# La mobilité du code dans les systèmes embarqués

Guy Lahlou Djiken



## ▶ To cite this version:

Guy Lahlou Djiken. La mobilité du code dans les systèmes embarqués. Automatique. Université Paris-Est; Université de Yaoundé I, 2018. Français. NNT : 2018PESC1112 . tel-02123855

## HAL Id: tel-02123855
## https://theses.hal.science/tel-02123855

Submitted on 9 May 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

**UNIVERSITE DE PARIS-EST CRETEIL**
**Ecole de Doctorale Mathématiques et STIC**
**Laboratoire d'Algorithmique, Complexité et Logique**

**UNIVERSITE DE YAOUNDE I**
**Centre de Recherche et de Formation Doctorale en Sciences, Technologies et Géosciences**
**Unité de Recherche et de Formation Doctorale en Mathématiques, Informatique, Bioinformatique et Applications**

**Thèse de doctorat en Informatique**

**DJIKEN GUY LAHLOU**

# Mobilité de code dans les systèmes embarqués

***Thèse en cotutelle dirigée par les professeurs Fabrice Mourlin et Laure Pauline Fotso***

**Composition du jury**

| | | |
|---|---|---|
| *Rapporteurs* : | Pommereau Franck | IBISC – Université Evry (Professeur) |
| | Bobda Christophe | TBA – Université de Arkansas (Maître de Conférences HDR) |
| *Examinateurs* : | Cervelle Julien | LACL – Université Paris-Est (Professeur) |
| | Farinone Jean-Marc | CEDRIC – CNAM (Maître de Conférences) |
| | Teguia Michel | LI - Université de Tours (Maître de Conférences) |
| | Charif Mahmoudi | Siemens Corporate Research (Software Architect) |
| *Directeurs* : | Fabrice Mourlin | LACL – Université Paris-Est (Maître de Conférences HDR) |
| | Laure Pauline Fosto | URFDMIBA – Université de Yaoundé I (Professeure) |

# REMERCIEMENTS

C’est un grand plaisir pour moi d’achever finalement ce manuscrit de thèse. C’est aussi l’occasion de remercier tous les gens qui m’ont aidé, soutenu ou supporté pendant toute la durée de ce projet. Mes premiers remerciements vont naturellement à mes directeurs de thèse les Professeurs Laure Pauline Fotso et Fabrice Mourlin.

Je remercie tout particulièrement Fabrice Mourlin pour les longues heures de travail stimulantes et toujours enrichissantes. Sa passion pour la recherche est inépuisable. Merci de m’avoir poussé dans mes derniers retranchements. Cette thèse lui doit vraiment beaucoup.

Ce travail n’aurait pas abouti sans l’aide du Docteur Charif Mahmoudi. Ses idées m’ont souvent sorti d’impasses où je m’étais aventuré. Je suis conscient de lui devoir beaucoup, qu’il en soit remercié par ces quelques lignes.

Je souhaite remercier très chaleureusement tous les membres du jury. Leur lecture minutieuse de ce document de thèse ainsi que leurs commentaires et conseils ont contribué à améliorer sa qualité.

Cette thèse a été réalisée dans deux laboratoires LACL et LIRIMA respectivement des universités de Paris-Est Créteil en France et de Yaoundé I au Cameroun. Les moments les plus marquants sont ceux vécus au sein du laboratoire LACL. Le laboratoire LACL a constitué un cadre favorable pour faire de la recherche, alliant des qualités scientifiques exceptionnelles et une grande convivialité.

Je remercie donc l’ensemble des membres du laboratoire et plus particulièrement son directeur Julien Cervelle pour son conseil et pour son attention.

Merci aussi au Docteur Cyril Dumont pour m’avoir donné des solutions à plusieurs reprises, à ma sœur Touko Carole et mon frère Ninko pour leur aide, leur soutien et leur encouragement.

Et surtout, je remercie mes parents et mes amis qui ont su me soutenir, me supporter, m’encourager, me remotiver... pendant toute la durée de ma thèse.

Pour finir, je veux remercier celle qui entre-temps est devenue ma femme, et qui a accepté de faire certains sacrifices pour me permettre de faire ma thèse dans ces excellentes conditions et merci aussi à mes fils Carl Asher et Gauthier Djiken qui ont fait de leur possible pour que je puisse me concentrer sur mon travail…

***A ma feu maman Yona Delphine***

# Résumé

Avec l'avènement du nomadisme, des périphériques mobiles, de la virtualisation et du Cloud Computing ces dernières années, de nouvelles problématiques sont nées au vu des considérations écologiques, de la gestion d'énergie, de la qualité de service, des normes sécuritaires et bien d'autres aspects liés à nos sociétés. Pour apporter une solution à des problèmes liés à la consommation d'énergie, la qualité de service, nous avons défini la notion de Cloudlet tel un Cloud local où peuvent se virtualiser des périphériques et ses applications embarquées. Ensuite, nous avons conçu une architecture distribuée basée sur ce pattern d'architecture lié au Cloud Computing et à la virtualisation de ressources. Ces définitions permettent de placer notre travail par rapport aux autres approches de déportation d'applications mobiles.

D'autre part, un réseau de Cloudlets permet la protection de l'activité effectuée sur un périphérique mobile par la déportation d'applications embarquées dans une machine virtuelle s'exécutant dans la Cloudlet, ainsi que le suivi des usagers dans leur déplacement.

Ces définitions nous ont guidées dans l'écriture de spécifications formelles via une algèbre de processus d'ordre supérieure. Elles autorisent le calcul de la sémantique opérationnelle pour les différentes études de cas basées sur ce concept de Cloudlet. Ces spécifications ont permis de décrire une nouvelle vision de la composition des périphériques virtuels applicables à tous les périphériques, les capteurs ou les actuateurs. L'ensemble des équations obtenues constitue une définition formelle de référence non seulement pour le prototypage d'une Cloudlet mais aussi pour la construction de modèle temporel.

En se basant sur la structure de nos spécifications, nous avons construit un modèle d'automates temporisés (automate fini doté d'horloge à valeurs réelles) pour un réseau de Cloudlets. Par l'emploi de technique de model checking, nous avons établi des propriétés temporelles montrant que toute exécution d'une application mobile sur un périphérique mobile pouvait être déportée dans une Cloudlet sous condition d'une structure applicative. Ces travaux ont abouti à des choix techniques donnant lieu à un prototype d'une telle architecture distribuée par l'emploi de serveurs OSGi. D'une part, nous fournissons une architecture logicielle d'application mobile. D'autre part, nous mettons en œuvre le principe de migration vers une Cloudlet voisine et son retour. Ces résultats sont une validation de nos choix initiaux et attestent de la réalité de nos travaux. Ils autorisent la prise de mesure permettant de définir le coût d'une migration vers une Cloudlet pendant une exécution, ainsi que son suivi au cours du déplacement de l'usager.



**Mots-clés :** Cloud Computing, Cloudlet, Migration, Déportation, Architecture logicielle, OSGi, Model checking, Prototypage

# Abstract

With the advent of nomadism, mobile devices, virtualization and cloud computing in recent years, new problems have arisen taking into account ecological concerns, energy management, quality of service, security standards and many other aspects related to our societies. To solve these problems, we define the concept of Cloudlet as a local cloud where virtual devices and embedded applications can be virtualized. Then, we design a distributed architecture based on this architectural pattern related to cloud computing and virtualization of resources. These notions allow us to position our work among other approaches to offload mobile applications in a Cloudlet.

On the other hand, a network of Cloudlets helps to secure the activity carried out on a mobile device by offloading embedded applications in a running virtual machine in the Cloudlet, and also to monitor users during their movements.

These definitions guided us towards writing formal specifications via a higher order processes of algebra. They facilitate the calculation of operational semantics for different case studies based on this Cloudlet concept. These specifications foster a new vision for designing virtual devices suitable to all devices, sensors or actuators. This set of equations constitutes a formal definition relevant not only for prototyping a Cloudlet but also for constructing a timed automata system.

Following the structure of our specifications, we built a model of timed automata for a network of Cloudlets. Exploiting the model checking techniques, we have established temporal properties showing that any execution of a mobile application on a mobile device could be offloaded in a Cloudlet depending on a given software architecture. This work resulted in making technical choices leading to a prototype of such a distributed architecture using an OSGi server. A first result leads us to define a software architecture for mobile applications. Secondly, we implement the principle of migration to a Cloudlet neighbor. Our tests validate our initial choices and confirm the hypotheses of our work. They allow taking measures in order to assess the cost of an offloading to a Cloudlet during runtime, as well as keeping track during user's movements.

**Keywords :** Cloud Computing, Cloudlet, Migration, Offloading, Software Architecture, OSGi, Model checking, Prototyping.

# TABLE DE MATIERES

REMERCIEMENTS ........ I
Résumé ........ III
Abstract ........ IV
TABLE DE FIGURES ........ IX
LISTE DE TABLEAUX ........ XII
INTRODUCTION ........ 1
I. Motivations et contributions ........ 1
II. Contenu de la thèse ........ 3
CHAPITRE 1 : Etat de l'art ........ 5
I. Mobilité informatique ........ 5
1.1. Atouts de la mobilité ........ 6
1.2. Limites de la mobilité ........ 15
1.3. Besoins de la mobilité ........ 19
II. Cloud local ........ 22
2.1. Définitions et Concepts ........ 22
2.2. Comparaison entre les différents Clouds locaux ........ 28
2.3. Communication entre un périphérique mobile et une Cloudlet ........ 29
III. Virtualisation ........ 33
3.1. Virtualisation des périphériques mobiles et serveurs ........ 34
3.2. Virtualisation du stockage de données ........ 35
3.3. Virtualisation du périphérique réseau ........ 35
3.4. Virtualisation des applications ........ 36
3.5. Famille de solution de virtualisation ........ 36
IV. Synthèse ........ 38
CHAPITRE 2 : Architecture pour la supervision d'une Cloudlet ........ 40
I. Architecture d'une Cloudlet ........ 40
1.1. Le monitoring dans un contexte de Cloudlets ........ 41
1.2. Défis de la surveillance logicielle mobile ........ 43
II. Les défis de la surveillance logicielle ........ 44
2.1. Monitoring d'une Cloudlet ........ 45
2.2. Le monitoring des périphériques ........ 48
III. Architecture pour la surveillance d'une Cloudlet ........ 52
3.1. Architecture à quatre couches ........ 53
3.2. Monitoring basé sur le répartiteur de ressources ........ 55
3.3. La création du VDR et l'application de déportation ........ 57

IV. Contraintes techniques pour la surveillance de la Cloudlet ..... 59
4.1. Les Frameworks pour la surveillance ..... 59
4.2. Corrective et développement spécifique ..... 60
4.3. Composants du système de surveillance ..... 61
V. Etude de cas ..... 64
5.1. Menaces liées à la sécurité des périphériques mobiles ..... 64
5.2. Expérimentations ..... 66
5.3. Résultats de nos expérimentations ..... 68
VI. Bilan ..... 70
CHAPITRE 3 : Spécification formelle des périphériques virtuels dans une Cloudlet ..... 72
I. Description d'une vision des périphériques virtuels composites ..... 72
1.1. Délégation des modules et des ressources ..... 73
1.2. Représentation des périphériques mobiles et des capteurs ..... 74
II. Paradigme de la virtualisation dans la Cloudlet ..... 75
2.1. Virtualisation et Cloudlet ..... 76
2.2. π-calcul ..... 78
III. Représentation des périphériques virtuels de la Cloudlet ..... 80
3.1. Composition des VDRs ..... 80
3.2. Orchestration et gestion du réseau virtuel ..... 87
IV. Architecture basée sur la Cloudlet ..... 92
4.1. Aspects techniques du réseau de Cloudlets ..... 93
4.2. Description des composants de notre architecture ..... 94
4.3. Projection des propriétés ACID ..... 95
V. Etude de cas ..... 97
5.1. Périphérique mobile ..... 97
5.2. Système ..... 98
5.3. Congruence structurelle ..... 99
VI. Bilan ..... 105
CHAPITRE 4 : Model-Checking appliqué à un réseau de Cloudlets ..... 106
I. Introduction ..... 106
II. La spécification formelle en automates ..... 108
2.1. Motivations ..... 108
2.2. Systèmes d'automates ..... 109
2.3. La logique temporelle ..... 112
2.4. UPPAAL comme outils de vérification ..... 115
III. Création d'un modèle temporel ..... 120
3.1. Démarche ..... 121

3.2. Construction du modèle ... 122
3.3. Migration de VDR dans le modèle ... 138
IV. Vérification ... 140
4.1. Démarche ... 140
4.2. Vérification d'exécution avec déportation et migration ... 140
4.3. Vérification d'exécution locale ... 145
V. Bilan ... 146
CHAPITRE 5 : Prototypage et résultats ... 148
I. Introduction ... 148
1.1. Serveur de composants dynamiques ... 149
1.2. Définition du protocole MOCP ... 151
II. Architecture logicielle pour une application embarquée mobile ... 153
2.1. Application à base de composants OSGi ... 153
2.2. Implémentation mobile du protocole MOCP ... 156
III. Architecture logicielle pour une Cloudlet ... 162
3.1. Architecture basée sur un serveur OSGi Karaf ... 162
3.2. Démarche logicielle pour la construction de composants OSGi ... 164
IV. Cas d'étude ... 174
4.1. Présentation de la configuration ... 174
4.2. Déroulement du scénario nominal ... 176
4.3. Description des mesures et analyse critique ... 178
V. Bilan ... 180
CHAPITRE 6 : Conclusions et perspectives ... 182
I. Contributions ... 182
1.1. Définition formelle d'une architecture logicielle ... 182
1.2. Développement du framework à base de composants ... 183
1.3. Développement d'outils de mesure ... 183
II. Perspectives ... 184
2.1. Optimisation du temps lors de la déportation ... 184
2.2. Etude de nouvelles propriétés ... 184
2.3. Mesure de l'impact de la migration et de l'usage d'une Cloudlet ... 185
LISTE DE PUBLICATIONS ... 186
Articles de Conférences ... 186
Article de Journal ... 186
BIBLIOGRAPHIE ... 187

# TABLE DE FIGURES

Figure 1.1 Convergence entre les normes et générations de normes ..... 8
Figure 1.2 Evolution des réseaux sans fil [13] ..... 8
Figure 1.3 Architecture de PhoneGap ..... 12
Figure 1.4 Architecture de Xamarin ..... 13
Figure 1.5 Ecosystème de réseaux sans fil 5G ..... 15
Figure 1.6 Services du Cloud vue sous forme de couches ..... 24
Figure 2.1 Architecture globale de Cloudlet ..... 41
Figure 2.2 Architecture locale d'une Cloudlet ..... 42
Figure 2.3 Comparaison des piles applicatives ..... 44
Figure 2.4 Outils de monitoring Linux ..... 46
Figure 2.5 Nagios check_ps.sh plug-in ..... 47
Figure 2.6 Architecture logicielle autour du serveur Sensu ..... 48
Figure 2.7 Intégration du monitoring dans l'architecture Android ..... 49
Figure 2.8 Niveaux d'interopérabilité introduite dans WSN ..... 51
Figure 2.9 Capacité de l'application NCAP ..... 52
Figure 2.10 Architecture pour le monitoring distribué ..... 53
Figure 2.11 Architecture en couches ..... 54
Figure 2.12 Processus de filtrage et de pondération ..... 56
Figure 2.13 Virtualisation de périphérique ..... 58
Figure 2.14 Virtualisation de capteur ..... 58
Figure 2.15 Diagramme de déploiement ..... 61
Figure 2.16 Pseudo code de l'algorithme de détection ..... 67
Figure 3.1 Connexion des VDRs ..... 81
Figure 3.2 Relation de composition entre VDRs ..... 81
Figure 3.3 Rôle de Event Bus ..... 82
Figure 3.4 Diagramme de séquence d'initialisation d'une SVDR ..... 85
Figure 3.5 Architecture globale ..... 93
Figure 3.6 Structure d'une Cloudlet et gestion du réseau ..... 95
Figure 3.7 Modèle d'implémentation de la DVDR ..... 96
Figure 3.8 Système utilisé ..... 99
Figure 4.1 Vue d’ensemble d’UPPAAL ..... 116
Figure 4.2 Architecture d'UPPAAL ..... 116
Figure 4.3 Exemple de déclaration de variables globales pour un réseau de Cloudlets ..... 117
Figure 4.4 Modèle d'automate temporisé UPPAAL en XML ..... 118

Figure 4.5 Représentation graphique d'un modèle d'automate temporisé UPPAAL .......................... 118
Figure 4.6 Déclaration de variables globales ........................................................................... 123
Figure 4.7 Déclaration du système et de ses composants ................................................................ 124
Figure 4.8 Modèle d'automate Orchestrator ............................................................................... 125
Figure 4.9 Modèle d'automate Provisioning ............................................................................... 126
Figure 4.10 Modèle de l'automate Launcher ............................................................................... 127
Figure 4.11 Modèle d'automate Run ........................................................................................... 127
Figure 4.12 Modèle d'automate Configuration ............................................................................ 129
Figure 4.13 Modèle d'automate Monitoring ................................................................................ 130
Figure 4.14 Modèle d'automate FrontEnd ................................................................................... 131
Figure 4.15 Modèle d'automate BackEnd .................................................................................... 132
Figure 4.16 Modèle d'automate de la SVDR ................................................................................ 133
Figure 4.17 Modèle d'automate VirtualSensor ............................................................................. 134
Figure 4.18 Modèle d'automate DevId ........................................................................................ 134
Figure 4.19 Modèle d'automate de la DVDR ................................................................................ 135
Figure 4.20 Modèle d'automate VirtualDevice ............................................................................. 136
Figure 4.21 Modèle d'automate de la CVDR ................................................................................ 137
Figure 4.22 Modèle d'automate CompositeDevice ......................................................................... 137
Figure 4.23 Modèle d'automate Process ...................................................................................... 138
Figure 4.24 Modèle d'automate Administrateur ........................................................................... 139
Figure 4.25 Modèle d'automate User .......................................................................................... 139
Figure 4.26 Aperçu de la vérification de la propriété 1 ................................................................ 141
Figure 4.27 Aperçu de la vérification de la propriété 2 ................................................................ 143
Figure 4.28 Aperçu de la vérification de la propriété 3 ................................................................ 144
Figure 4.29 Aperçu de la vérification d'une exécution locale sans déportation ................................ 145
Figure 4.30 Aperçu de la vérification d'une exécution locale pour intra = cx .................................. 146
Figure 5.1 Documentation du protocole MOCP ............................................................................ 152
Figure 5.2 Couches OSGi (source : OSGi Alliance, 2007) ............................................................. 154
Figure 5.3 Approche orientée service dynamique OSGi ................................................................. 155
Figure 5.4 Architecture logicielle d'une application mobile ............................................................ 156
Figure 5.5 Demande de migration d'une application ...................................................................... 158
Figure 5.6 Interaction entre une VDR et une VSR ......................................................................... 159
Figure 5.7 Interaction entre une Cloudlet et un Cloud ................................................................... 160
Figure 5.8 Migration d'une VDR entre deux Cloudlets ................................................................... 161
Figure 5.9 Initialisation du protocole dans Swagger UI .................................................................. 165
Figure 5.10 Méthode POST /identify .......................................................................................... 166
Figure 5.11 Méthode GET /notify .............................................................................................. 167

Figure 5.12 Méthode POST /vdr/{vdrId}/migrate ............................................................................. 167
Figure 5.13 Diagramme de composant illustrant une interaction de la figure 5.6.............................. 169
Figure 5.14 Simple test Suite pour agréger les cas de test ................................................................ 171
Figure 5.15 Bilan de Test ..................................................................................................................... 173
Figure 5.16 Diagramme de déploiement de notre cas d'étude............................................................ 175
Figure 5.17 Support de la phase d'évaluation : application mobile ImageScore ............................... 177
Figure 5.18 Log du périphérique mobile............................................................................................ 178
Figure 5.19 Impact de la migration sur la performance ..................................................................... 179
Figure 5.20 Migration du processus de distribution........................................................................... 180

# LISTE DE TABLEAUX

Tableau 2.1 Description de l'orchestration des composants ... 62
Tableau 2.2 Description de composants du périphérique / capteur ... 62
Tableau 2.3 Description de composants de Sensu ... 63
Tableau 2.4 Code malveillant par plate-forme ... 65
Tableau 2.5 Actions de codes malveillants ... 66
Tableau 2.6 Résultats de notre solution ... 68
Tableau 2.7 Résultat du Capsa ... 69
Tableau 2.8 Comparaison des résultats ... 69
Tableau 3.1 Construction du Pi-calcul ... 78
Tableau 3.2 Règle du Pi-calcul ... 79
Tableau 5.1 Test unitaire simplifié pour l'opération POST ... 171

# INTRODUCTION

La place des objets connectés ne fait que croître dans nos vies et dans nos villes. De nouvelles limites technologiques sont atteintes et nous cherchons à nous en affranchir. Les plus évidentes sont la consommation d'énergie, la charge des réseaux de communication, la surveillance des données personnelles, etc. Nous nous attachons dans ce document à appréhender ces questions avec la volonté de proposer une solution concrète même si parfois, celles-ci n'est que partielle.
Ces questions apparaissent aujourd'hui car l'informatique a fait des avancées considérables, nous autorisant de nouvelles possibilités : en architecture logicielle, en communication réseau ainsi qu'en virtualisation. Ainsi, le Cloud Computing est un terme désormais familier pour désigner la livraison de ressources et de services à la demande. Il offre une solution au stockage et à l'accès aux données via Internet, mais aussi aux traitements avec une grande disponibilité. L'offloading (ou déportation de code) est l'externalisation du calcul et du stockage des données à l'extérieur des périphériques mobiles vers des serveurs puissants ayant des ressources non limitées en général pour l'énergie, le calcul et le stockage.

## I. Motivations et contributions

Le but premier de mon travail de thèse est de contribuer à une solution réaliste d'un problème actuel et ainsi de repousser des limites de la mobilité qui aujourd'hui restreignent nos usages. Aussi face au problème de la limitation des ressources de calcul, de stockage et d'énergie de nos périphériques mobiles, ma motivation est d'offrir une solution pour améliorer leurs usages et permettre aux usagers que nous sommes, de devenir des citoyens connectés dans de meilleurs conditions.
Les applications mobiles apportent au secteur de la mobilité une variété de nouveaux services et un grand potentiel d'innovation. Pour les entreprises, les applications mobiles favorisent la transformation des processus métiers et permettent d'optimisent les déplacements tout en permettant aux salariés nomades d'être plus productifs et de simplifier le travail à domicile, source de fidélisation des talents. De même, les utilisateurs des périphériques mobiles ont recours aux applications mobiles à tout moment pour plusieurs usages. Elles offrent également de multiples possibilités dans l'ensemble de services de mobilités comme les services d'information, les extensions de service public et des services alternatifs. Cependant, toutes ces technologies naissantes ont de nombreuses limitations, plus spécifiquement du côté des périphériques mobiles utilisés que du côté du Cloud.
L'usager est amené à se déplacer d'un environnement à un autre sans se préoccuper du transfert d'une donnée sur un serveur dédié d'un Cloud. De ce fait, il se pose un problème de communication ou d'accès aux ressources au niveau des Clouds car l'interopérabilité n'est pas assurée au niveau du Cloud. Par

exemple, sur deux Clouds différents un même service peut avoir des interfaces différentes. Il devient primordial d'avoir un cache de ce Cloud ou Cloudlet afin de continuer le transfert dans de meilleures conditions. La continuité de service pour les composants déportés au cours du déplacement peut être assurée par l'utilisation d'un réseau de Cloudlets. Pour cela, l'ensemble de Cloudlets doit partager une même spécification formelle et la logique temporelle associée à un outil de vérification permet de vérifier les propriétés temporelles. Une Cloudlet utilise le même principe qu'un Cloud mais la déportation s'effectue dans les réseaux privés (Cloud local ou serveur de proximité) situés dans un voisinage géographique.

La définition d'un réseau de Cloudlets regroupe un ensemble de Cloudlets interconnectées entre elles et aussi à un ou plusieurs Clouds. Le réseau de Cloudlets permet de mettre en exergue un ensemble de composants interconnectés. Cela nous conduit au développement d'un framework à base de composants qui propose un ensemble d'outils intégrant le monitoring et la déportation des applications mobiles dans un tel réseau.

Partant d'un réseau de Cloudlets dans une entreprise, la question demeure celle-ci : comment permettre aux personnels d'utiliser leurs applications en mode déporté au sein de l'entreprise ? Eventuellement réimporter l'application qui avait été déportée.

Le monitoring d'un tel réseau peut se reposer sur un besoin d'effectuer le suivi au niveau des personnels d'applications mobiles dans leurs déplacements. Il autorise également la collecte et l'analyse des données d'où un besoin de sécurité de données et des infrastructures. Pour cela, nous définissons une politique de gestion des droits sur les données et des applications au cours du déplacement du personnel. De plus, il est utile de virtualiser les périphériques ainsi que chaque infrastructure d'un réseau de Cloudlets. Ceci permet grâce aux représentations virtuelles de périphériques de continuer l'exécution des backends applicatifs au niveau de la Cloudlet. Cette vision des périphériques virtuels est utilisée par tous les périphériques, les capteurs et les actuateurs disponibles sur le réseau, mais un périphérique peut être composé d'un GPS, d'un baromètre, d'un indicateur de vitesse ainsi qu'un capteur de proximité. Dans cette situation, la représentation composite apporte une valeur ajoutée aux périphériques virtuels en permettant la composition de plusieurs périphériques d'une manière harmonieuse.

Dans le cadre d'une intervention de premier secours, l'utilisation des applications mobiles pour la reconnaissance faciale est consommatrice d'énergie et nécessite souvent une connexion réseau haut débit et l'autonomie du poste mobile est alors réduite. Dans le but de repousser cette limite, il faut réduire la consommation d'énergie du poste mobile et la notion de Cloudlet a pour but de répondre à ces besoins.

Une autre motivation porte sur la définition d'un protocole de migration entre un périphérique mobile intelligent et une Cloudlet (MOCP – Migration Oriented Cloudlet Protocol). Notre démarche de définition du protocole MOCP consiste à définir tous les APIs comme des paires de messages Demande/Réponse échangées entre les différentes entités présentes dans un réseau de Cloudlets. Cette démarche suit un processus itératif dont les étapes (Analyse – Conception – Codage – Test) s'enchainent

et en cas de modification, il faut revoir le processus de définition. Cela garantit une démarche évolutive dans le temps avec des livrables consultables par tout le monde. Notre protocole MOCP utilise un protocole binaire au-dessus du protocole http et est considéré comme un socle pour la concrétisation de notre démarche. La validation de nos objectifs permet de mettre en évidence la similitude entre l'exécution locale d'une application mobile et sa déportation dans un réseau de Cloudlets.

