établissement des services *Cloud Networking* et plus spécifiquement l'instanciation de certains nœuds et liens pour connecter les ressources de Cloud distribuées.

4.6 Spécification OCNI

La spécification de base d'OCNI sous format JSON est dans l'annexe B.

4.7 Implémentation Open Source d'OCNI

L'implémentation d'OCNI est en Python et elle est disponible en tant que projet Open Source à [50]. Cette implémentation a servi comme base pour la validation des concepts et l'évaluation des performances. Elle est connue sous le nom Python Open Cloud Networking Interface (pyOCNI) et elle est également listée comme une implémentation de référence

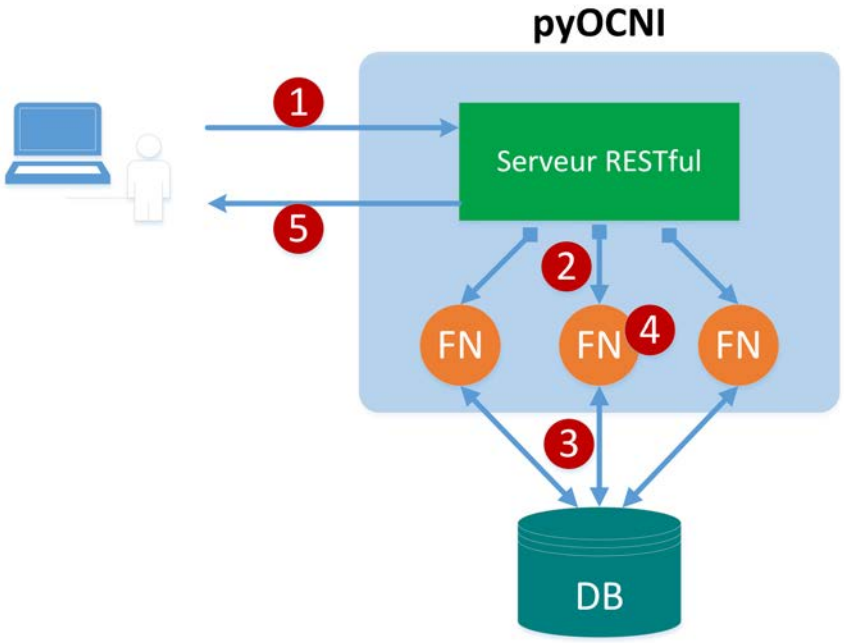

Figure 4.10 – *Processus du fonctionnement abstrait de pyOCNI*

sur le site officiel d'OCCI [51]. pyOCNI a été exploité par plusieurs projets de *Cloud Computing* et *Cloud Networking* (exemple le projet SAIL [52] et CompatibleOne [26]).

Le framework pyOCNI a été déployé et évalué sur une plate-forme expérimentale dédiée, voir annexe D, intégrant de vrais logiciels et commutateurs en plus des ressources *Cloud Computing*. En plus d'être une implémentation d'OCNI, pyOCNI intègre plusieurs nouveaux composants tels que le format JSON et le gestionnaire des catégories. Plus de détails de pyOCNI sont décrits dans la suite de cette section.

4.7.1 Architecture logicielle de pyOCNI

En plus de l'implémentation complète d'OCCI, pyOCNI ajoute l'extension du *Cloud Networking* OCNI et définit la façon dont les mécanismes internes doivent fonctionner. En effet, comme illustré dans la figure 4.10 qui expose un scénario simpliste du fonctionnement de l'implémentation pyOCNI, le processus est composé de cinq étapes :

1 Un utilisateur envoie une requête au serveur pyOCNI.

2 Grâce aux données de la demande et le format d'URL, le serveur RESTful à l'intérieur du serveur pyOCNI décide et choisit la fonction pour canaliser et traiter cette demande.

3 Une fois que la fonction appropriée obtient les données d'entrée, elle communique avec la base de données pour récupérer ce qui est nécessaire et par la suite faire le traitement de la demande.

4 Après le traitement des données, la fonction stocke, si nécessaire, les résultats dans la base de données, puis renvoie une réponse au serveur RESTful.

5 Le serveur RESTful formalise la réponse et l'envoie à l'utilisateur.

pyOCNI est implémenté en Python. Il utilise la base de données NoSQL CouchDB pour le stockage persistant et Eventlet comme un serveur web natif (Web Server Gateway Interface (WSGI)). Eventlet est choisie puisqu'il est évolutif et non bloquant. L'architecture logicielle de pyOCNI, comme illustrée dans la figure 4.11, est composée principalement de six éléments :

- ***Server*** : reçois la demande HTTP. Il est le cœur de pyOCNI et l'orchestrateur des commandes entre les autres composants ;
- ***Serialization*** : fais la sérialisation des données. Actuellement, deux formats de sérialisation sont développés, le HTTP et JSON ;
- ***Registry*** : stocke et gère les données. La base de données CouchDB est gérée par ce composant.
- ***Specification*** : implémente les spécifications OCCI et OCNI. Cet élément est utilisé dans la phase de vérification de conformité des données par rapport à la spécification ;
- ***Backend*** : contiens les technologies des contrôleurs (par exemple VPN-L2, VPN-L3, OpenFlow, OpenStack) ;
- ***Client*** : envoie les requêtes OCNI. Les utilisateurs n'ont pas à utiliser obligatoirement ce composant.

4.7.2 Gestionnaire de catégorie

La spécification OCCI se focalise sur la façon dont les données doivent être acceptées et comment elles devraient être envoyées. L'implémentation classique d'OCCI intègre les classes *'Kind'* et *'mixins'* déjà codées en dur. Il est évident que cette démarche n'est ni souple ni extensible. D'ailleurs, pour ces raisons nous avons ajouté à pyOCNI le potentiel non seulement de traiter les demandes concernant les *'kinds'*/*'mixins'* relatives à l'extension du *Cloud Networking* ci-dessus, mais aussi la possibilité de décrire facilement d'autres extensions. Grâce à pyOCNI, il est possible de décrire, via des requêtes HTTP REST, les classes *'Kind'* et *'mixins'* qui ne sont plus codées en dur.

4.7.3 Sérialisation JSON dans pyOCNI

Aujourd'hui, beaucoup de développeurs préfèrent utiliser JavaScript Object Notation (JSON) comme format de donnée dans plusieurs APIs. En effet, JSON offre beaucoup d'avantages et voici quelques-uns des aspects qui font sa puissance :

Simplicité : JSON est un format simple qui mappe directement sur les structures de données utilisées dans les langages de programmation modernes ;

Extensibilité : JSON est extensible de nature. JSON n'est pas un langage de balisage de document, il n'est pas nécessaire de définir de nouvelles balises ou des attributs pour représenter les données ;

Interopérabilité : JSON est largement utilisé et possède un grand potentiel d'interopérabilité ;

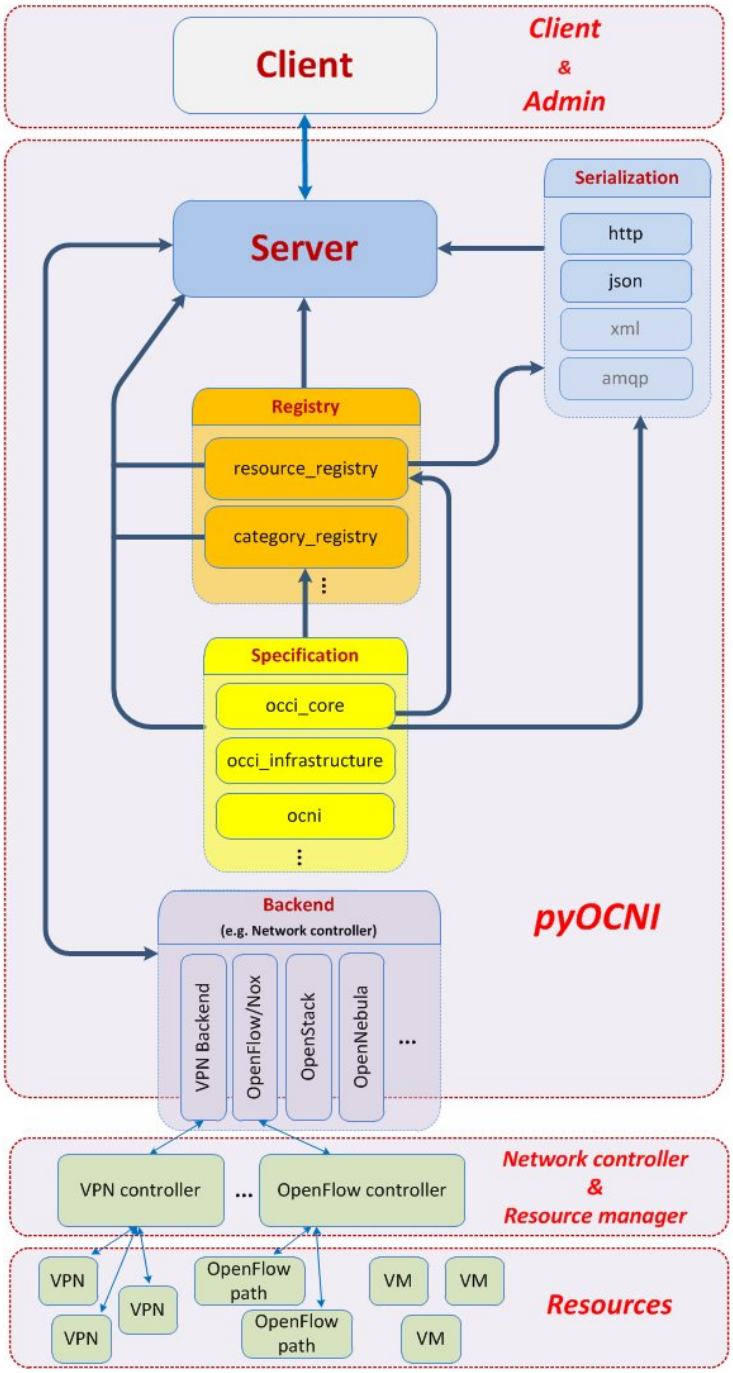

Figure 4.11 – *Architecture logicielle de pyOCNI*

Simple sérialisation : JSON est un format d'échange de données léger. Il est facile pour l'homme à lire et à écrire. De même, c'est facile pour les machines de l'analyser et le générer.

Le format JSON a un grand intérêt dans les interfaces *Cloud Computing*, c'est pourquoi le groupe de travail OCCI a commencé à normaliser le format JSON en plus du format HyperText Transfer Protocol (HTTP). Sachant que le format JSON est beaucoup plus simple et évolutif que le HTTP, il était évident qu'OCNI adopterait non seulement le format HTTP, mais aussi le format JSON. Cela dit, une interaction avec le groupe de travail d'OCCI existe pour pousser notre vision du format JSON et aussi pour mettre à jour OCNI avec les recommandations du standard.

4.7.4 Exemple de requêtes OCNI

Une liste exhaustive de requêtes OCNI est disponible dans l'annexe B et le site web [50]. Ici, par raison de lisibilité, nous fournissons seulement l'exemple d'une instanciation de ressources (requête POST) en utilisant le format JSON. La ressource est un lien *Cloud Networking* qui est défini comme un VPN de couche 3 :

Requête

```
POST networklink HTTP/1.1
Host: localhost:80
Accept: application/ocni+json
{
    "resources": [
        {
            "id": "ResourceID",
            "title": "Cloud Networking Link (L3VPN)",
            "kind": "http://exemple.com/ocni#networklink",
            "mixins": [
                "exemple.com/ocni#L3VPNMixin",
                "exemple.com/ocni#scalabilityMixin"
            ],
            "summary": "L3VPN instance description",
            "attributes": {
                "ocni": {
                    "networklink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active",
                        "bandwidth": "100Mbps",
                        "latency": "100ms",
                        "jitter": "",
                        "loss": "0.01%",
                        "routing_scheme": "unicast"
                    },
                    "L3VPNMixin": {
                        "routing_scheme": "unicast",
                        "infrastructure_request_id": "2222",
                        "customer_id": "0987",
                        "service_type": {
                            "type": "CaaS",
                            "layer": "L3",
                            "class_of_service": "guaranteed"
                        },
                        "service_description": [
                            {
                                "endpoint_id": "0987",
                                "in_bandwidth": "100Mbps",
                                "out_bandwidth": "100Mbps",
                                "max_in_bandwidth": "120Mbps",
                                "max_out_bandwidth": "120Mbps",
```

```
                                "latency": "100ms",
                                "max_latency": "200ms"
                            },
                            {
                                "endpoint_id": "1001",
                                "in_bandwidth": "100Mbps",
                                "out_bandwidth": "100Mbps",
                                "max_in_bandwidth": "120Mbps",
                                "max_out_bandwidth": "120Mbps",
                                "latency": "100ms",
                                "max_latency": "200ms"
                            }
                        ]
                    },
                    "scalability": {
                        "supported": "yes",
                        "setup_time": "10s"
                    }
                }
            }
        }
    ]
}
```

La réponse à la requête POST contient un schéma d'URI qui identifie la ressource allouée.

Réponse

```
HTTP/1.1  200 OK
Server: ocni-server
Content-Type: application/ocni+json
{
    "X-OCCI-Location": [
        "http://localhost:80/networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

4.7.5 Évaluation des performances de pyOCNI

Toutes les expérimentations sont menées dans un environnement réel et d'une façon distribuée (les clients et le serveur pyOCNI ne sont pas co-localisés). Les demandes des clients ont été envoyées à partir d'emplacements distants via Internet à un serveur pyOCNI situé dans un DataCenter. Ce serveur fonctionne dans une machine virtuelle ayant comme caractéristique un processeur à deux vCPU, 2 Go de RAM, 100 Go de disque et un système d'exploitation Linux de distribution Ubuntu. Le serveur pyOCNI a principalement deux types de requêtes. Le premier type concerne la définition et caractérisation des catégories qui décrivent les ressources. Le deuxième type concerne l'instanciation et gestion de ressources. Pour chaque type, les commandes CRUD (Create, Read, Update et Delete) sont définies. Pour une meilleure visibilité, seules les évaluations les plus pertinentes sont décrites.

L'évaluation porte sur les performances du temps de réponse de pyOCNI pour créer des catégories (par exemple, une nouvelle catégorie de ressource ou service *Cloud Networking*), créer des données de ressources, récupérer des informations d'une ressource existante et mettre à jour des informations de ressources. Les durées nécessaires à l'instanciation des ressources *Cloud Networking* ne sont pas prises en compte, car cela relève de la responsabilité des contrôleurs de réseau. Ces durées sont étroitement liées aux technologies de communication à utiliser. Nous sommes intéressés seulement à l'évaluation des perfor-

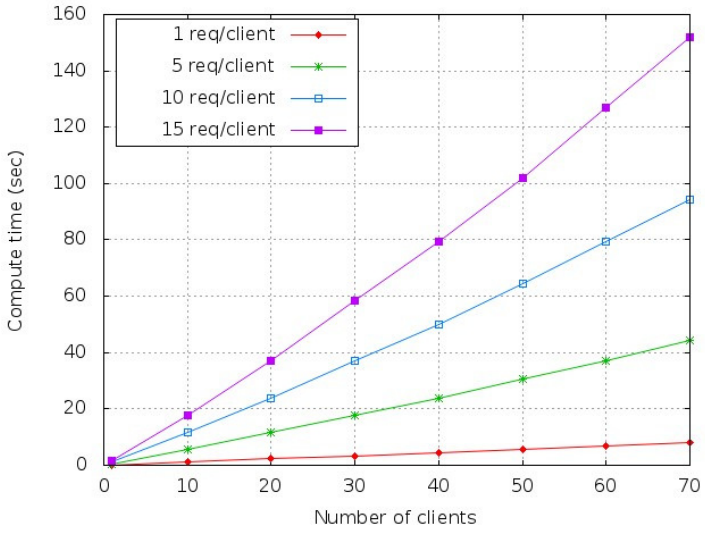

FIGURE 4.12 – *Performance de création d'une description de catégorie*

mances de pyOCNI en agissant comme une couche intermédiaire efficace entre les clients et les technologies de réseau sous-jacents pour la connexion des ressources Cloud distribuées par les fournisseurs de réseau.

Dans la figure 4.12, nous évaluons le temps nécessaire par pyOCNI pour créer une nouvelle catégorie en fonction de clients simultanément impliqués et le nombre de demandes que ces clients génèrent parallèlement. Il faut par exemple 7 secondes pour créer 70 nouvelles catégories envoyées, en parallèle, par 70 clients et 34 secondes pour créer 300 nouvelles catégories envoyées par 30 clients, chacun envoyant 10 requêtes. Les résultats pour une augmentation de charge avec 70 clients et 15 demandes par client sont de l'ordre de quelques minutes (moins de 3 minutes à honorer toutes les demandes reçues simultanément). Cette performance pour la création des catégories est en fait excellente, car la création de catégories se reproduit de temps en temps voire une seule fois. En effet, les requêtes de description des catégories ne sont sollicitées que lorsque de nouvelles technologies de *Cloud Computing* ou *Cloud Networking* sont créées ou modifiées.

La figure 4.13 illustre le temps nécessaire pour l'instanciation d'une nouvelle description de ressource en fonction des clients et de leurs demandes. Le temps nécessaire à la création de 50 nouvelles ressources envoyées par 50 clients est seulement de 11 secondes. Dans le cas des charges élevées, pour l'instanciation de 300 demandes envoyées simultanément par 20 clients (15 demandes par client), le temps est de 68 secondes. Ces résultats montrent que pyOCNI présente de très bonnes performances dans l'instanciation des descriptions de ressources.

La même observation peut être faite pour d'autres indicateurs de performance. La figure 4.14 présente les temps d'exécution pour extraire des informations à partir du serveur pyOCNI. D'après les émulations faites, il faut 7,5 secondes pour obtenir les descriptions des 50 instances de ressources demandées en parallèle par 50 clients. Environ 14 secondes sont nécessaires pour obtenir la description de 100 ressources demandées en parallèle par

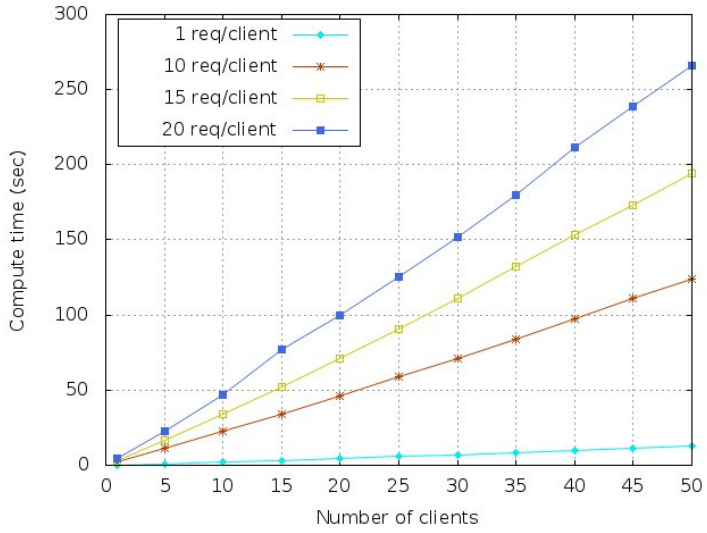

FIGURE 4.13 – *Performance de création d'une description de ressource*

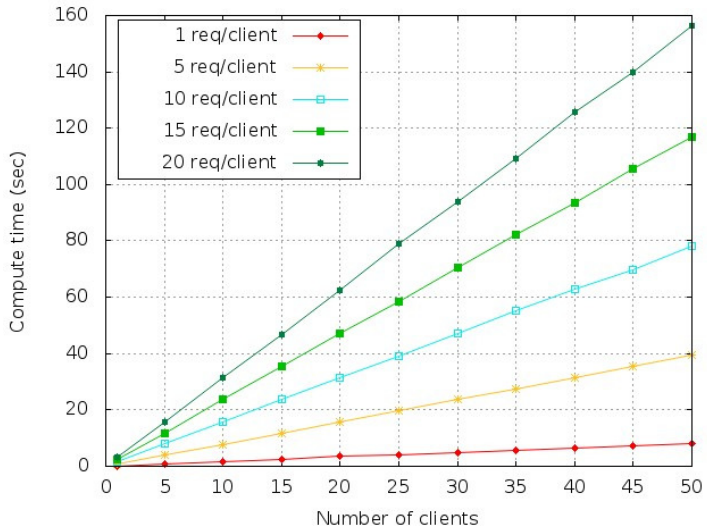

FIGURE 4.14 – *Performance d'extraction d'une description de ressource*

10 clients (avec 10 requêtes envoyées par chaque client). Pour 1000 requêtes parallèles provenant de 50 clients (générant chacun 20 demandes), 155 secondes sont nécessaires.

Les résultats de performance illustrés dans les figure (4.12,4.13,4.14) sont obtenus pour le format JSON. Les résultats obtenus avec le format HTTP se sont révélés être légèrement inférieurs et les différences étaient insignifiantes pour justifier leurs descriptions dans le présent document. En substance, les deux formats HTTP et JSON conduisent à des résultats similaires. Avec pyOCNI, chaque demande est traitée dans un *'thread'* séparé. Malgré que les performances de l'exécution d'un grand nombre de demandes traitées en parallèle restent bonnes, il est cependant possible d'améliorer ces performances. Ceci peut se faire en augmentant la taille de la mémoire de l'environnement d'exécution de Python (pour gérer un plus grand nombre de *'threads'*) ou en utilisant la répartition de charge (*'Load balancing'*) pour gérer plusieurs instances de pyOCNI. L'architecture et

l'implémentation de pyOCNI ont été réalisées en respectant toutes les bonnes pratiques pour le passage à l'échelle.

4.8 Frameworks utilisant OCNI/pyOCNI

OCNI et pyOCNI ont été développés principalement dans le cadre des deux projets SAIL [52] et CompatibleOne [26] où ils ont été utilisés comme une base pour de nombreuses solutions dont nous détaillons dans la suite :

4.8.1 libNetVirt

libNetVirt[146, 147] est une bibliothèque qui permet de gérer des réseaux virtuels d'une façon abstraite et indépendante des technologies sous-jacentes. libNetVirt est une abstraction des réseaux similaires à LibVirt dans l'abstraction des systèmes de virtualisation. Seuls les points de terminaison à l'interconnexion sont nécessaires pour créer le réseau. Parmi les technologies supportées par libNetVirt, il y a OpenFlow et MPLS. Son code source est disponible sur github [39]. libNetVirt utilise OCNI pour décrire les données des requêtes et se base sur pyOCNI comme serveur HTTP REST. Voici un exemple de requête de création d'une ressource réseau avec libNetVirt :

NetworkNode avec libNetVirt (requête POST)

```
POST networknode HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "resources": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#networknode",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NetworkNode for libnetvirt",
            "summary": "This is a NetworkNode resource with libnetvirt mixin",
            "mixins": [
                "http://schemas.ogf.org/occi/ocni#libnetvirt"
            ],
            "attributes": {
                "ocni": {
                    "networknode": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "location": "48.624456,2.443857",
                        "state": "active"
                    },
                    "libnetvirt": {
                        "uuid": "1",
                        "endpoints": [
                            {
                                "uuid": "12",
                                "swid": "1",
                                "port": "2"
                            },
                            {
                                "uuid": "21",
                                "swid": "2",
                                "port": "2",
                                "vlan": "10"
                            },
                            {
```

```
                                    "uuid": "41",
                                    "swid": "4",
                                    "port": "1",
                                    "vlan": "20"
                                }
                            ],
                            "of_controller": "192.168.56.1",
                            "of_controller_port": "2000",
                            "constraint": []
                        }
                    }
                },
                "actions": [],
                "links": []
            }
        ]
}
```

Avec libNetVirt, un travail de modélisation d'Openflow à base d'OCNI a été effectué. La technologie OpenFlow est modélisée via plusieurs *mixins*. L'un de ces *mixins* est intitulé *OpenFlowLink*. Ci-dessous un exemple de requête de création d'un lien OpenFlow :

Requête libNetVirt d'OpenFlow à base d'OCNI