# II. Contenu de la thèse

L'organisation du document est la suivante :

- Le chapitre 1 est un état de l'art sur la mobilité et le concept de virtualisation à trois niveaux (Périphériques – Applications - Réseau). Nous y abordons la mobilité et l'utilisation du Cloud Computing qui apporte des solutions aux besoins ou limites de la mobilité. Nous présentons la virtualisation et un bilan des besoins en architecture logicielle pour les applications mobiles distribuées actuelles.
- Le chapitre 2 décrit un modèle architectural basé sur un système de monitoring pour une Cloudlet. Dans ce chapitre, nous présentons une architecture de virtualisation d'une Cloudlet dans un milieu de Cloud Computing, les contraintes techniques pour la mise en œuvre ainsi qu'une approche de monitoring.
- Le chapitre 3 fournit une spécification formelle écrite avec le langage π-calcul qui définit la représentation des périphériques virtuels dans la Cloudlet. Dans ce chapitre, nous définissons le protocole de communication MOCP au cœur des interactions entre composants et/ou services durant la migration. Nous présentons une étude de cas montrant une congruence structurelle entre la déportation et l'exécution locale d'une application mobile, tout en illustrant la similitude des arbres sémantiques entre le terme local et celui déporté dans un réseau de Cloudlets.
- Le chapitre 4 porte sur la création d'un modèle temporel de notre architecture à base de Cloudlet et vérifie des propriétés temporelles liées à la migration des backends applicatifs depuis un périphérique mobile. Dans ce chapitre, nous présentons des apports de la logique TCTL associée à un outil de model-checking ainsi qu'un état de l'art sur des techniques de spécification en automates et des outils de vérification. Cette spécification d'un système en logique temporelle est construite par un système d'automates temporisés. Chacun d'eux fournit une description rigoureuse d'un composant au cours du temps qui respecte des contraintes liées à son contexte d'évaluation. Ces informations enrichissent notre modèle et ajoutent des propriétés à préserver dans l'implémentation. Nous présentons également le suivi des applications virtualisées dans une Cloudlet en cas de mobilité de l'usager final.
- Le chapitre 5 présente l'implémentation de la Cloudlet respectant la spécification des chapitres précédents et les propriétés liées. Il aborde des outils qui sont utilisés pour évaluer la solution

et présente les contraintes d'implémentation associées à leur rôle logiciel. Ce chapitre donne une réalité à la notion de Cloudlet en prototypant tout un écosystème autour de machines virtuelles, contenant entre autre un serveur OSGi[1] pour l'accueil dynamique de composants. De plus, dans ce chapitre nous mettons en place une démarche logicielle pour la définition du protocole d'échange qui est supporté par notre architecture.

Pour finir, nous concluons ce document par une synthèse de nos contributions. Nous présentons des propositions faites ainsi que les perspectives importantes et ouvertes par notre travail. A la fin du document, figure la liste de références utilisées dans nos cinq chapitres.

Cette brève introduction a permis de placer notre thème de recherche et nos motivations pour atteindre nos objectifs de meilleurs usages. Enfin le découpage des chapitres offre une vue générale du document afin de mieux percevoir les interdépendances.

[1] OSGi : Open Service Gateway interface

# CHAPITRE 1 : Etat de l'art

Avec l'avènement du nomadisme, des périphériques mobiles, de l'usage du Cloud et des techniques de virtualisation, le marché du logiciel s'est transformé en profondeur suite à ces nombreuses évolutions technologiques. Dans une première approche, nous pouvons considérer les périphériques mobiles comme des dispositifs mobiles utilisables de manière autonome lors d'un déplacement [1]. Un tel périphérique n'est pas seulement portatif, aujourd'hui, il inclut la capacité d'interaction et de communication, plus généralement une connectivité au réseau Internet. Ces périphériques mobiles tels que les smartphones et tablettes associés à un ensemble de capteurs ont rempli un nouveau besoin informatique [2].

La demande des smartphones au niveau mondial totalise 353 millions d'unités en 2016, en hausse de 7% en glissement annuel[2] [3]. Selon les dernières estimations du cabinet Strategy Analytics, le nombre d'utilisateurs de smartphones atteint 2,1 milliards d'unités et est estimé à plus de 3,3 milliards en 2018 [4]. De plus, ces chiffres tendent à s'accroître avec le « leapfrogging[3] africain » vu l'estimation de 97 % d'Africains utilisant les périphériques mobiles en 2017.

Dans ce chapitre, nous débutons par le concept de la mobilité tout en insistant sur la mobilité et le nomadisme. Il est question également de mettre en valeur les atouts, les limites et les besoins de cette mobilité. Ensuite, nous explorons l'utilisation du Cloud Computing qui apporte des solutions aux besoins ou limites de la mobilité. Puis, nous abordons la communication entre applications mobiles – Cloud local – Cloud Computing. Enfin, le concept de la virtualisation est développé à trois niveaux périphériques – applications – réseau. Le chapitre se termine par un bilan des besoins en architecture logicielle pour les applications mobiles distribuées actuelles.

## I. Mobilité informatique

La mobilité caractérise ce qui peut se mouvoir, ce qui peut changer de place, de position [5]. Dans les systèmes distribués, on parle de mobilité informatique qui consiste à rendre accessible toutes sortes de services, n'importe où, tout en masquant le périphérique support. Par contre, le nomadisme est la capacité d'un système à fournir à chaque utilisateur son environnement de travail sur n'importe quelle station [6]. Comme exemple de nomadisme, nous pouvons citer les ordinateurs portables et les consoles de jeux portables de Nintendo, où l'usage du réseau est primordial.

[2] Glissement annuel c'est un calcul permettant de suivre la performance d'une valeur, d'une statistique ou autre sur les 365 derniers jours

[3] Saut de grenouille c'est-à-dire une étape technologie pour passer directement à une autre

La mobilité est un concept clé du nomadisme de l'infosphère[4]. L'avènement d'Internet et l'étendue grandissante de la communication réseau ont mis à bas des idées de distance et de frontières et plus généralement de limites, que celles-ci soient techniques ou physiques [7]. L'idée est d'accéder à l'information en n'importe quel point du globe et à tout moment. La nouveauté à présent est celle de mobilité connectée. La mobilité informatique peut indifféremment désigner la mobilité matérielle ou la mobilité logicielle. La mobilité matérielle est le déplacement d'un terminal (périphérique) physique tel qu'un smartphone, une tablette ou un capteur. Cette mobilité, encore appelée d'usage, s'applique à la mobilité d'un utilisateur qui interagit avec un périphérique mobile. Alors que la mobilité logicielle est le déplacement d'un programme logiciel entre deux périphériques physiques. L'entité logicielle est alors appelée composant, agent, code mobile.

Les spécialistes du marketing y voient la possibilité alléchante de connecter perpétuellement l'utilisateur/consommateur. Les passionnés de technologie de l'information préfèrent croire en l'avènement d'une ère de communication globale et instantanée [8]. La notion de mobilité informatique a pris son essor dans le courant des années 90 au Japon, pays où la téléphonie mobile est déjà massivement utilisée comme accès au réseau et aux médias de divertissement [9]. En Afrique, l'arrivée du haut débit et l'expansion des technologies sans fil, particulièrement du réseau Wi-Fi (Wireless Fidelity) donnent le coup d'envoi de la course à la mobilité. L'idée d'accéder à l'information à l'aide de technologies "embarquées", qu'elle soit de type téléphone mobile, agenda électronique, ordinateur portable, ou autre est en train de se généraliser. Dans le sillage de ce que l'on désigne déjà comme la seconde révolution de l'information, des défis techniques apparaissent et donc des compétences adaptées à ce nouveau type de challenges [10] : interface Homme-Machine adaptée, synchronisation et sécurité des données, nouvelles formes d'architectures des systèmes, unification des protocoles de communication, etc.

## 1.1. Atouts de la mobilité

Dans le cadre de leurs activités, les individus comme des entités (voitures, trains, …) se déplacent et ont régulièrement besoin de communiquer, d'accéder à de l'information ou de recueillir des données pendant leurs déplacements. La combinaison des infrastructures de communication sans fil et des dispositifs informatiques portables pose les bases de la mobilité informatique et permet aux différents utilisateurs d'accéder à des informations, de collaborer avec d'autres dispositifs en mouvement [11]. Elle offre des avantages considérables et dans cette section, nous soulignons quelques-uns des atouts de la mobilité informatique.

[4] Infosphère désigne à la fois un environnement global, constitué d'informations, ainsi que tous les types de données qui y transitent ou y sont stockés

### 1.1.1. Au niveau des ressources matérielles

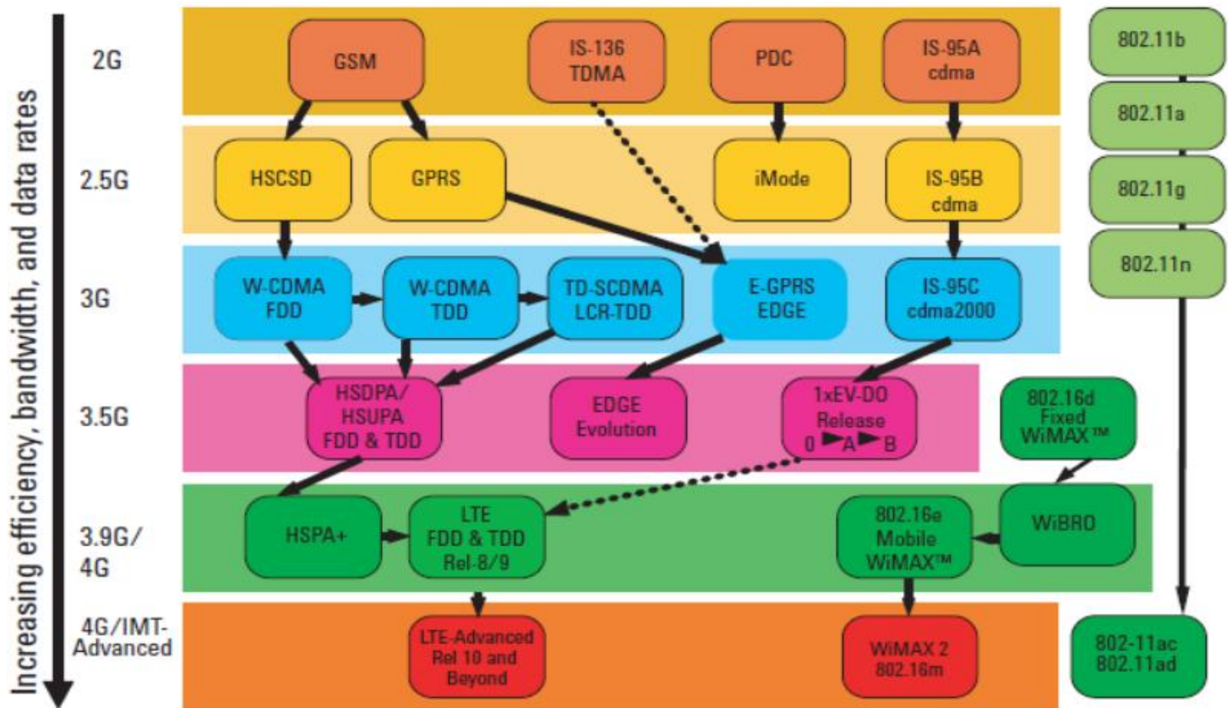
Les premières caractéristiques des ressources matérielles sont définies par la taille et la forme, le poids, les capacités de calcul processeur, les capacités de stockage primaire et secondaire, la taille de l'écran et le type, les moyens d'entrée, les moyens de sortie, la durée de vie de la batterie, les capacités de communication, l'évolutivité et la durabilité du matériel [11]. Deux aspects matériels sont également pris en compte à savoir les normes (pour les dispositifs téléphonies mobiles et réseaux) et la convergence des accès réseaux sans fil. Chacune de ces caractéristiques présente des atouts dont nous énumérons certains.

a) Les normes pour dispositif téléphonie mobile et Wi-Fi

L'échange d'informations (voix et/ou données) entre périphériques mobiles utilise des ondes radioélectriques dans les bandes de fréquence de 900 à 1800 MHz. Les technologies de téléphonie mobile sont normalisées pour être compatibles d'un pays à l'autre, et ce, même si de nombreuses normes mondiales cohabitent. Toute téléphonie mobile doit respecter des normes et deux ensembles se partagent ainsi le monde : les normes d'origine américaine (normes ANSI-41 / CDMA) et celles d'origine européenne (GSM ET UMTS). Les normes Wi-Fi permettent de créer des réseaux locaux sans fil à haut débit entre des périphériques (ordinateurs, tablettes, smartphones, capteurs). Ces normes sont toutes rassemblées sous la référence IEEE 802.11 qui présente plusieurs variantes, plus ou moins performantes [12]. Même si la 4G n'est pas encore déployée sur les territoires, la 5ème génération a pour objectif de réduire les temps de latence dans les transmissions de données. Pour le moment, le temps de réaction des réseaux mobiles est de l'ordre de la demi-seconde. Cela peut sembler peu, mais c'est aussi beaucoup lorsqu'il s'agit d'une communication issue d'un véhicule en réseau urbain. Ce futur standard doit s'adapter à plusieurs usages de demain. La marge de progression des réseaux 4G est encore grande, mais l'objectif de réduction de la consommation d'énergie ne pourra être atteint qu'après le changement technologique majeur de la 5G.

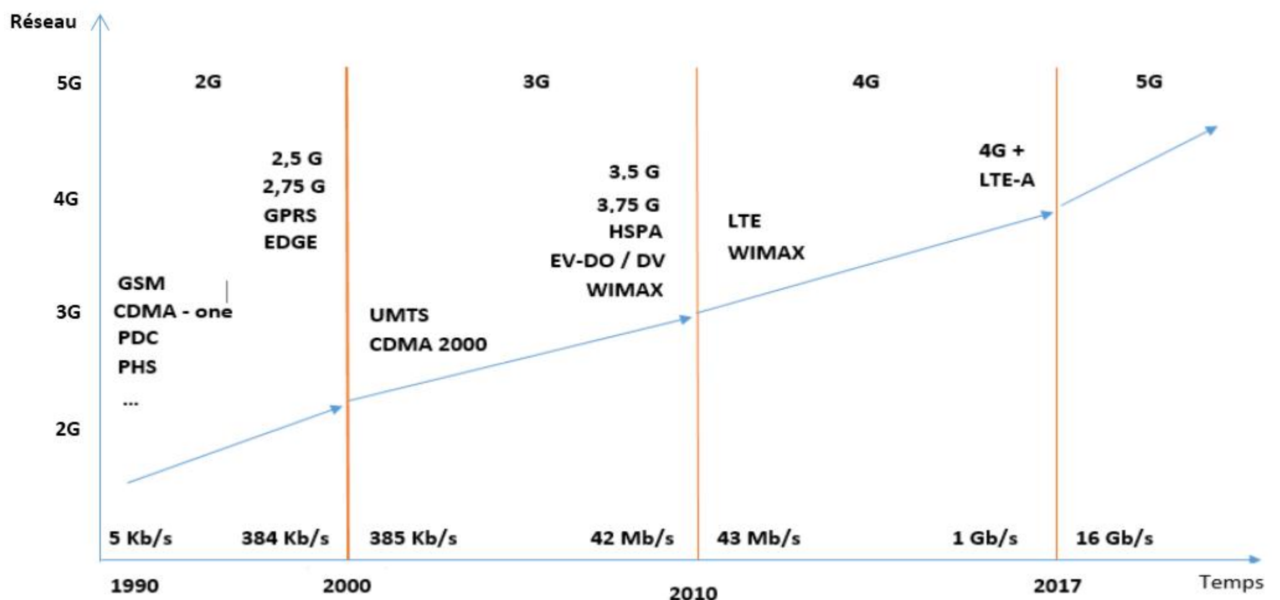
La Figure 1.1 illustre la convergence des normes pour les dispositifs mobiles et Wi-Fi. Cette figure est découpée en 5 bandes horizontales qui représentent les normes et standards de différentes générations. Les flèches pointillées entre les boîtes montrent une relation d'appartenance par contre les flèches pleines montrent l'évolution ou le passage d'un standard à un autre. Il est à noter une augmentation de l'efficacité, de la bande passante et des débits de données partant de la 2ème génération à la 4ème génération aux standards LTE-Advanced. Les normes extérieures aux bandes représentent les 5 normes différentes (802.11a/b/g/n/ac) du Wi-Fi [13]. Chacune représente une évolution par rapport à la précédente schématisée par une flèche pleine.

*Figure 1.1 Convergence entre les normes et générations de normes[5]*




La Figure 1.2 montre l'évolution des normes réseaux en fonction du temps, partant de la 2G avec les standards de communication GSM à la 4G LTE-Advanced. Elle annonce la 5ème génération qui reste encore en gestation et grâce à laquelle nous pouvons naviguer à partir de smartphones et tablettes à une vitesse maximale théorique de 20 Gbit/s, jusqu'à 20 fois plus vite que la vitesse maximale théorisée pour la 4G (1 Gbit/s) [14].

*Figure 1.2 Evolution des réseaux sans fil*




[5] http://opengarden.net/Opengardening

b) Capacités de calcul processeur et répartition dynamique de charge

Les périphériques mobiles actuels sont dotés des processeurs de 64 bits, l'utilisation de ces processeurs, non seulement, permet d'adresser plus de mémoire vive (c'est-à-dire plus d'adressage mémoire) mais aussi la gestion de plusieurs applications simultanées. Nous notons la réduction de l'utilisation de la mémoire vive ou du cache, concentrant l'effort dans les processeurs plus rapide. Les applications qui s'exécutent dans un contexte distribué peuvent se déployer vers de nouveaux terminaux et profiter ainsi des ressources physiques des terminaux d'accueil telles que le processeur [15] et les capteurs. Dans le cadre hétérogène d'une architecture matérielle constituée de périphériques mobiles et machines traditionnelles, une gestion intelligente des ressources de calcul peut améliorer grandement les performances de l'ensemble de l'application distribuée. Un déploiement judicieux des applications permet d'équilibrer dynamiquement la charge à travers le système distribué. Ce déplacement ne peut être entièrement statique du fait de la dynamicité de la configuration du réseau. Ainsi il apparaît naturel que les composants d'une même application distribuée puissent être répartis sur les ressources de calcul présentes à un instant donné. La gestion de ce redéploiement doit être faite avec attention afin que celui-ci ne se produise qu'en situation favorable. Ainsi, de nouvelles architectures logicielles sont possibles, avec de nouveaux séquencements.

c) Capacité de stockage

Les dispositifs mobiles tels que les ordinateurs portables, les smartphones, tablettes diffèrent des postes de travail traditionnels, car ils nécessitent des composants légers, peu coûteux, avec une faible consommation d'énergie et de bonnes performances interactives. Cependant, ils disposent d'une faible capacité de stockage des informations [16]. Dans le chapitre 3 de l'ouvrage « Mobile Computing » [16], Dougli et Al. proposent trois périphériques de stockage alternatifs pour les dispositifs mobiles à savoir des disques durs magnétiques, émulateurs de disque de mémoire flash et cartes mémoire flash. Ils mettent en évidence les différences entre les performances des trois périphériques et parviennent à l'évidence que l'usage de la carte mémoire flash (carte mémoire de capacité variable) peut réduire la consommation d'énergie et le temps de réponse en lecture de deux ordres de grandeur par rapport au disque magnétique tout en offrant des performances d'écriture acceptables. Les technologies de stockage actuelles offrent deux solutions de stockage de fichiers sur des dispositifs mobiles : les disques durs magnétiques (ou disques durs externes USB) et la carte mémoire flash. De nos jours, les fabricants de dispositifs mobiles intègrent directement la technologie USB OTG[6]. Grâce aux adaptateurs USB OTG [17], les disques durs surtout les disque durs USB en mode Host offrent une grande capacité à faible coût et ont un débit élevé pour les transferts importants. Le principal inconvénient des disques est leur consommation d'énergie élevée et le temps de rotation considérable pour accéder à la donnée. La mémoire flash consomme relativement peu d'énergie, a une faible latence et un débit élevé pour les accès en lecture [18]. Cette capacité de stockage offre une nouvelle répartition des données applicatives.

[6] OTG pour On-The-Go permet de connecter une clé USB, un disque dur externe, une souris ou même un clavier.

Ainsi, les données sont distribuées en relation avec le découpage logiciel. Depuis le stockage d'informations issues de capteurs jusqu'à la réception radio de données en provenance d'un Cloud privé ou public, ce support de persistance est un atout pour un meilleur équilibrage de l'accès aux données, mais aussi pour la journalisation d'activité en cas de manque de communication.

d) Capacité de représentation graphique et sonore

La progression des outils technologiques a rendu disponibles les éléments nécessaires à la création de systèmes audionumériques pilotés par le geste [19]. Les périphériques mobiles combinent à la fois les fonctionnalités d'affichage d'un écran ordinaire et celle d'un dispositif de pointage. Ils permettent donc de réduire le nombre de périphériques (entrées/sortie) nécessaires pour la manipulation d'un système [20]. Par contre, le smartphone cristallise la convergence de technologies et d'usages de terminaux antérieurs : téléphone mobile, ordinateur, téléviseur, baladeur musical, console de jeux [21]. Ainsi, le smartphone, terminal de communication, offre des services de représentations des entrées/sorties qui lui confère un rôle nouveau dans une architecture distribuée. Contrairement aux téléviseurs portatifs étudiés par L. Spiegel [22], un téléphone est un écran audiovisuel véritablement mobile, de format réduit. Cette mobilité participe à l'appropriation du smartphone comme écran audiovisuel en autorisant une variété inédite de contextes d'utilisation et de consommation des contenus rendus accessibles (visionner une émission de télévision dans un bus, partager une vidéo depuis un site communautaire tout en marchant). Elle permet également de renforcer l'individualisation et la privatisation de ces pratiques. Sa surface tactile interactive supprime les appendices nécessaires à l'interaction avec les écrans antérieurs (télécommande, clavier, souris). Comme le souligne J. Coutaz dans le cas d'une interface digitale, l'instrument est supprimé. La main contrôle directement l'entité-système représentée sur la surface [23]. L'utilisateur a donc au moins l'illusion de manipuler directement les contenus et données. Les APIs (Application Programming Interface) comme OpenGL [24] et OpenSL [25] fournissent une approche standardisée à haute performance et à faible latence pour accéder aux fonctionnalités graphiques et audio des développeurs d'applications natives pour des périphériques mobiles. Huot [26] propose de nouveaux types de menus (ArchMenu et ThumbMenu) pour les périphériques mobiles. Ces menus permettent l'interaction avec le pouce de la main qui tient le périphérique ainsi, l'utilisateur peut manipuler l'ensemble d'une seule main. L'usage de haut-parleurs et écouteurs permet d'avoir une représentation sonore. Associé à des microphones et de nouvelles formes d'interaction, un périphérique mobile permet aussi le pilotage vocal de dispositif IoT[7] (Internet des Objets). Tel est le cas du produit Echo fourni par Amazon [27] pour un usage d'organisation domotique (gestion de listes, interactions audio, …).

e) Usage des capteurs

Des thermomètres jusqu'aux caméras de surveillance, les capteurs sont des outils qui sont aujourd'hui utilisés dans presque tous les domaines pour observer n'importe quel phénomène. Les dispositifs mobiles

[7] IoT pour Internet of Things représente l'extension d'Internet à des choses et à des lieux du monde physique.

comme des tablettes, smartphones sont pour la plupart équipés d'un grand nombre de capteurs. Ces capteurs comprennent entre autre un GPS pour la localisation, une boussole numérique, un gyroscope, un accéléromètre pour la vitesse et la direction de mouvement, des caméras avant et arrière pour l'enregistrement de vidéos et la prise de photos ; et un microphone pour les enregistrements audio [28]. L'usage de ces capteurs permet d'améliorer les conditions de vie. Dans un milieu urbain, la plupart des capteurs sont accessibles par différents réseaux tels que Bluetooth ou Wi-Fi et sont facilement accessibles par les réseaux mobiles 3G/4G [29]. Les données issues de ces capteurs vont s'ajouter à la masse d'informations personnelles que les smartphones voient passer [30]. Les périphériques mobiles peuvent être également munis de capteurs de proximité[8] et se connecter via une multitude d'interfaces réseaux de types Wi-Fi, Bluetooth, RFID, LoRA ou NFC. Les périphériques mobiles qui embarquent toutes sortes de capteurs, permettent d'analyser et de collecter des informations en fonction de leurs caractéristiques. Les formats de données issues de ces capteurs sont souvent des tableaux de flottants. L'encodage de ces données est laissé à l'éditeur logiciel. Android fournit une API où les données sont proposées dans une structure gérant les accès concurrents. L'auteur A. Lesas [31] montre que le smartphone avec ses capteurs et sa connectivité, contribue à la surveillance et la détection des séismes pour alerter les utilisateurs et collecter des données géolocalisées et horodatées.

Le terme réseau de capteurs est utilisé pour définir un ensemble de dispositifs spatialement distribués utilisant des capteurs pour surveiller des grandeurs à différents endroits, tels que la température, le son, les vibrations, la pression, le mouvement ou les polluants. Un capteur Web fait référence aux capteurs accessibles sur le Web, qui exposent leurs données sur le protocole http. Ils peuvent être découverts et accessibles à l'aide de protocoles standard et d'interfaces de programmation d'application (API) [32]. Dans le monde du Web, le terme de capteur sociétal est employé par analogie pour désigner la fourniture de données issues de réseaux sociaux tels que Twitter, Facebook, etc. Ces réseaux sont d'ailleurs accessibles depuis les périphériques mobiles les plus courants. La prise en compte de capteurs et la localisation sont effectuées par l'installation de plug-ins dédiés. De nouveaux plug-ins peuvent être développés pour la gestion de nouvelles aptitudes matérielles telles que le protocole LoRa.

### 1.1.2. Au niveau des ressources logicielles

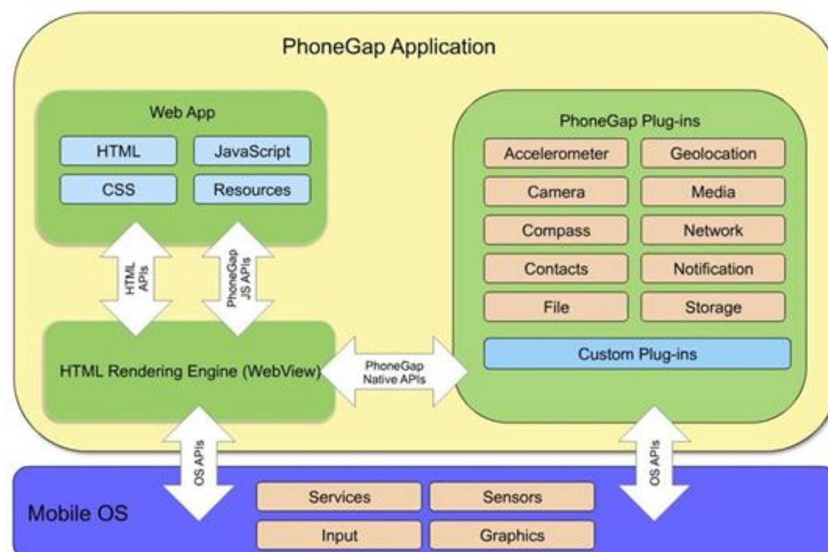
Le développement d'applications mobiles est un domaine d'activité qui concentre un grand nombre de concepteurs et de programmeurs. De même, le développement des périphériques mobiles a abouti à diverses formes de plate-formes auxquelles se sont greffées une multitude de systèmes d'exploitation (iOS, Android, Tizen, Windows Phone, etc.) [33]. Des problèmes traditionnels de portabilité sont apparus avec des solutions partielles telles que la génération de code dédié (solution PhoneGap) [34] ou la recherche des solutions communes (solution Xamarin) [35] [36].

[8] Un capteur doit être en mesure de détecter la présence d'un objet physique sans contact physique. Voir : http://www.w3.org/TR/2012/WD-proximity-20121206

a) PhoneGap : Framework hybride

PhoneGap propose un environnement pour créer des applications multi-plate-formes sur un navigateur Web. Ce logiciel fait partie de la famille des produits proposés par Adobe depuis 2011[36]. Il permet de créer des applications sur un navigateur, mais il permet aussi d'accéder aux fonctionnalités de l'appareil cible via des frameworks JavaScript. Niveau développement, des technologies du Web peuvent être utilisées telles que JavaScript, HTML5 ou le CSS. Le fonctionnement, illustré grâce à la Figure 1.3, réside dans le concept où le développeur peut définir les éléments graphiques via des pages HTML et CSS puis lier ces éléments graphiques à des actions écrites en JavaScript. Le tout est compilé pour chaque plate-forme. Grâce à un tel logiciel, les problèmes de comptabilité ne se posent plus. Les applications iOS, Android et Windows Phone sont basées sur un seul et unique code, que ce soit au niveau FrontEnd (partie d'interface graphique), mais aussi BackEnd (partie métier, persistance).

*Figure 1.3 Architecture de PhoneGap*




*Source 1 : phonegap-architecture-by-ibm-29-july-2011-modules*

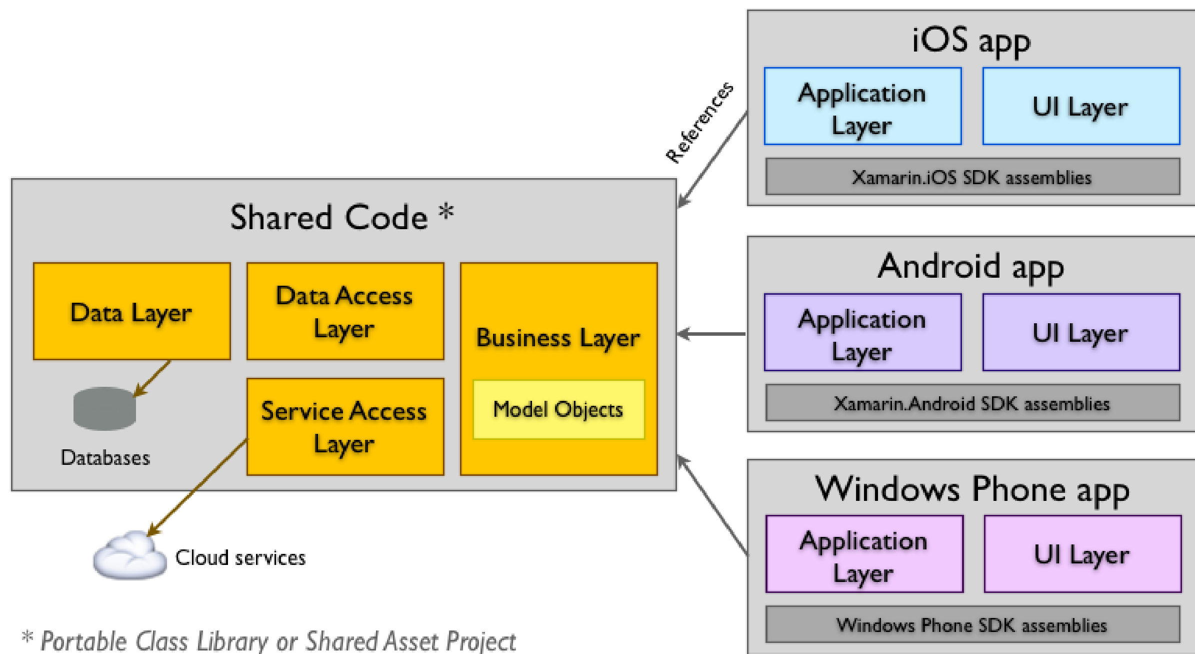
b) Xamarin : MonoTouch

Xamarin permet la création et la maintenance d'applications multi-plate-formes sur des appareils tels que les téléphones, les tablettes, mais aussi les appareils embarqués sous licence iOS, Android et Windows. Il possède son propre environnement de développement utilisant le langage de développement C# et le .Net Framework. Il autorise la création d'applications de type natif en utilisant les différentes API et interfaces-utilisateurs de chaque plate-forme. Xamarin permet de faire du développement multi-plate-forme grâce à sa bibliothèque Xamarin_Forms. Cela permet d'écrire en C# tout l'univers de l'application [37].

Deux types de projet (Shared App et Portable App) sont implémentés dans Xamarin et décrits dans la Figure 1.4. La Figure 1.4 définit chaque accès de données dans un projet shared et permet de partager les librairies, d'utiliser un plus grand nombre d'Add-Ons ou de plug-in proposés dans la bibliothèque

Xamarin ou NuGet dans un projet Portable App. Elle comprend des couches Data Layer, Data Access Layer, Business Layer, Service Access Layer et Application Layer [37]. Ce sont autant de pilotes pour accéder au contexte du périphérique. Par exemple, la classe SensorManager permet d'accéder en lecture aux capteurs du périphérique en tant que source d'information. Elle gère entre autres la réservation et la libération de capteurs tels qu'une caméra ou un GPS. Dans le dernier cas, la classe LocationManager fournit les coordonnées spatiales.

*Figure 1.4 Architecture de Xamarin*




*Source 2 : Xamarin Inc, 2015*

Des applications monolithiques ou autonomes s'exécutent sur un poste mobile sans nécessité d'autres ressources. C'est le cas d'utilitaires tel que l'utilisation d'une boussole pour l'orientation, l'utilisation de Talk Back pour aider les malvoyants à se servir de leur mobile [38]. Des applications de type Client/serveur existent où le poste mobile joue le rôle de client « embarqué ». Elles nécessitent un accès à Internet sans un coût d'installation d'équipements LAN et présentent un grand nombre d'avantages. Les informations qui y transitent, sont régulièrement téléchargées à partir du système d'information de l'organisation ou des serveurs distants vers les périphériques mobiles, puis téléchargées à partir du périphérique au système d'information ou des serveurs. Les mises à jour des informations sont disponibles dans les deux sens [39]. C'est le cas des logiciels de synchronisation de données largement utilisés dans le domaine des transports.

De nouvelles architectures apparaissent où le périphérique mobile expose ses services voire ses capteurs. C'est le cas lorsque l'on souhaite utiliser son téléphone portable comme hot spot Wi-Fi depuis une station de travail n'ayant plus de connexion réseau ou de gérer son smartphone Android depuis n'importe quel navigateur Internet via l'application PAW [40]. Dès lors la plate-forme mobile joue le

rôle de serveur pour d'autres stations de travail mobiles ou non mais partageant le même réseau (souvent Wi-Fi).

### 1.1.3. Au niveau de la communication

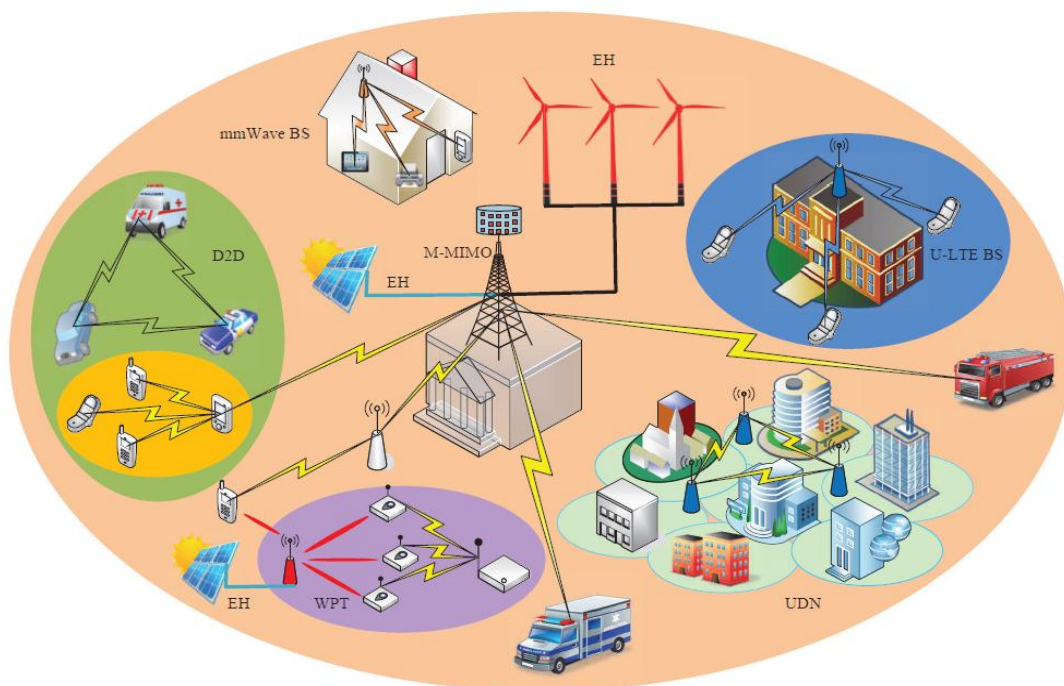
Les dispositifs informatiques adaptés aux utilisateurs mobiles présentent de grandes différences au niveau des capacités de communication. Ces dispositifs mobiles se connectent habituellement par services Web via des connexions sans fil [41]. Les services Web basés sur REST ou SOAP utilisent les formats d'échange de données JSON, XML ou Fast Infoset [42]. La plupart des services Web disponibles sur les smartphones utilisent le format XML ou JSON pour la représentation des données qui doivent être transférées entre le serveur Web et les périphériques mobiles ou entre périphériques. Johnsrud et Al. [43] ont analysé l'efficacité de l'utilisation de différentes techniques de compression XML pour avoir une représentation visuelle de qualité.

Aujourd'hui, un nouveau standard pour la vidéo/audio/données peer-to-peer en temps réel est élaboré au W3C et à l'IETF [44]. Ce standard est nommé WebRTC (Web Real-Time-Communication), il se compose de protocoles et d'APIs JavaScript et couvre également les appels en streaming vidéo, audio, et les échanges de données [41]. Les réseaux sans fil qui permettent aux périphériques mobiles de transférer leurs données ont des technologies différentes. Deux grandes catégories de réseaux existent à savoir les réseaux longue portée et les réseaux à courte portée.

Les réseaux longue portée, comme Sigfox, LoRa, NB-IoT (NarrowBand LTE for IoT) [45] ou encore les technologies cellulaires (GSM, 2G, 3G, 4G, …) sont capables de faire transiter des données d'un périphérique à l'autre sur des distances importantes. Par exemple, ils sont utilisés par les entreprises qui veulent connecter des kilomètres d'infrastructures à Internet ou dans les projets de smart-cities. Les réseaux à courte portée comme le Wi-Fi, le Z-Wave, le ZigBee, IrDA, le Bluetooth, permettent de transférer des données sur de faibles distances. Ils sont beaucoup utilisés dans la domotique ou des wearables grand public [46] [47]. Dans les réseaux longue portée, Sigfox a une portée pouvant atteindre les 50 kilomètres et consomme très peu d'énergie pour envoyer les données sur le réseau, de même que le protocole LoRa qui permet la transmission des données à des distances allant de 2 à 45 kilomètres avec un débit de 0,3 à 50 kilobits par seconde (kbps). En revanche, la taille des messages est bornée ainsi que le nombre d'échanges par jour dans le cas de Sigfox, ce qui limite les domaines d'applications distribuées. Enfin, Sigfox contraint ses clients/usagers à faire transiter leurs données par le biais de son Cloud propriétaire. Ce modèle économique l'oppose au réseau LoRa qui est davantage ouvert et n'impose pas d'abonnement mensuel à un Cloud et surtout offre la possibilité de chiffrer les échanges aisément. Contrairement aux deux autres, les réseaux cellulaires (GSM, 2G, 3G, 4G, …) permettent de transférer d'importantes quantités de données et sont gourmands en énergie. La technologie NB-IOT est un protocole basé sur l'évolution des réseaux cellulaires voire la 4G, c'est l'avènement de la 5G (voir Figure 1.5) et l'explosion du BigData afin d'extraire des données pertinentes pour améliorer le

comportement de l'ensemble du système. Dans les réseaux courte portée, le Wi-Fi permet de transférer un grand nombre de données rapidement jusqu'à 600 mégabits par seconde (mbps) et reste très énergivore. Par contre, le Z-Wave à une portée de 30 mètres, soit 10 mètres de plus que le ZigBee et permet de faire circuler moins de données que ce dernier. De même, le Bluetooth largement utilisé dans le monde, consomme environ 20 fois moins d'énergie que le Wi-Fi et permet de transporter moins d'informations que ce dernier [48]. Nous constatons aisément qu'il n'existe pas de protocole réseau idéal, mais chaque cas d'étude possède une solution privilégiée où la localisation de l'émetteur et du récepteur a un rôle essentiel. La Figure 1.5 est constituée d'un ensemble de technologies comme : D2D (Device to Device), M-MIMO (Massive –Multiple Input Multiple Output), WPT (Wireless Power Tranfert), UDN (Ultra Dense Network), mm Wave BS (millimeter Wave Base Station) et EH (Energy Hybrid).

*Figure 1.5 Ecosystème de réseaux sans fil 5G*




## 1.2. Limites de la mobilité

Que ce soient les smartphones, les tablettes, les capteurs, des serveurs Web, tout système logiciel ou matériel admet des limites au niveau des ressources. Et même la mobilité impose de nombreuses contraintes que nous résumons dans cette sous-section.

### 1.2.1. Insuffisance de la bande passante

Les milliards de smartphones et tablettes donnent accès à de nouveaux services nécessitant toujours plus de débit, additionnés aux milliards d'objets connectés formant l'IoT, cela va inévitablement provoquer

dans le futur une augmentation drastique du trafic de données transitant sur les réseaux mobiles [49]. Selon la capacité de transmission de données des réseaux des différents opérateurs de téléphonie mobile, la bande passante du signal audio est toujours plus ou moins limitée pour des raisons d'économie. Sur les réseaux 2G, le son est strictement en bande étroite (3,2 kHz), le réseau 3G autorisant quant à lui une transmission large bande (7 kHz). Malgré l'avènement de la 4G et les nouvelles technologies, l'accès à Internet mobile est généralement plus lent que les connexions directes par câble, car les facteurs physiques ont une influence sur la bande passante. En utilisant des technologies telles que GPRS et EDGE, et plus récemment les réseaux 4G, il est difficile de prendre en compte le niveau d'utilisation des périphériques mobiles, car l'accès au réseau varie que nous soyons sur terre ou sous terre, en mer ou sous l'eau, ou en altitude. La qualité de la transmission et de la réception peut influer elle aussi sur la qualité audio. Le système réduit automatiquement le débit réseau quand les conditions sont mauvaises. Cela nuit à la qualité audio, même si le son reste prioritaire dans les transmissions (par rapport aux images, par exemple) [50].

### 1.2.2. Norme sécuritaire

La sécurité est devenue une préoccupation principale afin de fournir une communication protégée entre les nœuds mobiles dans un environnement hostile [51] par exemple. Contrairement aux réseaux filaires, les problèmes de sécurité liés aux réseaux mobiles (sans fil) se produisent par l'interception de leurs signaux radio par le piratage et par la non-gestion entière du réseau par l'utilisateur, car la plupart des réseaux sans fil sont dépendants d'autres réseaux privés gérés par d'autres. Quelques-uns des principaux problèmes de sécurité du Mobile Computing standard sont les attaques par déni de service (DOS) [52], l'analyse de trafic [53], le mode écoute [54], l'interception de session et la modification de messages [55], le spoofing [56] et la technique de capture et retransmission de données [57].

Le Mobile Computing introduit un risque important pour les utilisateurs et les entreprises, car les périphériques mobiles sont vulnérables à de nouveaux types d'attaques de sécurité et vulnérables en cas de perte, de vol ou d'altération du périphérique mobile, car 75 % de nos données privées y sont stockées. Cela est également démontré par les documents de violation de données publiés en 2008 par Privacy Rights International que les 20 % des failles de sécurité liées aux données étaient dues à des pertes de périphériques mobiles [58]. L'informatique mobile, comme tout logiciel informatique, peut être endommagée par des logiciels malveillants tels que Virus, Spyware et Trojan [59] [60]. Cependant, la mobilité des utilisateurs et de leurs données introduisent des menaces de sécurité dues à la réplication des données et de profils d'utilisateurs à différents endroits pour permettre l'accès aux données personnelles et sensibles en tout lieu et à tout moment. Mais la répétition de données sensibles sur différents sites augmente les menaces de la sécurité [61]. Le risque est le même lors de la perte du signal ou d'une déconnexion fréquente, lorsque les périphériques mobiles traversent des zones différentes [62]. Il est difficile de prendre en compte la sécurité physique comme dans les systèmes figés. De plus, lorsque

nous travaillons de façon mobile, nous sommes dépendants de réseaux publics, ce qui nécessite une utilisation prudente de réseau privé virtuel (VPN). La sécurité est une préoccupation majeure en ce qui concerne les normes de la mobilité informatique. On peut attaquer le réseau VPN à travers un grand nombre de réseaux interconnectés. Comme illustration, nous avons le cas de la vulnérabilité HeartBleed [63] découverte dans OpenSSL.

### 1.2.3. Gestion d'énergie

Les téléphones mobiles ont subi une évolution fulgurante au cours des deux dernières décennies, à partir de simples appareils dotés uniquement de services vocaux vers des smartphones ou ordiphones offrant des services novateurs tels que l'Internet mobile, la géolocalisation et les cartes, les services multimédias, une connectivité haut débit et bien d'autres [64]. Les téléphones intelligents sont alimentés par batterie pour permettre un plus grand degré de liberté pour l'utilisateur et la batterie doit pouvoir assurer toutes les nouvelles fonctionnalités d'un smartphone en termes de consommation énergétique.

Aujourd'hui, les industries de périphériques mobiles optent pour deux types de batteries :

- Batteries Lithium-ion qui sont réputées comme légères et ont l'avantage d'être moins impactées par l'effet de la mémoire. On parle d'effet de mémoire quand une batterie, rechargée avant son épuisement total, refuse de délivrer la totalité de son énergie ;
- Batteries Lithium-ion-polymère qui sont aussi très légères. Elles peuvent prendre toutes formes et sont parfaitement adaptées aux contraintes des périphériques.

Les deux types de batteries sont identiques en termes de performances et de durée de vie, mais les batteries Lithium vieillissent, même si elles ne sont pas utilisées. Actuellement, la plupart des smartphones sont alimentés par des batteries lithium-ion [65]. Ces piles sont populaires, car elles peuvent offrir plusieurs fois l'énergie d'autres types de piles dans une fraction de temps. Quand les prises de courant ou d'un générateur portable ne sont pas disponibles, les périphériques mobiles doivent compter entièrement sur la puissance de leurs batteries. Pour la gestion de l'énergie, la batterie reste d'actualité dans le domaine du Mobile Computing. Certains chercheurs de l'université de Stanford utilisent la nanotechnologie [66] pour fabriquer des piles capables de produire dix fois plus d'électricité que les piles au lithium-ion existantes. D'autres chercheurs tentent d'exploiter le mouvement de l'utilisateur pour recharger la batterie du téléphone [67], mais ce ne sont pour le moment que des axes de recherche.

Actuellement, les piles sont limitées en capacité et en puissance. Alors que la capacité de stockage de l'énergie a un impact direct sur le temps pendant lequel un périphérique mobile est opérationnel, la limitation de puissance est liée au chauffage du périphérique mobile. Le fait de rendre les périphériques mobiles de plus en plus petits, entraîne un énorme problème de chauffage, faute de dissipation de chaleur. Les auteurs P. Perrucci et al. [68] ont montré qu'un périphérique mobile augmente

significativement sa température lorsque des parties de l'interface radio sans fil sont activées. La consommation d'énergie excessive limite l'évolution des smartphones, car l'amélioration de la capacité de la batterie est assez modérée [69] par rapport à l'augmentation de la complexité due au nouveau matériel et aux services. En fait, comme les batteries peuvent stocker une quantité fixe d'énergie, le temps de fonctionnement offert à un utilisateur pour utiliser son smartphone dans un cycle de charge est également limité. L'auteur Frank Fitzek [65] explique que les parties les plus énergivores d'un téléphone mobile sont les technologies sans fil et non l'écran ou la CPU comme c'est le cas pour les ordinateurs portables. Et pour la communication à courte portée, le Bluetooth doit être utilisé dans le cas où seulement quelques données doivent être échangées et le Wi-Fi dans le cas contraire, c'est-à-dire si plus de données doivent être transmises.

### 1.2.4. Interférences de transmission

La mobilité implique un changement d'environnement. La réception du signal et la qualité du signal sont également fonction du milieu d'utilisation du périphérique mobile [70]. Dans un métro ou dans un tunnel, sur un terrain ou sous terre, nous pouvons soit subir des interruptions momentanées, soit une interférence, soit une perte complète du signal [71]. Par contre en altitude, la portée du signal est meilleure et souvent médiocre dans certains tunnels, certains bâtiments et dans les zones rurales mal desservies. La perte de signal signifie généralement arrêt de la communication et souvent cela entraîne la perte d'un état d'un composant logiciel.

### 1.2.5. Potentiels dangers pour la santé

Certaines personnes utilisent des périphériques mobiles pendant la conduite ou sont souvent distraites, dans les salles d'opération des hôpitaux. Ces personnes sont plus susceptibles d'être impliquées dans des accidents pouvant entraîner la perte de vie. Les périphériques mobiles peuvent interférer avec les appareils médicaux sensibles [70]. Il est établi que les signaux de périphériques mobiles peuvent causer des problèmes de santé à partir de certaine durée d'exposition (trouble de l'audition, addiction, nomophobie, troubles de sommeil, douleurs à la nuque, radiation, …). Cependant, les protocoles mobiles sont bien plus irradiants que le Wi-Fi à cause de leur portée.

### 1.2.6. Interface Homme-Machine

Bien que les performances des périphériques mobiles soient aujourd'hui proches de celles des ordinateurs de bureau, les capacités d'interaction (modalités en entrée et en sortie) sont limitées par la petite taille du périphérique et par les conditions de mobilité (Exemple : marcher dans la rue en portant un sac). En effet, la nécessité pour ces périphériques d'être mobiles, implique une taille d'écran limitée et un faible nombre de boutons physiques [72]. Les écrans et les claviers des périphériques mobiles ont

tendance à être de petite taille, ce qui peut les rendre difficiles à utiliser. Les méthodes d'entrées et de sorties alternatives telles que la parole ou la reconnaissance de l'écriture manuscrite nécessitent une formation. Par conséquent, la taille de l'écran est une limitation importante pour les terminaux mobiles. Le contenu affiché sur un moniteur de 30 pouces nécessite 5 écrans pleins sur un petit écran de 4 pouces. Ainsi, les utilisateurs mobiles doivent supporter un coût plus élevé d'interaction afin d'accéder à la même quantité d'informations et comptent sur leur mémoire à court terme pour faire référence à des informations qui ne sont pas visibles à l'écran. Il n'est donc pas surprenant que le contenu mobile soit deux fois plus difficile à exploiter avec des terminaux de petites tailles.

### 1.2.7. Gestion d'espace mémoire

La plupart des smartphones ont une capacité de stockage fixe. Les smartphones à très bas prix n'embarquent en effet en moyenne 8 Go de stockage interne ; la mémoire flash de stockage bien que limitée est souvent la variable d'ajustement pour atteindre une capacité acceptable. Mais au-delà de l'aspect économique, embarquer si peu de mémoire est problématique à l'usage, pour la sécurité entre autres, et cela condamne l'utilisateur à effacer certains contenus pour en héberger d'autres, comme le précieux SMS que l'on supprimait jadis.

Dans cette section, ont été listées des limites contraignantes des usages des périphériques mobiles. Apporter des solutions pour repousser ces limites a pour objectif d'étendre ou de développer les usages des périphériques mobiles et ainsi offrir aux usagers finaux une meilleure offre de service. Au-delà des limites que nous avons relevées, nous souhaitons contribuer à de meilleurs usages en réduisant certaines barrières.

## 1.3. Besoins de la mobilité

Caractérisés par leur polyvalence, les périphériques mobiles combinent plusieurs appareils spécialisés pour usage unique (baladeur, appareil photo, caméra, …). La maturité des technologies y est pour beaucoup. Cependant, les limitations des ressources observées précédemment doivent pouvoir être résolues ou bien repoussées sans attendre une hypothétique révolution technologique.

### 1.3.1. Sécurité

Les mécanismes d'authentification, de confidentialité, d'intégrité sont à la base des exigences en sécurité [73] dans le monde de l'informatique traditionnelle. Mais avec l'informatique mobile les exigences de sécurité sont devenues plus complexes, en particulier en ce qui concerne la sécurité des données personnelles. L'une des mesures stratégiques de sécurité est le maintien de la dernière mise à jour des éléments du réseau et de leurs logiciels.

Il existe différentes exigences et techniques de sécurité valables tant pour les périphériques mobiles que pour les réseaux, dont certaines incluent :

- Chiffrement : s'il y a une information capitale qui est stockée dans un périphérique mobile, elle devrait être chiffrée pour la protéger de l'accès non autorisé par un tiers externe ou dans le cas où un périphérique est volé. Il contribue également aux aspects de sécurité au niveau de la confidentialité et de l'intégrité [74].
- Authentification : elle doit s'assurer que les périphériques mobiles sont protégés et ont un ensemble d'exigences comme : le verrouillage, les sauvegardes, les logiciels antivirus, et une protection par mot de passe fort [75].
- Solutions de contrôle d'accès réseau (NAC) : il s'agit d'un système utilisé pour vérifier quels périphériques mobiles tentent de se connecter au réseau, c'est-à-dire protéger le réseau contre toute infection ou tout code malveillant pouvant endommager les périphériques mobiles [76].
- Contrôle d'accès : il contrôle l'accès aux fonctions des systèmes informatiques mobiles en fonction de l'emplacement courant de l'utilisateur, et il existe déjà des modèles de sécurité qui identifient de telles approches liées à la localisation. Ces modèles proviennent directement des applications tarifaires de réseau.

Les réseaux sans fil ont relativement plus d'exigences de sécurité que le réseau câblé. Un certain nombre d'approches ont été proposées avec l'utilisation du chiffrement ou de capteurs comme les lecteurs rétiniens et d’empreintes [77].

### 1.3.2. Bande passante

Le besoin en bande passante est lié aux attentes des utilisateurs mobiles en fonction de l’accès à leurs différentes ressources. Comme les limites de la bande passante dépendent de la technologie utilisée et du périphérique mobile, l’utilisation de la bande passante peut être améliorée par la gestion de files d’attente (opérations en bloc contre les demandes de courte durée) et la compression de données avant la transmission. En outre, la technique de mise en cache des données fréquemment consultées peut jouer un rôle important dans la réduction des conflits dans les réseaux sans fil liés à la bande passante. Les données mises en cache peuvent aider à améliorer le temps de réponse des requêtes.

### 1.3.3. Localisation intelligente

Par définition, les périphériques mobiles se déplacent spatialement, ils rencontrent des réseaux avec des caractéristiques différentes. Un périphérique doit être capable de passer du mode infrarouge en mode radio comme il se déplace de l'intérieur vers l'extérieur. En outre, il doit être capable de passer du mode de fonctionnement cellulaire au mode satellitaire lorsqu’il se déplace en zones urbaines ou rurales. Dans la mobilité informatique, les périphériques mobiles travaillent dans des cellules et sont desservis par

différents fournisseurs de réseau, la distance physique peut ne pas refléter la vraie distance du réseau. Un petit mouvement peut entraîner une trajectoire beaucoup plus longue si les limites des cellules ou du réseau sont croisées. Elle permet également de mettre à jour des informations dépendantes de l'emplacement comme décrit ci-dessus. Cela peut augmenter la latence du réseau, ainsi que le risque de déconnexion. Les connexions de service doivent être transférées dynamiquement sur le serveur le plus proche. Cependant, lorsque l'équilibrage de charge est une priorité, cela peut ne pas être possible.

### 1.3.4. Consommation d’énergie et d’espace de stockage

Les périphériques mobiles ont pour principale source d’énergie leurs batteries. La section précédente nous a permis de montrer les limites des batteries, ce qui implique un besoin énergétique, car il n’existe pas encore de solution. En dehors des batteries auxiliaires, la recharge régulière des périphériques reste obligatoire, mais quelques règles, ou bonne pratique pour une meilleure autonomie sont à observer :

a) Faire preuve de bon sens en :
   - Baissant la luminosité de l’écran ou en évitant des fonds d’écran animés ;
   - Limitant la synchronisation des comptes (Mails, WhatsApp ou autres) qui actualisent constamment les données et les notifications ;
   - Désactivant la géolocalisation et l’interface Bluetooth en cas de non besoin ;
   - Évitant le vibreur ou en fermant les applications non utiles.

b) Surveiller la surchauffe du périphérique mobile : elle réduit considérablement son autonomie ainsi que la durée de vie de la batterie. Dans ce cas, nous pouvons consulter les paramètres des périphériques sur l’utilisation de batterie et repérer l’application responsable de ce dysfonctionnement ;

c) Télécharger une application pour l’optimisation de la batterie : les optimiseurs les plus fiables comme Clean Master, permettent de libérer de l’espace sur la mémoire interne. Ils proposent également l’économie de la consommation, un mode de recharge qui préserve la durée de la vie de la batterie ainsi que la surveillance de la température du périphérique.