```
POST networklink HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "resources": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#networklink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "OF_Link1",
            "summary": "This is a NetworkLink resource with openflow mixin",
            "mixins": [
                "http://schemas.ogf.org/occi/ocni#OpenFlowLink"
            ],
            "attributes": {
                "ocni": {
                    "networklink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "location": "48.624456,2.443857",
                        "state": "active",
                        "bandwidth": "100Mbps",
                        "latency": "100ms",
                        "jitter": "",
                        "loss": "0.01%",
                        "routing_scheme": "unicast"
                    },
                    "OpenFlowLink": {
                        "infrastructure_id": "1234",
                        "customer_id": "5",
                        "Ether_src": "",
                        "Ether_dst": "",
                        "IPv4_src": "12.0.0.12",
                        "IPv4_dst": "12.0.0.8",
                        "IPv4_proto": [
                            "tcp",
                            "icmp"
                        ],
                        "vlan_id": "",
                        "port_src": "",
                        "port_dst": ""
                    }
                }
            },
            "actions": [],
            "links": [
                1
            ]
        }
    ]
```

```
55 }
```

4.8.2 Contrôleur L3VPN

Dans le cadre du projet SAIL[52], OCNI a été utilisé pour pouvoir interagir avec le contrôleur VPN niveau 3. Pour se faire, cinq *mixins* ont été modélisés pour avoir une requête L3VPN à base d'OCNI telle qu'illustrée dans l'exemple de la section 4.7.4. Les attributs des mixins sont illustrés dans les tableaux suivants :

Attribut	Type	Description
endpoint_id	String	Unique identifiant du *endpoint*
in_bandwidth	String	La capacité de transfert des données du trafic entrant à l'*endpoint*
out_bandwidth	String	La capacité de transfert des données du trafic sortant à l'*endpoint*
max_in_bandwidth	String	La capacité maximale de transfert des données du trafic entrant à l'*endpoint*
max_out_bandwidth	String	La capacité maximale de transfert des données du trafic sortant à l'*endpoint*
latency	String	Valeur actuelle de la latence du *endpoint*
max_latency	String	Valeur maximale de la latence du *endpoint*

Table 4.1 – *Attributs du mixin L3VPN Service Description*

Attribut	Type	Description
type	String	Le type du service (exemple L3VPN)
layer	String	Le niveau du service (exemple *Layer 3*)
class_of_service	String	La classe du service. Exemple CaaS *(Communication as a Service)*

Table 4.2 – *Attributs du mixin L3VPN Service Type*

Attribut	Type	Description
supported	Enum {yes, no}	La capacité de la ressource à supporter l'élasticité
reconfiguration_time	String	Temps nécessaire pour la reconfiguration de la ressource

Table 4.3 – *Attributs du mixin L3VPN Elasticity*

Attribut	Type	Description
supported	Enum {yes, no}	La capacité de la ressource à supporter l'élasticité
setup_time	String	Temps nécessaire pour le déploiement des changements

Table 4.4 – *Attributs du mixin L3VPN Scalability*

Attribut	Type	Description
request_id	UUID	L'identifiant de la requête
customer_id	UUID	L'identifiant de l'utilisateur
service_type	Mixin (Service_type)	Le type du service
service_description	Mixin (Service_description)	La description du service
elasticity	Mixin (Elasticity)	L'élasticité du service
scalability	Mixin (Scalability)	La mise à l'échelle du service

Table 4.5 – *Attributs du mixin L3VPN*

4.8.3 Contrôleur Cloud Computing (OpenStack et OpenNebula)

OCNI se base sur la spécification d'OCCI. De ce fait, la gestion des ressources *Cloud Computing* classiques reste réalisable avec pyOCNI tout en utilisant le format JSON. Afin d'uniformiser les interfaces de gestion de tous les contrôleurs *Cloud Computing* et *Cloud Networking* du démonstrateur proposé par le projet SAIL, les développements des interfaces utilisant pyOCNI pour OpenStack et OpenNebula ont été proposés par les partenaires. Pour OpenStack, c'est *Ericsson* qui c'est chargé de le développer et pour OpenNebula, c'était *PT Inovaçao*. Un exemple de démonstrateur du projet SAIL utilisant

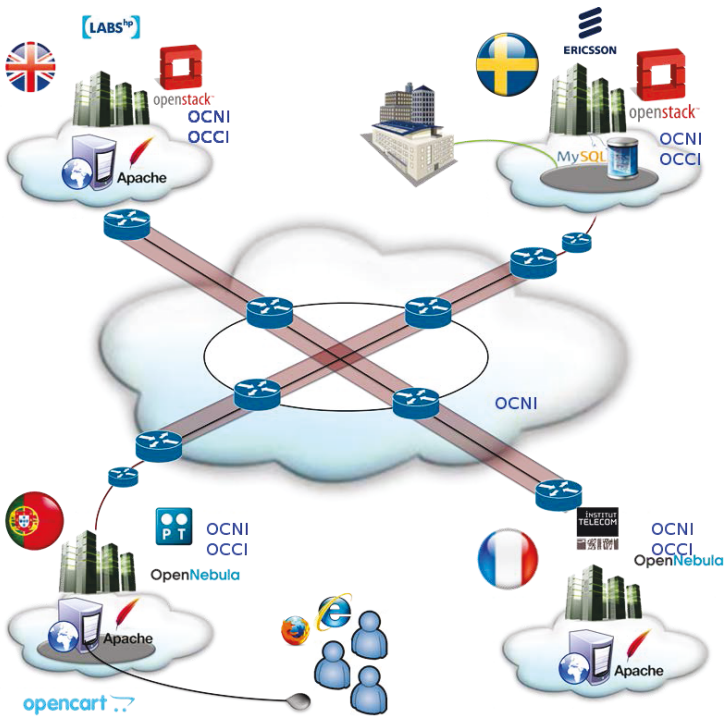

FIGURE 4.15 – *Démonstrateur Cloud Networking de SAIL[151]*

OCNI est illustré dans la figure 4.15[125]. Ce démonstrateur sert à déployer une application web sur une infrastructure Cloud distribuée dont l'allocation et la gestion des ressources s'effectuent via OCNI (les ressources sont de type *Compute*, *Storage* et *Network*).

4.8.4 CONTRÔLEUR D'ÉCHANGE DE MESSAGE

OCNI est à la fois un langage de descriptions et une interface WEB HTTP REST. De ce fait, il est possible de n'utiliser que la partie de description des ressources avec d'autres technologies d'échange d'information. C'est le cas, par exemple, pour le courtage Cloud à base d'échange de message entre plusieurs domaines et fournisseurs. Ce cas sera étudié en détail dans le chapitre 5.

4.8.5 CONTRÔLEUR DE PASSERELLE CLOUD NETWORKING

Un contrôleur Cloud Networking Gateway (CNG) [24, 25] assure la connectivité entre des ressources acquises auprès de fournisseurs de Cloud distribués et masque l'hétérogénéité des technologies d'interconnexion. L'idée principale du contrôleur CNG est de déployer, configurer et gérer des passerelles virtuelles (principalement une passerelle par fournisseur). Le contrôleur CNG utilise pyOCNI comme serveur OCCI pour les requêtes HTTP REST. Ci-dessous, un exemple de requête d'une passerelle *Cloud Networking* :

Requête intercng

```
1 POST intercng HTTP/1.1
2 Host: localhost:80
3 Accept: application/occi+json
4 User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
5 {
6     "resources": [
```