Les différents besoins que nous évoquons, sont parfois en opposition et il est alors utile de prioriser. Ainsi, l’ajout de disque externe ou l’utilisation de plusieurs cartes SIM répondent à un besoin de repousser une limite mémoire entre autres. En revanche, de tels ajouts grèvent de manière conséquente l’autonomie en énergie.

### 1.3.5. Révision architecturale

Les besoins métiers des applications mobiles sont toujours plus riches. Pour y répondre de nouvelles technologies sont développées, qui permettent de définir de nouvelles architectures logicielles. Pour

fournir une connectivité complète entre les usagers mobiles et les systèmes d'information traditionnels d'entreprise, il convient de réviser pour intégrer la connectivité mobile. En outre, les architectures des applications d'entreprise doivent également être révisées pour satisfaire aux demandes imposées par la connectivité mobile. Le Cloud Computing apporte un début de réponse à ces différents besoins, mais nécessite une connexion réseau haut débit.

Nous avons fait un constat partiel des limites et des besoins qu'apporte l'usage de la mobilité. Envisager des solutions nouvelles, nécessitent d'exploiter la mobilité différemment et plus particulièrement les usages réseaux afin de faire partager ces limites et besoins avec les autres stations accessibles via le réseau. Le partage avec des machines désignées par leurs identifications a toujours été possible mais réservé à des situations à forte technicité. L'usage du Cloud permet de masquer cette identification et cacher ainsi la localité des ressources matérielles utilisées par les périphériques mobiles pour repousser ses limites et ainsi satisfaire de nouveaux besoins.

# II. Cloud local

Les usages du Cloud sont aujourd'hui fort répandus et diversifiés. Pour schématiser, nous les rangeons en deux grandes catégories : le stockage et le calcul. Des Clouds sont mis à notre disposition dans le cadre professionnel ou privé. Ainsi, Orange offre une capacité de stockage de plusieurs Giga-octets à ses abonnés. Google permet le déploiement gratuit d'une vingtaine d'applications grâce à App-Engine [78]. Il est ainsi possible d'accéder depuis n'importe quelle machine, ayant des aptitudes réseau, à des données sauvegardées dans un cloud ou à des applications déployées dans un cloud.

La norme ISO 29151, pour mieux responsabiliser les acteurs du Cloud, couvre l'intégralité du cycle de vie de la donnée dans le Cloud. En France, la CNIL (Commission Nationale de l'Informatique et des Libertés) a établi les recommandations qui sont basées sur une analyse de risques réalisée au préalable par les clients et des engagements de transparence des prestataires vis-à-vis de leurs clients [79]. Le terme Cloud dans ce document fait référence à une plate-forme de Cloud Computing.

## 2.1. Définitions et Concepts

Afin de comprendre les notions de Cloud Computing que nous allons exploiter, nous présentons un ensemble de définitions des diverses facettes du Cloud.

Selon Lozano et al. [80], le Cloud Computing est un type de calcul qui offre un accès simple, sur demande d'un ensemble de ressources informatiques hautement élastique. Ces ressources sont fournies comme service sur Internet. Le Cloud permet aux utilisateurs de minimiser le coût d'utilisation, sans se préoccuper de la façon dont les ressources sont gérées, ni où elles se trouvent.

Selon Vaquero et al. [81], le Cloud est un vaste regroupement de ressources virtuelles, facilement accessibles et utilisables. Ces ressources peuvent être dynamiquement reconfigurées pour s'adapter à une charge variable, ce qui permet une utilisation optimale des ressources. Ce regroupement de ressources est généralement exploité par un modèle pay-per-use, dans lequel des garanties sont offertes par le fournisseur de services au moyen des contrats de niveau de service.

Selon F. Gens et al. [82], le Cloud Computing est un modèle émergeant, de développement, de déploiement et de livraison, qui permet d'offrir des services en temps réel sur Internet.

Selon l'organisme NIST (National Institute of Standards and Technologie), Cloud Computing est un modèle informatique qui permet un accès facile et à la demande par le réseau à un ensemble partagé de ressources informatiques configurables (serveurs, stockage, applications et services) qui peuvent être rapidement provisionnées et libérées par un minimum d'efforts de gestion ou d'interaction avec le fournisseur du service [83]. En analysant ces définitions, nous constatons que la définition du Cloud tourne autour de ses principales caractéristiques, que nous pouvons diviser en deux parties :

- Du point de vue fournisseur de services Cloud : avec les définitions [81, 82], nous trouvons les caractéristiques suivantes : l'élasticité, mise à jour automatique, évolutivité massive, auto-approvisionnement de ressources, et ressources partagées.
- Du point de vue utilisateur final du Cloud : avec les définitions [83, 84], nous trouvons les caractéristiques suivantes : l'instantanéité, la disponibilité, l'accès sur demande, pay per use.

Le Cloud Computing peut aussi être perçu comme un ensemble d'outils de virtualisation et de management de ressources, basé sur de puissantes machines. Ces machines sont regroupées en un seul système, dont les ressources sont éventuellement distribuées, et qui doit assurer et satisfaire un ensemble de caractéristiques, et cela, pour faciliter l'accès à un large spectre de ressources. Le concept du Cloud se différencie de l'informatique traditionnelle par trois aspects importants, à savoir, les services avec mise à jour automatique du contenu, les données sont en libre-service et le paiement s'effectue à l'usage et la mutualisation et l'allocation dynamique de capacité [84]. Il existe différentes offres de service de Cloud Computing sur le marché, qui peuvent être distinguées selon trois modèles de services et trois modèles de déploiement.

### 2.1.1. Modèle de service

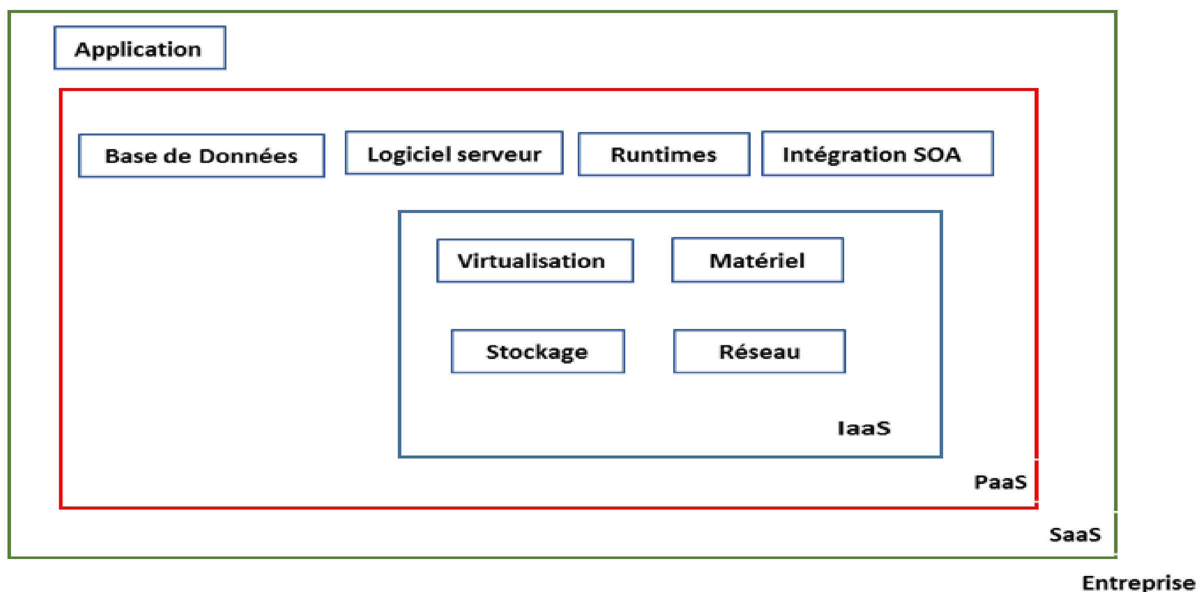
Le Cloud Computing offre trois niveaux de service, à savoir, le Software as a Service (SaaS), la Plate-forme as a Service (PaaS) et l'Infrastructure as a Service (IaaS).

Le Software as a Service est la partie applicative du Cloud Computing. Pour les personnes ou l'entreprise utilisatrice, il s'agit de l'accès à distance grâce au réseau Internet à des applications hébergées et exploitées par un fournisseur de services qui facture le droit d'usage, c'est-à-dire la fourniture de logiciel en ligne où tout est complètement transparent pour les utilisateurs. Il concerne les applications

d’entreprise telles que : CRM, outils collaboratifs, messagerie, BI, ERP, etc. Ce modèle consiste à déporter une application chez un fournisseur de service. Il convient à certaines catégories d’applications qui se doivent d’être globalement identiques pour tout le monde [85]. Un périphérique mobile est un bon candidat pour accéder à une application déportée dans un Cloud.

La Platform as a Service concerne davantage les environnements middleware de développement et de test. Ce modèle consiste à mettre à disposition un environnement prêt à l’emploi, fonctionnel et performant, y compris en production. C’est le lieu idéal pour l’installation d’un serveur OSGi tel que Apache Karaf. Il présente un certain nombre d’avantages comme la flexibilité, l’adaptabilité, la sécurité et bien d’autres. C’est le fournisseur du service cloud qui administre le système d'exploitation et ses outils. Le client peut installer ses propres applications si besoin. L’Infrastructure as a Service consiste à mettre à disposition une infrastructure informatique hébergée telle que serveurs, moyens de stockage et d’autres capacités réseau. L’accès à la ressource est complet et sans restriction, équivalent à la mise à disposition d’une infrastructure physique réelle à travers une connexion publique, généralement Internet. Il présente les avantages comme l’extensibilité, l’indépendance de la localité, la sécurité des sites de Datacenter et bien d’autres. C’est le niveau d’installation d’un Cloud local ou micro Cloud.

*Figure 1.6 Services du Cloud vue sous forme de couches*




La Figure 1.6 illustre certaines des fonctionnalités pouvant être incluses dans les différentes couches de services. La couche SaaS englobe les services de la couche PaaS qui englobe également les services de la couche IaaS.

### 2.1.2. Modèle de déploiement

En plus des modèles de service qui permettent de concrétiser les services Cloud, on trouve un ensemble de modèles de déploiement du Cloud Computing. Ces modèles permettent de définir le degré d'accès de l'utilisateur final aux fournisseurs du Cloud. Ces modèles sont divisés en trois grandes catégories.

#### a) Cloud public

Le Cloud public, appelé aussi Cloud externe [85], représente le Cloud traditionnel utilisé par la majorité des clients sur Internet. C'est une solution de Cloud Computing ouverte, gérée par un fournisseur tiers. Le terme « Public » veut dire collaboratif et accessible via un réseau public et non que les services sont gratuits ou les données des utilisateurs sont rendues publiques et visibles par les autres utilisateurs. Le Cloud public est une bonne solution pour les entreprises qui ont besoin d'un service immédiat. Dès la finalisation de l'abonnement, le service est disponible et prêt à être utilisé. Les fournisseurs du Cloud public les plus connus sont Google et Amazon avec Google App Engine, Amazon OpsWork / Elastic Beanstalk et Heroku. C'est la solution simple pour déployer un service REST accessible depuis un smartphone (utilisation de SaaS).

Les avantages du Cloud public :

- Il est utilisable instantanément et accessible à tous les budgets ;
- Il est adapté au développement et à l'expérimentation ;
- Le Cloud public est parfaitement "élastique" afin de s'adapter à l'augmentation des besoins d'une entreprise.

Les inconvénients du Cloud public :

- Le Cloud public, bien que flexible, n'est pas forcément adapté à tous les besoins d'une entreprise, n'étant pas fait sur-mesure comme le Cloud privé ;
- Le Cloud public n'assure pas la portabilité avec d'autres Clouds publics ;
- Le Cloud public étant facturé en fonction de l'utilisation du service, son coût peut s'élever rapidement. L'utilisation de services devra donc être surveillée afin d'éviter des dépassements budgétaires.

#### b) Cloud privé

Le Cloud privé est une solution de Cloud Computing entièrement dédiée à une entreprise, qui est la seule à pouvoir l'exploiter. Il s'agit donc le plus souvent d'un Cloud interne, dont l'accès est sécurisé et n'est possible qu'au sein de l'entreprise via des réseaux privés [86]. Il permet un contrôle total des données et des applications. A contrario, les organisations qui ciblent ce genre de modèle, doivent elles-mêmes posséder, gérer, et restreindre le nombre des personnels nécessaires à la construction et l'utilisation des ressources du Cloud. Ceci nécessite de grands investissements d'argent et de personnel

pour mettre en œuvre leur Cloud privé. La différence principale entre un Cloud privé et un Cloud public est le fait que les ressources d'un Cloud privé sont destinées seulement aux clients autorisés par l'organisation qui possède les ressources, et elles ne peuvent pas être partagées avec d'autres clients extérieurs. OpenStack, OpenNebula et Eucalyptus sont des exemples de solution pour la mise en place du Cloud privé. C'est une solution simple pour déployer une machine virtuelle représentant un smartphone (utilisation IaaS).

Les avantages du Cloud privé :

- Il est conçu sur-mesure pour l'entreprise ;
- Son coût est fixe (déterminé en fonction de la taille de l'infrastructure).

Les inconvénients du Cloud privé :

- Il s'agit d'un investissement coûteux, dont il faut prévoir l'amortissement ;
- Les contraintes liées à l'évolution des besoins de l'entreprise ;
- Le temps nécessaire pour adapter la taille de l'infrastructure aux besoins de l'entreprise peut être trop élevé par rapport à la vitesse d'évolution de celle-ci.

#### c) Cloud hybride

Le Cloud hybride est une solution de Cloud Computing mixte. Elle permet à une entreprise d'alterner entre le Cloud privé et le Cloud public en fonction de ses besoins. Grâce au Cloud hybride, les ressources disponibles dans le Cloud public permettent d'anticiper les dépassements de capacité du Cloud privé. Le Cloud hybride est une bonne alternative pour les entreprises ayant des besoins complexes. Comme exemple, nous pouvons avoir un Cloud dédié pour les données et un autre pour les applications [87].

Les avantages du Cloud hybride :

- Chaque donnée est naturellement conservée dans l'environnement Cloud le plus adapté ;
- Cette solution permet de combiner les avantages majeurs du Cloud public (flexibilité, rapidité de la mise en place, développement et expérimentation) et ceux du Cloud privé (sécurité et contrôle total des données).

Les inconvénients du Cloud hybride :

- Cette solution est cependant exposée aux inconvénients des différents types de Cloud et aux risques lors du déploiement de chaque solution Cloud ;
- L'utilisation de deux types de Cloud différents augmente la gestion nécessaire et apporte des problèmes d'interopérabilité entre Clouds.

### 2.1.3. Architecture logicielle adaptée au Cloud

Les architectures logicielles adaptées au Cloud font généralement apparaître l'usage du Cloud par la couche SaaS. Cela signifie par l'emploi de Web Services éventuellement sécurisés. Tout d'abord, ces architectures répondent aux principales difficultés entourant le traitement de données à grande échelle. Dans le traitement traditionnel des données, il est difficile d'obtenir autant de machines que souhaité à une date convenue. De plus, des problèmes traditionnels apparaissent lors de la distribution et la coordination des traitements distribués sur un ensemble de machines. Enfin, la tolérance aux incidents ajoute encore un niveau de difficulté avec le besoin de reprise sur l'incident. Les applications construites sur les architectures logicielles adaptées au Cloud sont telles que l'infrastructure de calcul sous-jacente est utilisée uniquement lorsqu'il est nécessaire d'extraire les ressources nécessaires à la demande (comme les serveurs de calcul et de stockage) et d'effectuer une tâche spécifique. À partir d'un modèle, il est possible de créer automatiquement, par exemple, une architecture multi-tiers composée de deux serveurs Web connectés à Internet et à trois serveurs de base de données sur un réseau séparé.

Le Cloud a pour principe de faire abstraction des couches plus basses et de permettre la mise en place d'architectures scalables tout en permettant une programmation respectant des designs patterns imposés. Un Cloud OpenStack permet aux utilisateurs de travailler dans des environnements isolés. Les services (XaaS[9]) et leurs fonctionnalités sont accessibles au niveau du Cloud par les APIs. Cette plate-forme permet la migration des applications d'un Cloud à un autre. Elle assure l'hébergement d'une application sur plusieurs Clouds. Les applications déployées sur ce type d'architecture sont entre autres des applications Web ou WebApp. De façon générale, une application offrant des services sur le Web est une bonne candidate pour une installation dans une architecture logicielle adaptée au Cloud. Elle sera ainsi déployée en assurant la transparence de localisation à ses usagers. Enfin, dans le cadre de la composition de services Web, il est convenu d'installer un bus logiciel (ou ESB) (Utilisation PaaS). Ce bus permet la construction de nouveau service en tant qu'orchestration de plusieurs services Web déjà déployés.

### 2.1.4. Mobile Cloud Computing

Le Mobile Cloud Computing réunit l'ensemble des nouvelles technologies du Cloud Computing classique, les périphériques mobiles et les réseaux sans fil. Kovachev et al. [88] ont défini le Mobile Cloud Computing par : «Mobile Cloud Computing is a model for transparent elastic augmentation of mobile device capabilities via ubiquitous wireless access to cloud storage and computing resources, with context-aware dynamic adjusting of offloading in respect to change in operating conditions, while preserving available sensing and interactivity capacibilities of mobile devices».

[9] XaaS (Anything as a Service / Tout en tant que service) est le concept regroupant toutes les solutions déployées en mode service.

De cette définition et du concept de Cloud, nous pouvons stipuler que le Mobile Cloud Computing consiste à externaliser les traitements, les calculs et le stockage des données à l'extérieur des appareils mobiles vers des plate-formes puissantes et centralisées sur Internet en préservant l'interactivité des dispositifs mobiles. La déportation des tâches de calcul d'utilisateurs mobiles vers des fournisseurs de ressources distants au lieu d'être calculées par les périphériques mobiles eux-mêmes est effectuée au niveau du Cloud tandis que l'affichage et l'usage des capteurs sont laissés au périphérique mobile. Il se réfère alors à la capacité d'exécution des applications mobiles et des calculs en utilisant des fournisseurs de ressources autres que les périphériques mobiles eux-mêmes. Les outils du Cloud classique proposent des solutions basées sur le traitement parallèle de données volumineuses et la flexibilité liée à la virtualisation des machines.

## 2.2. Comparaison entre les différents Clouds locaux

Les travaux actuels dans les réseaux et le Mobile Cloud Computing visent à améliorer la bande passante, le stockage, la sécurité, la consommation d'énergie et bien d'autres limites. Pour cela, plusieurs techniques ont été utilisées et impactent négativement la latence [89]. Pour pallier à ce problème, le rapprochement du Cloud des périphériques mobiles qui les utilisent permet de diminuer cette latence. D'où l'intérêt de Cloud local qui représente des couches intermédiaires situées entre le Cloud et les périphériques mobiles. De cette façon, au lieu d'accéder au Cloud distant, les utilisateurs se connectent au Cloud local le plus proche géographiquement grâce à un périphérique mobile ayant un accès à un réseau sans fil local. Dans la littérature, plusieurs architectures ont été définies pour illustrer le Cloud local. Certaines d'entre-elles sont abordées dans la section suivante.

### 2.2.1. Différents Clouds locaux

Le concept de Cloud local est apparu récemment, connu également sous le nom de serveur de proximité ou Cloudlet [90], Edge Computing [91], Fog Computing [92], Small Cell Cloud [93], Follow Me Cloud [94], etc. En termes d'objectifs, ils sont pratiquement les mêmes, mais d'une part les usages sont différents, d'autre part les technologies mises en œuvre restent ciblées sur le cas d'utilisation auquel elles répondent (distribution, réactivité, etc.). Dans la section suivante, nous allons mettre un accent sur les concepts de Micro Cloud et Cloudlet. Le terme Cloudlet est utilisé dans le reste du document pour illustrer le Cloud local.

### 2.2.2. Micro Cloud Computing

Le Micro Cloud Computing est une extension de la mobilité du Cloud Computing dans un environnement de réseau ad-hoc basé uniquement sur les périphériques mobiles. C'est un concept architectural permettant de faire usage de ressources sur les périphériques individuels pour fournir un Cloud mobile virtuel. Il permet de fournir des services mobiles de stockage et de traitement des données

des utilisateurs sans accéder au Cloud. Le matériel se compose en tout ou en partie d'appareils mobiles. L'idée principale de Micro Cloud Computing est de rapprocher les services de calcul et de stockage des utilisateurs, où des petits serveurs ou centres de données qui peuvent héberger des applications du Cloud, sont répartis sur le réseau et connectés directement à des entités (telles que des stations de base cellulaires) au niveau du réseau.

Les Micro Cloud Computing sont installés à des emplacements fixes tels que les points d'accès (APs) et peuvent être utilisés pour de nombreuses applications nécessitant une latence faible, une capacité de traitement de données élevée ou une fiabilité élevée [95]. Ils connaissent un développement rapide avec la croissance de nouvelles applications mobiles comme strava, doctisia et de smartphones plus avancés, et sont également plus robustes que les systèmes de Cloud Computing traditionnels.

### 2.2.3. Cloudlet

Dans le principe du Cloud Computing, la déportation d'applications ou de données se fait dans des data centers distants de très grande capacité. Une Cloudlet utilise à peu près le même principe, mais la déportation s'effectue dans des réseaux privés situés dans un voisinage géographique.

Satyanarayanan et al. [91] évoquent le concept de Cloudlet comme la mise en place de serveurs de proximité sans état, connectés à Internet auxquels les équipements mobiles peuvent se connecter directement via un réseau Wi-Fi. De ce concept, la Cloudlet est un système autogestionnaire et riche en ressources qui offre un accès rapide aux services Internet et Cloud. Elle se compose d'une ou plusieurs machines en réseau avec une connectivité interne à grande vitesse et des ressources informatiques, facilement disponibles. Ces équipements sont typiquement déployés comme des hotspots Wi-Fi dans les salles de Travaux Pratiques, universités, magasins, etc. Les Cloudlets offrent de nombreux avantages par rapport au Cloud. Nous pouvons citer : une latence plus faible, une bande passante plus élevée, une disponibilité hors ligne ou une rentabilité. Satyanarayanan et al. [96] ont implémenté OpenStack++ qui est une extension de OpenStack. OpenStack++ fournit un ensemble d'APIs spécifiques au Cloudlet et permet d'intégrer efficacement les Cloudlets avec OpenStack. Il est possible de considérer une Cloudlet comme un cache local d'un Cloud distant. On peut dans ce cas parler de Cloudlet publique et Cloudlet privée en se référant aux notions émises en section 2.1.2.

## 2.3. Communication entre un périphérique mobile et une Cloudlet

L'évolution des technologies et des services du Web a engendré un large éventail de normes et de protocoles d'interaction entre les composants logiciels. Les exemples vont des applications Web, des

mashups[10], des applications mobiles et des périphériques mobiles aux services d'entreprise et des utilisateurs. Plus particulièrement la Cloudlet peut être considérée comme l'industrialisation des prestations de services sur Internet en utilisant les technologies établies à partir du Web [97]. Pour communiquer avec les services Web situés dans la Cloudlet, les périphériques mobiles utilisent la norme MIME (Multipurpose Internet Mail Extensions) pour spécifier les types de contenu. Trois des langages les plus influents pour l'échange des informations dans le Web sont HTML, XML et JSON, mais des échanges au format octet-stream sont aussi possibles.

### 2.3.1. Communication Web Client - Client

Un protocole récent du côté client, a été pris en charge par de nombreux navigateurs modernes, c'est Web Real-Time Communications (WebRTC, voir section 1.1.3.), pour permettre une interaction directe entre les clients. Il permet des échanges d'informations en temps réel sans la nécessité d'un service tiers pour les appels vidéo et vocaux pour éviter les retards et les goulets d'étranglement du réseau. WebRTC utilise UDP comme transport et a encore besoin d'un service pour la découverte et la signalisation entre les clients, mais aussi pour traiter les translations d'adresses (NAT) ou les pare-feux sur le chemin entre deux clients. Pour l'échange d'informations générales, WebRTC dispose d'une chaine de données RTC appelée à établir une association SCTP (Stream Control Transmission Protocol) directe, protégée par DTLS (Datagram Transport Layer Security), entre deux clients [98]. L'interaction dans WebRTC est basée sur le message et une association peut fournir un ordre et un transfert fiable. En termes de modèles, une chaîne de données RTC prend en charge l'interaction d'envoi et de réception entre les clients.

### 2.3.2. Architecture d'interaction entre Cloudlet-Cloudlet et Périphérique-Cloudlet

Une telle architecture combine des protocoles, des langages et des modèles d'interaction de service pour la prestation de services. Les architectures se distinguent en deux visions l'une orientée Web et l'autre orientée service.

a) Vue orientée Web

Le World Wide Web (WWW) présente du contenu multimédia comme le texte, l'audio et la vidéo d'une manière non-linéaire, où les utilisateurs peuvent accéder à l'information par le biais d'hyperliens. Les applications Web sont considérées comme l'architecture de référence pour la diffusion hypermédia tandis que la syndication Web est une forme de composition. La syndication Web permet de diffuser des notifications d'événements ou de transférer du contenu des services vers les clients et vers d'autres applications Web pour l'interaction service-service. Un mashup compose des composants dits Web, par

[10] mashup est une application qui combine du contenu ou du service provenant de plusieurs applications plus ou moins hétérogènes

exemple des ressources multimédias ou du code script de différentes origines en un nouveau site Web. Un service mashup, également appelé intégrateur, peut être distingué en fonction de l'emplacement, où l'intégration des composants Web a lieu.

b) Vue orientée services

Pour offrir des services dans des environnements Cloudlet, quatre approches classiques sont utilisées :

- Appels de procédures à distance ou RPC (Remote Procedure Call)

RPC est un style architectural simple mais puissant pour offrir un service en exposant les fonctions accessibles au réseau. En termes de modèles, RPC est bilatérale Send-Receive entre un client et un service. Le client déclenche l'interaction et, sauf indication contraire, un appel de fonction réseau dans RPC est synchrone et les interfaces sont statiquement déclarées. La spécification d'un service RPC nécessite un accord. L'évolution de l'architecture de base pour un Web RPC est XML-RPC. L'information d'appel et de retour est sérialisée grâce à un mapping de données dont le format peut-être du XML ou du texte. JSON-RPC permet de spécifier les appels et retours via les formats basés JSON. Remarque : il est possible de faire un service Web SOAP en utilisation RPC.

- Services Web SOAP

Un service Web traite des documents XML et de l'encapsulation de documents pour échapper à la complexité du calcul distribué sur les objets partagés. La technologie de base utilisée ici est le protocole simple d'accès aux objets (SOAP) pour exprimer des messages en tant que documents XML. Pour décrire les services Web métiers, le protocole SOAP utilise le WSDL (Web Service Description Language) pour sa déclaration. Tout service Web métier possède un contrat de service décrit en WSDL. Avec une simple transformation XSL-T, il est possible de l'utiliser pour engendrer une requête SOAP. L'élément le plus important à regarder dans un document WSDL est le port-type ou interface d'appel qui donne les points de connexion au service Web. Un client appelant un service Web défini sur un serveur lui envoie une requête SOAP et le serveur lui retourne une réponse utilisant le protocole SOAP dans le cas d'un message exchange pattern (MEP) du type Request/Response. Actuellement, il est convenu de construire des services métier qualifiés de style Document (et non RPC) et d'encodage Literal, c'est-à-dire sous forme de chaîne de caractères bruts.

Ce protocole SOAP a pour particularité de disposer d'un entête (Header) optionnel dans lequel il est possible d'ajouter des contraintes de sécurité, des informations pour la gestion de transaction logicielle ou des directives de qualité de service.