```
7           {
8               "kind": "http://schemas.ogf.org/occi/infrastructure#intercng",
9               "attributes": {
10                  "occi": {
11                      "intercng": {
12                          "name": "First Network Configuration Example",
13                          "publicaddrCNGsrc": "1.1.1.1",
14                          "privateaddrCNGsrc": "192.168.1.1",
15                          "privateNetToCNGsrc": "192.168.1.0/24",
16                          "ethernameCNGsrc": "eth0",
17                          "providerCNGsrc": "site1",
18                          "publicaddrCNGdst": "2.2.2.2",
19                          "privateaddrCNGdst": "10.10.10.1",
20                          "privateNetToCNGdst": "10.10.10.0/24",
21                          "ethernameCNGdst": "eth0",
22                          "providerCNGdst": "site2",
23                          "linkType": "openvpn",
24                          "reusable": "1",
25                          "account": "userTest"
26                      }
27                  }
28              }
29          }
30      ]
31  }
```

4.9 Conclusion

Nous avons présenté dans ce chapitre notre contribution de l'interface Cloud - OCNI. Avant la description détaillée d'OCNI, nous avons introduit tout d'abord le contexte et l'objectif de ce genre d'interface, suivi par la description des interfaces Cloud existantes et les langages de description réseau utilisés. Par la suite, nous avons défini l'interface OCNI et nous avons détaillé la spécification proposée. Enfin, nous avons présenté notre implémentation Open Source d'OCNI et quelques frameworks qui l'ont utilisée et adoptée. Tous les travaux d'OCNI détaillés dans ce chapitre ont été adoptés et validés dans le cadre du projet européen SAIL et projet français CompatibleOne.

Comme présenté dans le chapitre 3, deux composants sont essentiels dans les architectures de fédération Cloud. Le premier est l'interface Cloud générique OCNI qui a été fait l'objet de ce chapitre. Dans le chapitre suivant, nous détaillons notre proposition du deuxième composant essentiel des architectures de fédération Cloud qui est le système d'échange de message.

Chapitre 5

Courtage Cloud à base de messages

Sommaire

5.1 Introduction

La fédération Cloud se base principalement sur un compostant essentiel qui est l'interface Cloud générique. Cette interface, nommée OCNI, a été détaillée dans le chapitre précédent. Cette interface est d'une grande importance dans les opérations d'allocation et de gestion des ressources. Cependant, elle est insuffisante pour les actions de coordination dans les environnements de fédération. Pour cela et comme expliqué pour les deux architectures de fédérations Cloud présentées dans le chapitre 3, un deuxième compostant est primordial pour assurer la fédération. Ce composent est le système d'échange de messages de type Cloud.

Tout au long de ce chapitre, nous présenterons les travaux de définition d'une solution d'échange et de courtage Cloud à base de messages. Cette solution est bien évidemment spécialisée pour les environnements Cloud. Comme illustré dans la figure 5.1, les travaux de cette solution font l'objet du dernier chapitre de cette thèse.

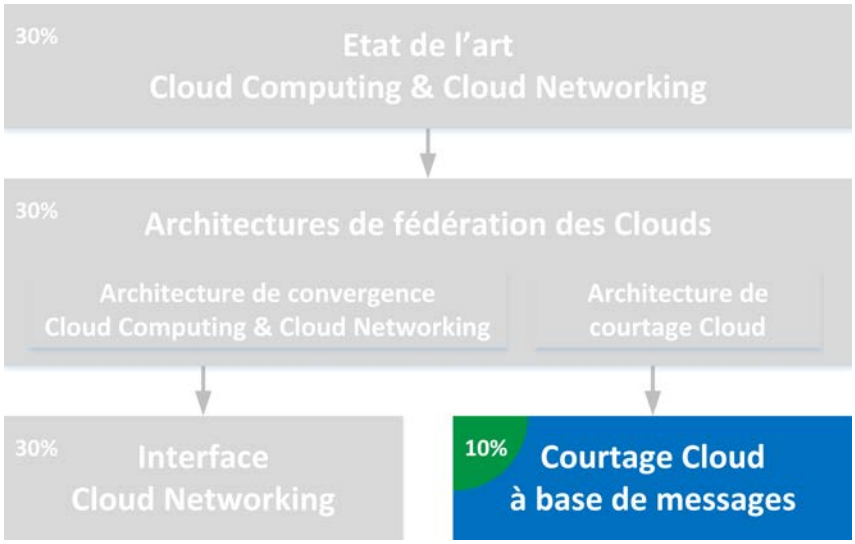

Figure 5.1 – *Structure du document - Chapitre de courtage Cloud à base de messages*

Dans la suite de ce chapitre, nous commençons dans la section 5.2 par introduire le concept général du MQS (*Message Queuing Service*). Durant cette section, nous définissons le MQS et nous présentons ses avantages, ses caractéristiques, ses *patterns*, ses protocoles et ses solutions existantes. Par la suite, nous détaillons dans la section 5.3 notre solution appelée CMBS (*Cloud Message Brokering Service*). Nous définissons en premier lieu le CMBS. En second lieu, nous détaillons le CMBS en décrivant ses requêtes, ses modèles de messages et ses composants. Ensuite, nous décrivons dans la section 5.4 notre implémentation de CMBS et ses cas d'utilisation. Enfin, nous concluons dans la section 5.5.

5.2 Message Queuing Service - MQS

5.2.1 Définition

Un **MQS** (*Message Queuing Service*) est un service dont le rôle principal est d'envoyer et recevoir des messages entre composants et systèmes distribués. Il fait partie des frameworks de type MOM (*Message Oriented Middleware*). Un MQS peut être soit le composant central d'une solution logiciel pour lier tous ses éléments, soit un moyen d'interaction et d'échange entre plusieurs solutions indépendantes. Dans les deux cas, tous les éléments peuvent être colocalisés ou distribués.

Dans ce qui suit, et comme illustré dans la figure 5.2, nous décrivons les avantages, les caractéristiques, les patterns et les protocoles les plus connus d'un MQS.

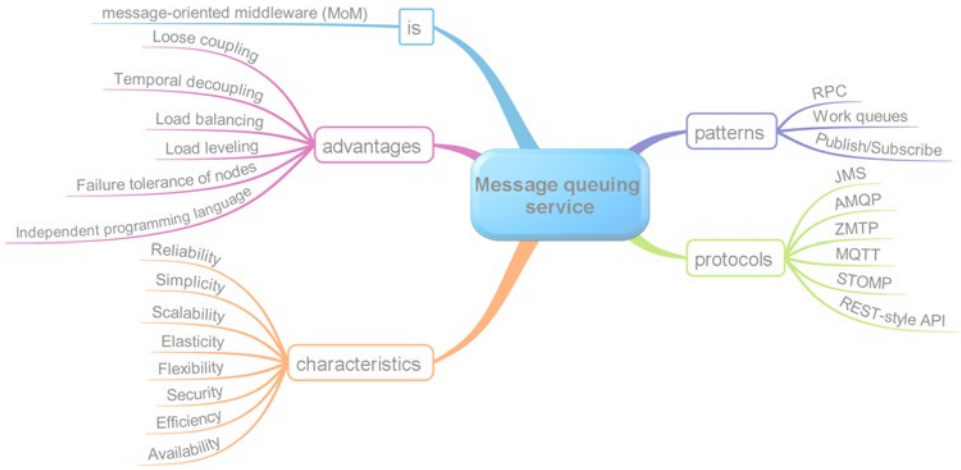

Figure 5.2 – *MQS - Message Queuing Service*

5.2.2 Avantages

Parmi les avantages du **MQS**, nous citons :

- **Couplage faible** (*Loose coupling*) : Dans une architecture à base de MQS, les composants, qui envoient et reçoivent des messages à travers le MQS, ne sont pas liés et chacun d'eux peut suivre son propre cycle de développement et fonctionnement. Par exemple, un composant peut être mis à jour sans affecter le fonctionnement et la stabilité des autres composants. De plus, la topologie du MQS peut évoluer sans affecter les composants logiciels qui l'utilisent ;
- **Découplage temporel** (*Temporal decoupling*) : Avec le découplage temporel, il n'y a aucune obligation d'avoir l'émetteur et le récepteur des messages connectés en même temps. En effet, le MQS peut, si nécessaire, enregistrer les messages en attente de leurs livraisons. Le découplage temporel résulte de l'utilisation d'un modèle d'échange de messages asynchrone (*'Asynchronous Messaging Pattern'*) ;
- **Équilibrage de charge** (*Load balancing*) : Dans le cas d'une augmentation de charge liée à un nombre élevé de messages, le MQS peut équilibrer cette charge en

répartissant efficacement le traitement des messages entre les consommateurs de message. De plus, en ajoutant un nouveau consommateur, ce dernier pourra être pris en charge par le MQS en temps réel ;

- **Nivellement de la charge** (*Load leveling*) : La charge gérée par chaque composant peut varier au fil du temps. D'un point de vue optimisation de l'utilisation de l'infrastructure physique, il est plus judicieux de faire le calcul sur la charge moyenne que sur la charge de pointe. Au moment des charges de pointe maximales, le MQS devra gérer cette situation en adaptant la taille des files d'attente et sa topologie. Dans ce cas de figure, il y a une diminution des coûts d'investissement hardware tout en garantissant le bon fonctionnement du système ;
- **Tolérance aux pannes** (*Failure tolerance*) : Chaque service peut être un producteur ou un consommateur de message ou un composant du MQS lui-même. Vu le faible couplage entre les composants, chacun d'eux peut être remplacé ou réparé dans le cas d'une panne. D'où l'avantage de la tolérance aux pannes dans une architecture à base de MQS ;
- **Indépendance du langage de programmation** (*Independent programming language*) : Un MQS n'a aucune exigence concernant le langage de programmation des composants. Les composants peuvent être développés en utilisant des langages de programmation différents, voire incompatibles. Cela n'empêche pas le bon fonctionnement du système global.

5.2.3 Caractéristiques

Un système MQS doit satisfaire un certain nombre de caractéristiques :

- **Fiabilité** (*Reliability*) : Même si le consommateur des messages (serveur) n'est pas en ligne, le producteur de message (client) devrait continuer à fonctionner sans blocage. Les messages à envoyer au serveur seront stockés dans une file et envoyés quand ce dernier sera en ligne.
- **Simplicité** (*Simplicity*) : Le système MQS doit être simple à mettre en place, à configurer et à utiliser.
- **Évolutivité** (*Scalability*) : Un MQS doit être évolutif par rapport aux besoins des composants d'un système et aussi face à la montée en charge.
- **Élasticité** (*Elasticity*) : En plus de l'évolutivité du système, un MQS doit s'adapter rapidement à une diminution de la charge afin d'optimiser les ressources physiques utilisées.
- **Flexibilité** (*Flexibility*) : Le MQS doit être flexible pour se métamorphoser, en changeant de topologie par exemple, et cela rapidement et facilement pour satisfaire les besoins applicatifs. De même, la flexibilité concerne la possibilité d'interagir avec des systèmes hétérogènes en assurant une interopérabilité totale.

- **Sécurité** (*Security*) : Un MQS doit être sécurisé et capable de garantir le bon acheminement des messages tout en respectant toutes les règles de routage définies.
- **Efficacité** (*Efficiency*) : Un système MQS doit être efficient dans sa réception des messages et surtout dans leur acheminement aux destinataires finaux.
- **Disponibilité** (*Availability*) : À tout instant, un MQS doit être disponible pour la réception et l'acheminement des messages indépendamment du niveau de la charge.

5.2.4 Modèles

L'échange de messages peut suivre plusieurs modèles *'Pattern'* selon le besoin de l'application qui utilise le MQS. Parmi les modèles les plus utilisés, nous citons :

- *RPC (Remote Procedure Call)* : Avec RPC, le producteur de message envoie un message à un consommateur de message et attend un retour de réponse ;
- *Work Queue* : Avec les Work Queue, un message peut être considéré comme une tâche à exécuter. Cette tâche peut s'exécuter par un ou plusieurs serveurs avec une stratégie d'ordonnancement bien définie ;
- *Publish/Subscribe* (Pub/Sub) : Le *Publish/Subscribe* sert à distribuer des messages à un ensemble de consommateurs. Les producteurs et les consommateurs de messages n'ont pas besoin de se connaître. Chaque message publié est identifié par des clés (*Keys* ou *Tags*). Les consommateurs de messages peuvent s'inscrire à une liste de clés et peuvent recevoir les messages publiés liés à ces clés.

5.2.5 Protocoles

Pour structurer et standardiser les MQS, plusieurs protocoles ont fait leurs apparitions. Parmi les plus connus, nous citons :

- JMS (*Java Message Service*) ;
- AMQP (*Advanced Message Queuing Protocol*) ;
- ZMTP (*ZeroMQ Message Transport Protocol*) ;
- MQTT (*Message Queuing Telemetry Transport*) ;
- STOMP (*Streaming Text Oriented Message Protocol*) ;
- API de type REST.

5.2.6 Solutions d'échange de message

Il existe sur le marché une panoplie de solutions de MQS et leur nombre ne cesse d'augmenter. Cette croissance est liée à leurs importances dans le bon fonctionnement des frameworks distribués et en particulier dans les environnements Cloud. Une liste de ces solutions est illustrée dans la figure 5.3 avec une classification des solutions selon quatre critères :

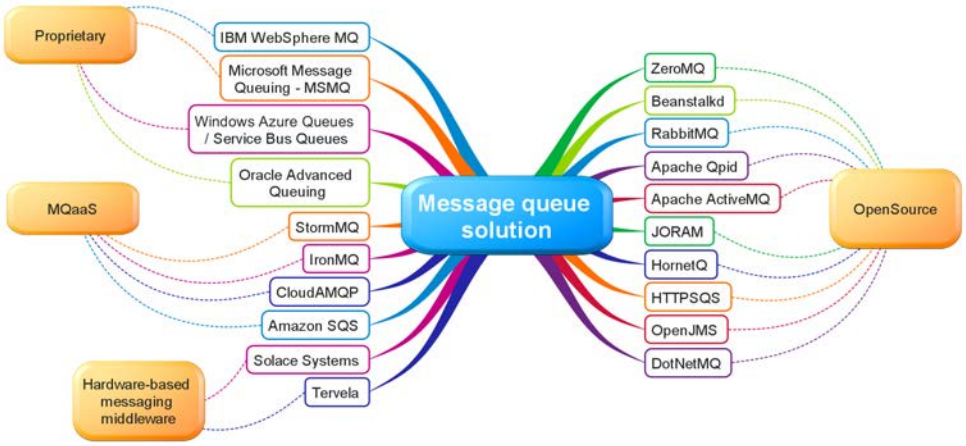

FIGURE 5.3 – *Les solutions de MQS*

- Solution propriétaire (*Proprietary*) : Des solutions propriétaires proposées par les gros acteurs comme Oracle, Microsoft ou IBM ;
- Solution MQaaS (*Message Queuing as a Service*) : Des solutions à utiliser à la demande d'une façon similaire au fonctionnement des solutions Cloud public ;
- Solution physique (*Hardware-based messaging middleware*) : Un matériel physique dont le rôle principal est le traitement et acheminement des messages avec des garanties de performance et de sécurité ;
- Solution OpenSource : Des solutions qui sont développées par des entreprises ou des communautés et qui sont en Open Source.

5.3 CLOUD MESSAGE BROKERING SERVICE - CMBS

5.3.1 DÉFINITION DE CMBS

Cloud Message Brokering Service (CMBS) est un service d'échange et de courtage Cloud à base de messages entre plusieurs domaines, fournisseurs ou entités dans le Cloud. En tant que service Cloud, CMBS doit également satisfaire les caractéristiques essentielles du Cloud. Comme mentionné dans [143], le modèle d'échange de message qui convient le mieux aux caractéristiques du Cloud est l'architecture orientée événements (EDA - *Event-Driven Architecture*). De plus, le Middleware Orienté Message (MoM - *Message Oriented Middeleware*) est le modèle de type EDA qui est le plus utilisé et le plus efficace. De ce fait, CMBS adopte l'architecture EDA et le modèle MOM. Pour le modèle MOM, il y a trois types distincts qui peuvent être utilisés : le type applicatif (exemple JMS), les protocoles couche de transport avec un courtier (exemple AMQP [22]) et les protocoles couche de transport sans un courtier (par exemple ZeroMQ [55]). Le principal objectif de CMBS concerne les informations échangées entre les entités d'un Cloud et non pas les fonctions de messagerie natives (comme *Publish/Subscribe*, diffusion, etc.). De ce fait, CMBS est construit au-dessus des cadres de messagerie existants. Pour le prototypage, nous avons choisi ZeroMQ et RabbitMQ/AMQP afin de couvrir la majorité des cas d'utilisation.

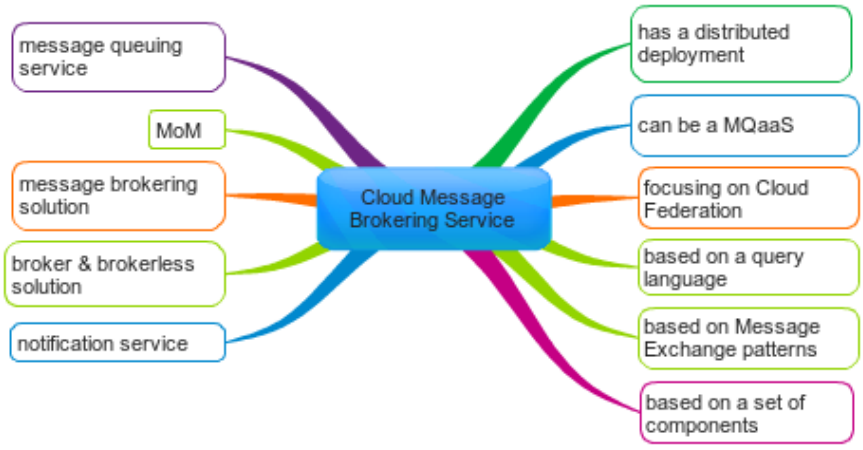

FIGURE 5.4 – *Éléments décrivant CMBS*

Comme illustré dans la figure 5.4, le CMBS peut être présenté d'une façon concise par un ensemble d'éléments décrits comme suit :

- Service d'échange de messages : CMBS est un service de type MQS *Message Queuing Service* ;
- MoM : CMBS adopte le modéle MoM (*Message Oriented Middeleware*) ;
- Solution de courtage de messages : CMBS est une solution de courtage de messages ;
- Solution avec ou sans Courtier principal : CMBS peut être déployé avec un courtier central ou sans courtier principal (*Broker & Brokerless Solution*) ;
- Service de notification : CMBS peut jouer le rôle d'un système de notification ;
- Déploiement distribué : CMBS peut être déployé d'une façon complètement distribuée sans aucun point central de type SPOF ;
- Peut être un MQaaS : CMBS peut être déployé et utilisé comme un MQaaS (*Message Queuing as a Service*) ;
- Se focalise sur le courtage Cloud : CMBS se focalise principalement sur le courtage Cloud malgré la possibilité de son utilisation dans d'autres contextes ;
- Basé sur un langage de requêtes : CMBS se base sur un langage de requêtes qui est bien défini (voir section 5.3.2) ;
- Basé sur un modèle d'échange de messages : CMBS se base sur un modèle d'échange de messages qui est bien défini (voir section 5.3.3) ;
- Basé sur un ensemble de composants : CMBS se base sur un ensemble de composants architectural et sur une liste de combinaison de ces composants qui sont bien définis (voir section 5.3.4).

5.3.2 Les requêtes de CMBS

Les requêtes de contrôle dans CMBS sont structurées selon un format bien défini. Ce format, illustré dans la figure 5.5, est composé de quatre groupes de requêtes :

- ***Provider*** : Ce groupe de requêtes englobe toutes les opérations liées à la gestion des entités qui envoient ou reçoivent des messages ;
- ***Topic*** : Ce groupe de requêtes englobe toutes les opérations liées à la définition et la gestion des sujets (*Topics*). Un *Topic* décrit une famille logique d'information. Grâce au *Topics*, il est possible d'avoir un échange de type un-à-plusieurs (*one-to-many*) afin d'établir des architectures d'échange de messages en mode *Publish/Subscribe*.
- ***Queue*** : Ce groupe de requêtes englobe toutes les opérations liées à la gestion des files d'attente dans CMBS ;
- ***Message*** : Ce groupe de requêtes englobe toutes les opérations liées à la création, envoi ou réception des messages.

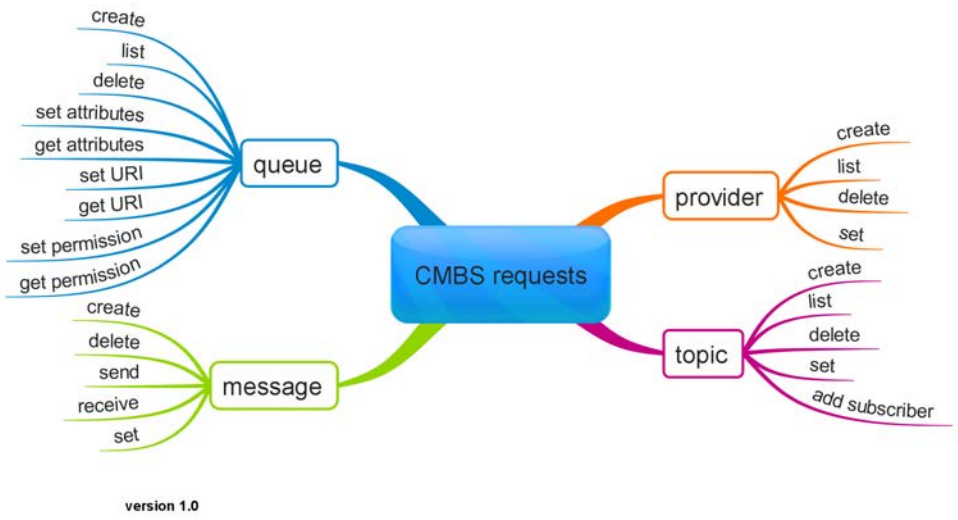

Figure 5.5 – *Format des requêtes CMBS*

Chaque groupe de requêtes inclut une liste de commandes REST bien définie. Dans ce qui suit les détails de ces commandes.

Le groupe de requêtes *Provider* a quatre commandes :

* create : créer un ou plusieurs *Providers* ;
* list : récupérer la liste des *Providers* ;
* delete : supprimer un ou plusieurs *Providers* ;
* set : modifier un ou plusieurs *Providers*.

Le groupe de requêtes *Topic* a cinq commandes :

* create : créer un ou plusieurs *Topics* ;

* list : récupérer la liste des *Topics* ;
* delete : supprimer un ou plusieurs *Topics* ;
* set : modifier un ou plusieurs *Topics* ;
* add subscriber : ajouter un *Provider* comme souscripteur d'un *Topic*.

Le groupe de requêtes *Queue* a neuf commandes :

* create : créer une ou plusieurs *Queues* ;
* list : récupérer la liste des *Queues* ;
* delete : supprimer une ou plusieurs *Queues* ;
* set attributes : fixer les attributs décrivant une *Queue* ;
* get attributes : récupérer la liste et contenu des attributs d'une *Queue* ;
* set URI : fixer l'URI d'une ou plusieurs *Queues* ;
* get URI : récupéré l'URI d'une ou plusieurs *Queues* ;
* set permission : fixer les permissions d'accès d'une ou plusieurs *Queues* ;
* get permission : récupérer les permissions d'accès d'une ou plusieurs *Queues*.

Le groupe de requêtes *Message* a cinq commandes :

* create : créer un ou plusieurs *Messages* ;
* delete : supprimer un ou plusieurs *Messages* ;
* send : envoyer un ou plusieurs *Messages* ;
* receive : recevoir un ou plusieurs *Messages* ;
* set : fixer le contenu d'un ou plusieurs *Messages*.

5.3.3 Modèle des messages CMBS

L'échange d'informations entre les fournisseurs peut suivre plusieurs modèles et peut se produire de différentes façons. Parmi ces modèles, il y a l'envoi d'information à un ou plusieurs fournisseurs (*Provider*) bien définis ou bien l'envoi des informations liées à un thème spécifique (*Topic*). Pour satisfaire les exigences des différents modèles d'échange d'informations, CMBS est basée sur un modèle d'échange de messages bien spécifié. Ce modèle, illustré dans la figure 5.6, est composé de cinq couches (*layers*) :

- ***layer 1*** - Espace de découverte des fournisseurs (*Cloud provider discovery space*) : le rôle de cette couche est tout simplement d'échanger des informations sur le type et les caractéristiques des fournisseurs *Provider*. Chaque *Provider* qui veut échanger des messages devrait s'annoncer et fournir ses caractéristiques à travers cette couche ;
- ***layer 2*** - Espace de diffusion (*Broadcast space*) : cet espace permet d'envoyer des informations à tous les *Providers* connus sans spécifier un *Topic* ou une liste des destinataires *Providers* ;

- ***layer 3.1*** - *Pour Providers* X (*For providers X*) : via cette couche, un *Provider* peut envoyer des informations à une liste particulière de *Providers* : $\{X_1, X_2, ..., X_n\}$;
- ***layer 3.2*** - *Concernant Topics* Y (*About topics Y*) : via cette couche, les *Provider* peuvent envoyer des messages concernant une liste de *Topic* $\{Y_1, Y_2, ..., Y_n\}$ à tous les *Providers* qui se sont inscrits à ces *Topics* ;
- ***layer 4*** - *Pour Providers* X *concernant Topics* Y (*For providers X about topics Y*) : via cette couche, un *Provider* peut envoyer des informations à propos d'une liste de *Topics* $\{Y_1, Y_2, ..., Y_n\}$ à un ensemble de *Providers* bien défini $\{X_1, X_2, ..., X_n\}$.

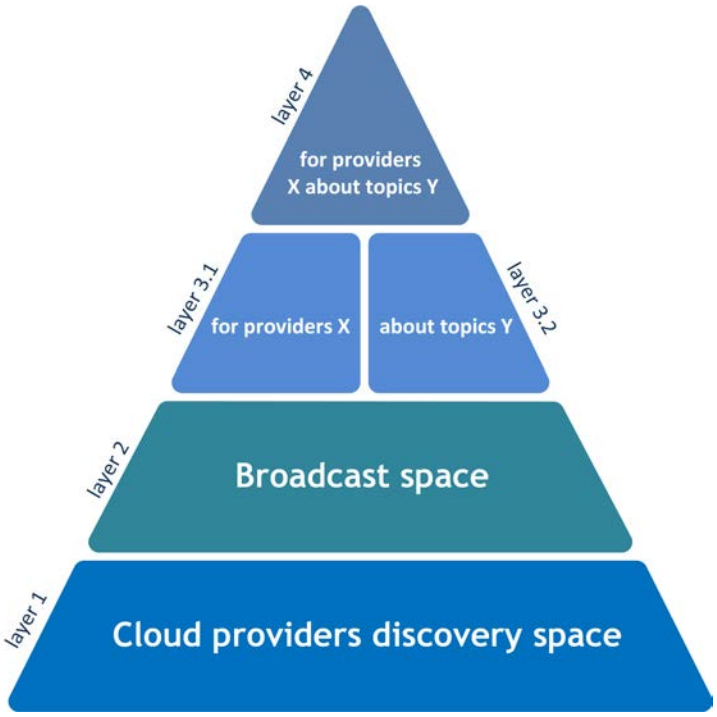

Figure 5.6 – *Modèle d'échange de messages de CMBS*

Dans CMBS, les messages échangés ont un format bien spécifique. Ce format, illustré par la figure 5.7, est composé de trois champs principaux :

- ***For Providers*** : contiens la liste des *Providers* destinataires du message ;
- ***About Topics*** : contiens la liste des *Topics* concernés par le message ;
- ***Message Content*** : contiens les informations utiles et consommables par le récepteur. Le format et la structuration du *Message Content* ne suivent pas des règles prédéfinies, car cela dépend de la nature des données à transférer. Cependant, dans le cadre de nos travaux, le format JSON du standard OCNI a été utilisé.

Comme le montre la figure 5.8, un framework d'échange de messages décentralisé peut être mis en place à base de CMBS. Ce framework facilite la coopération entre les différentes entités d'un Cloud pour assurer la fédération du Cloud.

FIGURE 5.7 – *Format de message CMBS*

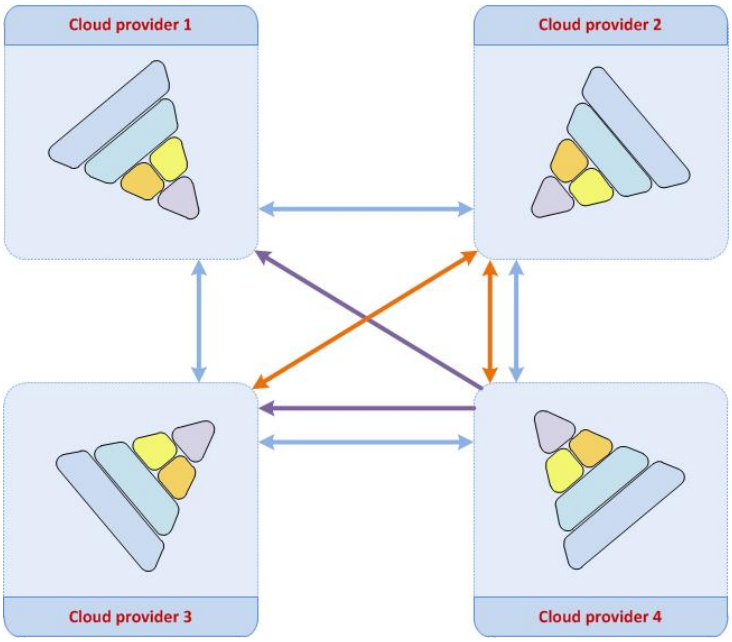

FIGURE 5.8 – *Échange entre Providers à base de CMBS*

5.3.4 LES COMPOSANTS DE CMBS

Pour mettre en place une architecture de fédération à base d'échange de messages, quatre composants ont été définis dans CMBS. Ces composants, illustrés dans la figure 5.9, sont :

- **MQ-S** (*Message Queuing Service*) : MQ-S est le composant périphérique d'une architecture CMBS. MQ-S est le plus proche des applications et des utilisateurs, car il initie l'envoi des messages. De plus, il est le destinataire final des messages et celui qui fait leurs traitements ;
- **MQ-LB** (*Message Queuing Load Balancer*) : Pour assurer la scalabilité et disponibilité du framework CMBS, les composants MQ-S peuvent se dupliquer. Le rôle du MQ-LB est de répartir la charge entre les différents MQ-S. La répartition de la charge peut être définie soit selon une approche classique à la *Round Robin* soit par l'utilisateur ;
- **MQ-B** (*Message Queuing Broker*) : MQ-B est l'élément essentiel dans la mise en place d'une architecture de fédération Cloud à base de CMBS. MQ-B est mis comme front-end pour chaque *Provider*. Il s'occupe de transférer et recevoir les messages à la liste des *Providers* concernés à travers leurs composants MQ-B respectifs. De même, il s'occupe de la réception des messages des autres *Providers* ;
- **Links** : Les *Links* ont le rôle d'établir des liens entre les composants MQ-S, MQ-LB

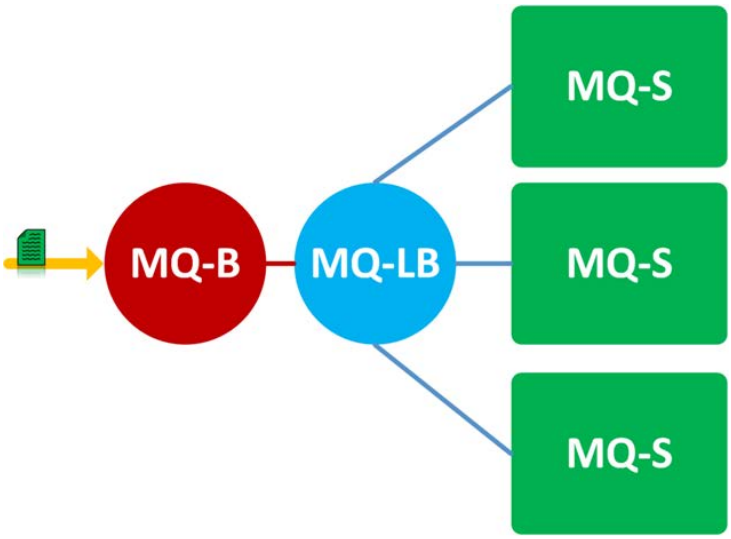

FIGURE 5.9 – *Les composants de CMBS*

et MQ-B.

En se basant sur les composants de CMBS, il est possible d'établir plusieurs modèles architecturaux d'échange de messages. De plus, pour les composants MQ-S et MQ-B, il est possible d'avoir un seul ou plusieurs serveurs de message (les consommateurs de message) qui permettent d'augmenter la liste des combinaisons. Pour simplifier l'illustration de l'ensemble des combinaisons possibles, un plan logique est illustré dans la figure 5.10. Dans l'axe des abscisses, il y a deux valeurs A et B : le A représente les combinaisons de bloc abstraites et le B représente les cas déjà définis en A en détaillant les combinaisons possibles sur le critère du nombre de serveurs. Dans l'axe des ordonnées, il y a les combinaisons qui se basent sur le nombre de chaque type des composants CMBS (ces nombres peuvent être *0*, *1* ou n). Comme présenté dans la figure 5.10, les combinaisons possibles sont illustrées par les symboles AX ou $BX.Y$ avec $X = \{1, 2, 3, 4, 5, 6\}$ et $Y = \{1, 2, 3, 4\}$. Toutes ces combinaisons sont illustrées dans les figures 5.11, 5.12, 5.13, 5.14, 5.15, 5.16, 5.17, 5.18.

Parmi les 18 combinaisons possibles des modèles architecturaux de CMBS, le modèle B5.4 de la figure 5.16 est composé de *1* MQ-B, *1* MQ-LB et n MQ-S. De plus, le MQ-B et les MQ-S sont composés par n serveurs.

Tous les modèles de combinaisons cités auparavant peuvent être déployés dans un domaine administratif. De plus, afin d'assurer la fédération Cloud, chaque domaine devrait avoir un ensemble de composants CMBS qui suit l'un des modèles prédéfinis. Ces modèles déployés ne sont pas forcément les mêmes. Ce cas d'exemple est illustré dans la figure 5.19 là où il y a quatre domaines administratifs. En effet, les messages inter-domaines passent via les MB-B et les liens entre ces MQ-B sont à définir selon l'architecture de fédération cible. Ces liens peuvent être multi-directionnels ou uni-directionnels.

Pour assurer le bon fonctionnement de CMBS en passant par les fournisseurs de réseau et dans des réseaux ayant des politiques et des protocoles hétérogènes, plusieurs MQ-B peuvent être déployés dans ces réseaux et entre les MQ-B périphériques. Le rôle de ces

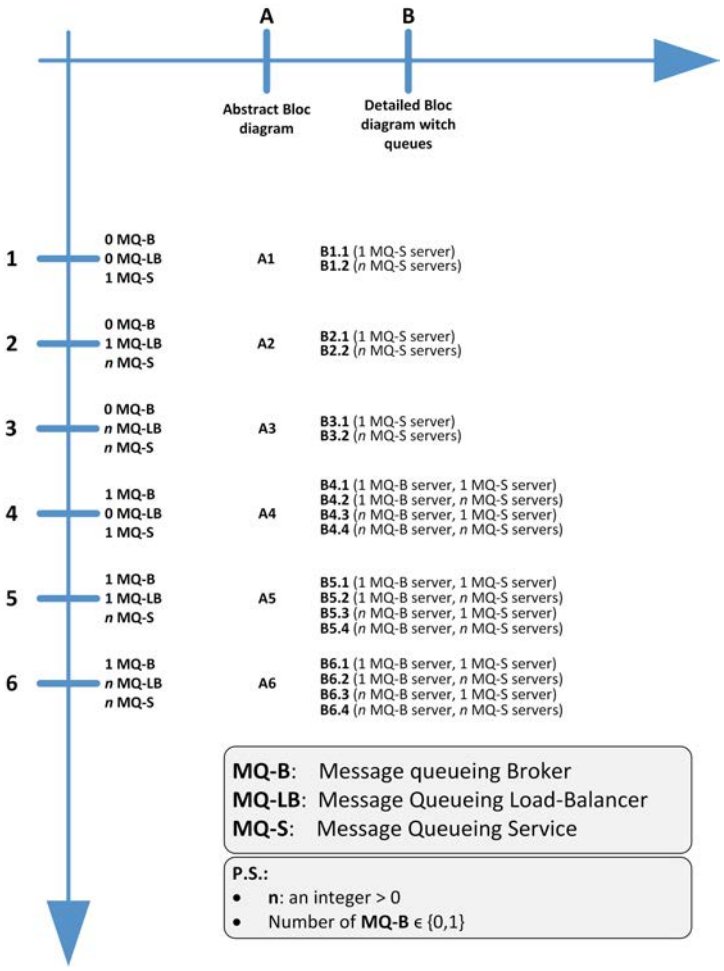

Figure 5.10 – *Liste des combinaisons possibles des composants du CMBS*

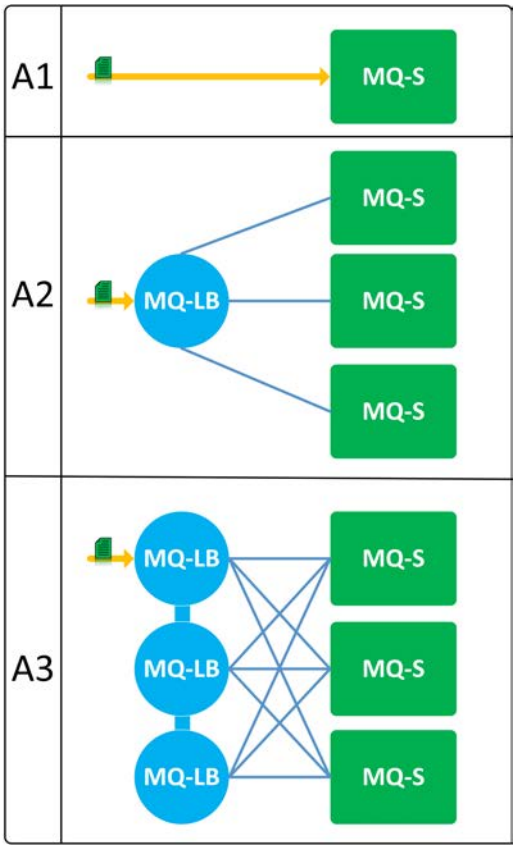

Figure 5.11 – *Combinaisons A1, A2 et A3*

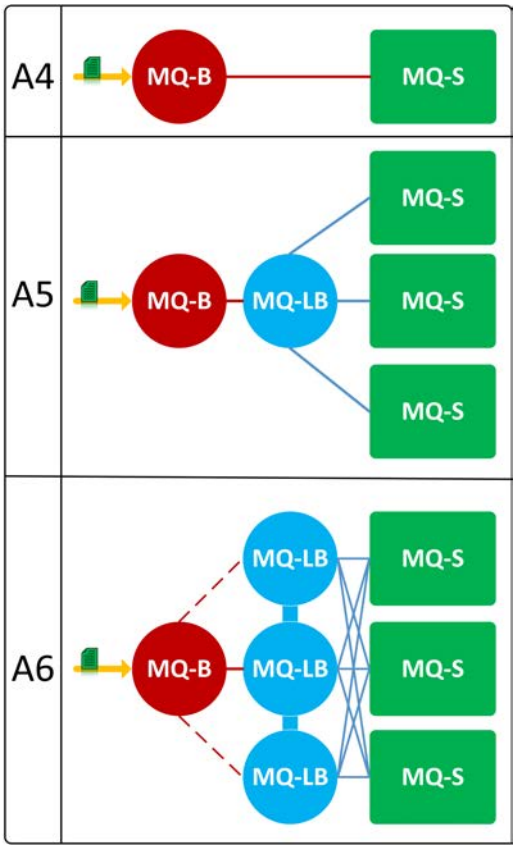

Figure 5.12 – *Combinaisons A4, A5 et A6*

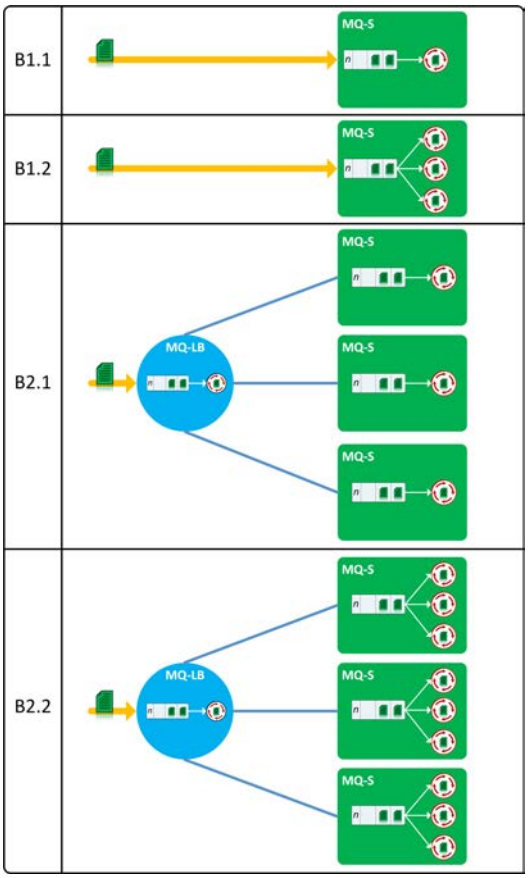

Figure 5.13 – *Combinaisons B1.1, B1.2, B2.1 et B2.2*

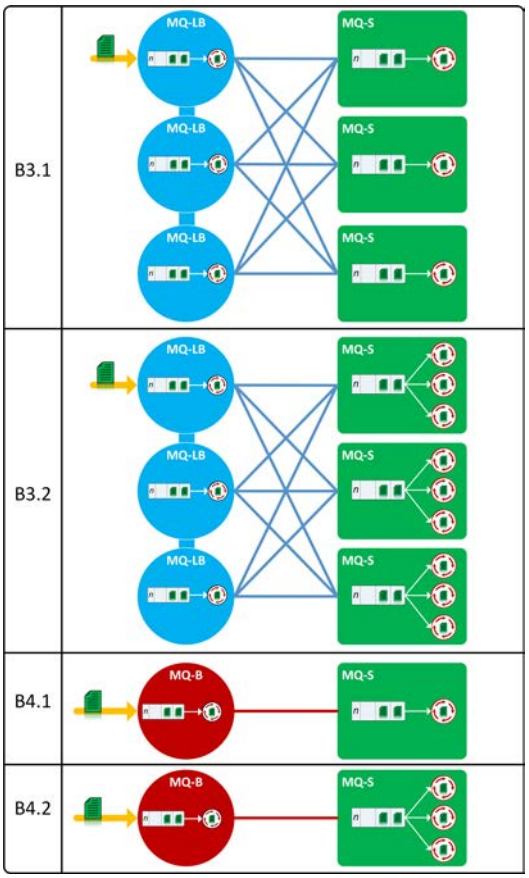

Figure 5.14 – *Combinaisons B3.1, B3.2, B4.1 et B4.2*

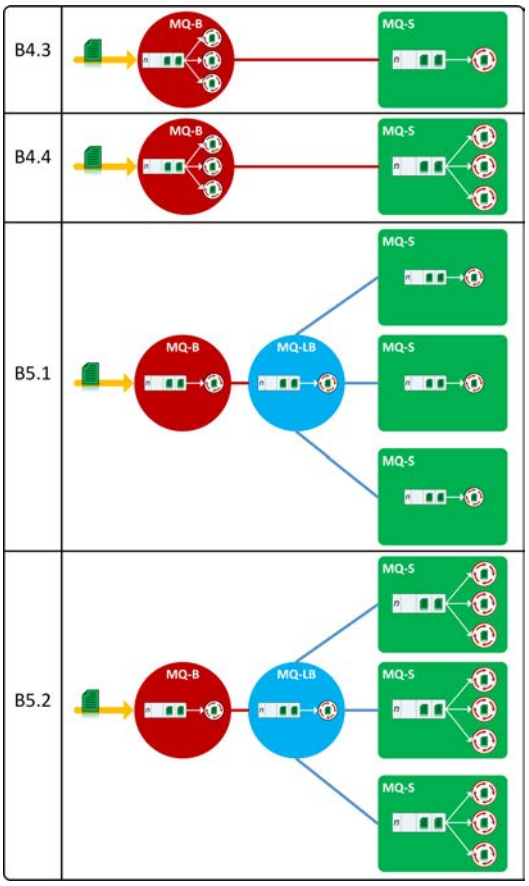

Figure 5.15 – *Combinaisons B4.3, B4.4, B5.1 et B5.2*

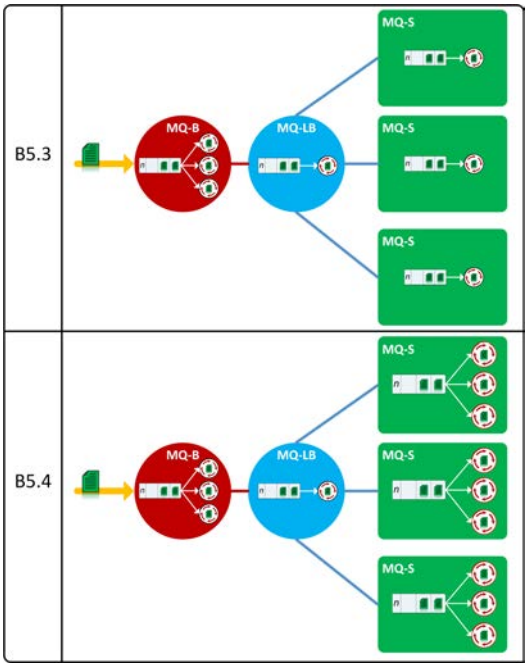

Figure 5.16 – *Combinaisons B5.3 et B5.4*

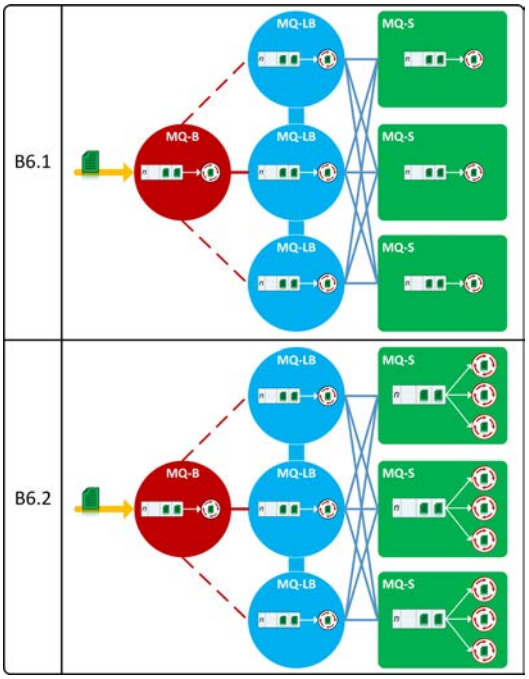

Figure 5.17 – *Combinaisons B6.1 et B6.2*

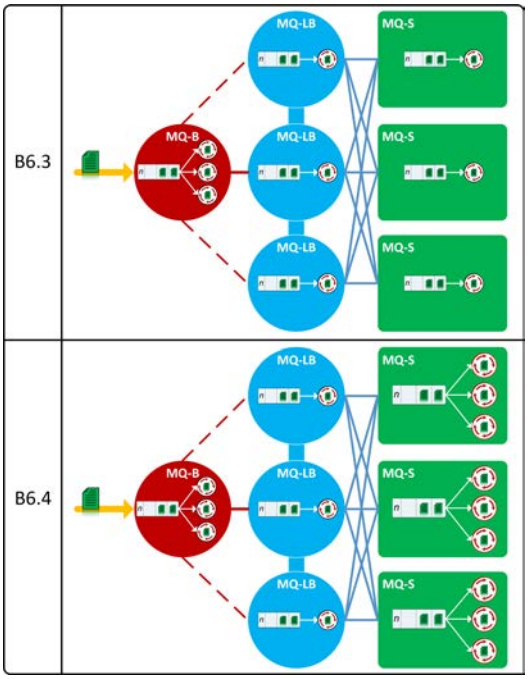

Figure 5.18 – *Combinaisons B6.3 et B6.4*

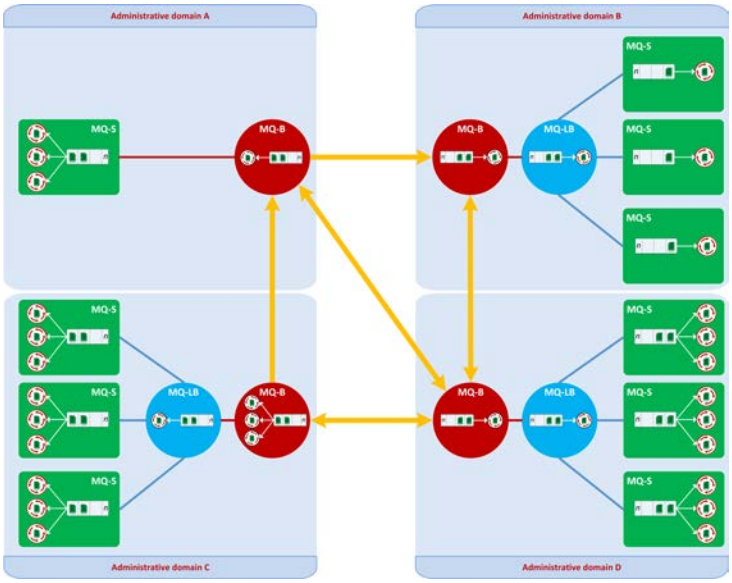

FIGURE 5.19 – *Exemple d'échange de message Cloud à base de CMBS*

MQ-B intermédiaire est d'assurer le bon fonctionnement du framework de fédération tout en améliorant les performances. La figure 5.20 illustre un exemple d'utilisation de ces MQ-B intermédiaires.

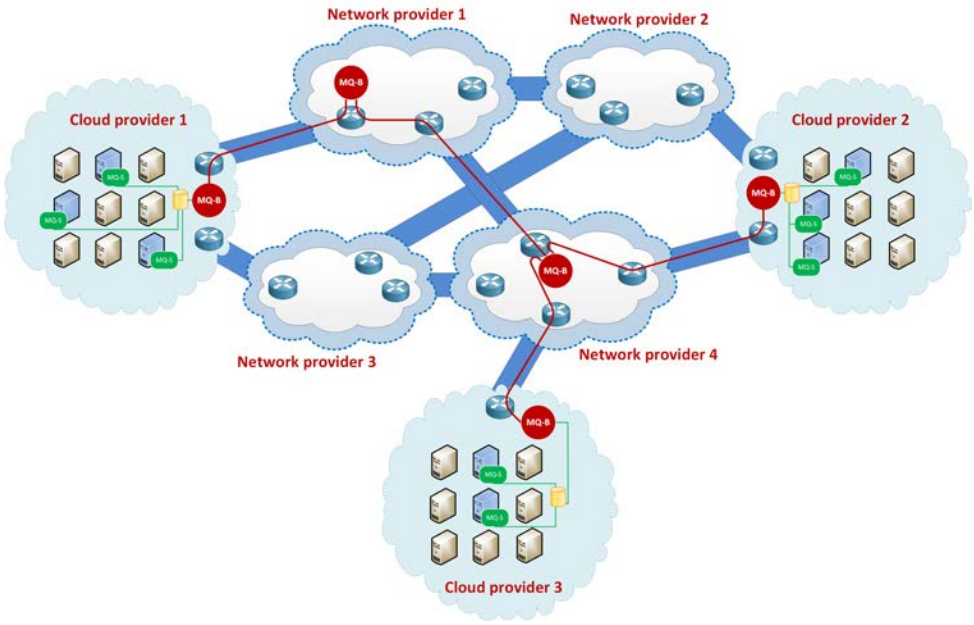

FIGURE 5.20 – *Exemple d'utilisation de MQ-B intermédiaire*

5.4 IMPLÉMENTATION ET CAS D'UTILISATION

L'implémentation de CMBS est en Python et elle est disponible en tant que projet Open Source à [49]. Cette implémentation a servi comme base pour la validation des concepts et modèle CMBS. Elle est connue sous le nom Python Cloud Message Brokering Service (pyCMBS). pyCMBS a été utilisé principalement dans le cadre du projet SAIL

[52].

Dans l'implémentation pyCMBS, chaque composant CMBS est encapsulé dans un serveur pyOCNI. Grâce à ce dernier, la gestion et le contrôle des composants CMBS se font via des requêtes HTTP REST bien définis. Coté backend des serveurs, des connecteurs ont été développés pour assurer les fonctions de messagerie natives. Ces connecteurs fonctionnent principalement pour deux technologies d'échange de messages Open Source : AMQP/RabbitMQ et ZeroMQ. Comme illustré dans le *Google Trend* de la figure 5.21, ces deux technologies ne cessent de gagner de l'importance dans le domaine informatique. L'utilisation conjointe de ces deux technologies permet de satisfaire la définition et les contraintes de CMBS définis dans la section 5.3.1. Cette utilisation conjointe satisfait particulièrement le fait d'avoir la possibilité de déployer une architecture avec ou sans *broker* central.

Pour récapituler, pyCMBS est une implémentation en Python de CMBS, utilisant zeroMQ et AMQP/RabbitMQ pour assurer les fonctions de routage de messages natifs, employant CouchDB comme base de données NoSQL persistante, s'encapsulant dans un serveur OCNI et utilisant le format JSON qui est formalisé avec le modèle OCNI.

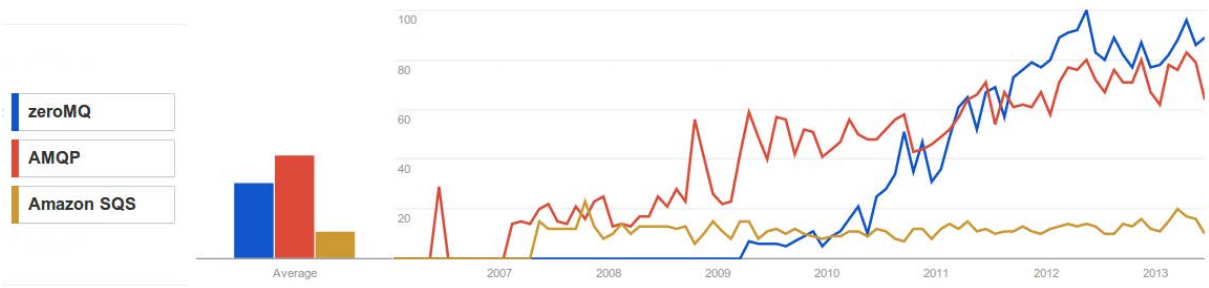

FIGURE 5.21 – *Fréquence de recherche : zeroMQ vs AMQP vs Amazon SQS*

CMBS peut être utilisé dans différents cas de figure. Généralement, l'utilisation de CMBS peut être classifiée selon trois niveaux dans le contexte d'une fédération Cloud. Comme illustré dans la figure 5.22, ces trois niveaux sont :

- Niveau applicatif : CMBS est utilisé par les applications déployées dans le Cloud. À ce niveau, CMBS est utilisé pour assurer le bon fonctionnement des applications ;
- Niveau fournisseur : CMBS est utilisé par les fournisseurs des ressources Cloud. À ce niveau, CMBS est utilisé pour assurer l'allocation et le contrôle des ressources entre plusieurs fournisseurs dans un environnement de Cloud Hybride ;
- Niveau contrôleurs de ressources : CMBS est utilisé par les contrôleurs de ressources Cloud. À ce niveau, CMBS est utilisé pour assurer la bonne configuration des ressources dans un environnement de Cloud Hybride.

Par la suite, nous nous focalisons sur un exemple de fonctionnement de CMBS au niveau applicatif. Sachant que l'efficacité de CMBS dans les deux autres niveaux a été aussi approuver dans le cadre du projet SAIL.

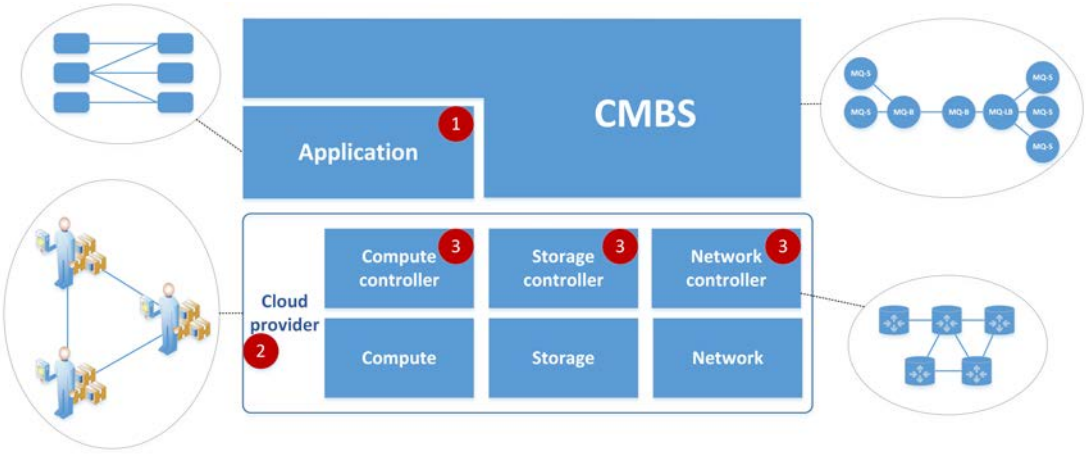

FIGURE 5.22 – *Trois niveaux d'utilisation de CMBS*

L'exemple applicatif choisi pour l'utilisation de CMBS dans les environnements Cloud est une application financière de bourse. Cette application sert à faire coopérer les acteurs de la bourse. Ces acteurs ont de fortes contraintes dans l'échange des données tout en respectant les exigences liées à la localisation et aux SLAs. CMBS peut satisfaire ces exigences et bien faire fonctionner ce genre d'application pour un déploiement Cloud. Grâce à CMBS, il est possible d'échanger les données utiles au bon moment tout en diminuant les coûts et en respectant les SLAs. De plus, CMBS fournit une solution indépendante des fournisseurs et des plateformes (sans *Vendor Lock-In*) tout en assurant l'interopérabilité.

Afin d'illustrer le cas d'utilisation d'une application financière utilisant CMBS, plusieurs éléments doivent être définis. Ces éléments sont groupés par la définition des acteurs de la bourse, des types de données échangés et l'architecture en couches de l'application. Ces trois groupes, schématisés dans la figure 5.23 sont définis comme suit :

Acteurs de la bourse :

1. Place boursière : Euronext Paris, Bourse de Londres, Bourse de New York, etc ;
2. Intermédiaire financier : Société de bourse, banque, conseiller financier, broker, etc ;
3. Émetteurs : Entreprises, états, etc ;
4. Investisseurs : Particulier, entreprises, investisseurs institutionnels, etc ;
5. Autorités du marché : Instances de régulation.

Données échangées :

1. Données de la bourse : Indices boursiers (CAC40, NASDAQ, Nikkei, indice des entreprises...) ;
2. Données des intermédiaires financiers ;
3. Données des émetteurs ;
4. Données des investisseurs ;
5. Données des autorités de marchés.

Architecture trois tiers :

1. Couche de présentation ;
2. Couche de traitement ;
3. Couche de stockage.

FIGURE 5.23 – *Éléments de base du cas d'utilisation*

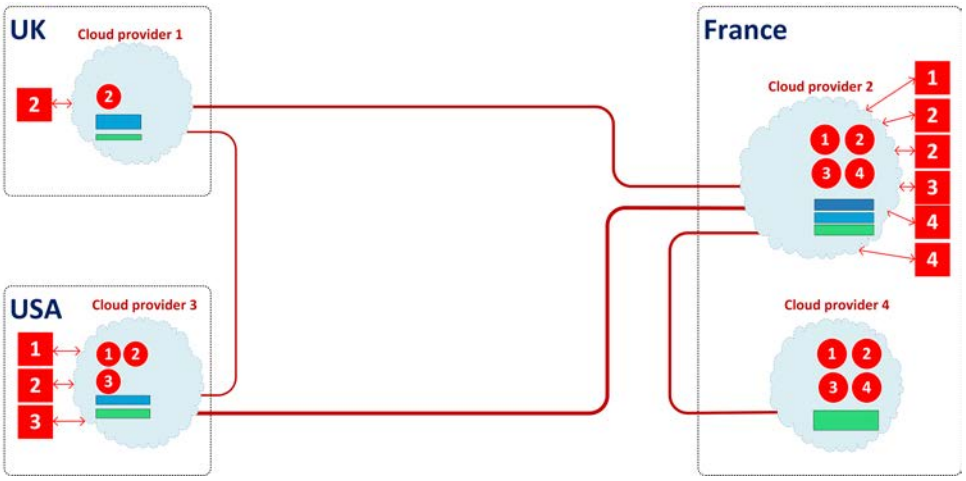

FIGURE 5.24 – *Exemple de requête de CMBS*

Un exemple de requêtes est schématisé dans la figure 5.24. Cette requête est envoyée par une application financière de bourse tout en se basant sur un framework CMBS. Dans cet exemple, quatre fournisseurs de ressources Cloud réparties dans trois pays sont nécessaires. Cette répartition est conduite par des règles de localisation et de qualité de service (QoS). Dans chaque fournisseur un ensemble de données sont à échanger selon la nature des acteurs et selon les couches applicatives à déployer. Par exemple, en Angleterre chez le fournisseur Cloud numéro 1 seul l'intermédiaire financier intervient et seules ses données sont stockées et traitées via les deux couches de stockage et de traitement. En France, deux fournisseurs de Cloud sont utilisés. Chez le premier fournisseur (le numéro 2), trois couches applicatives sont utilisées et quatre types de données sont manipulées. Ces types de données sont liés aux quatre acteurs (place boursière, deux intermédiaires financiers, un émetteur et deux investisseurs) qui interagissent avec les ressources déployées chez ce fournisseur. Quant au fournisseur de Cloud numéro 4, il est utilisé principalement comme site de stockage et backup, car il ne contient que la couche applicative de stockage. Pour

les liens d'échange d'informations, le fournisseur de Cloud numéro 2 doit interagir avec les trois autres fournisseurs de Cloud et le fournisseur de Cloud numéro 1 doit interagir avec le fournisseur de Cloud numéro 3.

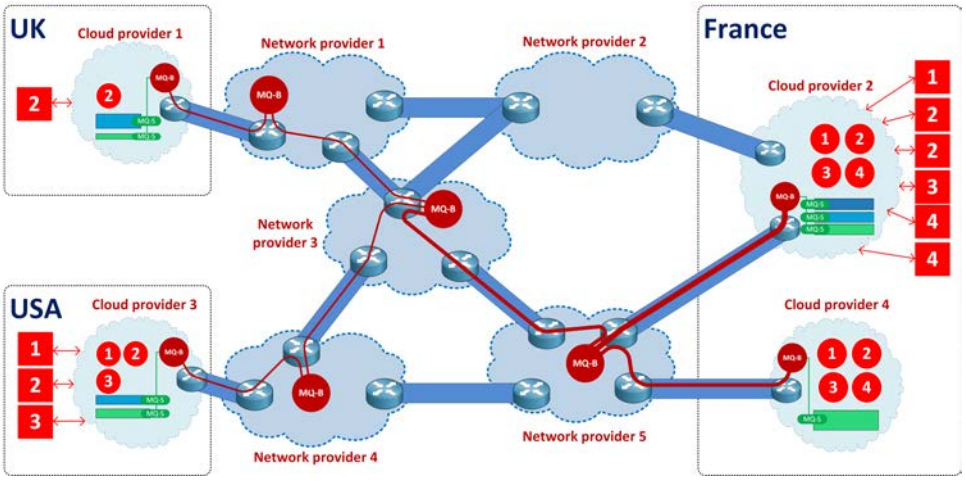

Figure 5.25 – *Exemple de déploiement de CMBS*

Le résultat de déploiement de CMBS qui peut satisfaire la requête demandée est illustré dans la figure 5.25. La configuration de CMBS mise en place est composée d'un ensemble de composants MQ-S attachés à chaque instance de couche applicative. Ces composants MQ-S sont liés du côté de chaque fournisseur à un composant MQ-B. Notez bien que chaque fournisseur a au moins une instance de MQ-B. Ces instances de MQ-B qui sont déployées chez les fournisseurs de Cloud, sont liées à d'autres instances MQ-B déployées chez les fournisseurs réseau. Les liens logiques demandés par l'application financière sont mappés en tant que règles de routage de message au niveau des instances MQ-B.

5.5 Conclusion

Nous avons présenté dans ce chapitre notre contribution du système d'échange et courtage Cloud à base de message - CMBS. Avant la description de CMBS, nous avons introduit au début du chapitre le concept général d'échange de message de type MQS (*Message Queuing Service*). Par la suite, nous avons détaillé CMBS en présentons ses requêtes, ses modèles de messages et ses composants. Ensuite, nous avons présenté une implémentation de CMBS et ses cas d'utilisation. Les travaux réalisés dans le cadre de CMBS ont été adoptés et validés dans le cadre du projet SAIL[52].

Grâce à CMBS, la fédération Cloud entre plusieurs domaines et plusieurs acteurs est devenue une réalité. De plus, CMBS est une solution de type Cloud qui respecte les caractéristiques Cloud comme celles liées à la simplicité, la flexibilité ou l'extensibilité. De ce fait, l'intégration et la mise en place des architectures de fédération à base de CMBS sont devenues simples, flexibles et extensibles.

Chapitre 6

Conclusion Générale

Présenté dans la littérature comme une nouvelle technologie, le Cloud Computing est devenu incontournable dans la mise en place et la fourniture des services informatiques. En effet, le Cloud offre d'importants avantages en termes d'extensibilité, d'élasticité et de coûts. L'objectif principal de ce travail de thèse était de proposer des solutions aux problématiques d'interopérabilité et de fédération dans le Cloud. Notre travail de recherche s'est articulé autour de deux grands axes. Le premier axe concerne le rapprochement et la convergence entre le Cloud Computing et le Cloud Networking. Le deuxième axe a visé la fédération et le courtage entre les solutions et services hétérogènes du Cloud.

En se basant sur nos travaux d'état de l'art au sujet du Cloud Computing et Cloud Networking, nous avons défini, dans cette thèse, deux architectures de fédération Cloud. La première architecture a permis le rapprochement entre le Cloud Computing et le Cloud Networking. Cette architecture introduit le concept d'un plan de contrôle distribué qui fonctionne sur un ou plusieurs domaines administratifs. Quant à la deuxième architecture, elle ramène une solution aux problèmes d'interopérabilité et courtage entre les services de type Cloud. La solution proposée est simple, flexible et évolutive. Ces caractéristiques assurent aussi bien l'intégration des nouveaux services Cloud que la mise à jour de l'existant.

L'étude des deux architectures de fédération que nous avons menée a fait ressortir deux composants primordiaux et essentiels pour assurer la fédération : une interface générique et un système d'échange de messages. Ces deux composants ont fait l'objet de deux contributions majeures de notre thèse. La contribution de l'interface générique Cloud a permis l'allocation et la gestion de n'importe quelles ressources et services Cloud. Cette interface, nommée Open Cloud Networking Interface - OCNI, est une extension de la spécification OCCI avec une spécialisation du Cloud Networking. Elle permet alors de combler l'écart entre les services d'interconnexion au niveau intra-Cloud et inter-Cloud. En outre, OCNI a facilité la participation des fournisseurs de réseaux *'Cloud Carrier'* dans la mise en place des services et l'émergence des déploiements hybrides du *Cloud Computing* via une interface Cloud standardisée et ouverte. Afin de valider notre proposition, nous avons fourni une implémentation Open Source d'OCNI qui a été adoptée dans le cadre du projet de recherche SAIL [52].

En plus de l'interface générique et comme détaillée dans nos contributions d'architecture, un système d'échange de messages est essentiel pour assurer la fédération Cloud. Ce système, nommé *Cloud Message Brokering Service* CMBS, est un service d'échange et de courtage Cloud à base de messages entre plusieurs domaines, fournisseurs ou entités dans le Cloud. Grâce à CMBS, la fédération Cloud entre plusieurs domaines et plusieurs acteurs est devenue une réalité. De plus, CMBS est une solution de type Cloud qui respecte les caractéristiques Cloud comme celles liées à la simplicité, la flexibilité ou l'extensibilité. De ce fait et comme validées dans le cadre des projets SAIL [52] et CompatibleOne [26], l'intégration et la mise en place des architectures de fédération à base de CMBS sont devenues simples, flexibles et extensibles.

Toutes les études et les contributions entreprises au cours de ce travail de thèse nous ont permis de mieux appréhender la fédération du Cloud. Cette thèse peut être le point de départ de plusieurs travaux d'extension dont nous citons quelques un en tant que perspectives. Comme première perspective au niveau des architectures de fédération et afin de valider leurs pertinences, nous comptons tester la fédération des récentes solutions fournies par les acteurs historiques des systèmes d'information (HP, IBM, VMware, Microsoft, Oracle...). Comme deuxième perspective liée à l'interface générique, nous pourrons étendre l'implémentation d'OCNI avec les nouveaux types de services et solutions Cloud. Parmi ces derniers, nous citons la virtualisation à base de conteneur type Docker, les récents contrôleurs réseaux à base de SDN et NFV et aussi les systèmes convergés et hyper-convergés. Quant à la contribution du système d'échange de messages, nous allons déployer notre contribution chez les principaux fournisseurs Cloud publics afin de valider le bon fonctionnement de notre solution CMBS dans de multiples environnements Cloud.

Annexe A

Modèles des langages de description des réseaux

A.1 NDL - Network Description Language

Exemple de code NDL selon la spécification RDF dans la version 2.4

```
<ndl:Location rdf:about="#Amsterdam1.netherlight.net">
  <ndl:name>Netherlight Optical Exchange</ndl:name>
  <geo:lat>52.3561</geo:lat>
  <geo:long>4.9527</geo:long>
</ndl:Location>
<ndl:Device rdf:about="#tdm1.amsterdam1.netherlight.net">
  <ndl:name>tdm1.amsterdam1.netherlight.net</ndl:name>
  <ndl:locatedAt rdf:resource="#amsterdam1.netherlight.net"/>
  <ndl:hasInterface rdf:resource="#tdm1.amsterdam1.netherlight.net
      :12/1"/>
  <ndl:hasInterface rdf:resource="#tdm1.amsterdam1.netherlight.net:6/1"/>
</ndl:Device>
<ndl:Interface rdf:about="#tdm1.amsterdam1.netherlight.net:12/1">
  <ndl:name>12/1</ndl:name>
  <ndl:connectedTo rdf:resource="#tdm3.amsterdam1.netherlight.net
      :501/2"/>
  <ndl:capacity rdf:datatype="&xsd;float">1.244E+9</ndl:capacity>
</ndl:Interface>
```

A.2 cNIS - Common Network Information Service

A.3 NM/perfSONAR - Network Measurement/perfSONAR

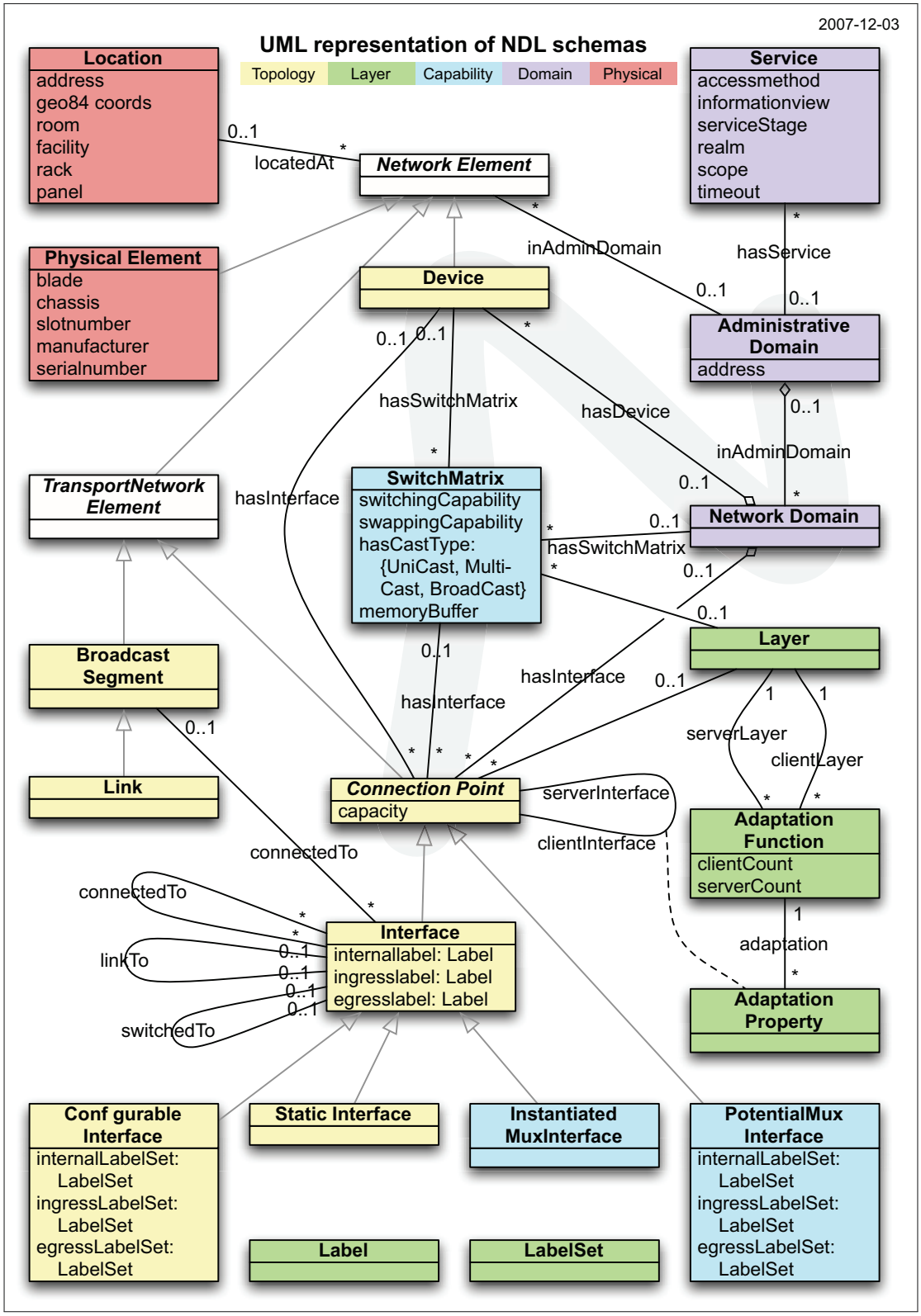

Figure A.1 – *Représentation UML du schéma NDL[17]*

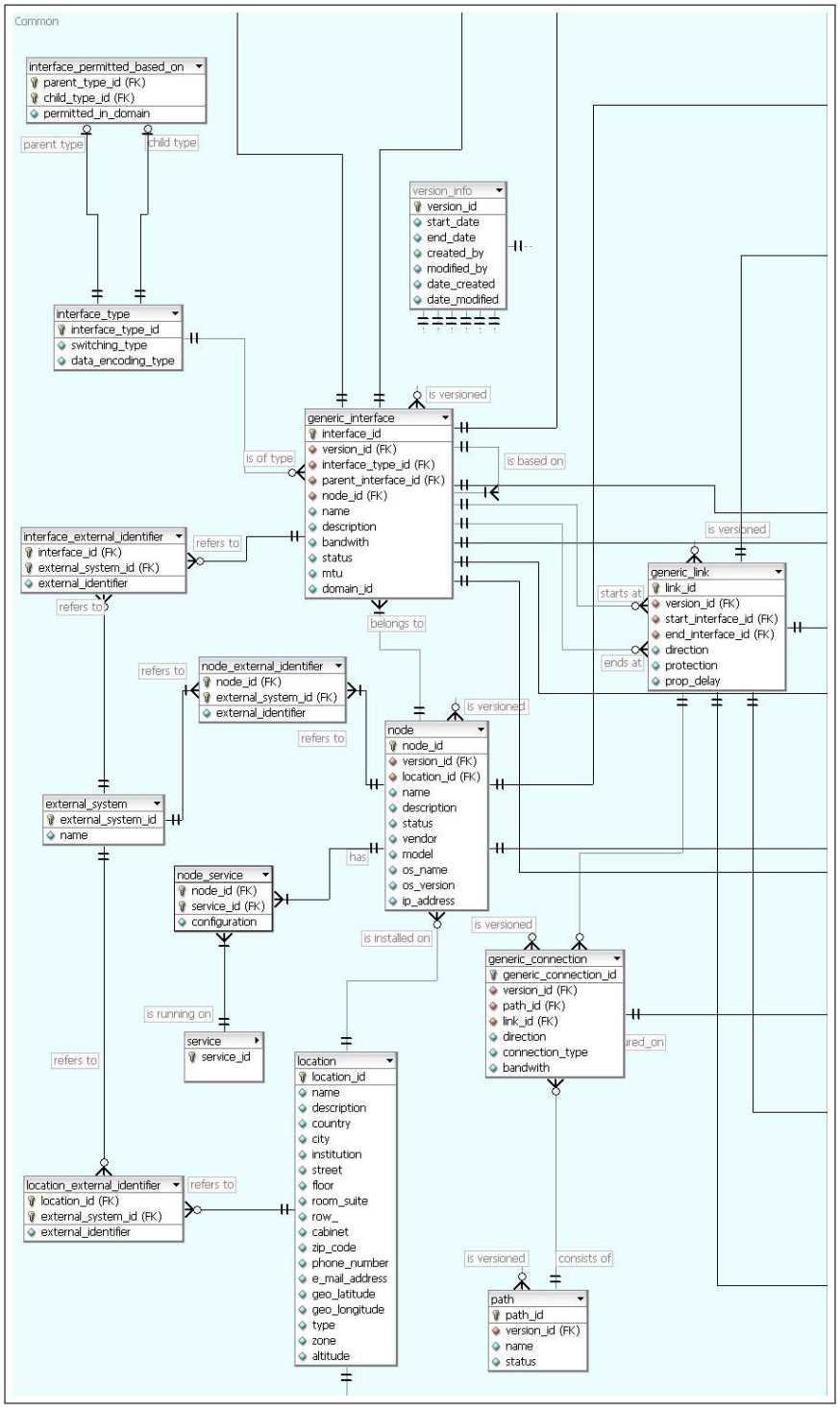

Figure A.2 – *Diagramme entité/relation des tables principales de cNIS[150]*

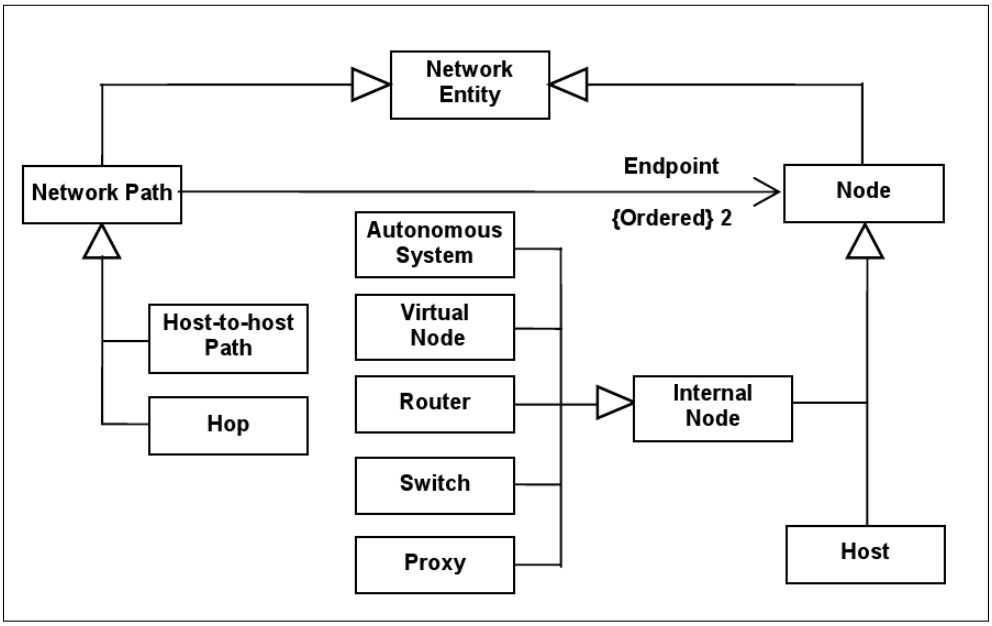

Figure A.3 – *La relation entre les éléments d'un réseau selon NM[111]*

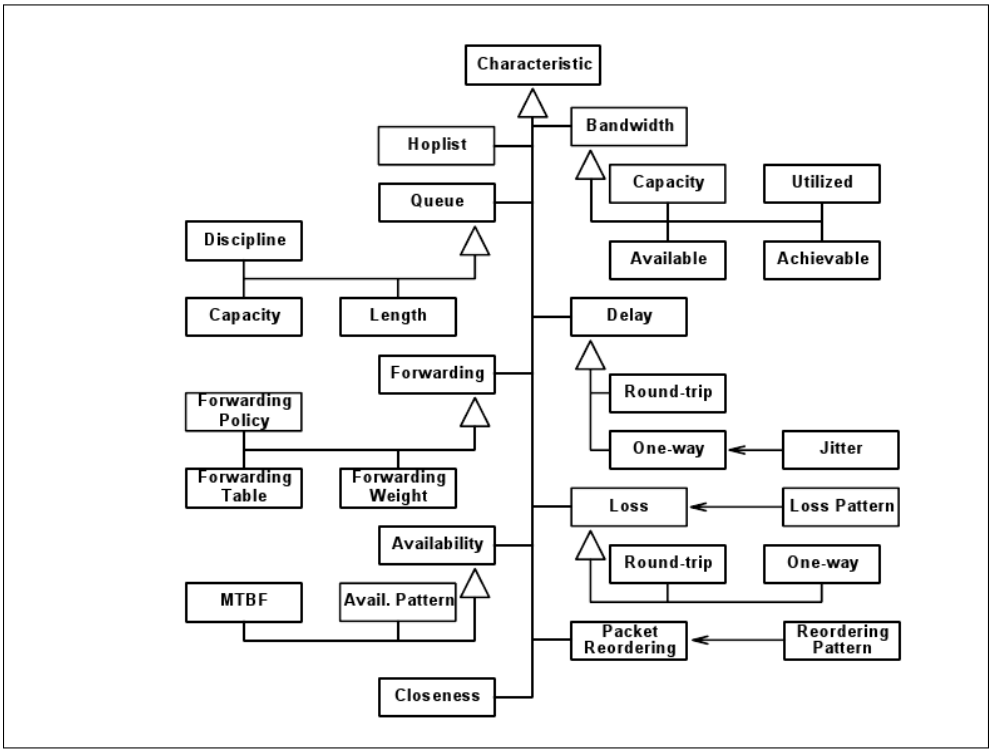

Figure A.4 – *Les caractéristiques utilisées pour décrire un réseau selon NM[111]*

Exemple de code XML de la spécification NM qui spécifie la bande passante d'un lien entre deux nœuds