- Services REST

Le style architectural REST devient de plus en plus un standard et s'impose sur le marché par rapport à SOAP. REST n'est ni un protocole ni un format, c'est un style d'architecture logicielle, c'est-à-dire un ensemble de contraintes bien adapté au Web. Les applications qui répondent aux contraintes imposées par ce style architectural sont dites "RestFul". Ce style architectural a été introduit par Roy Fielding [99]

dans son travail de doctorat en 2002. Pour Roy Fielding, une application Web n'est autre qu'un réseau de pages Web. L'une des contraintes du style architectural REST est que les applications doivent être constituées de clients de serveurs séparés par une interface uniforme. L'information de base dans une architecture REST est appelée ressource. Le style architectural REST permet d'utiliser une interface uniforme pour accéder aux services Web, son utilisation entraîne une meilleure gestion de la bande passante. Un exemple de framework pour le développement de logiciel RestFul est HATEOAS (Hypermedia As The Engine Of Application State) de Spring [100]. À partir d'une structure de données métier, ce framework construit dynamiquement l'ensemble des services REST permettant une navigation à chaque niveau. Ainsi, lors de la navigation par un client, une requête de celui-ci fournit non seulement les données auxquelles il veut accéder, mais aussi toutes les URLs offrant les accès aux niveaux voisins. D'un point de vue simplicité, le développement REST apparaît plus prometteur que SOAP. Mais la gestion d'une politique de sécurité peut-être mise en œuvre en SOAP alors qu'une architecture REST impose l'usage du protocole oauth2 [101].

- Middleware orienté Message

Pour faire face aux exigences croissantes en matière d'évolutivité, de flexibilité et de fiabilité, un middleware orienté message (MOM) est une infrastructure pour une communication intraprocessus faiblement couplée dans un bus de service d'entreprise (ESB) ou des Clouds. Surtout, dans les Cloudlets, le couplage permet d'exposer rapidement les producteurs de messages et les consommateurs. Un message concernant un MOM est une entité autonome qui modélise un événement, un document ou une commande et se sépare en un entête et un corps appelé payload. Le middleware fournit des moyens techniques d'échange, de sorte qu'un pair peut échanger des messages avec d'autres pairs connectés. Un message SOAP répond à cette structure.

Un concept central dans un MOM est la notion de message, File d'attente (ou canal) pour stocker, transformer et transférer des messages. Il représente d'ailleurs un pattern d'intégration de base [102]. Il existe deux approches différentes de MOM en utilisant des files d'attente de messages comme :

- La messagerie pair-to-pair : un composant middleware unifié de chaque pair coordonne la découverte et l'interaction entre pairs.
- La messagerie basée sur le fournisseur : le middleware agit comme un courtier pour fournir une infrastructure de messagerie entre les pairs hétérogènes.

Les pairs peuvent participer en tant que client, service ou les deux. Un fournisseur réduit la complexité de la communication entre un certain nombre de pairs. Cela peut entraîner des retards dans les applications en temps réel, car une procédure supplémentaire de stockage et de renvoi est nécessaire. Enfin, un système de messagerie permet de supprimer des blocages dus aux messages exchange patterns synchrones, il offre ainsi des propriétés de compositions logicielles.

Parmi ces services orientés, certains sont plus adaptés pour l'accès à un micro Cloud ou une Cloudlet. Ainsi, l'emploi de requêtes REST peut devenir un moyen de déclenchement de la déportation d'un calcul depuis un poste mobile vers une Cloudlet, voire son rapatriement sur le poste mobile.

c) Offloading d'application dans une Cloudlet ou une micro Cloud

La déportation de calcul se trouve étroitement liée au Cloud Computing et à la Cloudlet. C'est un mécanisme qui consiste à déporter l'exécution des tâches lourdes sur des machines plus puissantes (Cloudlet). La demande croissante d'applications mobiles pour des capacités de calcul et de stockage élevé avec une mobilité libre des utilisateurs a fait des Cloudlets, une solution efficace pour la déportation de la charge de travail des utilisateurs finaux. La mobilité des utilisateurs finaux entraîne le besoin de faire un suivi d'activité de ces utilisateurs sur un réseau de Cloudlets. Ainsi, la déportation d'une application d'un utilisateur depuis une plate-forme mobile n'impose en rien à cet utilisateur de rester dans le voisinage de la Cloudlet voisine lors de son action de déportation. Aussi, ce suivi correspond à des échanges inter Cloudlets dont l'ensemble doit couvrir la zone de déplacement de l'utilisateur.

Au cours de cette section, nous avons pu observer le rôle crucial du Cloud local dans les échanges avec un réseau mobile. De nouveaux problèmes apparaissent tels que la déportation d'applications et son retour sur le poste initial ou encore la migration de l'application déportée afin d'assurer une proximité avec l'utilisateur de cette application. Il apparaît alors que plusieurs utilisateurs peuvent partager l'usage d'un réseau de Cloudlets et qu'il devient essentiel d'isoler les applications déportées afin que l'exécution de l'une soit sans effet sur l'exécution des autres. Pour cette raison, des concepts de virtualisation sont à étudier.

# III. Virtualisation

Le concept de "virtualisation" couvre l'ensemble des techniques permettant de dissocier les caractéristiques physiques d'un système matériel ou logiciel des applications orientées utilisateurs. La virtualisation est utilisée pour permettre le fonctionnement de plusieurs machines virtuelles disposant chacune de leur système d'exploitation spécifique partageant la même infrastructure physique. Elle facilite la mutualisation et la déportation des ressources. Les spécificités techniques des unités informatiques de traitement et de stockage du Cloud Computing sont transparentes pour l'utilisateur. Techniquement, le Cloud Computing résulte du fruit des technologies existantes comme Internet et la virtualisation. La souplesse de montée en charge avec une capacité théoriquement infinie n'est pas le moindre des avantages. La virtualisation donne lieu à plusieurs aspects, à savoir les périphériques, les applications, le stockage et le réseau.

## 3.1. Virtualisation des périphériques mobiles et serveurs

Dans cette section, nous présentons la technologie de la virtualisation qui a pour objectif d'émuler le matériel qui héberge un ensemble de systèmes virtualisés.

### 3.1.1. Virtualisation des postes de travail et des périphériques mobiles

Le déploiement de postes de travail sous la forme d'un service géré offre la possibilité de réagir plus rapidement à l'évolution des besoins et des opportunités. Les entreprises sont extrêmement focalisées sur les utilisateurs finaux pour ce qui est de la personnalisation des applications, des outils et de la mobilité, car elles souhaitent leur offrir une expérience utilisateur efficace en matière de virtualisation des clients. Par conséquent, le centre de toutes les attentions et la clé du succès de ce type de technologie de virtualisation sont l'utilisateur final [103]. Le Mobile Computing permet aux utilisateurs finaux de travailler au bureau et en déplacement. La mobilité des utilisateurs finaux est l'une des principales facilitatrices du déploiement de la virtualisation des postes de travail. Les connexions réseau, la nécessité d'un accès distant et les techniques d'optimisation WAN diffèrent selon les utilisateurs. Par conséquent, pour réduire la complexité informatique, les responsables informatiques doivent traiter les groupes d'utilisateurs en fonction de leur emplacement. L'infrastructure de périphériques virtuels (VDI) offre un espace de travail pour les utilisateurs finaux virtuels, qui permettent de livrer virtuellement les périphériques et/ou les applications, à partir du Cloud. Cette infrastructure gère et contrôle les données qui peuvent être enregistrées sur les périphériques mobiles. Elle prend en charge les applications qui ne peuvent être utilisées en mode natif. Comme avantage, la virtualisation permet l'emploi optimal des ressources d'un groupe de périphériques (répartition des périphériques virtuels sur les périphériques physiques en fonction des charges respectives) et l'installation, les tests, les développements avec l'économie sur le matériel (consommation électrique, entretien physique, surveillance). L'exemple le plus représentatif de cette technologie est Qemu [104].

### 3.1.2. Virtualisation des serveurs

De nos jours, la majorité des entreprises optent de plus en plus pour réduire le nombre de leurs serveurs à cause du coût important qu'engendre la maintenance, la mise à jour et le fonctionnement de ces équipements [105]. La plupart des serveurs utilisent moins de 15 % de leurs capacités, ce qui favorise leur prolifération et leur complexité. La virtualisation des serveurs résout ces problèmes d'efficacité en permettant d'exécuter plusieurs systèmes d'exploitation sur un même serveur physique sous la forme de machines virtuelles, dont chacune peut accéder aux ressources de calcul du serveur sous-jacent [106].

L'étape suivante consiste à regrouper un cluster de serveurs en une seule et même ressource consolidée, de façon à optimiser l'efficacité globale et à réduire les coûts. La virtualisation des serveurs permet également d'accélérer le déploiement des charges de travail, de stimuler les performances applicatives et de maximiser la disponibilité. En revanche, cette mutualisation des coûts n'est pas sans limite et certaines propriétés telles que la taille mémoire, celle du disque amènent des contraintes d'usages. Il redevient important de pouvoir redimensionner ces propriétés pour ne pas risquer de stopper l'exécution d'une telle machine virtuelle. L'exemple le plus significatif de cette technologie est VMware vSphere [107].

## 3.2. Virtualisation du stockage de données

Avec la virtualisation du stockage, de multiples dispositifs indépendants, répartis sur le réseau et de niveaux différents (mémoires flash, disques, stockage dans le Cloud) apparaissent. Du point de vue de la gestion, ils sont considérés soit comme un unique pool de stockage, monolithique et partagé, soit comme différents pools pouvant être affectés au gré des besoins et gérés de manière centralisée. La virtualisation du stockage isole les disques et les lecteurs Flash installés sur les serveurs et les combine dans de vastes pools de stockage haute performance, qu'elle met à disposition sous forme de logiciel. Une nouvelle approche de stockage offrant un modèle opérationnel beaucoup plus efficace, le Software-Defined Storage (SDS) est la solution [108]. La virtualisation du stockage se compose en deux catégories principales : la virtualisation de blocs et la virtualisation de fichiers. La virtualisation de blocs correspond plus précisément aux technologies de réseau de stockage (SAN) et de stockage en réseau (NAS). Au niveau du SAN, la virtualisation de l'espace de stockage se rencontre d'abord dans les unités de stockage, avec l'introduction il y a plusieurs années, de la première forme de virtualisation du stockage : le RAID [109]. Les exemples de produits suivants permettent la virtualisation de stockage : DataCore, EMC Invista, Symantec Veritas Storage Foundation et IBM SAN Volume Controller [110].

## 3.3. Virtualisation du périphérique réseau

Avec l'accroissement constant des infrastructures réseaux actuelles, un nouveau modèle de fonctionnement et de déploiement devient nécessaire. SDN (Software Defined Networking) et la virtualisation des réseaux sont des technologies conçues pour répondre à ces nouvelles problématiques. Elles permettent de rationaliser l'utilisation des ressources disponibles, tout en offrant une flexibilité que des équipements physiques dédiés ne peuvent fournir. Elles permettent également de résoudre un certain nombre de problèmes liés au stockage, à la capacité de calcul, à la consommation d'énergie et à l'accessibilité aux données en permanence. La virtualisation de réseau prend en compte la virtualisation des périphériques réseau et plusieurs outils sont mis en œuvre.

Concernant la mobilité des machines virtuelles, un des grands enjeux de la configuration en cluster étendu est la mise en place d'un réseau de niveau couche 2 (liaison de données ou L2) étendu. La mise en œuvre d'un réseau de niveau L2 étendu sur deux sites induit une complexité au niveau du réseau physique et des coûts très importants. Grâce à la virtualisation des réseaux, il est possible d'encapsuler un réseau de niveau L2 dans un réseau de niveau couche 3 (réseau, IP) [111]. Cette simplification de la mise en œuvre d'un réseau de niveau L2 permet de déplacer des machines virtuelles simplement d'un data center à un autre sans avoir à modifier les adresses IP à l'intérieur des VMs. Les solutions comme OpenNebula, Proxmox, Archipel, OpenVSwitch le switch virtuel ou GlusterFS et Ceph, sont des hyperviseurs classiques pour les systèmes de gestion de stockage distribué et du réseau. V. Autefage et al. [112] proposent un outil NEmu ayant pour but de générer des réseaux virtuels dynamiques à la demande afin de tester et de valider des prototypes d'applications réseaux avec un contrôle complet de la topologie et des propriétés de liens.

## 3.4. Virtualisation des applications

La virtualisation d'application correspond à ce que l'on a appelé « client léger ». Elle implique de pouvoir exécuter un logiciel sans toutefois l'installer physiquement sur le système auquel l'utilisateur est connecté. Elle permet aussi de dissocier l'application du système d'exploitation hôte et des autres applications présentes afin d'éviter les conflits. En d'autres termes, virtualiser une application, c'est la transformer en une simple donnée stockée quelque part dans l'attente d'être déportée, à la demande d'un utilisateur, vers un système d'exploitation (périphérique ou serveur) où elle est utilisée avec ses pleines capacités. Softgrid de Microsoft est un excellent exemple de déploiement de la virtualisation d'application. Bien que l'on puisse parfaitement utiliser Microsoft Word 2007 sur son ordinateur portable, le stockage, la gestion et la fourniture des informations personnelles et l'état opérationnel sont administrés par Softgrid. Microsoft Terminal Services et les applications par navigateur constituent d'autres types de virtualisation d'applications. Docker [113] est un exemple de logiciel libre qui automatise le déploiement d'applications dans des conteneurs logiciels.

## 3.5. Famille de solution de virtualisation

Les solutions de virtualisation sont nombreuses et présentent des caractéristiques diverses. Nous parlons de l'isolation qui n'est pas une réelle technique de virtualisation et des trois familles de virtualisation du matériel que sont : la para-virtualisation, la virtualisation complète et la virtualisation assistée par le matériel.

### 3.5.1. Virtualisation par isolation

La virtualisation par isolation est scindée en deux types : la virtualisation applicative et la virtualisation au niveau du noyau. La virtualisation applicative est une technique permettant d'emprisonner l'exécution des applications dans des contextes ou zones d'exécution. Dans un système UNIX par exemple, on peut utiliser la commande chroot pour isoler un processus dans une sous-arborescence du système de fichier, de même que BSD Jail. La virtualisation au niveau du noyau fait croire à la présence de plusieurs machines, mais il n'est pas possible d'utiliser des noyaux différents en même temps.

L'isolation est une solution simple techniquement et peu consommatrice en termes de surcoût lié à la virtualisation. Les ressources matérielles telles que les disques ou cartes réseau sont facilement partagées entre la zone principale et les containers qu'elle embarque. Comme outil pour l'isolation au niveau noyau, nous pouvons citer OpenVZ et LXC OpenVZ qui créent plusieurs conteneurs Linux isolés et sécurisés sur un seul serveur physique permettant une meilleure utilisation du serveur.

### 3.5.2. Virtualisation du matériel

La virtualisation du matériel consiste à faire fonctionner un système d'exploitation comme un simple logiciel. Ce procédé permet à plusieurs systèmes d'exploitation d'utiliser une même machine physique par le biais d'un hyperviseur. Un Moniteur de machine virtuelle (VMM) ou hyperviseur [114] peut s'exécuter directement sur le matériel ou de manière logicielle au-dessus du système d'exploitation d'une machine physique. Cette caractéristique ajoute un niveau d'abstraction permettant de fournir des environnements variés sans modifier la configuration d'une machine, facilitant ainsi l'administration d'un cluster. Elle permet une grande modularité dans la répartition des charges et la reconfiguration des outils en cas d'évolution ou de défaillance momentanée.

a) Para-virtualisation

La para-virtualisation permet à une machine virtuelle d'utiliser des ressources physiques. L'architecture virtuelle exposée est légèrement différente de l'architecture physique sous-jacente. Cette différence empêche la compatibilité entre les pilotes des systèmes d'exploitation et la virtualisation de l'architecture physique. Il est donc nécessaire de modifier le code du système d'exploitation pour rétablir cette compatibilité, ce désavantage est le principal handicap de la para-virtualisation. Pour autant, la para-virtualisation permet une plus grande liberté dans l'implémentation des interactions entre le système d'exploitation et l'architecture virtuelle. La para-virtualisation est utilisée depuis longtemps dans des hyperviseurs tels que VM/370 [115] et Disco [116]. Pour cela, ils utilisaient une combinaison d'instructions, de registres et de périphériques vers des architectures virtuelles afin d'en améliorer les performances. Nous pouvons citer : Citrix XenServer, VMware vSphere (ESXi), Microsoft Hyper-V [117].

### b) Virtualisation complète

La virtualisation est dite complète lorsque le système d'exploitation invité n'a pas conscience d'être virtualisé. L'OS qui est virtualisé n'a aucun moyen de savoir qu'il partage le matériel avec d'autres OS. Ainsi, l'ensemble des systèmes d'exploitation virtualisés s'exécutant sur un unique périphérique, peuvent fonctionner de manière totalement indépendante les uns des autres et être vus comme des périphériques à part entière sur un réseau. Les technologies comme VirtualBox, KVM (Kernel-based Virtual Machine), VMware Server permettent de virtualiser un système comme Android dans un système comme Linux.

### c) Virtualisation assistée par support matériel

Les extensions matérielles pour machine virtuelle sont fournies sur les processeurs Intel (Intel Virtualization Technology (Intel VT) [118]) et AMD (AMD-V [119]). Ces extensions permettent de supporter deux types de logiciels : l'hyperviseur qui se comporte comme un hôte réel et qui a le contrôle complet du processeur et des autres parties matérielles et le système invité, qui est exécuté dans une machine virtuelle (VM). Chacune des machines virtuelles s'exécute indépendamment des autres et utilise la même interface pour le processeur, la mémoire et autres périphériques fournis par la plate-forme physique. Ces extensions améliorent les performances des applications lancées au sein d'une machine virtuelle.

# IV. Synthèse

Nous avons présenté dans ce chapitre le contexte de recherche dans lequel se situe cette thèse. De ce qui découle des sections précédentes, nous nous orientons vers la Cloudlet, comme concept d'architecture logicielle permettant la virtualisation de périphériques.

Ce choix de Cloudlet a une incidence directe sur les choix conceptuels pour les réalisations logicielles. En effet, nous allons pouvoir protéger ou simplifier l'activité effectuée sur un périphérique mobile et déporter la majeure partie dans une machine virtuelle s'exécutant dans la Cloudlet. Mais celle-ci reste locale au point d'usage des utilisateurs. C'est sur une architecture basée sur un réseau de Cloudlets que nous souhaitons aboutir pour effectuer le suivi des usagers dans leur déplacement. Cette gestion peut se simplifier par l'emploi d'un Cloud public afin de faire communiquer les Cloudlets, même si des techniques plus élaborées permettent de s'en affranchir.

Dans la suite de ce manuscrit, nous allons :

- Fournir au chapitre 2 une architecture qui réponde aux besoins et limites de la mobilité que nous avons mis en lumière dans ce chapitre ;
- Définir au chapitre 3 cette architecture logicielle afin de pouvoir partager et uniformiser notre définition avec d'autres approches ;

- Ressortir au chapitre 4 une définition des propriétés temporelles par rapport à cette architecture en particulier au regard des opérations de migration de code ;
- Prototyper au chapitre 5 cette architecture de Cloudlet afin de valider nos choix par quelques exemples et mesurer l'impact de la migration et le coût d'usage d'une Cloudlet afin d'assurer un gain aux usagers et de promouvoir son application.

# CHAPITRE 2 : Architecture pour la supervision d'une Cloudlet

Ce chapitre vise à décrire un modèle architectural basé sur un système de monitoring pour une Cloudlet. Notre travail porte sur la collecte de données liées aux ressources allouées aux périphériques mobiles complexes en cours d'exécution. Il crée une base pour la déportation et la migration des applications mobiles à partir des périphériques mobiles vers la Cloudlet. Pour calculer le coût de la migration qui est la base de la décision de migration, nous avons besoin des mesures et des données recueillies à partir des périphériques concernés et des infrastructures de la Cloudlet.

Dans ce chapitre, nous présentons notre architecture de virtualisation ainsi que les contraintes techniques pour la mise en œuvre d'une étude de cas. Pour cela, nous partons du constat que les périphériques mobiles remplissent de multiples usages : calculer, stocker, communiquer, mais aussi gérer des capteurs. Afin d'observer l'ensemble de ces facettes, il faut définir une architecture logicielle pour effectuer cette surveillance. Les perturbations de l'existant étant contraires à de bonnes observations, nous nous sommes orientés vers la virtualisation de périphériques mobiles dans une Cloudlet, où l'ensemble des capteurs constitue des sources d'information au modèle virtualisé.

Dans la première partie de ce chapitre, nous détaillons l'architecture qui résulte d'une Cloudlet dans un milieu de Cloud Computing. Dans la suite, nous abordons les défis lors de la surveillance de ces périphériques virtualisés. Ensuite, une approche du monitoring de la Cloudlet est proposée, ainsi que des contraintes techniques en vue d'une implantation de cette architecture. Cela consiste en un socle logiciel adapté à la containerisation de machine et d'application. Enfin, nous présentons une étude de cas de notre approche de monitoring de périphérique mobile.

## I. Architecture d'une Cloudlet

Une Cloudlet (Cloud local) [91] est un nouveau concept architectural qui résulte de la convergence des dispositifs mobiles et du Cloud Computing. Ce concept a été introduit à la section 2.2.3 du chapitre 1. Les premiers travaux sur les Cloudlets viennent du NIST[11] (National Institute of Standards and Technology) et ces travaux sont effectués à la division ANTD (Advanced Network Technology Division) dirigée par le Professeur A. Battou au sein du laboratoire ITL (Information Technology Laboratory). Un projet nommé MCC [120] a été initié en 2014 au sein de cette division. Ses résultats ont apporté un autre regard sur la migration d'application dans le monde des périphériques mobiles.

[11] http://www.nist.gov/itl

Dans notre travail, nous utilisons ces résultats pour la virtualisation de périphériques mobiles dans un Cloud local ou Cloudlet afin d'étudier le comportement de périphériques ainsi virtualisés.

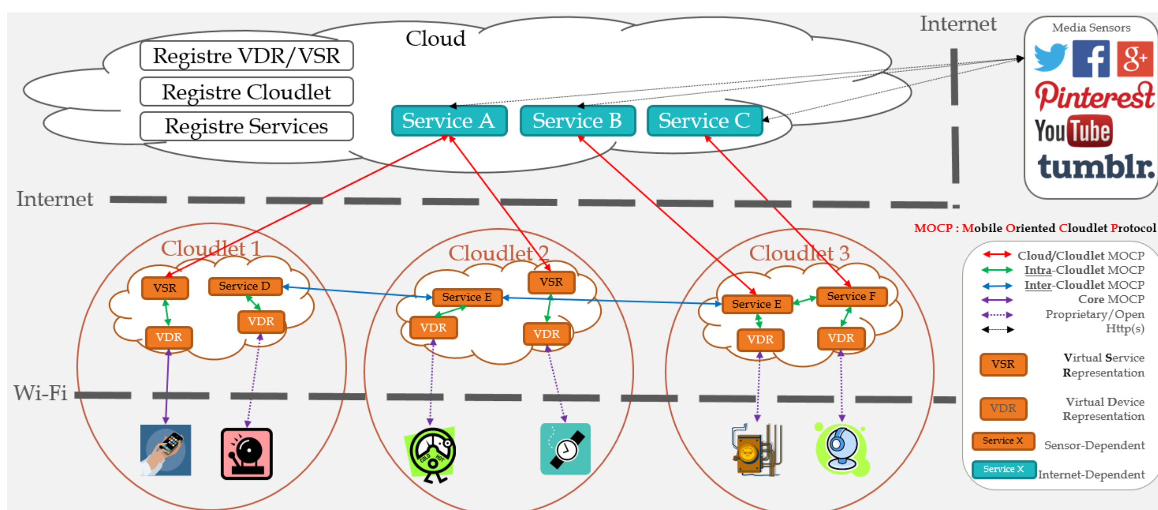
## 1.1. Le monitoring dans un contexte de Cloudlets

Dans un contexte de Cloudlet, le monitoring est un aspect clé dans les mécanismes de prise de décision. Les applications à l'intérieur des périphériques nomades utilisent des ressources pour leurs besoins. En raison de l'aspect distribué de ces applications, différents protocoles sont choisis pour échanger des données. Les systèmes de monitoring classiques ne sont pas adaptés pour assurer la collecte de données utilisées dans un tel contexte. Les méthodes de monitoring classiques sont basées sur la surveillance d'un périphérique central, qui est situé à l'intérieur du système d'information. Les périphériques surveillés envoient les données utiles à ce périphérique central pour analyser des systèmes et déclencher ainsi des opérations de maintenance de support informatique.

### 1.1.1. Architecture globale d'une Cloudlet

La notion de Cloudlet introduit la virtualisation des périphériques comme le montre la Figure 2.1, de sorte que les dispositifs surveillés sont également exécutés dans la Cloudlet. Cette figure illustre que des périphériques mobiles tels qu'un smartphone, un appareil de surveillance (en bas de figure) peuvent être virtualisés dans un Cloud local ou Cloudlet (au centre de la figure). L'usager d'un périphérique mobile peut se déplacer, il est essentiel de faire le suivi de la virtualisation de ce périphérique lorsqu'il sort de la zone géographique de la Cloudlet de départ pour rejoindre une autre. L'usage de service du Cloud (en haut de la figure) assure ce suivi.

*Figure 2.1 Architecture globale de Cloudlet*




La Figure 2.1 montre également que les services de datation issus du Cloud peuvent fournir à chaque Cloudlet un système d'estampille temporelle qui peut être partagé à défaut d'être global. De plus, cette

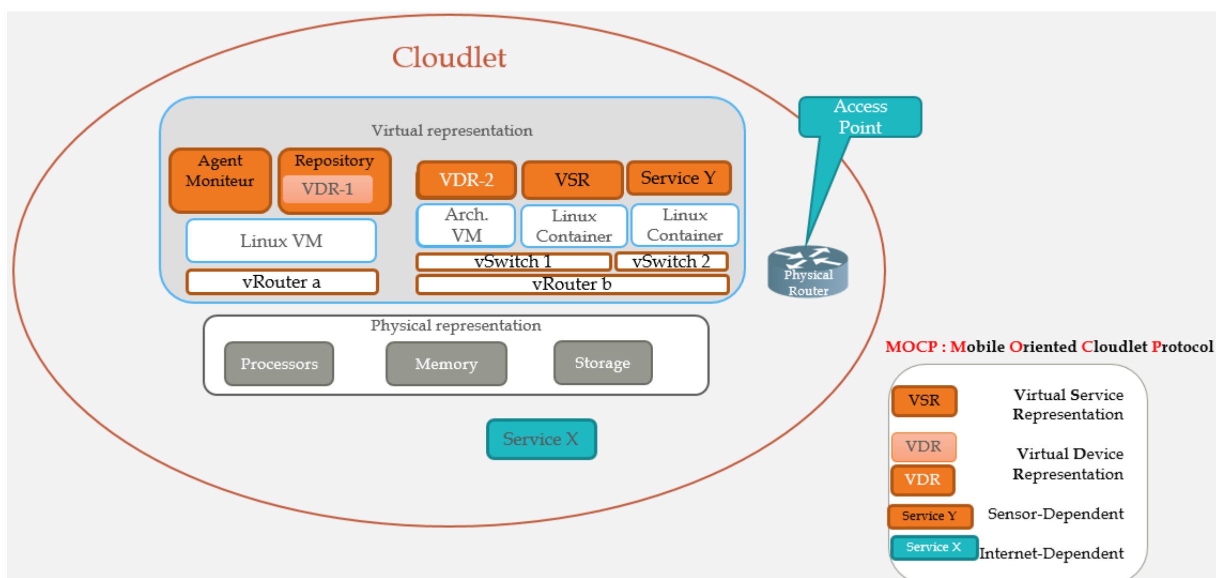
architecture à trois niveaux reste opérationnelle quand bien même l'accès à un Cloud ne serait pas possible pour des problèmes réseau ou autre. Alors, le problème de datation des événements reviendrait à l'usage de vecteur d'horloge comme présenté dans les travaux de L. Lamport [121].

Ce type de comportement oblige les supports de Technologie de l'Information (IT) et d'autres utilisateurs du monitoring à changer leur source de données pour s'adapter au contexte d'une Cloudlet. En outre, la virtualisation implique la migration des applications et des services au sein d'un réseau de Cloudlets et de périphériques nomades. Le système de monitoring de Cloudlet doit prendre en compte des défis tel que le temps de synchronisation des données en mémoire dans un système embarqué.