```
<nmwg:metadata id="bandmeta1">
  <band:subject id="bandsub1">
    <nmwgt:endPointPair>
      <nmwgt:src type="hostname" value="blackseal.pc.cis.udel.edu" port
         ="21132"/>
      <nmwgt:dst type="hostname" value="ellis.internet2.edu" port
         ="14837"/>
    </nmwgt:endPointPair>
  </band:subject>
  <band:parameters id="bandparam1">
    <nmwg:parameter name="valueUnits">ms</nmwg:parameter>
    <nmwg:parameter name="numBytes">64</nmwg:parameter>
    <nmwg:parameter name="numBytesUnits">bytes</nmwg:parameter>
  </band:parameters>
</nmwg:metadata>
```

A.4 SDL - System Description language

Exemple de description d'une ressource en SDL

```
<computer id="SFR1">
  <interface id="eth0" technology="Ethernet" protocol="1000BaseT"
     connector="RJ45">
    <addr type="hw">00:09:3D:00:28:33</addr>
    <addr type="ipv4" netmask="255.255.255.248">10.0.10.1</addr>
  </interface>
  <os idRef="Debian"/>
  <service id="web server">
    <protocol idRef="pwrhttp"/>
    <sap addr="*" transport="tcp" port="80"/>
    <software idRef="DebianApache"/>
  </service>
</computer>
```

A.5 VXDL - Virtual eXecution Infrastructures Description Language

Exemple de description d'une ressource en VXDL

```
<vxdl:resource>
  <vxdl:name>Node_Cluster_Borderline</vxdl:name>
  <vxdl:ramMemory>
    <vxdl:max>32</vxdl:max>
    <vxdl:maxUnit>GB</vxdl:maxUnit>
```

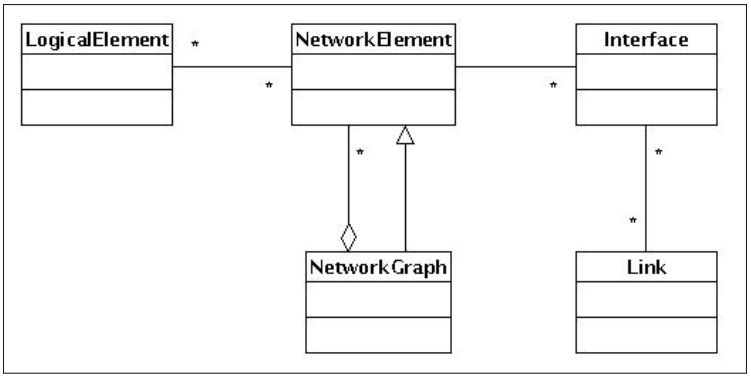

Figure A.5 – *Le diagramme UML de la structure générale de SDL*

```
  </vxdl:ramMemory>
  <vxdl:cpuFrequency>
    <vxdl:max>2.6</vxdl:max>
    <vxdl:maxUnit>GHz</vxdl:maxUnit>
  </vxdl:cpuFrequency>
  <vxdl:cpuProcessors>
    <vxdl:max>4</vxdl:max>
  </vxdl:cpuProcessors>
  <vxdl:hdSize>
    <vxdl:max>600</vxdl:max>
    <vxdl:maxUnit>GB</vxdl:maxUnit>
  </vxdl:hdSize>
</vxdl:resource>
```

Exemple de description d'un lien en VXDL