### 1.1.2. Architecture locale d'une Cloudlet

Une Cloudlet est une représentation de proximité des ressources du Cloud et des périphériques nomades. Les ressources virtuelles de la Cloudlet sont des représentations virtuelles de ressources du Cloud et des périphériques nomades. Par contre, les ressources physiques sont un ensemble de représentations (services, équipements) locales ou dédiées à la Cloudlet. Les échanges entre les différentes ressources s'effectuent grâce aux protocoles MOCP (Mobile Oriented Cloud Protocol) issus du projet MCC (chapitre 5, section 1.2). À l'intérieur d'une Cloudlet, il s'agit des échanges inter-Cloudlet MOCP.

*Figure 2.2 Architecture locale d'une Cloudlet*




a) Représentation virtuelle

La virtualisation des différents services et équipements du Cloud s'effectue au niveau de la Cloudlet, de même que les périphériques nomades des usagers. La Figure 2.2 désigne les différents composants dont nous avons besoin, leur conception et leur construction sont abordées dans les chapitres suivants.

- VSR (Virtual Service Representation), ce composant est lié à la virtualisation d'un service du Cloud ;

- VDR (Virtual Device Representation), ce composant est une représentation virtuelle d'un périphérique comme un smartphone, un capteur. Selon le type du périphérique, on peut retrouver le descripteur du périphérique, le transducteur (Manager d'état, controleur), le périphérique virtuel (App Manager, Agent moniteur, Backend, conteneur OSGi) (voir la section 3.3) ;
- vRouter et vSwitch sont respectivement liés aux représentations virtuelles du routeur et du switch. Leurs buts sont d'assurer les propriétés réseau des périphériques virtualisés.

b) Représentation physique et logicielle

Dans une Cloudlet, nous retrouvons les points d'accès réseau comme des routeurs, les services dédiés ou locaux et des systèmes logiciels basés sur le noyau linux. Les équipements serveurs sont représentés par des processeurs, des mémoires et des unités de stockage (voir la Figure 2.2).

## 1.2. Défis de la surveillance logicielle mobile

Si nous considérons l'analyse basée sur des logs applicatifs pour une application donnée, deux défis apparaissent pour assurer le bon fonctionnement de cette analyse : la définition d'un format de données et le stockage de données. Dans un environnement centralisé, les solutions existent depuis longtemps, mais dans un système embarqué, il devient utile de mettre en place un mécanisme basé sur des agents de collecte. Ceux-ci sont généralement mobiles dans un contexte adaptatif, afin d'améliorer la réponse en cas d'incident.

Un autre cas d'utilisation intéressant, est l'observation des échecs ou des erreurs. En pratique, les principales activités sont la sauvegarde des données dans un domaine de stockage indépendant et l'enrichissement de flux de données en utilisant des méta-informations telles que la date et l'heure des opérations de collecte ou encore le type d'agent de collecte concerné par ces données. Les outils d'analyse utilisent ce principe afin de calculer de façon sommaire des statistiques sur les taux d'échec. Après virtualisation d'une application mobile dans une Cloudlet, les techniques de surveillance doivent nécessairement évoluer. En particulier, le suivi d'une application d'une Cloudlet à une autre nécessite que le système de datation d'événements puisse supporter ce suivi. Du fait qu'une Cloudlet soit intrinsèquement distribuée, nous nous interrogeons sur la gestion du temps, afin de savoir si le temps pris en compte est celui d'une Cloudlet ou d'un périphérique mobile. En outre, la représentation des périphériques mobiles dans une Cloudlet se fait par l'utilisation de VDR (Virtual Device Representation). Une VDR joue le rôle de la virtualisation d'un périphérique mobile. Aussi, nous nous interrogeons également sur le temps d'horloge d'une VDR. Les outils analytiques permettent de savoir si des VDR sont synchronisées à une horloge centrale ou si chaque VDR est synchronisée avec le périphérique physique, ou s'il n'y a pas de synchronisation, et même si des VDRs ont à la base leur propre horloge.

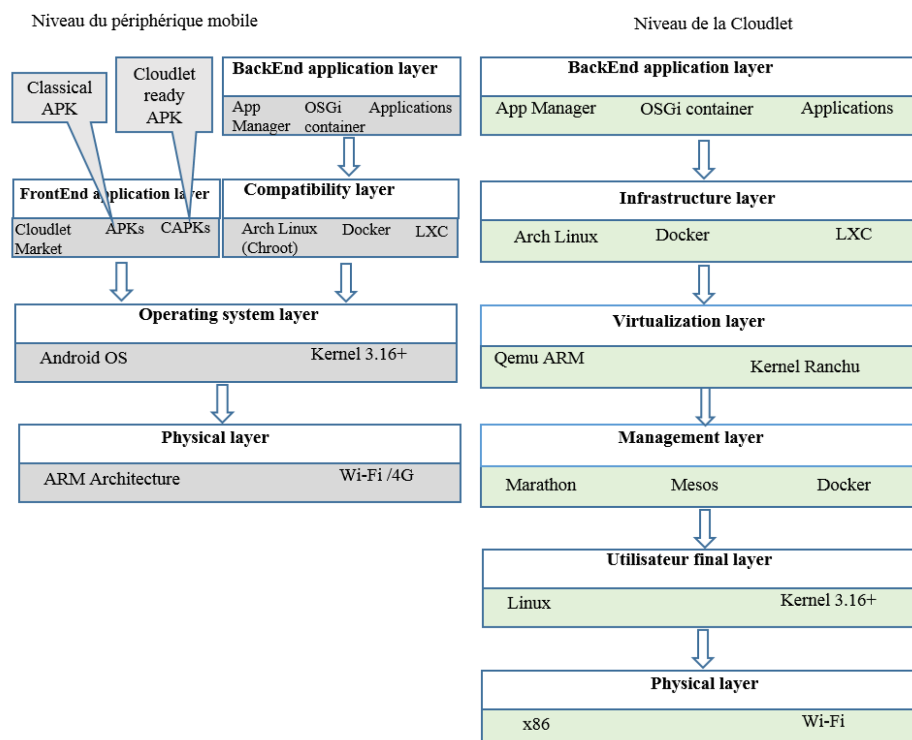
Il est important de bien définir le format de données des messages de monitoring, même si le système de traitement doit être conçu pour pouvoir s'adapter facilement à différents cas. L'utilisation d'un gabarit commun permet de simplifier l'exploitation des messages dans le serveur de monitoring et des tableaux de bord. Le format de données dans un contexte de surveillance est également fonction de l'outil utilisé. La plupart des applications développées en interne et des solutions commerciales écrivent des fichiers de log locaux, souvent à l'aide de frameworks de journalisation tels que log4j ou log4net, des services de journalisation intégrés dans les serveurs d'applications tels que WebLogic, WebSphere et JBoss, ou encore .Net, PHP, etc. De façon générale, les formats de données peuvent être de type XML, JSON et bien d'autres, mais suffisamment riche pour autoriser des représentations.

# II. Les défis de la surveillance logicielle

On peut considérer deux types de solutions de monitoring qui peuvent s'appliquer à l'approche du monitoring de Cloudlet :

- Serveur de surveillance utilisé dans les Cloudlets
- Appareils de monitoring utilisés dans les périphériques mobiles

*Figure 2.3 Comparaison des piles applicatives*




Comme illustrés sur la Figure 2.3, les Cloudlets et les appareils ont des piles applicatives différentes, ainsi qu'une architecture matérielle et des contraintes énergétiques différentes. Ceci implique la

nécessité de repenser l’approche de monitoring afin de recueillir toutes les informations pertinentes de suivi.

Dans cette section, nous mettons l’accent sur les outils existants pour la surveillance d’applications mobiles et abordons ces solutions pour la résolution efficace des problèmes.

## 2.1. Monitoring d’une Cloudlet

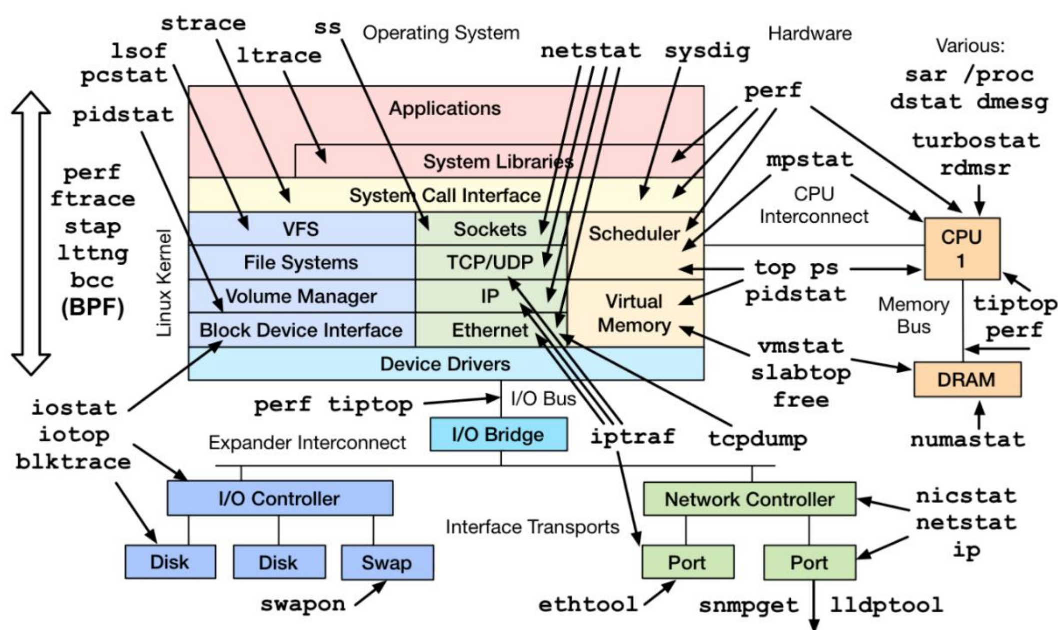
Nous considérons une Cloudlet comme possédant un point de connexion susceptible de recevoir des données depuis des périphériques mobiles. Ainsi, une Cloudlet doit être apte à gérer ces données par l’emploi d’un système d’exploitation.

Les Cloudlets restent un environnement de calcul similaire à celui d’un serveur classique du point de vue du matériel et du système d'exploitation. Donc, nous pouvons utiliser un outil classique de monitoring en intégrant certains de nos composants de surveillance dans la pile applicative d’une Cloudlet.

### 2.1.1. Outils de monitoring

Dans un environnement Linux, il y a beaucoup d’outils d’analyse tels que perf, ftrace, netstat, lldptool et bien d’autres, en particulier pour la performance [122] comme le montre la Figure 2.4. Ces outils, représentés par les flèches bidirectionnelles et unidirectionnelles, peuvent être utilisés par un administrateur système. Ils sont axés sur des aspects spécifiques de la surveillance logicielle : l’analyse de l’utilisation mémoire, les ressources de calcul, l’état des threads, l’état des sémaphores, etc. C’est le cas de l’outil dstat [123] qui permet un monitoring complet pour Linux. Bien que très utiles, tous ces outils restent très proches de la couche système, et il nous serait plus utile de disposer d’un outil fédérateur de niveau d’abstraction plus élevée. En particulier, l’usage d’une console de commandes déportée s’avère indispensable, afin de piloter la surveillance d’un périphérique virtualisé.

*Figure 2.4 Outils de monitoring Linux*[12]

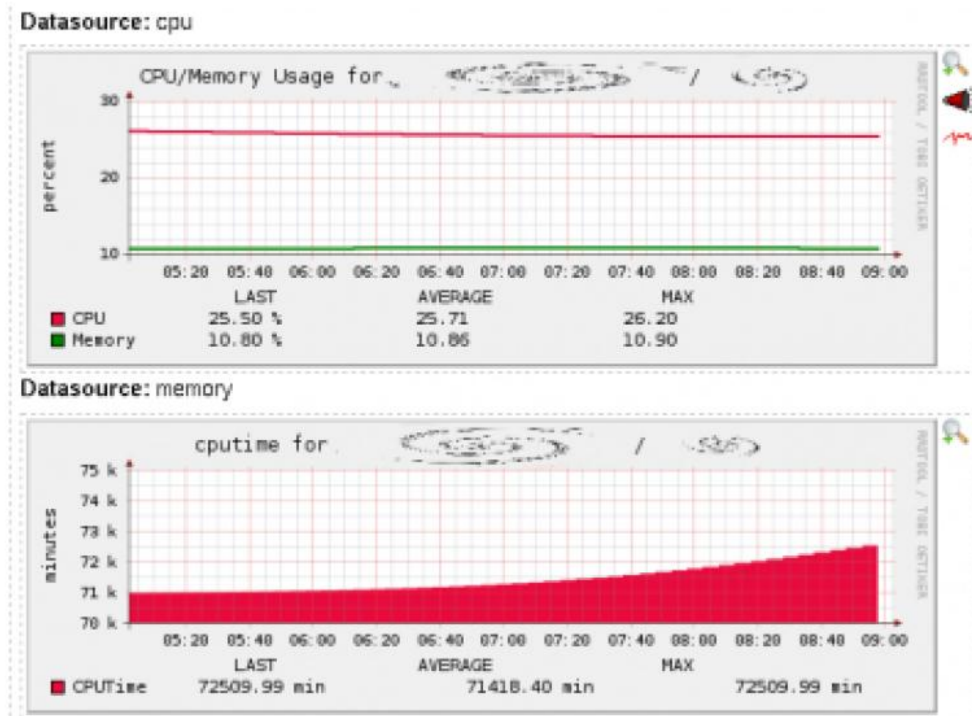



Le monitoring d'une Cloudlet est une activité de surveillance et de mesure de l'activité informatique qui s'y déroule. Ainsi, le monitoring d'un ensemble de Cloudlets permet la surveillance du système d'information interne d'une Cloudlet, du réseau informatique au sens large, allant de la disponibilité d'un équipement jusqu'à la disponibilité d'un service. De nos jours, des études ont été menées sur des outils de monitoring [124] [125]. Nagios [126], l'outil le plus approprié, permet de faire un monitoring distant vers d'autres plates-formes qu'Unix. En effet, il est possible de surveiller les machines Windows, grâce à son plugin Check_nt. Grâce à ses fondements UNIX, la plate-forme MacOS X et d'autres comme Android sont aussi supportées. Parmi les sources d'informations mises à disposition, sont accessibles, les métriques du système d'exploitation, l'état du service, l'état du processus, l'utilisation des fichiers du système, etc. Nagios fournit une surveillance de haut niveau, qui peut être intégrée à une Cloudlet comme agent de monitoring pour recueillir des données sur l'utilisation des processus. Les données recueillies sont ensuite regroupées avec les données collectées à partir des périphériques.

En plus du noyau de Nagios, des plug-ins sont également disponibles. À titre d'exemple, le plug-in "Check_ps.sh" permet de vérifier des processus spécifiques, tel qu'illustré par la Figure 2.5. Toutes les données recueillies par le noyau et les plug-ins sont stockés dans une base de données à l'intérieur de la Cloudlet et sont à la disposition des services réseaux (SMTP, POP3, http, SSH, etc.).

[12] http://www.brendangregg.com/linuxperf.html

*Figure 2.5 Nagios check_ps.sh plug-in*




D’autres fonctionnalités intéressantes de Nagios sont également prévues pour le monitoring des infrastructures. Elles donnent une visibilité globale sur la disponibilité de ressources et des services sur les nœuds de Cloudlet, et les aident à prendre de bonnes décisions en terme de déploiement du processus cible.

Cependant, Nagios présente un inconvénient majeur : la restriction d'accès à de nombreuses fonctionnalités nécessaires comme le tableau de bord. Les plug-ins sont l’une des alternatives, mais ils ne sont pas constamment mis à jour. Argus [127] apporte des solutions aux versions précédentes de Nagios. C’est une solution gratuite, open source et utilisable en réseau. Considéré comme une solution académique, Argus est peu utilisé dans les entreprises. En plus des fonctions couvertes, son fonctionnement modulaire permet d’ajouter des fonctions qui ne seraient pas disponibles dans un premier temps. Une autre alternative à Nagios est Zenoss [128]. C’est une solution complète pour le monitoring du réseau et du système. Zenoss offre une visibilité du réseau depuis l’infrastructure physique jusqu’au niveau application. Elle fournit une interface Web qui permet de contrôler la disponibilité, l’inventaire et la configuration, les performances et les événements. Les limites de Zenoss sont plus ou moins identiques à celles de Nagios : les limites d'évolutivité et d'intégration dans un contexte distribué.

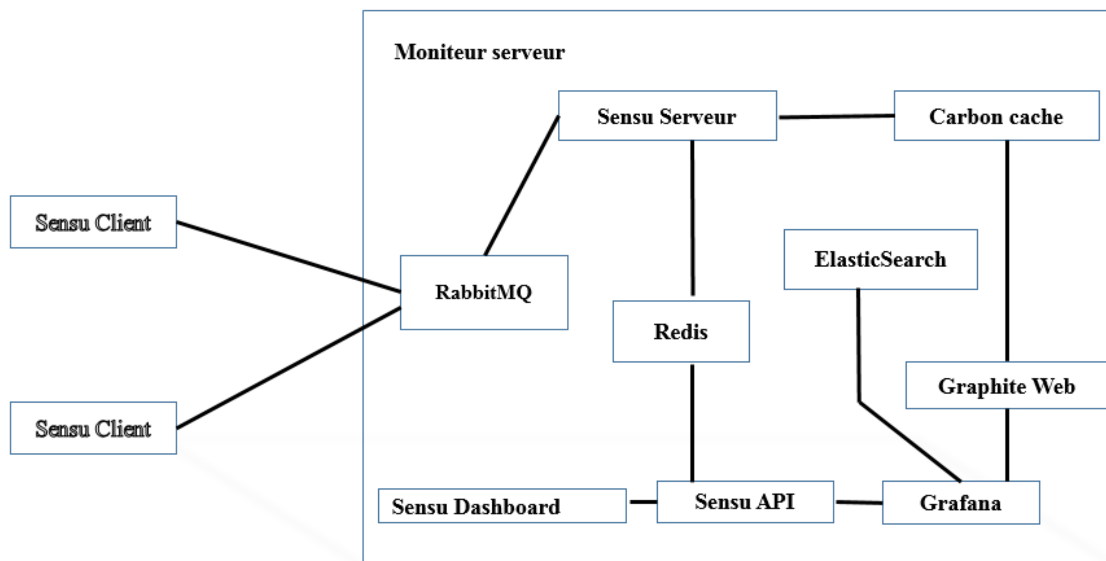
### 2.1.2. Notre solution pour le monitoring

Pour être capable de faire évoluer le monitoring d’un système distribué, la tendance réelle est de construire une solution dite « adaptative ». Cela ne veut pas dire qu’il est nécessaire de réinventer la

roue, mais nous devons intégrer et composer une solution adaptée à la base sur les nécessités des produits émergents. Pour ce genre de solution, nous avons identifié un framework appelé Sensu [129]. Le but est de disposer un framework ouvert pour construire le socle d'une solution complète de monitoring. Ce framework est évolutif et est enrichi par une grande communauté internationale de développeurs. Le noyau de Sensu dans un routeur de surveillance peut être réparti à travers de multiples périphériques connectés. Chaque instance de ce noyau peut détenir des éléments de configuration sur les hôtes et les services du cluster. Ce noyau n'admet aucun outil graphique. Pour la représentation et l'analyse de données collectées par le framework Sensu, il est primordial d'utilisation des outils comme Graphite et Grafana et d'autres frameworks.

Comme le montre la Figure 2.6, notre architecture basée sur le socle Sensu fonctionne grâce à un routeur de messages RabbitMQ. L'usage de messages permet de découpler les appelants et appelés et d'assurer la fiabilité des échanges par l'emploi de messages persistants. Elle montre également l'intégration des outils comme Graphite et Grafana et des frameworks comme ElasticSearch et Carbon Cache qui aident à analyser une partie du système de monitoring. De même, Redis joue le role d'une base de données NoSQL pour la sauvegarde d'informations lors de la collecte et ElasticSearch est un moteur de recherche utilisé par Grafana pour filtrer les données stockées par Carbon Cache en tant que fichiers JSON.

*Figure 2.6 Architecture logicielle autour du serveur Sensu*




## 2.2. Le monitoring des périphériques

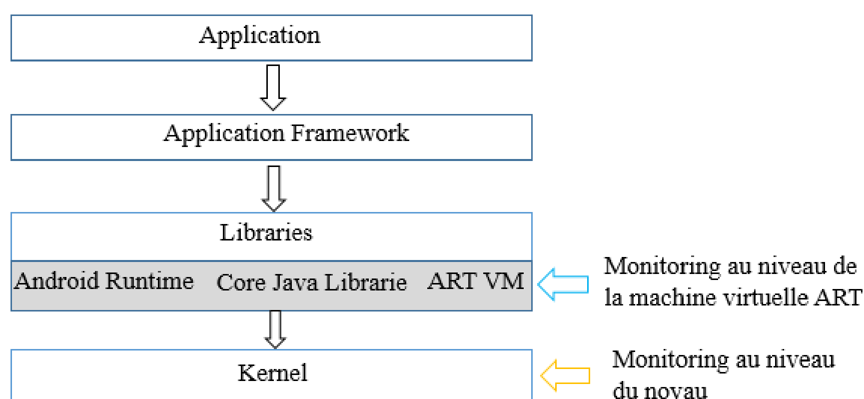
La configuration matérielle des périphériques diffère de celle d'une Cloudlet. Cela signifie que les outils utilisés dans le contexte d'une Cloudlet, ne peuvent pas être utilisés directement sur des périphériques nomades. Les approches standards délèguent au serveur de monitoring, qui agit comme un collecteur de données au niveau central [130]. Dans le contexte de Cloudlet, ces approches ne peuvent pas être

adaptées, du fait que chaque périphérique reste autonome. Cela signifie que chaque périphérique possède ses propres services de collectes et d'analyse de données.

### 2.2.1. Configuration matérielle et logicielle

Actuellement, il y a de nombreux systèmes d'exploitation pour les périphériques nomades, Chrome OS, iOS, Cloud OS, Ubuntu ou Tizen en sont des exemples. Pour nos travaux, nous considérons Android comme le système d'exploitation référent supporté par les périphériques nomades de type tablette ou téléphone mobile dans un réseau de Cloudlets. Ce choix s'explique par une grande disponibilité des sources, des mises à jour et de la documentation. Notre choix est aussi étayé par la mise à disposition de machine virtuelle Android récente [131]. Son architecture est composée de quatre couches empilées comme le montre la Figure 2.7 où la surveillance est assurée aux niveaux du noyau et des librairies. Nous nous sommes intéressés à recueillir des données du monitoring à deux niveaux : la machine virtuelle ART (Android RunTime ou anciennement Dalvik) et le noyau Android. À ces niveaux, Android ne fournit pas d'APIs pour le monitoring. Cependant, Android fournit plusieurs outils livrés avec le Kit de développement logiciel (SDK) pour aider les développeurs avec le débogage, le monitoring et le profilage (étude du déroulement dans le but de découvrir les points faibles à optimiser).

*Figure 2.7 Intégration du monitoring dans l'architecture Android*




Deux outils ont retenu notre attention. Le premier est Dalvik Debug Monitor Server (DDMS) qui est un outil de débogage fournissant une vue sur les codes, les applications et un ensemble d'informations sur le périphérique, le logcat, le processus et un rapport de l'état d'informations. Il est à noter que la perspective DDMS n'a pas changé de nom sur les versions Androïd supérieures ou égales à 5.0. En outre, il assure la simulation des appels, des SMS, un accès à l'emplacement des données écoutées et d'autres observations sur les threads courants, la gestion mémoire, etc. Le second outil est le CPU Monitor (le monitoring au niveau du noyau).

Remarque : la version d'Android utilisée dans le cadre de ce travail est la version 5.0.2.

### 2.2.2. Intégration du monitoring

Dans le cadre du monitoring des systèmes embarqués, il est fréquent qu'il se fasse depuis une plate-forme distante. Mais les appareils sont reliés à un réseau sans fil. De ce fait, deux approches peuvent être conçues. La première concerne le monitoring classique à distance, elle consiste à enregistrer ce qui est exécuté sur le périphérique mobile. La seconde est l'utilisation de la machine virtuelle ART [132] comme outil de monitoring. Elle consiste à enregistrer les données localement sur l'appareil. Ensuite, ces données sont collectées et assemblées en toute sérénité.

Une machine virtuelle Java peut être observée par les applications écrites au-dessus du framework JPDA (Java Platform Debugging Architecture) [133]. Une approche similaire peut être appliquée avec une machine virtuelle ART à part que la plupart des instructions doivent être écrites en langage C. Plusieurs paramètres peuvent être enregistrés à savoir : la durée des méthodes ou des fonctions, la cartographie des données en mémoire, la métrique entre l'allocation de données et son utilisation, le blocage, etc.

À cette approche, nous pouvons adjoindre le monitoring des VDRs. De ce point de vue, le périphérique joue un rôle de périphérique simulé. Dans la réalité, un périphérique mobile doit toujours être disponible pour les autres périphériques. De ce fait, nous pouvons évaluer la disponibilité et l'état d'accès afin de connaître ceux qui sont cohérents avec le périphérique physique et ceux qui ne le sont pas. Ceci permet d'avoir une métrique sur l'adéquation de l'utilisation de Cloudlet.

### 2.2.3. Le monitoring de capteurs

K. Kail [134] décrit un système de monitoring reprogrammable pour les capteurs. Il utilise un composant de surveillance centralisé qui communique avec des capteurs filaires connectés. Ce type d'approche n'est pas adapté à notre cas d'étude. Tout d'abord, dans un contexte de mobilité, nous avons besoin d'une approche de monitoring évolutif que nous pouvons répartir sur les différents emplacements des Cloudlets. Deuxièmement, les capteurs dans notre cas d'étude n'utilisent pas une connexion filaire, d'où la mobilité de ces derniers.

#### a) Interopérabilité des réseaux de capteurs sans fil

Dans l'architecture d'une Cloudlet, la partie regroupant des capteurs est considérée comme des WSNs (Wireless Sensor Networks). Pour être capable d'effectuer le monitoring dans cette partie, ces capteurs doivent être interopérables, c'est la capacité que possède les capteurs ou des réseaux de capteurs à fonctionner avec d'autres ou réseaux de capteurs existants ou futurs, et ce, sans restriction d'accès ou de mise en œuvre. La contribution de J. Higuera [135] introduit une méthodologie pour atteindre l'interopérabilité du WSN. Comme l'illustre la Figure 2.8, on utilise cette contribution en tant que recommandation, pour dissocier la pile de communication du capteur à trois niveaux d'interopérabilité.