```
<vxdl:link>
  <vxdl:name>Raw Data</vxdl:name>
  <vxdl:bandwidth>
    <vxdl:min>38</vxdl:min>
    <vxdl:minUnit>Gbps</vxdl:minUnit>
  </vxdl:bandwidth>
  <vxdl:direction>uni</vxdl:direction>
  <vxdl:pair>
    <vxdl:source>Cluster Data Source</vxdl:source>
    <vxdl:destination>Cluster Filtering</vxdl:destination>
  </vxdl:pair>
</vxdl:link>
```

A.6 NML - Network Mark-up Language

Le diagramme UML de la spécification NML :

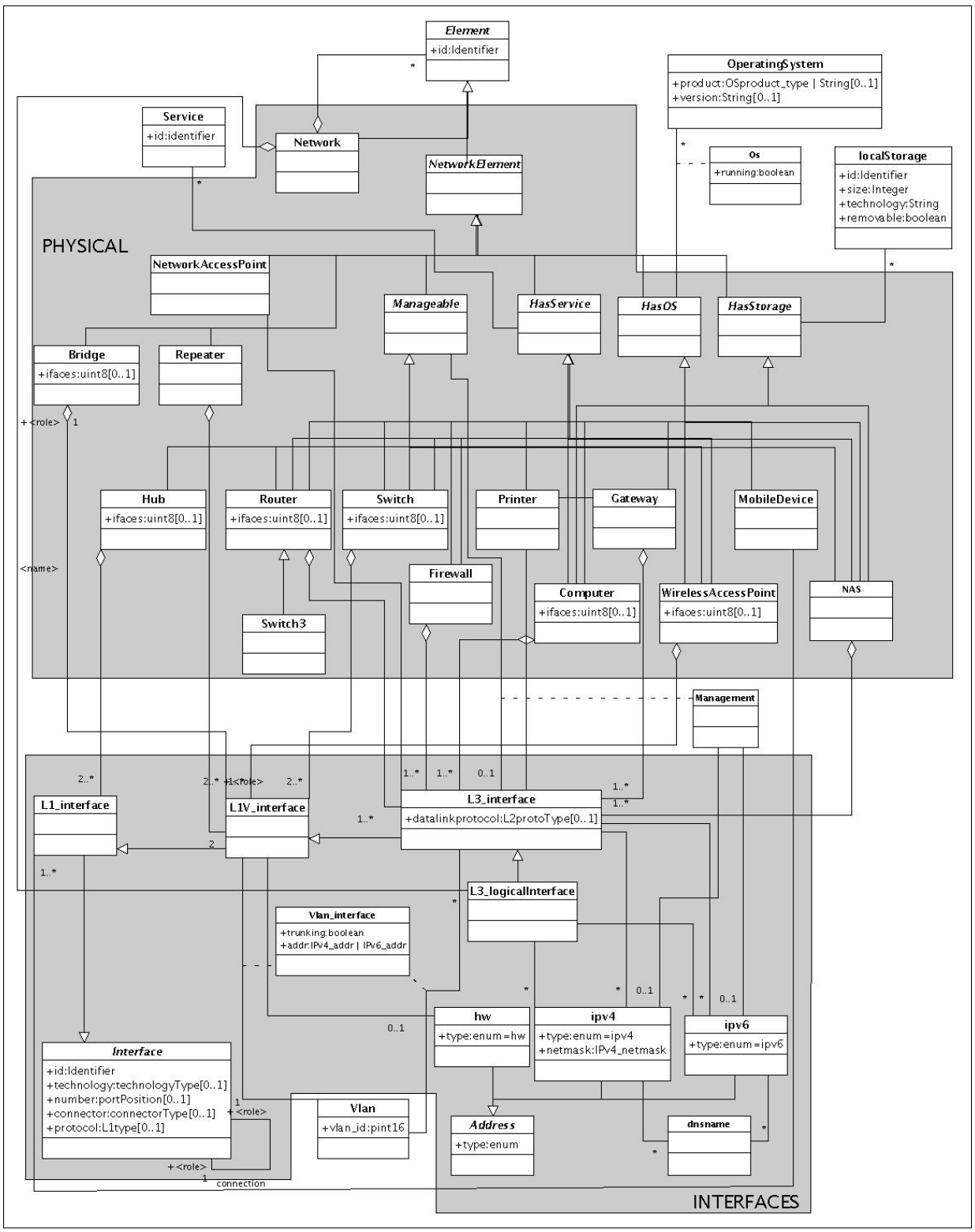

Figure A.6 – *Diagramme UML des noeuds et des interfaces dans SDL*

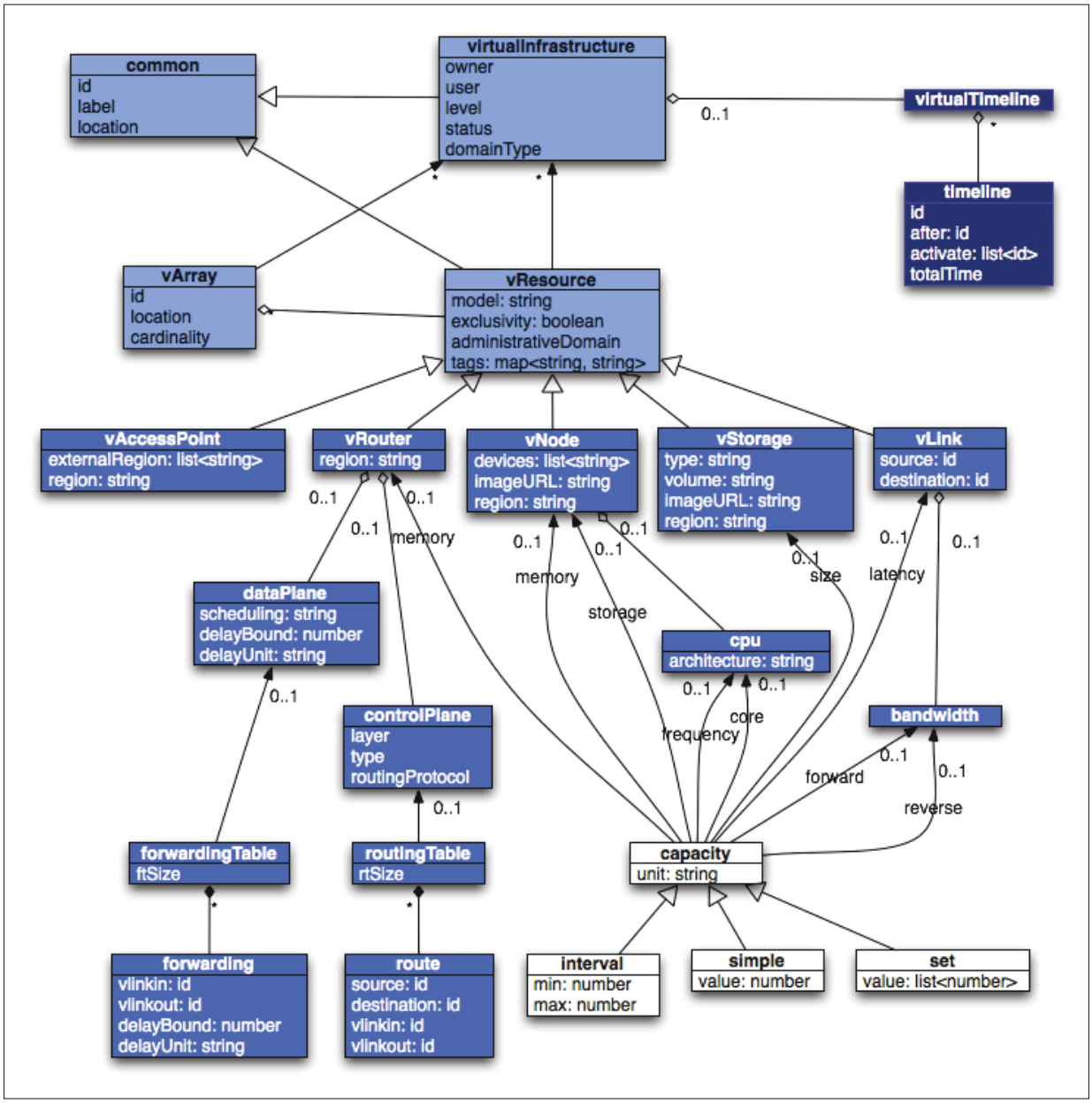

Figure A.7 – *Diagramme UML de VXDL[12]*

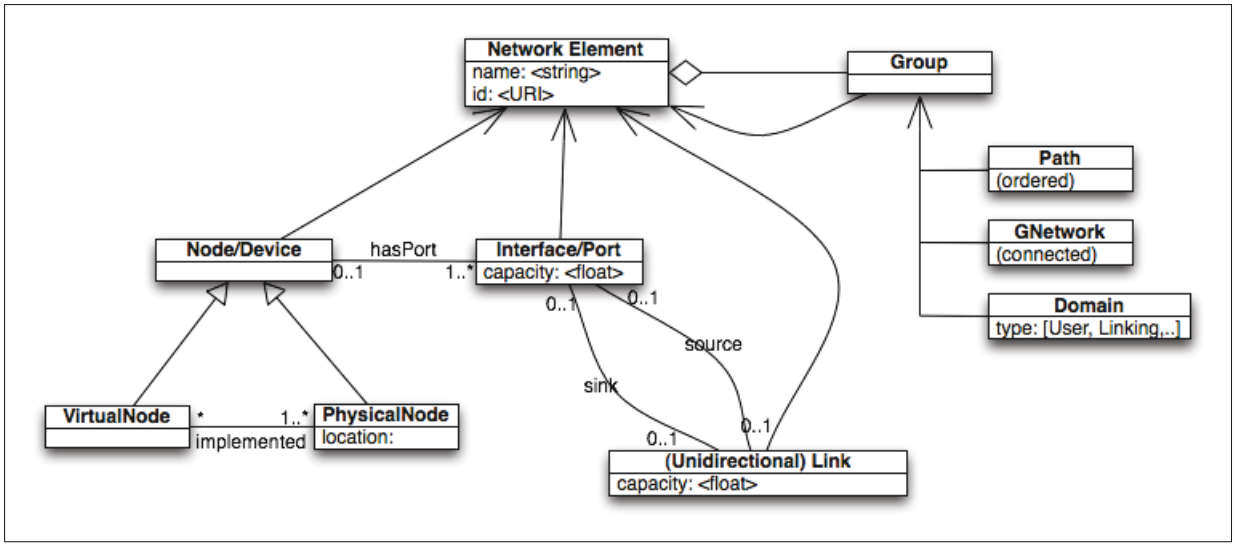

Figure A.8 – *Diagramme UML de NML [8]*

A.7 INDL - Infrastructure et Réseau Description Language

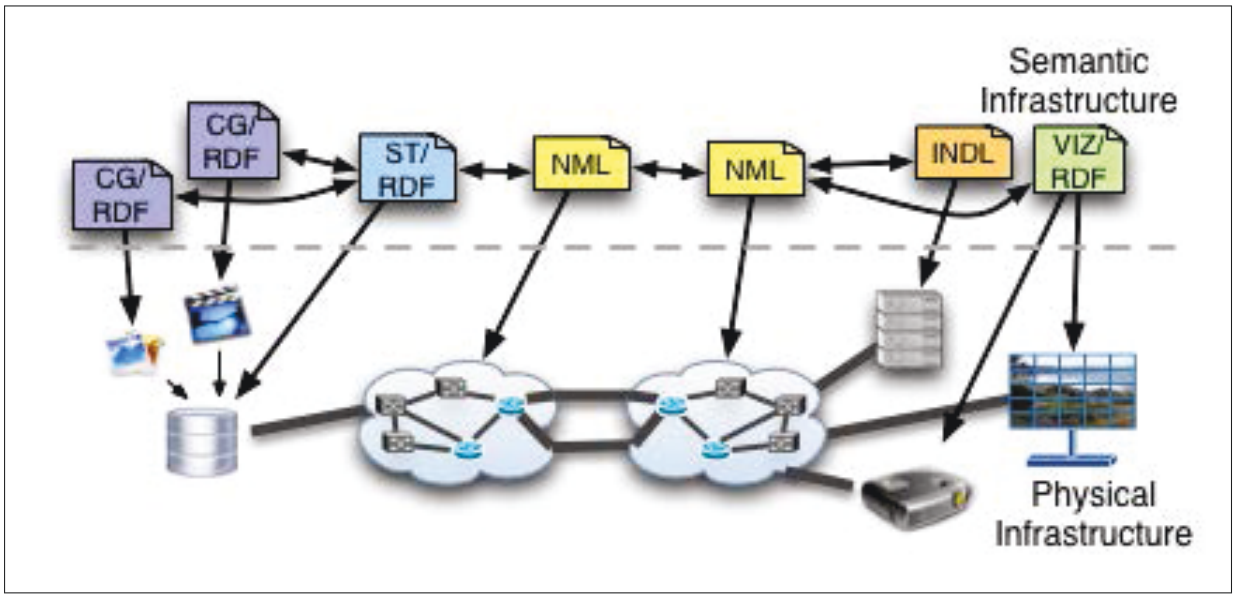

Figure A.9 – *Utilisation d'INDL dans la description des infrastructures et des réseau [17]*

A.8 NNDL - Network Node Description Language

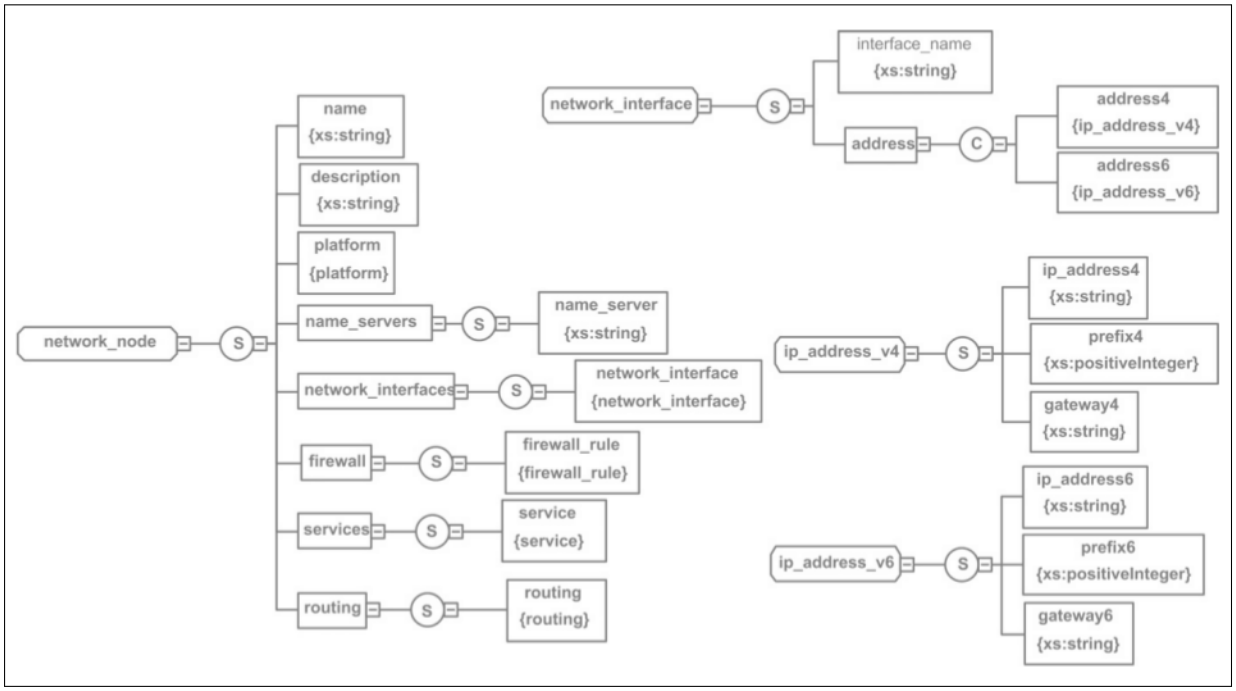

Figure A.10 – *Schéma XML de NNDL d'un nœud réseau [83]*

Annexe B

Spécification OCNI (en JSON)

B.1 Ressource NetworkNode

Le *NetworkNode* définit un nœud ou service de communication du *Cloud Networking*.

B.1.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
location	String	0..1	Mutable	Emplacement actuel de l'instance
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.1 – *Attributs de NetworkNode*

B.1.2 Commandes

Commande 'READ'

Requête

```
1 GET networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
2 Host: localhost:80
3 Accept: application/occi+json
4 User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
1 HTTP/1.1  200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
5 {
```

```
    "resources": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#networknode",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NetworkNode resource",
            "summary": "This is a NetworkNode resource",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "networknode": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "location": "48.624456,2.443857",
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "links": []
        }
    ]
}
```

Commande 'CREATE'

Requête

```
POST networknode HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "resources": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#networknode",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NetworkNode resource",
            "summary": "This is a NetworkNode resource",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "networknode": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "location": "48.624456,2.443857",
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "links": []
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

```
{
    "X-OCCI-Location": [
        "http://localhost:80/networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "resources": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#networknode",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NetworkNode resource",
            "summary": "This is a NetworkNode resource",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "networknode": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "location": "48.624456,2.443857",
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "links": []
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
DELETE networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

B.2 Ressource NetworkLink

Le *NetworkLink* définit un lien ou service de communication du *Cloud Networking*.

B.2.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance
bandwidth	bits/second	0..1	Mutable	La capacité de transfert de données de l'instance.
latency	Time	0..1	Mutable	Le temps nécessaire pour la livraison de paquets par l'instance.
jitter	Time	0..1	Mutable	Les variations du temps de la livraison de paquets par l'instance..
loss	%	0..1	Mutable	% des paquets perdus par l'instance.
routing_scheme	Enum unicast, multicast, broadcast, geocast	0..1	Mutable	Le type de transmission de l'instance.

Table B.2 – *Attributs de NetworkLink*

B.2.2 Commandes

Commande 'READ'

Requête

```
1 GET networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
2 Host: localhost:80
3 Accept: application/occi+json
4 User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
1 HTTP/1.1  200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
```

```
{
    "resources": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#networklink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NetworkLink resource",
            "summary": "This is a NetworkLink resource",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "networklink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active",
                        "bandwidth": "10G bit/sec",
                        "latency": "1microsecond",
                        "jitter": "",
                        "loss": "0.01%",
                        "routing_scheme": "unicast"
                    }
                }
            },
            "actions": [],
            "links": []
        }
    ]
}
```

Commande 'CREATE'

Requête

```
POST networklink HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "resources": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#networklink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NetworkLink resource",
            "summary": "This is a NetworkLink resource",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "networklink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active",
                        "bandwidth": "10G bit/sec",
                        "latency": "1microsecond",
                        "jitter": "",
                        "loss": "0.01%",
                        "routing_scheme": "unicast"
                    }
                }
            },
            "actions": [],
            "links": []
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "X-OCCI-Location": [
        "http://localhost:80/networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "resources": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#networklink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NetworkLink resource",
            "summary": "This is a NetworkLink resource",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "networklink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active",
                        "bandwidth": "10G bit/sec",
                        "latency": "1microsecond",
                        "jitter": "",
                        "loss": "0.01%",
                        "routing_scheme": "unicast"
                    }
                }
            },
            "actions": [],
            "links": []
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
DELETE networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
```

```
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
```

B.3 Ressource VirtualInfrastructure

Le *VirtualInfrastructure* définit une infrastructure virtuelle. Cette infrastructure peut être composé par plusieurs instances de ressources.

B.3.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance
resources	Resource	1..*	Mutable	Les instances de type *Resource* qui composent l'instance VirtualInfrastructure.
links	Link	1..*	Mutable	Les instances de type *Link* qui composent l'instance VirtualInfrastructure.

Table B.3 – *Attributs de VirtualInfrastructure*

B.3.2 Commandes

Commande 'READ'

Requête

```
1 GET virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
2 Host: localhost:80
3 Accept: application/occi+json
4 User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
1 HTTP/1.1  200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
5 {
6     "resources": [
7         {
8             "kind": "http://schemas.ogf.org/occi/ocni#virtualinfrastructure",
9             "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
10            "title": "VirtualInfrastructure resource",
```

```
            "summary": "This is a VirtualInfrastructure resource",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "virtualinfrastructure": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active",
                        "resources": [],
                        "links": []
                    }
                }
            },
            "actions": [],
            "links": []
        }
    ]
}
```

Commande 'CREATE'

Requête

```
POST virtualinfrastructure HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "resources": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#virtualinfrastructure",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "VirtualInfrastructure resource",
            "summary": "This is a VirtualInfrastructure resource",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "virtualinfrastructure": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active",
                        "resources": [],
                        "links": []
                    }
                }
            },
            "actions": [],
            "links": []
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "X-OCCI-Location": [
        "http://localhost:80/virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29"
```

```
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "resources": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#virtualinfrastructure",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "VirtualInfrastructure resource",
            "summary": "This is a VirtualInfrastructure resource",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "virtualinfrastructure": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active",
                        "resources": [],
                        "links": []
                    }
                }
            },
            "actions": [],
            "links": []
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
DELETE virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

B.4 Lien CNLink

Le *CNLink* relie deux instances de ressources l'une de type *'Compute'* et l'autre de type *'NetworkNode'*.

B.4.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.4 – *Attributs de CNLink*

B.4.2 Commandes

Commande 'READ'

Requête

```
GET cnlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#cnlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "CNLink link",
            "summary": "This is a CNLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "cnlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "compute/996ad860-2a9a-504f-8861-aeafd0b2ae01",
            "target": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Commande 'CREATE'

Requête

```
POST cnlink HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#cnlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "CNLink link",
            "summary": "This is a CNLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "cnlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "compute/996ad860-2a9a-504f-8861-aeafd0b2ae01",
            "target": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "X-OCCI-Location": [
        "http://localhost:80/cnlink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT cnlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#cnlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "CNLink link",
            "summary": "This is a CNLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "cnlink": {
                        "availability": [
                            {
                                "start": "08:00",
```

```
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "compute/996ad860-2a9a-504f-8861-aeafd0b2ae01",
            "target": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
DELETE cnlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

B.5 Lien CLLink

Le *CLLink* relie deux instances de ressources l'une de type *'Compute'* et l'autre de type *'NetworkLink'*.