*Figure 2.8 Niveaux d'interopérabilité introduite dans WSN*

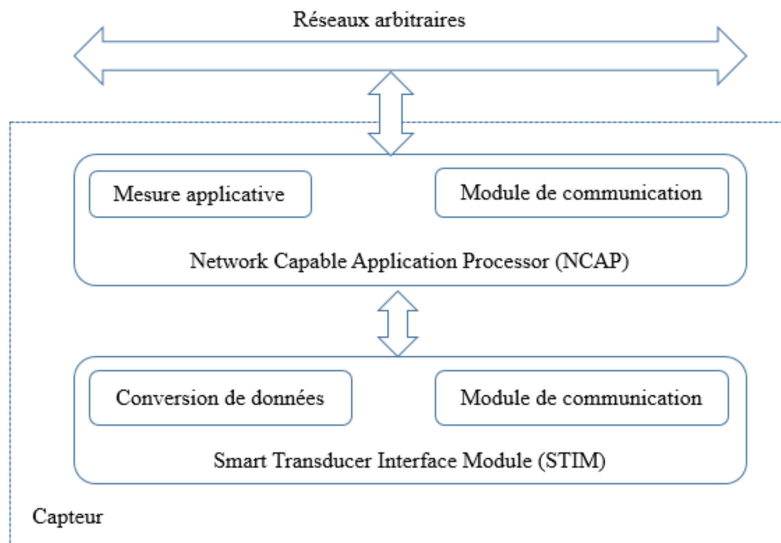
| Niveau | Protocole et standard |
|---|---|
| Intégration | Web Service (Soap, Rest)<br>SMPP<br>JMS |
| Interopérabilité | IEEE1451<br>XML |
| Technique | IEEE Standard<br>802.1 (Bluetooth)<br>802.11 (Wifi)<br>IEEE 802.11af (Wifi)<br>802.15.4 (ZigBee, …)<br>802.15.4a (UWB) |

- Le niveau technique met l'accent sur la transparence de la connectivité du réseau de capteurs. Ce niveau couvre la mise en place de la communication au niveau de la couche physique et de la couche liaison de données ;
- Le niveau interopérabilité implique des formats normalisés d'envois des commandes de contrôle et des métadonnées. Il couvre les couches : réseau et application. Dans ce niveau, les normes IEEE 1451 et XML sont utilisées pour échanger des messages entre le capteur et le coordinateur (Cloudlet) ;
- Au niveau intégration, les réseaux de capteurs exercent une communication efficace. Pour communiquer entre eux, ils utilisent les langages du World Wide Web Consortium (W3C) et la norme Internet Engineering Task Force (IETF). Au sein d'une architecture SOA ou REST, un capteur peut être considéré en partie comme un composant ou une ressource.

#### b) Capacité de l'outil NCAP

L. Kang [136] a défini une architecture intéressante qui utilise la norme IEEE 802.11 et 1451.5. Cette architecture introduit des mesures applicatives, illustrées sur la Figure 2.9, qui collectent des données du monitoring provenant des capteurs et les transmettent au serveur de surveillance. Les travaux de L. Kang montrent que cette approche peut donner de bons résultats dans les réseaux sans fil. Ils fournissent une base pour la mise en œuvre d'applications de capteurs sans fil basées sur les normes IEEE 1451.0 et IEEE 1451.5. L'outil NCAP (Network Capable Application Processor) est un dispositif entre le circuit STIM (Smart Transducer Interface Module) et le réseau. Il effectue les communications réseau, STIM, les fonctions de conversion de données et les fonctions applicatives. Cet outil alimente également les circuits STIM et contient généralement un contrôleur d'interface réseau plus large qui peut prendre en charge d'autres nœuds.

*Figure 2.9 Capacité de l'application NCAP*




L'un des principaux avantages du protocole Internet (IP ou Internet Protocol) dans la gestion de réseaux WSNs est de permettre l'utilisation de services Web standard basés généralement sur les architectures SOA et REST. Les protocoles utilisés au niveau Intégration, permettent aux applications de s'appuyer sur les services couplés, qui peuvent être exposés dans le réseau pour être partagés et réutilisés. Dans une architecture REST ou SOA, une ressource est considérée comme une abstraction identifiée par une URI (Uniform Resource Identifier). Toutes les ressources sont découplées des services ou des capteurs qu'elles représentent.

# III. Architecture pour la surveillance d'une Cloudlet

Plusieurs modèles basés sur les Cloudlets [137] ont été proposés pour résoudre le Cloud Computing Mobile (MCC). Notre travail, qui proroge celui de Y. Jararweh [138], s'intéresse à trois axes :

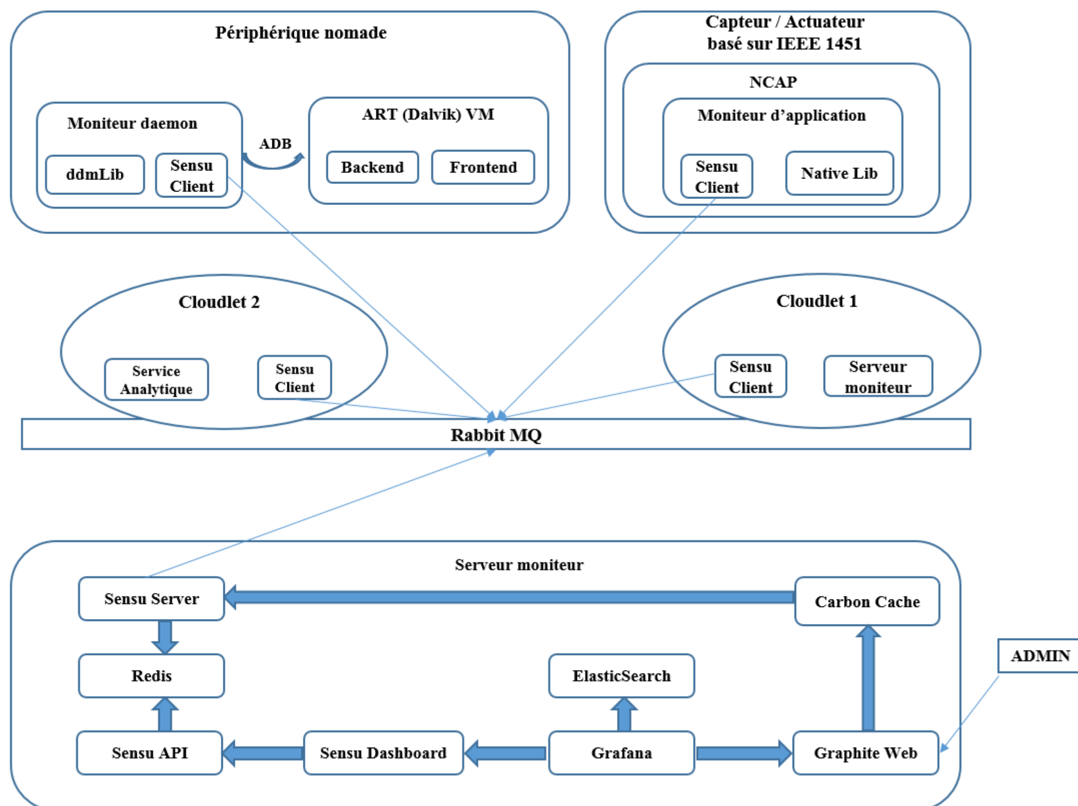
- Le premier vise à définir de nouveaux concepts pour gérer la fiabilité, la disponibilité des services et des terminaux mobiles ;
- Le deuxième vise à introduire un support de développement du code natif supporté par la couche de compatibilité ;
- Le dernier axe vise à fournir des données de contrôle utiles qui sont nécessaires pour la partie applicative afin de calculer le coût de la déportation de l'application.

Nous décrivons dans cette section, les trois axes tout en nous concentrant sur le monitoring. Nous proposons une architecture MCC basée sur les Cloudlets, c'est-à-dire composée d'un réseau de Cloudlets réparties géographiquement pour permettre la virtualisation de périphériques utilisant des services

déployés sur des Cloudlets. De plus, toutes ces Cloudlets sont reliées à un système centralisé de monitoring de Cloudlet.

La Figure 2.10 montre nos choix d'architecture logicielle où les interactions entre composants (outils) sont représentées avec les flèches. Dans ce contexte, les périphériques nomades et les capteurs communiquent directement avec la Cloudlet qui est connectée au routeur du monitoring. Comme le montre la Figure 2.10, le composant « Serveur moniteur » peut être vu comme un service dans l'une des Cloudlets actives. Chaque élément à surveiller utilise un matériel spécifique, mis en œuvre dans la partie Sensu Client qui recueille des données suivies et les envoie au niveau du routeur de monitoring pour l'analyse. Graphite réalise le stockage de séries de données numériques et le déclenchement de calcul de graphe issus de ces données. Carbon cache reçoit des métriques et en conserve la trace sur le disque.

*Figure 2.10 Architecture pour le monitoring distribué*




## 3.1. Architecture à quatre couches

Notre architecture respecte un pattern Layer de quatre couches comme illustrée par la Figure 2.11. Les deux premières couches appelées, « Utilisateur final » et « Infrastructure » sont considérées comme la source de données contrôlées.

Il y a trois différences majeures entre ces deux couches (Utilisateur final et Infrastructure) décrites ci-après.

a) Au niveau de l'architecture matérielle

Les éléments de la couche « Utilisateur final » sont plus susceptibles d'utiliser un processeur RISC basé sur les architectures ARM qui sont adaptées pour équiper des capteurs et des appareils à faible consommation d'énergie tandis que les éléments de la couche « Infrastructure » sont fondés sur des architectures matérielles AMD/Intel classiques, qui ont un ensemble d'instructions différentes. Cela signifie que le monitoring coté client doit être publié spécifiquement pour chaque couche/plate-forme cible.

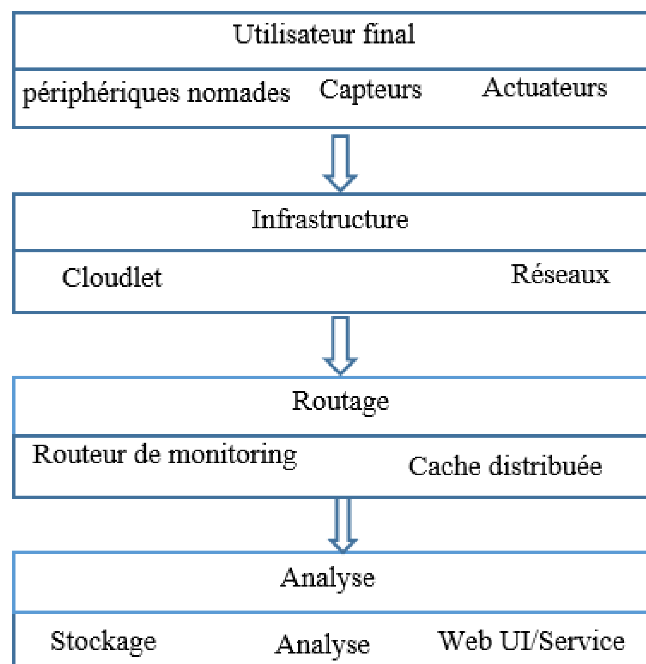
b) Au niveau du système d'exploitation utilisé

Les éléments de la couche « Utilisateur final » utilisent des outils spécifiques à des frameworks pour les capteurs et les servocommandes. Les appareils nomades utilisant Linux doivent avoir comme système d'exploitation de base Android. Le monitoring coté client doit prendre en compte cette contrainte pour pouvoir fonctionner dans ce système hétérogène.

c) Au niveau des services

La couche « Infrastructure » n'est pas seulement un fournisseur de données contrôlées, mais également un consommateur. Cela signifie que les services exécutés dans la couche « Infrastructure » essentiellement l'orchestrateur (le répartiteur ou l'ordonnanceur) [139], exploitent les données de suivi recueillies à partir de ces deux premières couches. Elles utilisent ces données afin de déterminer si les applications ont besoin d'être déportées, si oui, où elles devraient l'être.

*Figure 2.11 Architecture en couches*



## 3.2. Monitoring basé sur le répartiteur de ressources

Dans cette sous-section, nous allons examiner les stratégies d'affectation des ressources pour effectuer efficacement une déportation d'application vers une Cloudlet. Par « efficacement », nous entendons la minimisation de la taille de ressources gaspillées dans le réseau de Cloudlets tout en répondant aux exigences de disponibilité.

De cette façon, nous améliorons l'utilisation globale de la Cloudlet en donnant la priorité à l'exécution de la VDR directement accessible sur la Cloudlet. Nous anticipons les mouvements de l'utilisateur de l'appareil. À la fin, nous basons notre algorithme d'allocation de ressources sur l'Équité de Ressource Dominante (DRF ou ERD) [140] qui respecte les trois propriétés suivantes :

a) Incitation au Partage : chaque élément de la couche "Utilisateur final" devrait être mieux partagé par la Cloudlet la plus proche, en utilisant exclusivement une Cloudlet spécifique. Cela signifie que chaque Cloudlet doit être capable de supporter au moins autant de VDR qu'un maximum de connexion d'appareil /capteurs ;
b) Absence de Préférence : chaque élément de la couche "Utilisateur final" ne devrait pas préférer l'attribution d'un autre utilisateur ;
c) Stratégie-imperméabilité : chaque élément de la couche "Utilisateur final" ne devrait pas être en mesure d'allouer les ressources à la demande. Le système de monitoring est l'élément clé pour assurer cette propriété.

En plus des propriétés ci-dessus, nous considérons la propriété suivante, importante dans le contexte MCC :

d) Accessibilité directe : chaque élément de la couche "Utilisateur final" devrait être en mesure d'instancier sa VDR directement accessible dans la Cloudlet en cas de besoin. Par exemple, en cas de panne permanente dans le réseau.

### 3.2.1. Allocation et ordonnancement de processus

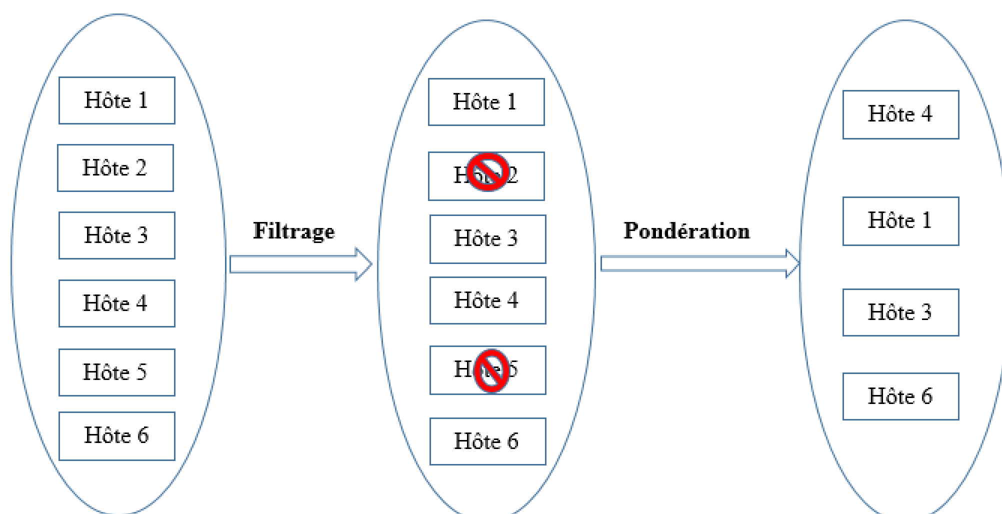
Il existe plusieurs approches qui répondent à l'allocation et l'ordonnancement des ressources dans une infrastructure du Cloud. Leur objectif est d'effectuer le travail de mise en place des machines virtuelles (VM) dans une machine. Elles dépendent du CPU, de la mémoire, des unités E/S et des besoins en bande passante du réseau pour les machines virtuelles. Elles s'assurent que la mise en place, garde toujours la totalité des ressources utilisées sur chaque serveur en dessous de la capacité de ce dernier.

La première approche est basée sur une heuristique comme dans le Cloud AZUR de Microsoft. Cette approche utilise l'outil de gestion appelé System Center Virtual Machine Manager (SCVMM) [141]. Elle est basée sur l'exploration de l'espace avec une stratégie de recherche pour indiquer la "meilleure"

solution à l'ordonnanceur [142] comme l'utilisation de l'algorithme RMS (Rate Monotonic Scheduling). Une autre solution utile est l'approche dite « filtrage », telle qu'elle est utilisée sur OpenStack++ [143]. Ces approches permettent une modification dynamique de l'algorithme d'allocation de ressources en fonction du besoin réel de la VM et également en fonction de l'infrastructure. Le principal avantage de ces approches est la possibilité d'intégration pour un pattern « Pipe & Filter ». Cette approche est utilisée pour composer des stratégies de filtrage complexe basées sur les stratégies simples bien définies. Comme illustrée par la Figure 2.12, cette approche prend en charge le filtrage et la pondération pour prendre des décisions sur l'endroit où, une nouvelle instance doit être créée. Dans cette figure, le filtrage permet de bloquer deux hôtes (hôte 2 et hôte 5) et la pondération permet de sélectionner l'hôte le mieux adapté d'un groupe d'hôtes valides en donnant des pondérations à tous les hôtes de la liste. « Nova » par son ordonnanceur OpenStack offre quatre stratégies de filtrage de base :

- Filtre d'hôtes : ce filtre ne fait aucune opération, car il laisse passer tous les hôtes disponibles ;
- Filtre de propriétés d'images : c'est un filtre d'hôtes basé sur les propriétés définies sur l'image de l'instance. Il laisse passer des hôtes qui vérifient les propriétés d'images spécifiques contenues dans l'instance ;
- Filtre hôte en zone de disponibilité : il laisse passer des hôtes qui respectent les conditions de la zone de disponibilité spécifiées dans les propriétés de l'instance ;
- Filtre de la capacité de calcul : il vérifie que les capacités fournies par le service de calcul satisfont toutes les spécifications supplémentaires associées avec le type d'instance. Ce filtre laisse passer des hôtes qui peuvent créer le type d'instance spécifiée.

*Figure 2.12 Processus de filtrage et de pondération*




### 3.2.2. Notre stratégie de filtrage

Les stratégies de filtrage précédentes ne couvrent pas nos besoins en termes de mobilité des périphériques. En effet, le monitoring des informations collectées donne des informations en temps réel

sur les capteurs, les périphériques et les Cloudlets. Nous introduisons dans notre architecture, le mécanisme de mobilité des supports VDRs entre les Cloudlets. En raison du traitement en temps réel que nous utilisons, l'ordonnanceur fonctionne en continu, mais il maintient le traitement de la VDR dans un intervalle défini. Cette activité provoque la migration des VDRs en cas de besoin de mises à jour recueillies pour le système de monitoring.

Nous avons conçu quatre nouveaux filtres qui prennent en compte l'état de la VDR en temps réel, l'architecture VDR du support, l'accessibilité directe à la Cloudlet par le périphérique et aussi la disponibilité des capteurs de VDRs nécessaires dans chaque hôte de la Cloudlet.

a) Filtre de VDR de propriétés en temps réel : il filtre des hôtes qui correspondent aux ressources estimées en utilisant des données analysées en temps réel à partir de la VDR. Ce filtre considère la moyenne des ressources utilisées dans la VDR et ajoute une marge de sécurité ;
b) Filtre de la capacité architecturale : il filtre des hôtes qui peuvent créer les types d'instances spécifiées sur la base des types d'émulateur disponibles sur les nœuds et de son architecture matérielle ;
c) Filtre de VDR de propriétés de connexions de Capteurs : il filtre des hôtes qui correspondent aux VDRs de capteurs nécessaires à la VDR du périphérique. Ce filtre est utilisé pour la pondération ;
d) Filtre de Cloudlet directement accessible : il filtre des hôtes qui sont dans la même zone physique dans laquelle le périphérique est connecté au moment du traitement.

Ces nouveaux filtres permettent à l'ordonnanceur de composer des stratégies plus efficaces pour gérer la migration d'une VDR d'une Cloudlet à l'autre. La migration est alors considérée comme une tâche de l'ordonnanceur en temps réel. L'ordonnanceur est responsable de trouver le meilleur emplacement pour l'exécution de la VDR.

Dans la sous-section suivante, nous décrivons notre approche de migration pour la création d'une VDR et le déplacement d'une VDR d'une Cloudlet à une autre.

## 3.3. La création du VDR et l'application de déportation

La VDR représente un périphérique mobile ou un capteur dans la Cloudlet. Dans notre architecture, nous associons une VDR à chaque périphérique mobile ou chaque capteur lors de leur première connexion à une Cloudlet. Cette VDR reste associée à ce périphérique tant qu'elle n'est pas réinitialisée à sa configuration d'usine. Les deux types de VDR (capteur et périphérique) sont composés de deux principales parties qui sont le descripteur du périphérique et le périphérique virtuel. Cependant, le contenu du périphérique virtuel est illustré dans la Figure 2.13 et la Figure 2.14.

La VDR du périphérique est composée d'un conteneur OSGi (Open Services Gateway initiative) qui est utilisé pour exécuter les Backends applicatifs. Par contre, la VDR du capteur est composée d'un vSTIM (virtual Smart Transducer Interface Module) qui est un module d'interface virtuel de capteur intelligent, pour maintenir la compatibilité avec le standard IEEE 1451 des capteurs.

*Figure 2.13 Virtualisation de périphérique*

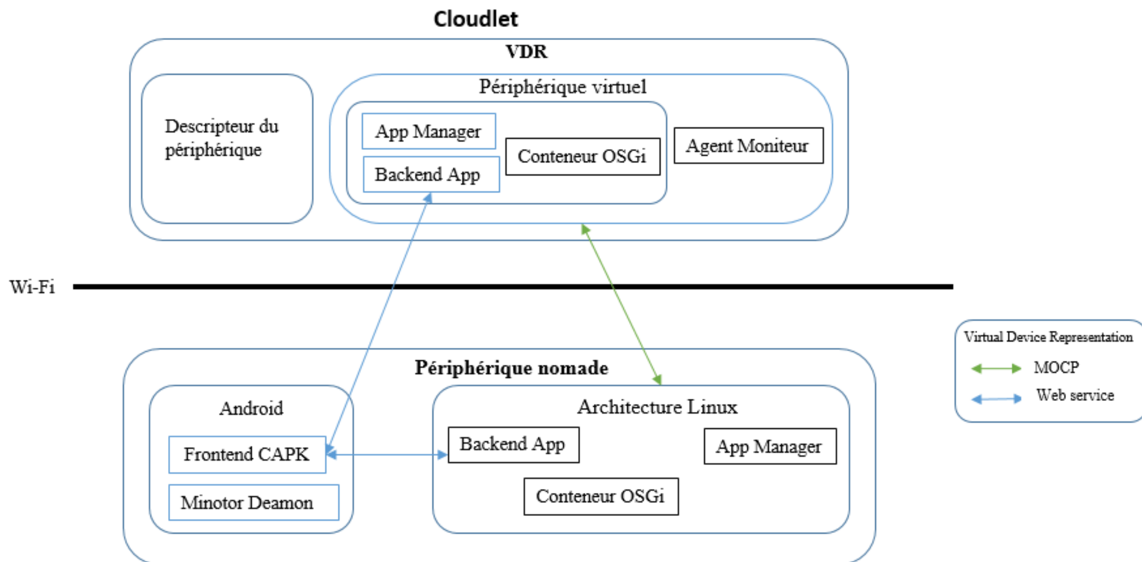
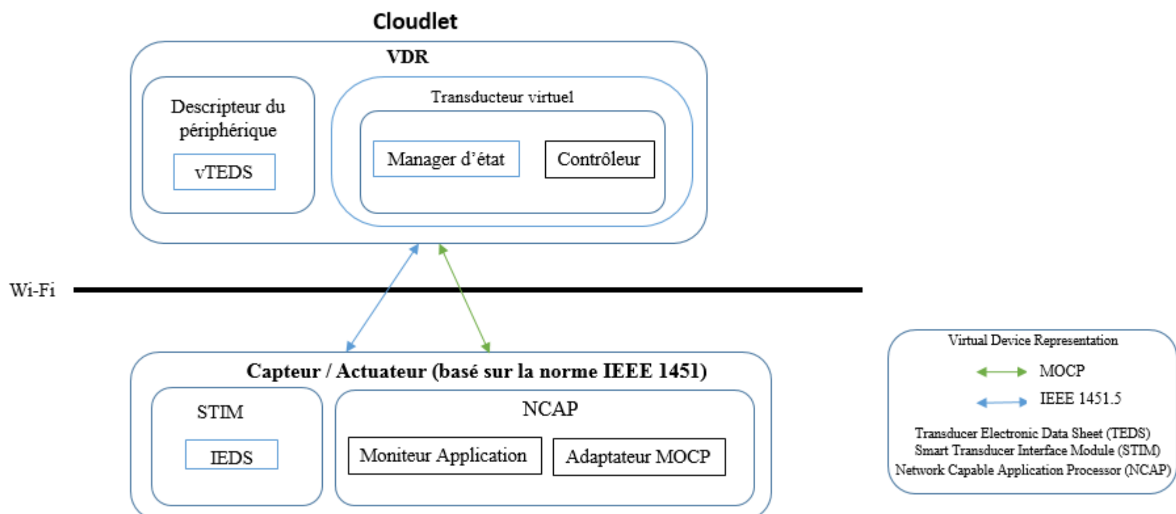



*Figure 2.14 Virtualisation de capteur*




La création de la VDR suit le flux de travail de l'orchestrateur du Cloud IaaS pour la création de la VM. En plus des filtres spécifiques à la Cloudlet, les VDRs des périphériques doivent être exécutées à l'intérieur des émulateurs pour adapter le processeur de l'hôte à l'architecture du processeur du périphérique. De cette façon, une application trouve le même jeu d'instructions entre la VDR et le périphérique. Maintenant, nous devons également initier la création de la VDR. Pour cela, nous utilisons un listener de surveillance pour démarrer l'instanciation de la VDR dès que le périphérique est connecté.

# IV. Contraintes techniques pour la surveillance de la Cloudlet

Cette section vise à décrire les contraintes techniques en vue d'une implantation de notre architecture. Tout d'abord, nous décrivons les frameworks utiles pour la mise en œuvre. Ensuite, nous détaillons les modifications apportées aux frameworks utilisés pour répondre aux contraintes techniques et architecturales pour compléter la surveillance de la Cloudlet. Enfin, nous décrivons les composants du système de surveillance tout en les illustrant par un diagramme de déploiement.

## 4.1. Les Frameworks pour la surveillance

Pour répondre aux besoins architecturaux du monitoring, nous avons identifié les contraintes techniques qui doivent être couvertes.

a) Routeur de monitoring : il couvre les contraintes d'évolutivité dans un réseau de Cloudlet. Le noyau de Sensu est utilisé pour créer un routeur de surveillance évolutif basé sur un cluster de nœuds de RabbitMQ installé dans chaque Cloudlet ;

b) Collection de données : elle couvre les contraintes de vérification des états des différentes ressources. Un nom générique "Sensu Client" est donné aux clients qui prennent en charge les vérifications basées sur Sensu. Derrière ce nom générique, nous pouvons trouver de nombreux frameworks en fonction de la plate-forme cible pour laquelle ce client est dédié. Nous utilisons le DDMS pour collecter des informations à partir des périphériques Android, un code embarqué en C pour collecter les données de notre prototype de capteur IEEE 1451 et les commandes Linux pour recueillir des données provenant des nœuds de la Cloudlet ;

c) Cache distribué : il couvre les contraintes d'auto-configuration lorsqu'une nouvelle Cloudlet est ajoutée au réseau ou lorsqu'un nouveau client est connecté. Pour cela, nous utilisons Redis comme une base de données distribuée et persistante en mémoire cache interne configurable à travers le réseau de Cloudlet ;

d) Métrique de stockage : elle couvre les contraintes relatives aux valeurs des métriques dans le cache. L'objectif de ce composant est de constituer le backend de stockage pour les données collectées. Pour cela, nous utilisons Carbon Cache qui collecte les métriques du disque sur un intervalle de temps et les restitue en cas de besoin ;

e) Analyse des données : elle couvre les contraintes de perception du réseau de Cloudlets. Pour cela, nous utilisons ElasticSearch qui fournit de nombreuses fonctionnalités analytiques ; comme par exemple : l'analyse en temps réel des données, la multi-location, la recherche du

texte intégral et bien d'autres. Ces fonctionnalités permettent d'analyser des données collectées lors des vérifications ;

f) Présentation graphique : elle couvre les contraintes de la partie graphique de la perception du réseau de Cloudlets. Pour cela, nous utilisons les fonctionnalités de visualisation de Graphite, Graphite FrontEnd essentiellement pour les séries de données chronologiques servant à la construction d'un tableau de bord.

Dans la section suivante, nous décrivons l'adaptation et l'intégration de ces frameworks. En outre, nous détaillons les composants que nous avons développés pour compléter notre solution de monitoring.

## 4.2. Corrective et développement spécifique

Une des correctives apportées porte sur les infrastructures de routage utilisées dans notre architecture. En effet, RabbitMQ est utilisé en même temps par l'orchestrateur de la Cloudlet (OpenStack++) et le routeur de monitoring (Sensu). Étant donné que chaque infrastructure utilise une des clés spécifiques pour sécuriser la connexion des composants de RabbitMQ, nous avons modifié les paramètres de sécurité de sorte à avoir un certificat de sécurité unique pour les deux infrastructures. Ce correctif de sécurité a permis la cohabitation de tous les composants qui utilisent le même routeur. De cette façon, la configuration du clustering est faite une fois et couvre tous les composants qui sont utilisés par notre prototype.