B.5.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.5 – *Attributs de CLLink*

B.5.2 Commandes

Commande 'READ'

Requête

```
GET cllink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#cllink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "CLLink link",
            "summary": "This is a CLLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "cllink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "compute/996ad860-2a9a-504f-8861-aeafd0b2ae01",
            "target": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Commande 'CREATE'

Requête

```
POST cllink HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#cllink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "CLLink link",
            "summary": "This is a CLLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "cllink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
```

```
                                {
                                    "start": "14:00",
                                    "end": "18:00"
                                }
                            ],
                            "state": "active"
                        }
                    }
                },
                "actions": [],
                "source": "compute/996ad860-2a9a-504f-8861-aeafd0b2ae01",
                "target": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29",
                "rel": ""
            }
        ]
    }
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "X-OCCI-Location": [
        "http://localhost:80/cllink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT cllink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#cllink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "CLLink link",
            "summary": "This is a CLLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "cllink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "compute/996ad860-2a9a-504f-8861-aeafd0b2ae01",
            "target": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
1 DELETE cllink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
2 Host: localhost:80
3 Accept: application/occi+json
4 User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
1 HTTP/1.1 200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
```

B.6 Lien LSLink

Le *LSLink* relie deux instances de ressources l'une de type *'NetworkLink'* et l'autre de type *'Storage'*.

B.6.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.6 – *Attributs de LSLink*

B.6.2 Commandes

Commande 'READ'

Requête

```
1 GET lslink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
2 Host: localhost:80
3 Accept: application/occi+json
4 User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
1 HTTP/1.1 200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
5 {
6     "links": [
7         {
```

```
            "kind": "http://schemas.ogf.org/occi/ocni#lslink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "LSLink link",
            "summary": "This is a LSLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "lslink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "target": "storage/996ad860-2a9a-504f-8861-aeafd0b2ae02",
            "rel": ""
        }
    ]
}
```

Commande 'CREATE'

Requête

```
POST lslink HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#lslink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "LSLink link",
            "summary": "This is a LSLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "lslink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "target": "storage/996ad860-2a9a-504f-8861-aeafd0b2ae02",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

```
{
    "X-OCCI-Location": [
        "http://localhost:80/lslink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT lslink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#lslink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "LSLink link",
            "summary": "This is a LSLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "lslink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "target": "storage/996ad860-2a9a-504f-8861-aeafd0b2ae02",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
DELETE lslink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

B.7 Lien SVLink

Le *SVLink* relie deux instances de ressources l'une de type *'Storage'* et l'autre de type *'VirtualInfrastructure'*.

B.7.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.7 – *Attributs de SVLink*

B.7.2 Commandes

Commande 'READ'

Requête

```
GET svlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#svlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "SVLink link",
            "summary": "This is a SVLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "svlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
```

```
30              "actions": [],
31              "source": "storage/996ad860-2a9a-504f-8861-aeafd0b2ae02",
32              "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
33              "rel": ""
34          }
35      ]
36  }
```

Commande 'CREATE'

Requête

```
1  POST svlink HTTP/1.1
2  Host: localhost:80
3  Accept: application/occi+json
4  User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
5  {
6      "links": [
7          {
8              "kind": "http://schemas.ogf.org/occi/ocni#svlink",
9              "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
10             "title": "SVLink link",
11             "summary": "This is a SVLink link",
12             "mixins": [],
13             "attributes": {
14                 "ocni": {
15                     "svlink": {
16                         "availability": [
17                             {
18                                 "start": "08:00",
19                                 "end": "12:00"
20                             },
21                             {
22                                 "start": "14:00",
23                                 "end": "18:00"
24                             }
25                         ],
26                         "state": "active"
27                     }
28                 }
29             },
30             "actions": [],
31             "source": "storage/996ad860-2a9a-504f-8861-aeafd0b2ae02",
32             "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
33             "rel": ""
34         }
35     ]
36 }
```

Réponse

```
1 HTTP/1.1  200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
5 {
6     "X-OCCI-Location": [
7         "http://localhost:80/svlink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
8     ]
9 }
```

Commande 'UPDATE'

Requête

```
1  PUT svlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
2  Host: localhost:80
3  Accept: application/occi+json
4  User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
5  {
6      "links": [
7          {
8              "kind": "http://schemas.ogf.org/occi/ocni#svlink",
```

```
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "SVLink link",
            "summary": "This is a SVLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "svlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "storage/996ad860-2a9a-504f-8861-aeafd0b2ae02",
            "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
DELETE svlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

B.8 Lien NVLink

Le *NVLink* relie deux instances de ressources l'une de type *'NetworkNode'* et l'autre de type *'VirtualInfrastructure'*.

B.8.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.8 – *Attributs de NVLink*

B.8.2 Commandes

Commande 'READ'

Requête

```
GET nvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#nvlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NVLink link",
            "summary": "This is a NVLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "nvlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae07",
            "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Commande 'CREATE'

Requête

```
POST nvlink HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#nvlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NVLink link",
            "summary": "This is a NVLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "nvlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae07",
            "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "X-OCCI-Location": [
        "http://localhost:80/nvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT nvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#nvlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NVLink link",
            "summary": "This is a NVLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "nvlink": {
                        "availability": [
                            {
                                "start": "08:00",
```

```
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae07",
            "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
DELETE nvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

B.9 Lien CVLink

Le *CVLink* relie deux instances de ressources l'une de type *'Compute'* et l'autre de type *'VirtualInfrastructure'*.

B.9.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.9 – *Attributs de CVLink*

B.9.2 Commandes

Commande 'READ'

Requête

```
GET cvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#cvlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "CVLink link",
            "summary": "This is a CVLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "cvlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "compute/996ad860-2a9a-504f-8861-aeafd0b2ae02",
            "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Commande 'CREATE'

Requête

```
POST cvlink HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#cvlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "CVLink link",
            "summary": "This is a CVLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "cvlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
```

```
                                {
                                    "start": "14:00",
                                    "end": "18:00"
                                }
                            ],
                            "state": "active"
                        }
                    }
                },
                "actions": [],
                "source": "compute/996ad860-2a9a-504f-8861-aeafd0b2ae02",
                "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
                "rel": ""
            }
        ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "X-OCCI-Location": [
        "http://localhost:80/cvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT cvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#cvlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "CVLink link",
            "summary": "This is a CVLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "cvlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "compute/996ad860-2a9a-504f-8861-aeafd0b2ae02",
            "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
1 DELETE cvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
2 Host: localhost:80
3 Accept: application/occi+json
4 User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
1 HTTP/1.1 200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
```

B.10 Lien LNLink

Le *LNLink* relie deux instances de ressources l'une de type *'NetworkLink'* et l'autre de type *'NetworkNode'*.

B.10.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.10 – *Attributs de LNLink*

B.10.2 Commandes

Commande 'READ'

Requête

```
1 GET lnlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
2 Host: localhost:80
3 Accept: application/occi+json
4 User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
1 HTTP/1.1 200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
5 {
6     "links": [
7         {
```

```
8               "kind": "http://schemas.ogf.org/occi/ocni#lnlink",
9               "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
10              "title": "LNLink link",
11              "summary": "This is a LNLink link",
12              "mixins": [],
13              "attributes": {
14                  "ocni": {
15                      "lnlink": {
16                          "availability": [
17                              {
18                                  "start": "08:00",
19                                  "end": "12:00"
20                              },
21                              {
22                                  "start": "14:00",
23                                  "end": "18:00"
24                              }
25                          ],
26                          "state": "active"
27                      }
28                  }
29              },
30              "actions": [],
31              "source": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae08",
32              "target": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29",
33              "rel": ""
34          }
35      ]
36  }
```

Commande 'CREATE'

Requête

```
1  POST lnlink HTTP/1.1
2  Host: localhost:80
3  Accept: application/occi+json
4  User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
5  {
6      "links": [
7          {
8              "kind": "http://schemas.ogf.org/occi/ocni#lnlink",
9              "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
10             "title": "LNLink link",
11             "summary": "This is a LNLink link",
12             "mixins": [],
13             "attributes": {
14                 "ocni": {
15                     "lnlink": {
16                         "availability": [
17                             {
18                                 "start": "08:00",
19                                 "end": "12:00"
20                             },
21                             {
22                                 "start": "14:00",
23                                 "end": "18:00"
24                             }
25                         ],
26                         "state": "active"
27                     }
28                 }
29             },
30             "actions": [],
31             "source": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae08",
32             "target": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29",
33             "rel": ""
34         }
35     ]
36 }
```

Réponse

```
1 HTTP/1.1  200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
```

```
{
    "X-OCCI-Location": [
        "http://localhost:80/lnlink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT lnlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#lnlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "LNLink link",
            "summary": "This is a LNLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "lnlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae08",
            "target": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
DELETE lnlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

B.11 Lien LVLink

Le *LVLink* relie deux instances de ressources l'une de type *'NetworkLink'* et l'autre de type *'VirtualInfrastructure'*.

B.11.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.11 – *Attributs de LVLink*

B.11.2 Commandes

Commande 'READ'

Requête

```
GET lvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#lvlink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "LVLink link",
            "summary": "This is a LVLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "lvlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
```

```
30              "actions": [],
31              "source": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae08",
32              "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
33              "rel": ""
34          }
35      ]
36  }
```

Commande 'CREATE'

Requête

```
1  POST lvlink HTTP/1.1
2  Host: localhost:80
3  Accept: application/occi+json
4  User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
5  {
6      "links": [
7          {
8              "kind": "http://schemas.ogf.org/occi/ocni#lvlink",
9              "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
10             "title": "LVLink link",
11             "summary": "This is a LVLink link",
12             "mixins": [],
13             "attributes": {
14                 "ocni": {
15                     "lvlink": {
16                         "availability": [
17                             {
18                                 "start": "08:00",
19                                 "end": "12:00"
20                             },
21                             {
22                                 "start": "14:00",
23                                 "end": "18:00"
24                             }
25                         ],
26                         "state": "active"
27                     }
28                 }
29             },
30             "actions": [],
31             "source": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae08",
32             "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
33             "rel": ""
34         }
35     ]
36 }
```

Réponse

```
1 HTTP/1.1  200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
5 {
6     "X-OCCI-Location": [
7         "http://localhost:80/lvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
8     ]
9 }
```

Commande 'UPDATE'

Requête

```
1  PUT lvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
2  Host: localhost:80
3  Accept: application/occi+json
4  User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
5  {
6      "links": [
7          {
8              "kind": "http://schemas.ogf.org/occi/ocni#lvlink",
```

```
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "LVLink link",
            "summary": "This is a LVLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "lvlink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networklink/996ad860-2a9a-504f-8861-aeafd0b2ae08",
            "target": "virtualinfrastructure/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
DELETE lvlink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

B.12 Lien NSLink

Le *NSLink* relie deux instances de ressources l'une de type *'NetworkNode'* et l'autre de type *'Storage'*.

B.12.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.12 – *Attributs de NSLink*

B.12.2 Commandes

Commande 'READ'

Requête

```
GET nslink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#nslink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NSLink link",
            "summary": "This is a NSLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "nslink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "target": "storage/996ad860-2a9a-504f-8861-aeafd0b2ae03",
            "rel": ""
        }
    ]
}
```

Commande 'CREATE'

Requête

```
POST nslink HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#nslink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NSLink link",
            "summary": "This is a NSLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "nslink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "target": "storage/996ad860-2a9a-504f-8861-aeafd0b2ae03",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "X-OCCI-Location": [
        "http://localhost:80/nslink/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT nslink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#nslink",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "NSLink link",
            "summary": "This is a NSLink link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "nslink": {
                        "availability": [
                            {
                                "start": "08:00",
```

```
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "networknode/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "target": "storage/996ad860-2a9a-504f-8861-aeafd0b2ae03",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
DELETE nslink/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

B.13 Lien AALink

Le *AALink* relie deux instances de ressources de même type "*A*" ("*A*" peut être une instance de ressource de type *Compute*, *NetworkLink*, *NetworkNode*, *VirtualInfrastructure* ou *Storage*).

B.13.1 Attributs

Attribut	Type	Multiplicité	Mutabilité	Description
availability	(Time, Time)	0..*	Mutable	Les proportions du temps cette entité se trouve dans une état de fonctionnement
state	Enum {active, inactive}	1	Immutable	L'état actuel de l'instance

Table B.13 – *Attributs de AALink*

B.13.2 Commandes

Commande 'READ'

Requête

```
GET {aa}link/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#{aa}link",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "{AA}Link link",
            "summary": "This is a {AA}Link link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "aalink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
                            {
                                "start": "14:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "{A}/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "target": "{A}/996ad860-2a9a-504f-8861-aeafd0b2ae03",
            "rel": ""
        }
    ]
}
```

Commande 'CREATE'

Requête

```
POST {aa}link HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#{aa}link",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "{AA}Link link",
            "summary": "This is a {AA}Link link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "aalink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "12:00"
                            },
```

```
                                {
                                    "start": "14:00",
                                    "end": "18:00"
                                }
                            ],
                            "state": "active"
                        }
                    }
                },
                "actions": [],
                "source": "{A}/996ad860-2a9a-504f-8861-aeafd0b2ae29",
                "target": "{A}/996ad860-2a9a-504f-8861-aeafd0b2ae03",
                "rel": ""
            }
        ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
{
    "X-OCCI-Location": [
        "http://localhost:80/{aa}link/996ad860-2a9a-504f-8861-aeafd0b2ae29"
    ]
}
```

Commande 'UPDATE'

Requête

```
PUT {aa}link/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
Host: localhost:80
Accept: application/occi+json
User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
{
    "links": [
        {
            "kind": "http://schemas.ogf.org/occi/ocni#{aa}link",
            "id": "996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "title": "{AA}Link link",
            "summary": "This is a {AA}Link link",
            "mixins": [],
            "attributes": {
                "ocni": {
                    "aalink": {
                        "availability": [
                            {
                                "start": "08:00",
                                "end": "18:00"
                            }
                        ],
                        "state": "active"
                    }
                }
            },
            "actions": [],
            "source": "{A}/996ad860-2a9a-504f-8861-aeafd0b2ae29",
            "target": "{A}/996ad860-2a9a-504f-8861-aeafd0b2ae03",
            "rel": ""
        }
    ]
}
```

Réponse

```
HTTP/1.1  200 OK
Server: pyocni-server
Content-Type: application/occi+json; charset=utf-8
Content-Length: xxx
```

Commande 'DELETE'

Requête

```
1 DELETE {aa}link/996ad860-2a9a-504f-8861-aeafd0b2ae29 HTTP/1.1
2 Host: localhost:80
3 Accept: application/occi+json
4 User-Agent: curl/7.13.1 (powerpc-apple-darwin8.0) libcurl/7.13.1 OpenSSL/0.9.7b zlib/1.2.2
```

Réponse

```
1 HTTP/1.1  200 OK
2 Server: pyocni-server
3 Content-Type: application/occi+json; charset=utf-8
4 Content-Length: xxx
```

Annexe C

Projets de recherche

C.1 SAIL

SAIL *"Scalable and Adaptive Internet Solutions"* est un projet européen FP7[52]. Il vise à définir une architecture et à mettre en œuvre une solution Cloud Networking au niveau Intra et Inter DataCenter. La solution utilise les frameworks de Cloud Computing existants comme OpenStack tout en intégrant les solutions de networking comme les SDNs.

C.2 CompatibleOne

CompatibleOne est un projet français [26] dont l'objectif est la modélisation et la mise en œuvre d'un broker Cloud Computing Open Source. Ce courtier sert d'intermédiaire entre les consommateurs et les divers fournisseurs du Cloud Computing (IaaS, PaaS, SaaS, NaaS, Public/Private/Hybrid...).

C.3 Marguerite

Marguerite est un projet régional Île-de-France. Son objectif est la modélisation et la mise en œuvre d'un service Cloud Computing qui offre aux étudiants, lycéens et chercheurs franciliens des ressources de stockage Cloud, de machines virtuelles et de logicielles à la demande.

C.4 Mobesens

Mobesens, *Mobility for Long Term Water Quality Monitoring*, est un projet européen[41]. Il vise la définition d'une architecture et la mise en œuvre d'un middleware de stockage, transformation, indexation, analyse et visualisation des données d'un réseau de capteurs pour la surveillance de la qualité de l'eau.

C.5 EASI-Clouds

EASI-Clouds, *Extendable Architecture and Service Infrastructure for Cloud-Aware Software*, est un projet européen ITEA[30]. L'objectif de ce projet est de pouvoir fédérer les ressources et services de différents fournisseurs de Cloud. Cette fédération passe par la simplification et la sécurisation des environnements d'exécution et de développement d'application. Un deuxième objectif de ce projet concerne la gestion des SLAs à un niveau avancé en négociant les niveaux de services au pré de chaque fournisseur.

C.6 FIT

FIT, *Future Internet of Things*, est un projet français[32]. L'objectif de FIT est de fournir une infrastructure expérimentale, fédérée et compétitive avec une visibilité internationale et un large panel de clients. Cette infrastructure est fournie avec un ensemble de composants complémentaires qui permettent l'expérimentation sur des services innovants pour les utilisateurs académiques et industriels. Le projet donnera aux acteurs français un moyen d'expérimentation innovant pour contribuer à la construction de l'internet du futur.

C.7 XLcloud

XLcloud, *High Performance Cloud Computing Platform*, est un projet français[54]. Ce projet vise à définir et démontrer les principes du HPC en tant que service de type Cloud. La plateforme Cloud fournis par XLcloud doit satisfaire toutes les exigences des applications qui impliquent des calculs hautes performances.

C.8 4ward

4ward est un projet européen FP7[21] dont l'objectif est la définition d'architecture et la mise en œuvre d'une solution de virtualisation des réseaux. L'idée est de concevoir une nouvelle architecture de réseau pour l'Internet de nouvelle génération.Ce projet ne présuppose aucune architecture de base, ni aucun protocole de base.

Annexe D

DataCenter NCF

Le DataCenter Network and Cloud Federation (NCF) est une plateforme expérimentale qui vise à fusionner les concepts, les technologies et les architectures du monde réseaux et du Cloud Computing sur un système commun.

Les utilisateurs NCF peuvent demander des ressources virtuelles réseau et des services de Cloud Computing (informatique, stockage ou réseau) pour déployer et valider leurs propres solutions et architectures.

Nous nous basons sur la plateforme pour assurer la convergence des réseaux et des principes du Cloud et pour développer des solutions Cloud ouvertes qui éliminent les obstacles d'interopérabilité et de portabilité provenant des fournisseurs.

Des propriétés d'élasticité et de mise à l'échelle automatique sont intégrées dans la plateforme pour permettre la fédération et la composition dynamique des nuages privés et hybrides. L'objectif à long terme est d'offrir des infrastructures, des plateformes et des logiciels comme un service (concepts de Software as a Service, Platform as a Service et Infrastucture as a Service) pour soutenir les développeurs d'applications et les fournisseurs par le biais d'interfaces de programmation ouvertes qui procurent des services ouverts, déverrouillés, portables et interopérables.

L'évolution de la plateforme se fait progressivement en utilisant une approche d'auto-expansion afin de répondre aux attentes d'ouverture, d'extensibilité, d'interopérabilité et de confiance demandées par les utilisateurs et les intervenants du domaine.

La figure D.1 schématise en détail tous les composants de NCF et globalement elle contient :

- 608 CPU Cores
- 3.2 To RAM
- 2 serveurs GPU (4 Nvidia TESLA)
- Serveur stockage : NAS 72 To
- Serveur stockage : SAN 48 To

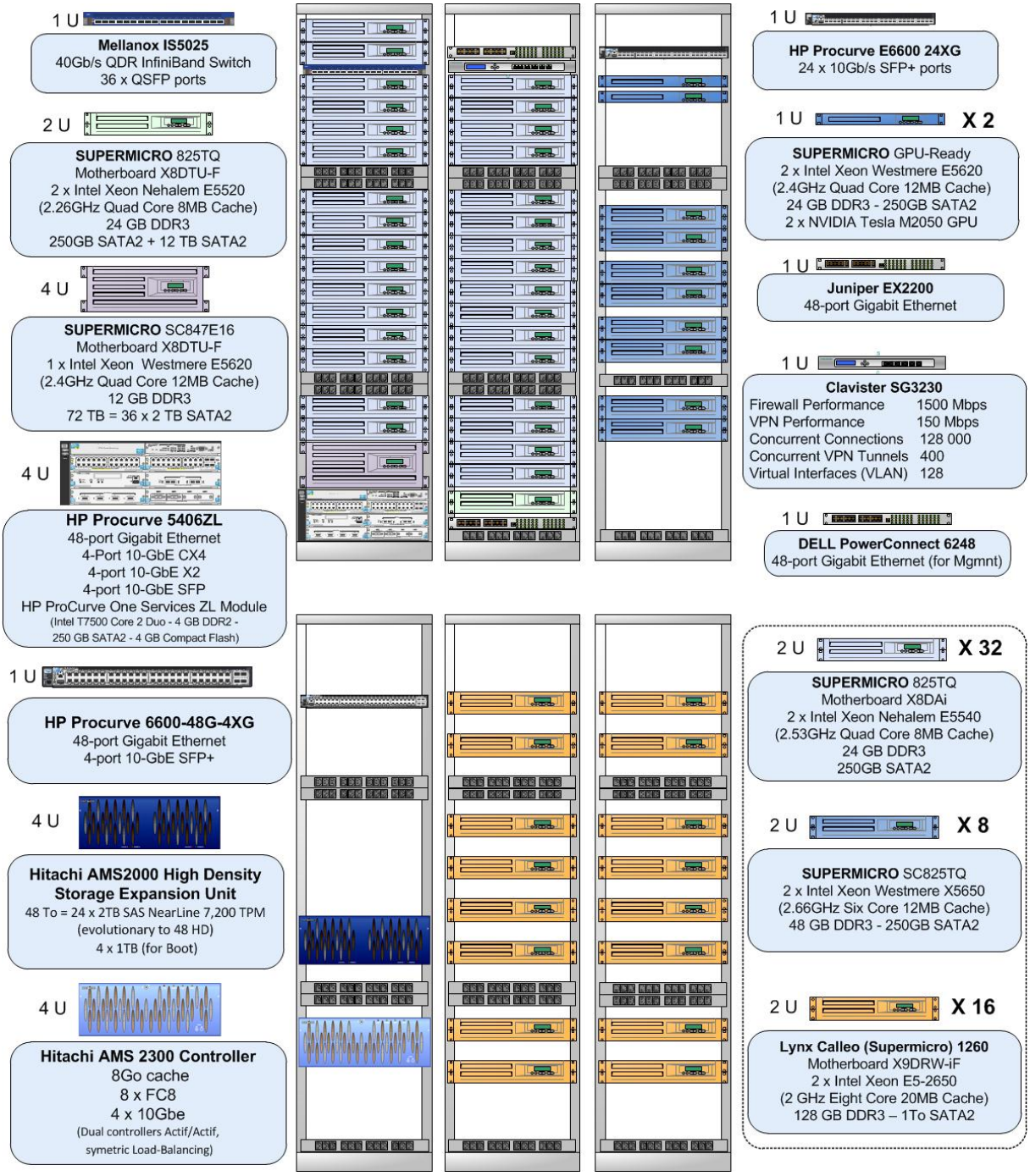

Figure D.1 – *DataCenter NCF - Network and Cloud Federation*

Bibliographie

[1] Spanning Tree Protocol (STP). Technical report, IEEE 802.1D - IEEE Standard for Local and Metropolitan Area Networks—Media access control (MAC) Bridges (Incorporates IEEE 802.1t-2001 and IEEE 802.1w), 2004.

[2] Provider Backbone Bridges (PBB). Technical report, IEEE Draft Standard for Local and Metropolitan Area Networks– Virtual Bridged Local Area Networks – Amendment 6 : (DRAFT Amendment to IEEE Std 802.1Q -REV), 2008.

[3] 802.1Qbg - Edge Virtual Bridging. Technical report, IEEE Standard for Local and Metropolitan Area Networks, www.ieee802.org/1/pages/802.1bg.html, 2009.

[4] Amazon EC2 - Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`, 2009. [Online. Last accessed : 2014-06-01].

[5] Amazon S3 - Amazon Simple Storage Service. `http://aws.amazon.com/s3/`, 2009. [Online. Last accessed : 2014-06-01].

[6] Crossbow : Network Virtualization and Resource Control. `http://hub.opensolaris.org/bin/view/Project+crossbow/`, 2009. [Online. Last accessed : 2011-10-10].

[7] Geant2 - Gigabit European Advanced Network Technology 2. `http://www.geant2.net/`, 2009. [Online. Last accessed : 2011-10-10].

[8] NML-WG - Network Mark-up Language Working Group. `http://www.ogf.org/gf/group_info/view.php?group=nml-wg`, 2009. [Online. Last accessed : 2011-10-10].

[9] NMWG - OGF Network Measurement Working Group. `http://nmwg.internet2.edu/`, 2009. [Online. Last accessed : 2011-10-10].

[10] Open Cloud Manifesto. Technical report, Open Cloud Manifesto Working Group, 2009.

[11] perfSONAR - Infrastructure for network performance monitoring. `http://www.perfsonar.net/`, 2009. [Online. Last accessed : 2011-10-10].

[12] VXDL - Virtual Resources and Interconnection Networks Description Language. `http://www.ens-lyon.fr/LIP/RESO/Software/vxdl/home.html`, 2009. [Online. Last accessed : 2011-10-10].

[13] Vyatta Core software(VC). http://www.vyatta.org/, 2009. [Online. Last accessed : 2014-06-01].

[14] Open Data Center Alliance. http://www.opendatacenteralliance.org, 2010. [Online. Last accessed : 2013-10-25].

[15] Open vSwitch - a production quality, multilayer virtual switch. http://openvswitch.org/, 2010. [Online. Last accessed : 2014-06-01].

[16] GLIF - The Global Lambda Integrated Facility. http://www.glif.is/, 2011. [Online. Last accessed : 2011-10-10].

[17] NDL - Network Description Language. http://www.science.uva.nl/research/sne/ndl/, 2011. [Online. Last accessed : 2011-10-10].

[18] Open Cloud Initiative, 2011.

[19] Shortest Path Bridging. Technical report, IEEE SA - 802.1aq-2012 - IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 20, 2012.

[20] Software-Defined Networking : The New Norm for Networks. Technical report, ONF Open Networking Foundation, 2012.

[21] 4ward Project - The European FP7 project. http://www.4ward-project.eu/, 2013. [Online. Last accessed : 2013-10-25].

[22] AMQP - Advanced Message Queuing Protocol. http://www.amqp.org/product/architecture, 2013. [Online. Last accessed : 2012-10-10].

[23] CDMI - Cloud Data Management Interface. http://www.snia.org/cdmi, 2013. [Online. Last accessed : 2013-10-15].

[24] CNG-Manager. Github repository. https://github.com/MarouenMechtri/CNG-Manager, 2013. [Online. Last accessed : 2014-06-01].

[25] CNG-Manager. Second Github repository. https://github.com/jordan-developer/CNG-Manager, 2013. [Online. Last accessed : 2014-06-01].

[26] CompatibleOne Project - The Open cloud Broker. http://compatibleone.org/bin/view/Main/, 2013. [Online. Last accessed : 2013-10-25].

[27] CSA - Cloud Security Alliance. https://cloudsecurityalliance.org/, 2013. [Online. Last accessed : 2013-10-15].

[28] CSCC - Cloud Standards Customer Council. http://www.cloud-council.org/, 2013. [Online. Last accessed : 2013-10-15].

[29] DMTF - Distributed Management Task Force. http://www.dmtf.org/, 2013. [Online. Last accessed : 2013-10-15].

[30] EASI-Clouds : Extendable Architecture and Service Infrastructure for Cloud-Aware Software. http://easi-clouds.eu/, 2013. [Online. Last accessed : 2014-06-01].

[31] ETSI - ETSI TC Cloud/Grid. http://www.etsi.org/technologies-clusters/technologies/grid-and-cloud-computing, 2013. [Online. Last accessed : 2013-10-15].

[32] FIT : Future Internet of Things. http://fit-equipex.fr/, 2013. [Online. Last accessed : 2014-06-01].

[33] Gartner Magic Quadrant Cloud IaaS - 2010. http://blogs.gartner.com/lydia_leong/2011/01/03/qualifying-for-the-next-cloud-iaas-magic-quadrant/, 2013. [Online. Last accessed : 2014-06-01].

[34] Gartner Magic Quadrant Cloud IaaS - 2013. http://blogs.gartner.com/lydia_leong/2013/08/21/the-2013-cloud-iaas-magic-quadrant/, 2013. [Online. Last accessed : 2014-06-01].

[35] Gartner WebSite. http://www.gartner.com/technology/home.jsp, 2013. [Online. Last accessed : 2014-06-01].

[36] Geni Project - Exploring networks of the future. http://www.geni.net/, 2013. [Online. Last accessed : 2013-10-25].

[37] IETF - The Internet Engineering Task Force. http://www.ietf.org/, 2013. [Online. Last accessed : 2013-10-15].

[38] Is Amazon the Official Cloud Standard ? http://www.cloudswitch.com/page/is-amazon-the-official-cloud-standard, 2013. [Online. Last accessed : 2013-10-15].

[39] libNetVirt Github repository. https://github.com/danieltt/libnetvirt, 2013. [Online. Last accessed : 2014-06-01].

[40] MEF - Metro Ethernet Forum. http://metroethernetforum.org/, 2013. [Online. Last accessed : 2013-10-15].

[41] Mobesens Project - Mobility for Long Term Water Quality Monitoring. http://www.mobesens.eu, 2013. [Online. Last accessed : 2013-10-25].

[42] NIST - National Institute of Standards and Technology. http://www.nist.gov/, 2013. [Online. Last accessed : 2013-10-15].

[43] OASIS - Organization for the Advancement of Structured Information Standards. https://www.oasis-open.org/, 2013. [Online. Last accessed : 2013-10-15].

[44] OCC - Open Cloud Consortium. http://opencloudconsortium.org/, 2013. [Online. Last accessed : 2013-10-15].

[45] OCCI - Open Cloud Computing Interface. http://occi-wg.org/, 2013. [Online. Last accessed : 2013-10-15].

[46] OCCI - Open Cloud Computing Interface : Implementaiton List. http://occi-wg.org/community/implementations/, 2013. [Online. Last accessed : 2013-10-15].

[47] OGF - Open Grid Forum. http://www.ogf.org/, 2013. [Online. Last accessed : 2013-10-15].

[48] ONF - Open Networking Foundation. http://www.opennetworking.org, 2013. [Online. Last accessed : 2013-10-26].

[49] pyCMBS (Python Cloud Message Mbrokering Service) Github repository. https://github.com/jordan-developer/pyCMBS, 2013. [Online. Last accessed : 2014-06-01].

[50] pyOCNI (Python Open Cloud Networking Interface) Github repository. https://github.com/jordan-developer/PyOCNI, 2013. [Online. Last accessed : 2014-06-01].

[51] pyOCNI (Python Open Cloud Networking Interface) OCCI Website reference. http://occi-wg.org/2012/02/20/occi-pyocni/, 2013. [Online. Last accessed : 2014-06-01].

[52] SAIL Project - Scalable and Adaptive Internet Solutions. An European FP7 project. http://www.sail-project.eu/, 2013. [Online. Last accessed : 2013-10-25].

[53] SNIA - Storage Networking Industry Association. http://www.snia.org/, 2013. [Online. Last accessed : 2013-10-15].

[54] XLcloud- High Performance Cloud Computing Platform. http://www.xlcloud.org/, 2013. [Online. Last accessed : 2014-06-01].

[55] ZeroMQ - The Intelligent Transport Layer. http://www.zeromq.org/, 2013. [Online. Last accessed : 2012-10-10].

[56] Aerohive. The Benefits of Cloud Networking. Technical report, 2013.

[57] Bengt Ahlgren, Pedro Aranda, Prosper Chemouil, Sara Oueslati, Luis Correia, Holger Karl, Michael Söllner, and Annikki Welin. Content, connectivity, and cloud : ingredients for the network of the future. *IEEE Communications Magazine*, 49(7) :62–70, 2011.

[58] Mohiuddin Ahmed, Abu Sina, Raju Chowdhury, Mustaq Ahmed, and Mahmudul Hasan Rafee. An Advanced Survey on Cloud Computing and State-of-the-art Research Issues. *IJCSI International Journal of Computer Science Issues*, 9(1) :201–207, 2012.

[59] Miha Ahronovitz, Dustin Amrhein, Patrick Anderson, Andrew De Andrade, and Joe Armstrong. Cloud Computing Use Cases Whitepaper, 2010.

[60] F. Anhalt, G. Koslovski, M. Pasin, J. Gelas, and P. Vicat-Blanc Primet. Les infrastructures virtuelles à la demande pour un usage flexible de l'internet. In *JDIR 09 : Journees Doctorales en Informatique et Reseaux*, Belfort, France, February 2009.

[61] ARISTA. Cloud Networking : A network approach that meets the requirements of cloud computing. Technical report, 2011.

[62] ARISTA. Impact of Virtualization on Cloud Networking. Technical report, 2013.

[63] Arista. Software Defined Cloud Networking. Technical report, ARISTA Networks, 2013.

[64] By Armbrust, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, and Ariel Rabkin. A view of cloud computing. *Communications of the ACM*, 53(4) :50–58, 2010.

[65] Michael Armbrust, Anthony D Joseph, Randy H Katz, and David A Patterson. Above the Clouds : A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[66] Bruno Nunes Astuto, Marc Mendonça, Xuan Nam Nguyen, Katia Obraczka, and Thierry Turletti. A Survey of Software-Defined Networking : Past, Present, and Future of Programmable Networks. October 2013.

[67] Siamak Azodolmolky, Philipp Wieder, and Ramin Yahyapour. Cloud computing networking : challenges and opportunities for innovations. *IEEE Communications Magazine*, 51(7), July 2013.

[68] Lee Badger, Tim Grance, Robert Patt Corner, and Jeff Voas. Cloud Computing Synopsis and Recommendations. Technical report, National Institute of Standards and Technology, 2011.

[69] Lounes Baleh, Lucian Suciu, and Jean-Marie Bonnin. Enriched connectivity-as-a-service for dynamic Mobile-Cloud. In *2012 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 655–660. IEEE, January 2012.

[70] Theophilus Benson and Aditya Akella. CloudNaaS : A Cloud Networking Platform for Enterprise Applications. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, pages 1–13. ACM Press, 2011.

[71] Andreas Berl, Roman Weidlich, Michael Schrank, Helmut Hlavacs, and H. de Meer. Network virtualization in future home environments. *Integrated Management of Systems, Services, Processes and People in IT*, 5841 :177–190, 2009.

[72] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud Computing : A Study Of Infrastructure Aa A Service (IAAS). *International Journal of Engineering and Information Technology*, 2(1) :60–63, 2010.

[73] Nabil Bitar, Steven Gringeri, and Tiejun Xia. Technologies and protocols for data center and cloud networking. *IEEE Communications Magazine*, 51(9), September 2013.

[74] Don Boulia. Platform as a Service : The IBM point of view. Technical report, IBM Corporation, 2012.

[75] N. M. Mosharaf Kabir Chowdhury Boutaba and Raouf. Network Virtualization : State of the Art and Research Challenges. *Communications Magazine*, 3(July) :20–26, 2009.

[76] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms : Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6) :599–616, 2009.

[77] Dennis Cai and Sai Natarajan. The Evolution of the Carrier Cloud Networking. In *2013 IEEE Seventh International Symposium on Service-Oriented System Engineering*, pages 286–291. IEEE, March 2013.

[78] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization, 2010.

[79] Yu-Hunag Chu, Yao-Ting Chen, Yu-Chieh Chou, and Min-Chi Tseng. A simplified cloud computing network architecture using future internet technologies. In *2011 13th Asia-Pacific Network Operations and Management Symposium*, pages 1–4. IEEE, September 2011.

[80] CISCO. Cisco FabricPath. Technical report, Cisco FabricPath : Introducing Routing Concepts at Layer 2, 2010.

[81] Salvatore D Agostino, Miha Ahronovitz, Joe Armstrong, Rizwan Ahmad, and All. Moving to the Cloud. Technical Report February, Cloud Computing Use Cases Discussion Group, 2011.

[82] B. Davie and J. Gross. A Stateless Transport Tunneling Protocol for Network Virtualization (STT). Technical report, IETF draft-davie-stt-04, 2013.

[83] Dalibor Dobrilovic, Zeljko Stojanov, Borislav Odadzic, and Branko Markoski. Using Network Node Description Language for modeling networking scenarios. *Advances in Engineering Software*, 43(1) :53–64, 2012.

[84] A. Doria, J. Hadi Salim, R. Haas, H. Khosravi, W. Wang, R. Gopal, and J. Halpern. Forwarding and Control Element Separation (ForCES) Protocol Specification. Technical report, IETF RFC 5810, 2010.

[85] Qiang Duan, Yuhong Yan, and Athanasios V. Vasilakos. A Survey on Service-Oriented Network Virtualization Toward Convergence of Networking and Cloud Computing. *IEEE Transactions on Network and Service Management*, 9(4) :373–392, December 2012.

[86] Andy Edmonds, Alexander Papaspyrou, Thijs Metsch, Platform Computing, and Copyright Notice. Open Cloud Computing Interface - Core. Technical report, OGF, 2011.

[87] Equinix Group. Entreprise, cloud et datacentre. Technical report, 2013.

[88] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. The Locator/ID Separation Protocol (LISP). Technical report, IETF RFC 6830, 2013.

[89] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). Technical report, IETF RFC 2784www.ietf.org/rfc/rfc2784.txt, 2010.

[90] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The Past, Present, and Future of Software Defined Networking. In *Communications of the ACM*, 2013.

[91] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *2008 Grid Computing Environments Workshop*, volume abs/0901.0, pages 1–10. Ieee, Ieee, 2008.

[92] Borko Furht and Armando Escalante. Cloud Computing Fundamentals. In Borko Furht and Armando Escalante, editors, *Handbook of Cloud Computing*, number May, chapter 1, pages 3–19. Springer US, 2010.

[93] Borko Furht and Armando Escalante. *Handbook of Cloud Computing*, volume chapter 13. Springer US, 2010.

[94] Keke Gai and Saier Li. Towards Cloud Computing : A Literature Review on Cloud Computing and Its Development Trends. In *2012 Fourth International Conference on Multimedia Information Networking and Security*, pages 142–146. IEEE, 2012.

[95] Vishy Ganti, Vince Lubsey, Mrigank Shekhar, and Chris Swan. OPEN DATA CENTER ALLIANCE USAGE MODEL : Software-Defined Networking. Technical report, Open Data Center Alliance, 2013.

[96] M. Ghijsen, J. van der Ham, P. Grosso, C. Dumitru, H. Zhu, Z. Zhao, and C. de Laat. A semantic-web approach for modeling computing infrastructures. *Computers and Electrical Engineering*, 2013.

[97] Mattijs Ghijsen, Jeroen van der Ham, Paola Grosso, and Cees de Laat. Towards an Infrastructure Description Language for Modeling Computing Infrastructures. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 207–214. IEEE, July 2012.

[98] Jeroen_van_der Ham, Freek Dijkstra, Roman Lapacz, and Jason Zurawski. Network Markup Language Base Schema. Technical report, OGF - NML-WG, 2012.

[99] David Hilley. Cloud Computing : A Taxonomy of Platform and Infrastructure-level Offerings. Technical Report April, Georgia Institute of Technology, 2009.

[100] C N Höfer and G Karagiannis. Cloud computing services : taxonomy and comparison. *Journal of Internet Services and Applications*, 2(2) :81–94, 2011.

[101] HP. Reducing network complexity, boosting performance with HP IRF technology. Technical report, HP White Paper http ://h17007.www1.hp.com/docs/reports/irf.pdf, 2010.

[102] Michael Jarschel, Simon Oechsner, Daniel Schlosser, Rastin Pries, Sebastian Goll, and Phuoc Tran-Gia. Modeling and performance evaluation of an OpenFlow architecture. In *2011 23rd International Teletraffic Congress (ITC)*, pages 1–7. University of Wurzburg, Institute of Computer Science, Wurzburg, Germany, IEEE, 2011.

[103] Inc Juniper Networks. Juniper Networks QFabric System. Technical report, 2013.

[104] Qura-tul-ain Khan, Shahid Naseem, Fahad Ahmad, and M Saleem Khan. Usage & Issues of Cloud Computing Techniques. *International Journal of Scientific Engineering Research*, 3(5) :1–7, 2012.

[105] G. Koslovski, P. Vicat-Blanc Primet, and A. S. Charão. Vxdl : Virtual resources and interconnection networks descriptionlanguage. In *GridNets 2008*, Oct. 2008.

[106] DS Kushwaha and A Maurya. Cloud Computing-A Tool For Future. *International Journal Of Mathematics and Computer Research*, 1(1) :09–14, 2013.

[107] T.V Lakshman, T Nandagopal, R. Ramjee, K. Sabnani, and T. Woo. The SoftRouter Architecture. In *Proc ACM SIGCOMM Worshop on Hot Topics in Networking*, 2004.

[108] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network Innovation using OpenFlow : A Survey. In *IEEE Communications Surveys & Tutorials*, volume PP, pages 1–20, 2013.

[109] M. Lasserre and V. Kompella. Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling. Technical report, Alcatel-Lucent RFC 4762, 2007.

[110] Fang Liu, Jin Tong, Jian Mao, Robert Bohn, John Messina, Lee Badger, and Dawn Leaf. NIST Cloud Computing Reference Architecture. Technical Report 9, NIST National Institute of Standards and Technology, 2011.

[111] B. Lowekamp, B. Tierney, L. Cottrell, R. Hughes-Jones, T. Kielmann, and M. Swany. A hierarchy of network performance characteristics for grid applications and services. Technical report, 2004.

[112] Ting Luo Ting Luo and Shaohua Yu Shaohua Yu. Control and communication mechanisms of a SoftRouter. In *2009 Fourth International Conference on Communications and Networking in China*, 2009.

[113] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. VXLAN : A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. Technical report, Work in Progress (RFC), 2013.

[114] Nick Mckeown, Tom Anderson, Larry Peterson, Jennifer Rexford, Scott Shenker, and St Louis. OpenFlow : Enabling Innovation in Campus Networks. *ACM SIGCOMM Computer Communication Review*, 38(2) :69–74, 2008.

[115] D. McPherson and B. Dykes. VLAN Aggregation for Efficient IP Address Allocation. Technical report, IETF RFC 3069, 2001.

[116] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. Technical Report 6, National Institute of Standards and Technology, 2011.

[117] Thijs Metsch, Platform Computing, Andy Edmonds, and Copyright Notice. Open Cloud Computing Interface - Infrastructure. Technical report, OGF, 2011.

[118] Jim Metzler and Ashton Metzler. The 2012 Cloud Networking Report. Technical report, Webtorials Analyst Division, Cisco Systems, Ipanema Technologies, Netscout, 2012.

[119] Shivaji P Mirashe and N V Kalyankar. Cloud Computing. *Journal of computing*, 2(3) :78–82, 2010.

[120] G Motta, N Sfondrini, and D Sacco. Cloud Computing : An Architectural and Technological Overview. In *Service Sciences IJCSS - 2012 International Joint Conference on Service Sciences*, pages 23–27, 2012.

[121] Paul Murray and El. Cloud Networking Architecture Description. Technical report, SAIL Project, WP-D (Scalable and Adaptable Internet Solutions), 2012.

[122] Panagiotis Papadimitriou. Implementing Network Virtualization for a Future Internet. *Specialist*, (May), 2009.

[123] R. Perlman, D. Eastlake, D. Dutt, S. Gai, and A. Ghanwani. Routing Bridges (RBridges) : Base Protocol Specification. Technical report, IETF RFC 6325, 2011.

[124] Ben Pfaff, J Pettit, T Koponen, K Amidon, M Casado, and S Shenker. Extending Networking into the Virtualization Layer. *Interfaces*, VIII :60, 2009.

[125] Hareesh Puthalath and El. Description of Implemented Prototype. Technical report, SAIL Project, WP-D (Scalable and Adaptable Internet Solutions), 2013.

[126] Hassan Rajaei and Jeffrey Wappelhorst. Clouds & grids : a network and simulation perspective. In *Proceedings of the 14th Communications and Networking Symposium*, CNS '11, pages 143–150. Society for Computer Simulation International, 2011.

[127] Sameer Rajan and Apurva Jairath. Cloud Computing : The Fifth Generation of Computing. In *2011 IEEE 3rd International Conference on Communication Software and Networks*, volume 15, pages 1–4. Department of Computer science and technology, CUMT, City XuZhou, JiangSu, China, IEEE, 2011.

[128] Srinivasa Rao, Nageswara Rao, and E Kusuma Kumari. Cloud Computing : An Overview. *Journal of Theoretical and Applied Information Technology*, 7(5) :1–5, 2009.

[129] Y. Rekhter, T. Li, and S. Hares. A Border Gateway Protocol (BGP), 2006.

[130] Robert F Roggio, Tetiana Bilyayeva, and James R Comer. Everyday Cloud Computing with SaaS. In *The 2012 International Conference on Software Engineering Research and Practice SERP12*, 2012.

[131] E. Rosen and Y. Rekhter. BGP/MPLS IP Virtual Private Networks (VPNs). Technical report, Cisco Systems and Juniper Networks RFC 4364, 2006.

[132] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture (MPLS). Technical report, IETF RFC 3031, 2001.

[133] Joachim Sachs, Ibtissam El Khayat, Stephan Baucke, Norbert Niebert, Ralf Keller, and René Rembarz. Network Virtualization : A Viable Path Towards the Future Internet, 2008.

[134] A. Sajassi, R. Aggarwal, N. Bitar, and Aldrin Isaac. Requirements for Ethernet VPN (EVPN). Technical report, IETF draft-ietf-l2vpn-evpn-req-05.txt, 2013.

[135] Ali Sajassi, Samer Salam, Sami Boutros, Nabil Bitar, Aldrin Isaac, and Lizhong Jin. Provider Backbone Bridging E-VPN (PBB-EVPN). Technical report, IETF draft-ietf-l2vpn-pbb-evpn-06, 2013.

[136] Lutz Schubert and Keith Jeffery. Advances in Clouds - Research in Future Cloud Computing. Technical report, European Commission, 2012.

[137] Lutz Schubert, Keith Jeffery, and Burkhard Neidecker-Lutz. The Future of Cloud Computing - Opportunities for European Cloud Computing beyond 2010. Technical report, European Commission, 2010.

[138] Azimeh Sefidcon and El. Refined Architecture. Technical report, SAIL Project, WP-D (Scalable and Adaptable Internet Solutions), 2013.

[139] M. Sridharan, Duda K., I. Ganga, A. Greenberg, G. Lin, M. Pearson, P. Thaler, C. Tumuluri, N. Venkataramiah, and Y. Wang. NVGRE : Network Virtualization using Generic Routing Encapsulation. Technical report, IETF, 2013.

[140] L.M. Surhone, M.T. Tennoe, and S.F. Henssonow. *Vyatta Book*. 2010.

[141] Damian A. Tamburri and Patricia Lago. Satisfying Cloud Computing Requirements with Agile Service Networks. In *2011 IEEE World Congress on Services*, pages 501–506. IEEE, July 2011.

[142] J. Touch and R. Perlman. Transparent Interconnection of Lots of Links (TRILL) : Problem and Applicability Statement. Technical report, IETF RFC 5556, 2009.

[143] Nam-Luc Tran, Sabri Skhiri, and Zimanyi Esteban. EQS : An Elastic and Scalable Message Queue for the cloud. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 391–398. IEEE, November 2011.

[144] Economic Trends. Towards the future internet. *Security*, 23(0) :359, 2009.

[145] Sunay Tripathi, Nicolas Droux, Thirumalai Srinivasan, and Kais Belgaied. Crossbow : from hardware virtualized NICs to virtualized networks. In *ACM workshop on Virtualized infrastructure systems and architectures*, pages 53–62. ACM, 2009.

[146] Daniel Turull, Markus Hidell, and Peter Sjödin. Evaluating {OpenFlow} in {libNetVirt}. In *The 8th Swedish National Computer Networking Workshop 2012 (SNCNW 2012)*, 2012.

[147] Daniel Turull, Markus Hidell, Peter Sjoedin, and Ieee. libNetVirt : the network virtualization library. In *2012 Ieee International Conference on Communications*. 2012.

[148] J. van der Ham, P. Grosso, R. van der Pol, A. Toonk, and C. de Laat. Using the Network Description Language in Optical Networks. *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007.

[149] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the Clouds Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39(1) :50, 2009.

[150] M Wolski, S. Osinski, P. Gruszczynski, M. Labedzki, A. Patil, and I. Thomson. Deliverable ds3.13.1 : common network information service schema specification. Technical report, GEANT2, 2007.

[151] Fetahi Wuhib, Joao Monteiro Soares, Vinay Yadhav, Daniel Turull, Suksant Sae Lor, Matthias Keller, Rolf Stadler, and Paul Perie. Integrated Prototype System for Selected Use Cases. Technical report, SAIL Project, WP-D (Scalable and Adaptable Internet Solutions), 2013.