Une autre corrective concerne la collecte de données. Le manque de documentation concernant la collecte de données dans les périphériques mobiles, l'hétérogénéité des processeurs des périphériques mobiles et la collecte de données à partir de VDR sont des défis importants que nous avons surmontés. Pour être en mesure de collecter des données à partir des périphériques Android, nous avons créé un vérificateur spécifique pour Android qui utilise le DDMS pour accéder aux données.

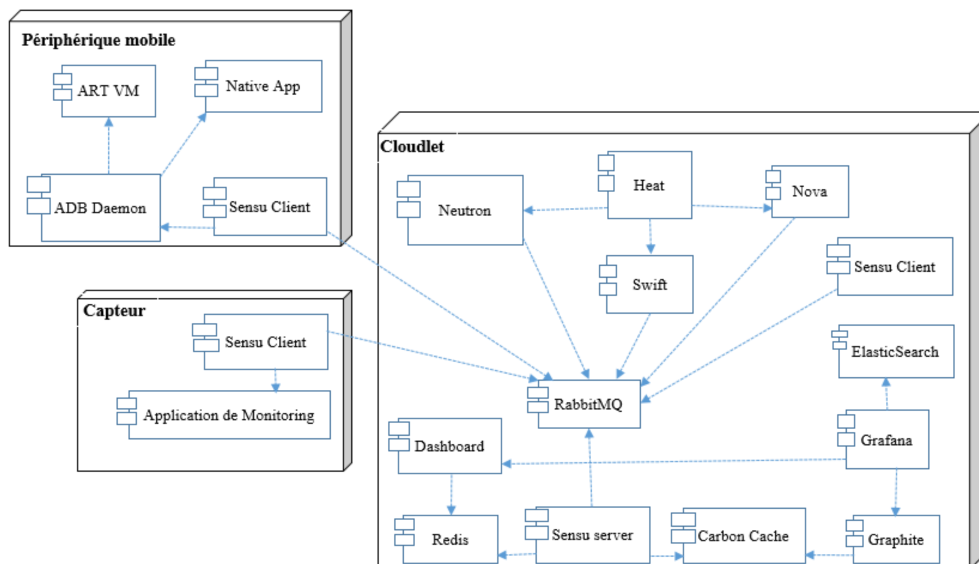
Pour l'intégration de ce vérificateur au système Sensu, nous avons utilisé le client Python Sensu qui s'exécute sur une version de Python que nous avons adaptée et compilée en utilisant les chaînes d'outils du compilateur croisé d'un Android natif. Avec ce client natif, nous avons la possibilité de compiler Python sur toutes les plates-formes basées sur Android ainsi, nous pouvons gérer l'hétérogénéité en utilisant la chaîne d'outils d'Android. En ce qui concerne la collection de données de la VDR, l'outil DDMS est utilisé sur l'interface du réseau virtuel employé pour la connexion de la VDR. ADB (Android Debug Bridge) prend en charge l'établissement d'une connexion sur TCP et permet à l'hyperviseur d'accéder aux données, dans ce cas, le client standard Sensu est utilisé. Pour intégrer la machine virtuelle d'Android dans l'orchestrateur du Cloud, nous avons modifié le code source du projet Android pour ajouter le module du noyau VirtIO qui permet aux machines virtuelles d'Android de communiquer avec l'hyperviseur en utilisant VirtLib.

De plus, un prototype a été développé pour un capteur de détection de pression en utilisant la plate-forme Arduino et une application de surveillance qui rapporte la température recueillie par le capteur et l'envoie au serveur Sensu. Pour les autres composants utilisés pour le cache distribué, les métriques de stockage et de présentation, une configuration par fichier est suffisante pour l'intégration de ces composants au réseau de Cloudlets. La liste exhaustive des composants utilisés et des objets est présentée à la section suivante.

## 4.3. Composants du système de surveillance

Pour illustrer les composants nécessaires au système proposé, la Figure 2.15 montre une vue statique d'un déploiement classique de notre solution. Notons que nous n'abordons pas les composants nécessaires pour les applications de déportation, nous nous focalisons sur les aspects de surveillance.

*Figure 2.15 Diagramme de déploiement*




Trois principaux environnements de déploiement sont illustrés à la Figure 2.15. Le composant « Sensu Client » est implémenté pour correspondre aux spécificités de la plate-forme cible, mais son rôle est exactement le même pour toutes les plates-formes.

Les composants décrits dans le Tableau 2.1 se rapportent à l'orchestrateur OpenStack++. Ces composants et leurs utilités visent à gérer les ressources physiques c'est à dire l'allocation et l'ordonnancement des ressources. Nous notons que RabbitMQ est utilisé en même temps par Sensu et OpenStack++.

*Tableau 2.1 Description de l'orchestration des composants*

| Composant | Description |
|---|---|
| Swift | C'est l'implémentation du stockage d'objets d'OpenStack++. Il permet un stockage objet redondant et évolutif s'appuyant sur des clusters de serveurs standards. |
| Neutron | Anciennement appelé Quantum, Neutron est le contrôleur de réseau du Cloud au sein d'un OpenStack. C'est l'API pour administrer les réseaux et les adresses IP utilisées par les instances de traitements. L'API s'appuie sur les différentes technologies traditionnelles. Elle permet des services de couches hautes comme VPN-as-a-Service, Firewall-as-a-Service, et Loadbalancing-as-a-Service. Neutron utilise OpenvSwitch pour assurer le routage des couches 2 et 3 définies par les Fonctions de Virtualisation du Réseau (NFV). |
| Nova | C'est l'implémentation de l'ordonnanceur qui gère les ressources physiques de traitements (CPU, mémoire, etc.). Nova fournit les APIs pour contrôler la planification à la demande des instances de traitements (par exemple des machines virtuelles) sur des technologies de virtualisation ou des technologies de containers. |
| Heat | C'est un service d'orchestration d'applications Cloudlet multi-composants d'OpenStack++. Il utilise le format de templating AWS CloudFormation, à travers l'API native REST d'OpenStack++ et une API compatible avec CloudFormation Query. |

Le *Tableau 2.2* vise à décrire les composants utilisés pour des périphériques mobiles et des capteurs.

*Tableau 2.2 Description de composants du périphérique / capteur*

| | |
|---|---|
| ADB Daemon | C'est le prestataire de services pour l'ADB qui écoute les connexions entrantes en utilisant l'outil de débogage DDMS. Il vise à être le pont entre l'ART VM et Native App d'une part et Sensu Client d'autre part. |
| ART VM | C'est la machine virtuelle qui exécute les applications non-natives Android. Elle est utilisée pour exécuter le code Java dexifié, écrit en utilisant l'API Android. |
| Native App | C'est l'application que nous avons développé pour surveiller le comportement du système. Elle a été développée en utilisant les outils natifs d'Android et n'utilise pas l'ART VM. L'API n'est pas accessible via l'outil DDMS. |
| Monitoring Application | C'est une application native que nous avons développée pour surveiller les capteurs. Elle est écrite en C et accède au système d'information à travers une API propriétaire. |

Le Tableau 2.3 vise à décrire les composants relatifs à Sensu, y compris les outils de collecte et d'analyse de données.

*Tableau 2.3 Description de composants de Sensu*

| | |
|---|---|
| RabbitMQ | C’est le serveur implémentant le protocole AMQP. Il propose un broker AMQP qui se trouve entre deux composants Nova ou Sensu. Il leur permet de communiquer à travers des messages dans un mode couplé, utilisant le paradigme producteur/consommateur. Ainsi, il est utilisé pour le découplage entre le client et le serveur, pour fournir un asynchronisme complet entre eux. C’est donc l’élément central de l’architecture qui permet la communication entre :<br>➢ Client et le Serveur Sensu<br>➢ Instances du serveur Sensu<br>➢ Instances d’OpenStack |
| Sensu Serveur | Il est responsable de l'orchestration du monitoring lié aux vérificateurs, aux cas de manipulation et des tâches de distribution. Chaque instance du serveur va inspecter chaque résultat vérifié (qui indique une panne de service ou contient des données telles que les paramètres) et communique le résultat à Carbon Cache. Les informations de configuration et de topologie sont stockées et partagées à l'aide de Redis. Elles sont accessibles à travers le tableau de bord (Dashboard) Sensu. |
| Sensu Client | Il est exécuté sur nos trois plates-formes cibles (Cloudlet, Appareil Mobile et Capteur) que nous voulons surveiller. Le rôle du client est de recevoir les demandes d'exécution de vérification, d’exécuter les vérifications, et de publier leurs résultats au serveur Sensu via RabbitMQ. |
| Redis | C’est une mémoire cache distribuée qui vise à assurer la connexion des informations aux différents composants de Sensu. En outre, il est utilisé pour fournir des informations de configuration pour le tableau de bord de Sensu (Sensu Dashboard). |
| Sensu Dashboard | Le tableau de bord de Sensu est construit sur Uchiwa d’où l'utilisation avec Go, NodeJS et AngularJS. Il vise à administrer la partie configurable d’un système de monitoring. Il est utilisé pour gérer les clients et les instances du serveur. |
| Carbon Cache | C’est la partie Backend du tableau de bord Graphite qui reçoit les métriques (prise en charge de divers protocoles) et les écrit sur disque. C’est à dire qu'il stocke les valeurs de métriques collectées par Sensu Server, dans la RAM comme il les a reçus. Il les aligne dans le disque selon un intervalle prédéfini à l’intérieur de la base de données Whisper. |
| Graphite | Il est utilisé pour filtrer et stocker des paramètres arbitraires pour une utilisation ultérieure. Ici, c’est la partie Frontend du tableau de bord Graphite qui vise à exposer sur demande les tracés de données stockées par Carbon Cache. |
| Grafana | C’est une extension de Graphite. C’est un tableau de bord qui offre un éditeur de requête avancée. Il permet d'étendre la navigation sur l'espace métrique. Grafana est utilisé conjointement avec ElasticSearch. |
| ElasticSearch | C’est un moteur de recherche distribué utilisé par Grafana pour filtrer les données stockées par Carbon Cache en tant que fichiers JSON. |

# V. Etude de cas

La Cyber-sécurité est une question clé en particulier pour les organismes publics comme le ministère de la Défense des États-Unis, la communauté du renseignement et les agences civiles fédérales. La National Institute of Standards and Technology (NIST) donne une orientation dans la publication spéciale 800-53 qui contient les recommandations relatives aux contrôles de la sécurité et des améliorations. Cette orientation a été mise à jour par les initiatives 2011-12 qui forment des éléments clés de la sélection du processus de commande. Elle se focalise sur les menaces initiées, des applications de sécurité, les réseaux sociaux, des périphériques mobiles et du Cloud Computing.

Nous estimons que notre approche peut enrichir l'implémentation des recommandations du NIST en intégrant une perspective interne, non prise en compte par les solutions classiques de monitoring du réseau. Dans le cadre de ces études, nous nous focalisons sur la bonne foi des initiés, qui peuvent introduire des codes malveillants dans le réseau à l'aide de leurs périphériques mobiles. Notre objectif est d'utiliser notre approche de monitoring du réseau pour montrer sa valeur ajoutée par rapport aux approches classiques.

## 5.1. Menaces liées à la sécurité des périphériques mobiles

La sécurité vise à protéger les systèmes d'information sensibles par un ensemble de moyens techniques, organisationnels, juridiques et humains nécessaires à la mise en place des moyens visant à empêcher l'utilisation non-autorisée, le mauvais usage, la modification ou le détournement du système. Les attaquants peuvent essayer de perturber les opérations normales tout en exploitant les vulnérabilités et en utilisant différents outils et techniques. Les attaquants ne sont pas nécessairement externes à l'organisation. Les employés ou les visiteurs malveillants peuvent nuire et perturber un système d'information même s'ils n'introduisent pas de façon volontaire un code malveillant. Les attaquants les plus dangereux sont généralement les initiés s'ils sont de mauvaise foi. Parce qu'ils ont connaissance de la plupart des solutions de sécurité qui sont utilisées dans l'organisation. Dans le cas où ils sont de bonne foi, ils peuvent introduire un code malveillant qui agit comme les attaques des initiés de mauvaise foi, qui initient un accès légitime au système. Les initiés peuvent aider à planter des virus, des chevaux de Troie, ou des vers. Ils peuvent parcourir le réseau dans lequel ils ont accès.

Comme le montre le Tableau 2.4, la plate-forme Android est une cible populaire des programmes mobiles malveillants, avec 97 % de logiciels malveillants mobiles dirigés à la plate-forme Android. La majorité des logiciels malveillants derrière ces attaques mobiles, illustrés dans le Tableau 2.5, sont originaires des marchés tiers.

Google via son application PlayStore a un contrôle de sécurité qui garantit l'authenticité des applications téléchargées depuis sa boutique officielle. Les chercheurs de McAfee Labs conseillent fortement d'installer uniquement des logiciels du marché officiel pour réduire le risque de compromission de son système Android.

*Tableau 2.4 Code malveillant par plate-forme*

| Plate-forme | Nombre de menaces | Pourcentage de menaces |
|---|---|---|
| Android | 57 | 97% |
| Symbian | 1 | 2% |
| Windows | 1 | 2% |
| iOS | 1 | 2% |

Les attaques les plus nocives sur le système d'information de l'organisation sont une porte dérobée ouverte sur les appareils vulnérables. Elle permet aux pirates d'exécuter les actions arbitraires en adhérant en même temps sur le réseau de l'organisation. Ils utilisent l'interface Wi-Fi de l'appareil et également un serveur de contrôle via la connexion du réseau mobile. Ce type de logiciels malveillants peut ouvrir un tunnel entre le réseau de l'organisation et un réseau externe qui n'est pas visible pour les systèmes classiques de monitoring. Ces systèmes classiques de monitoring contrôlent seulement le réseau de l'organisation et sont totalement transparents sur les interfaces qui ne sont pas associées à leur réseau.

Comme illustré dans le Tableau 2.5, les chevaux de Troie sont largement présents sur les appareils Android. Aujourd'hui, ils prennent le contrôle de l'appareil d'un employé et lancent d'autres logiciels malveillants. À titre d'illustration, voici trois exemples intéressants de logiciels malveillants qui peuvent prendre le contrôle à distance d'un appareil. Ces exemples ont été analysés par les chercheurs de McAfee Labs [144].

a) FoncyDropper.A : il fait appel à un processus de niveau super-utilisateur pour prendre le contrôle du smartphone et lancer un robot IRC (Internet Relay Chat) qui reçoit des commandes de l'auteur de l'attaque. Il envoie également des SMS surfacturés sur la base du pays de la carte SIM ;

b) Rootsmart.A : il s'appuie sur un processus appelé « augmentation de privilège » qui permet, une fois installé, de télécharger un cheval de Troie de type porte dérobée à partir d'un serveur distant en cachant ces transferts de données dans le flux normal d'utilisation du smartphone infecté ;

c) Stiniter.A : il s'appuie également sur le processus appelé « augmentation de privilège » qui permet, une fois installé, de télécharger d'autres logiciels malveillants et envoie des

informations du smartphone vers des sites sous le contrôle de l'auteur de l'attaque. Il envoie également des SMS à des numéros payants. Le serveur de contrôle du pirate modifie le corps du message et le numéro composé par le smartphone piraté.

*Tableau 2.5 Actions de codes malveillants*

| Détails de catégories de Menaces | 2013 | 2012 |
|---|---|---|
| Vols de données de l'appareil | 17 % | 27 % |
| Espionnage sur l'utilisateur | 28 % | 12 % |
| Envois de SMS surtaxés | 5 % | 11 % |
| Téléchargements | 8 % | 11 % |
| Porte dérobée | 12 % | 13 % |
| Empreinte de position | 3 % | 3 % |
| Modification de paramètres | 8 % | 5 % |
| Spam | 3 % | 2 % |
| Vols de Media | 3 % | 2 % |
| Augmentation de privilèges | 2 % | 3 % |
| Cheval de Troie bancaire | 3 % | 2 % |
| Publiciel / Ennuyeux | 9 % | 8 % |
| Utilitaire des attaques par déni de service distribué | 0 % | 1 % |
| Outils de piratage | 0 % | 1 % |

## 5.2. Expérimentations

Pour expérimenter notre approche, nous avons développé un programme Cheval de Troie non auto-reproductif que nous appelons « Trojunnel ». Ce cheval de Troie est basé sur la définition du Dropper de Symantec. Son objectif est de créer un tunnel de connexion entre l'interface Wi-Fi connectée à un réseau de sécurité critique et un réseau mobile connecté à Internet.

Trojunnel est développé comme une application native d'Android embarquée dans un jeu Snake sur Android du projet GitHub de Mariano Eloy Fernandez dont le chemin relatif est : mefernandez / android-snake-example. Dans ce projet, nous avons ajouté une autorisation nécessaire - <uses-permission

android:name="android.permission.INTERNET" /> - au package de l'application Snake pour permettre à Trojunnel d'utiliser le réseau internet. Concernant le Snake installé dans l'appareil, il copie le Trojunnel binaire dans le dossier /system/bin et modifie le fichier init.rc pour ajouter des informations et permettre le démarrage automatique de Trojunnel. Cette application est lancée par l'utilisateur de l'appareil. Pour être en mesure d'accéder à un appareil en utilisant le réseau mobile, nous avons une adresse IP statique du fournisseur d'accès sans fil. Dans notre cas, ce service supplémentaire est fourni par SFR (la Société Française de Radiotéléphone qui est l'un des opérateurs de télécommunication française).

Pour établir un tunnel de connexion, Trojunnel détecte les interfaces connectées et s'exécute sur un serveur en ouvrant un port sur l'interface du réseau mobile. Et puis, il transmet les données destinées à n'être utilisées que dans le réseau privé Wi-Fi par l'intermédiaire du réseau mobile public. De cette façon, les nœuds de routage dans le réseau privé ne se rendent pas compte que la transmission fait partie du réseau public.

Notre mécanisme de détection est un mécanisme en deux étapes. La Figure 2.16 montre un algorithme simplifié de détection :

*Figure 2.16 Pseudo code de l'algorithme de détection*

```
For (p in running_processes) {
      set malicious_processes to true;
       if p is installed from trusted store
           set malicious_process to false;
        else
             if(p open port on mobile network
                or
                p use a socket on mobile network)
                and
                p open a socket on Wi-Fi network
                     set malicious_process to false;
         if malicious_process
               kill malicious_process
               put malicious_process binary in quarantaine
               report the malicious_process using Sensu Client
 }
```

La première étape est consacrée à l'identification des processus suspects qui utilisent les ressources du réseau et qui ne sont pas installés par le biais d'une boutique officielle. Notre objectif est de trouver les processus qui ont le profil d'un code malveillant, capable de configurer un tunnel entre le réseau privé et le réseau public. Une fois identifiés, ces processus sont d'abord arrêtés en les tuant. Les binaires associés à ces processus sont ensuite placés en quarantaine. Ce qui signifie que les binaires sont copiés dans un répertoire sécurisé où l'exécution n'est pas autorisée. Enfin, un rapport est envoyé en utilisant Sensu Client au serveur du monitoring pour permettre à l'administrateur de décider si ce sont des processus autorisés ou vraiment malveillants.

En plus de notre solution de détection, nous avons installé un logiciel commercial fourni par Colasoft LLC, pour disposer de données supplémentaires afin de comparer avec les données générées par notre solution. Nous avons opté pour une version d'évaluation de Capsa Enterprise. Notre motivation pour l'utilisation de ce logiciel vient du fait que Colasoft Capsa admet un algorithme dédié à la détection des portes ouvertes. Il est effectivement utilisé dans de nombreuses entreprises. Cet algorithme est basé sur la détection des ports. Capsa contrôle les ports : 31337, 31335, 27444, 27665, 20034, 9704, 6063, 5999, 5910, 5432, 2049, 1433, 444 et 137-139, afin de déterminer si le réseau est infecté par une porte ouverte. S'il y a une communication au niveau de ces ports, Capsa isole l'hôte infecté pour assurer la sécurité du réseau.

## 5.3. Résultats de nos expérimentations

Nous avons trois itérations de 1000 séries de tests dans un environnement Sandbox en utilisant Oracle Virtual Box pour virtualiser le système en cours de test. Nos tests de fonctionnement sont divisés sur 500 tests utilisant Android VM et les 500 autres utilisant cinq appareils physiques : Motorola Nexus 6, Samsung Galaxy S5, Samsung Galaxy S4, Samsung Galaxy Alpha, et Udoo. Nous avons utilisé chaque appareil pour exécuter 100 tests au cours de notre expérimentation.

La première itération est dédiée à la détection et l'exécution de Trojunnel passif où le programme est chargé dans la mémoire, mais n'envoie pas d'information sur le réseau, il est en attente pour la connexion du contrôleur. La seconde itération est dédiée à l'exécution de Trojunnel actif où le programme est activement transmis via un tunnel entre le Wi-Fi et le réseau mobile. La troisième itération est dédiée à un cheval de Troie qui est déjà identifié par l'anti-spyware.

Quelques remarques intéressantes : notons que Trojunnel utilise un port aléatoire pour établir le tunnel. La seule contrainte est que le port doit être disponible sur l'appareil mobile. En ce qui concerne l'appareil de contrôle (la machine du pirate), il est utilisé pour scanner le réseau privé d'un réseau public. Concernant Udoo Board, nous n'utilisons pas un réseau mobile, mais un réseau câblé en raison de la limitation de matériels. Enfin, concernant l'émulateur, nous utilisons deux interfaces virtuelles. Une interface est connectée à un routeur virtuel et utilisée pour émuler le réseau mobile. L'autre interface est associée à l'interface du réseau hôte.

*Tableau 2.6 Résultats de notre solution*

| Appareil | Trojunnel Actif | Trojunnel Passif | Dropper.A |
|---|---|---|---|
| VM | 95 % | 95 % | 85 % |
| NEXUS 6 | 91 % | 91 % | 76 % |
| GALAXY S5 | 99 % | 99 % | 91 % |

| GALAXY S4 | 99 % | 99 % | 93 % |
|---|---|---|---|
| GALAXY ALPHA | 97 % | 97 % | 83 % |
| UDOO | 95 % | 95 % | 80 % |

Le Tableau 2.6 montre le pourcentage de détection des processus suspects et variants comme Trojunnel, Dropper.A sur plusieurs appareils. Il en ressort que notre solution admet de très bons résultats pour la détection de Trojunnel due à la perspective du monitoring interne et l'algorithme qui est considéré. Nous notons que nous avons eu des résultats très similaires entre les exécutions du Trojunnel passif et actif en raison du fait que notre solution n'attend pas qu'une communication ait eu lieu pour détecter le code malveillant. En outre, notre solution a un bon résultat en ce qui concerne la détection de Dropper.A.

*Tableau 2.7 Résultat du Capsa*

| Périphérique | Trojunnel Actif | Trojunnel Passif | Dropper .A |
|---|---|---|---|
| VM | 85 % | 1 % | 88 % |
| NEXUS 6 | 65 % | 0 % | 86 % |
| GALAXY S5 | 74 % | 0 % | 92 % |
| GALAXY S4 | 78 % | 0 % | 91 % |
| GALAXY ALPHA | 87 % | 0 % | 89 % |
| UDOO | 75 % | 1 % | 89 % |

Le Tableau 2.7 illustre les résultats de Capsa qui sont obtenus en utilisant la technique d'inspection approfondie recommandée en cas d'hôtes suspects. Notons que c'est une opération laborieuse en plus, elle n'est pas entièrement automatisée.

*Tableau 2.8 Comparaison des résultats*

| Solution | Trojunnel Actif | Trojunnel Passif | Dropper .A |
|---|---|---|---|
| La Notre | 95 % | 95 % | 84 % |
| Capsa | 80 % | 1 % | 88 % |

La comparaison des résultats, illustrée par le Tableau 2.8, montre que notre solution a de meilleurs résultats pour la détection du Trojunnel actif. Les résultats montrent également le fait que le mode passif

est probablement indétectable avec une solution classique. Grâce à notre approche, nous obtenons le même taux de détection en utilisant le mode actif. Les résultats concernant Dropper.A sont comparables.

# VI. Bilan

Dans ce chapitre, nous avons détaillé l'architecture qui résulte d'une Cloudlet et du Cloud Computing. Les défis lors de la surveillance des périphériques mobiles et des capteurs ont été relevés afin d'aborder la conception d'un framework pour la surveillance d'un réseau de Cloudlets. Ce framework a pour avantage l'omniprésence des périphériques mobiles et crée un outil de surveillance permettant la collecte de données pour fournir d'un point de vue interne, les informations qui se déroulent à l'intérieur des périphériques en temps réel. En plus de l'architecture proposée, nous avons présenté des études de cas sur la cyber-sécurité portant sur les chevaux de Troie et traitant d'une sécurité connexe.

Le travail présenté dans ce chapitre est encore préliminaire, il présente un système qui autorise la collecte de données lors de la déportation et la migration des applications mobiles à partir des périphériques mobiles vers la Cloudlet. En effet, pour calculer le coût de la migration qui est la base de la décision de migration, nous avons besoin des mesures et des données recueillies à partir des périphériques concernés et des infrastructures de la Cloudlet. L'architecture de monitoring proposée dans ce chapitre autorise la collecte des données qui nous permettent de valider que la déportation d'applications mobiles s'est effectuée suivant les conditions demandées. Dans le cas contraire, elle fournit les raisons du refus de migration.

Dans le chapitre suivant, nous allons développer une spécification formelle écrite avec le langage $\pi$-calcul qui définit la représentation des périphériques virtuels dans le MCC. En outre, nous allons décrire une nouvelle vision des périphériques virtuels composites qui peuvent être utilisés par tous les périphériques, les capteurs et les actuateurs disponibles sur le réseau. Enfin, nous allons aborder la déportation des applications et la gestion en réseau des périphériques virtuels sur des Clouds mobiles.

# CHAPITRE 3 : Spécification formelle des périphériques virtuels dans une Cloudlet

Dans le domaine émergent du Mobile Cloud Computing, deux aspects sont considérés comme fondamentaux. D'un côté, nous avons les aspects de virtualisation qui affectent les Datacenters et d'autre part, nous avons les périphériques mobiles qui se sont révélés être des outils les plus efficaces et pratiques dans la vie quotidienne. Au niveau du Cloud Computing, le développement s'accentue autour de la mobilité surtout en termes d'espaces de travail, d'interaction avec les périphériques connectés et des capteurs. L'objectif de ce chapitre est de fournir une spécification formelle écrite avec le langage $\pi$-calcul qui définit la représentation des périphériques virtuels dans la Cloudlet. Pour cela, nous décrivons notre vision des périphériques virtuels composites qui peuvent être utilisés par tous les autres périphériques, les capteurs et les actuateurs disponibles sur le réseau. Dans la suite, nous abordons la déportation d'applications et la gestion en réseau des périphériques virtuels de la Cloudlet. Notre modèle architectural est organisé autour d'un réseau de Cloudlets. Enfin, nous présentons nos spécifications formelles pour l'architecture d'une Cloudlet. Puis nous définissons une congruence structurelle que nous appliquons à l'évaluation des termes décrivant l'exécution locale et distante d'une application mobile respectant une structure ad-hoc. Grâce à la sémantique de cette opération structurelle, nous montrons que les exécutions d'une application mobile dans le périphérique mobile et de la même application déportée sur une Cloudlet sont similaires.

## I. Description d'une vision des périphériques virtuels composites

De nos jours, les périphériques mobiles sont de plus en plus utilisés, essentiellement comme outil de communication ou comme source de données. La non-limitation en temps et en espace d'utilisation de ces périphériques permet aux utilisateurs mobiles d'accumuler une riche expérience de ses divers services et applications. L'exécution de ces services ne se limite pas au périphérique mobile lui-même. Ainsi, de plus en plus d'applications utilisent des serveurs distants via des réseaux sans fil pour interagir avec ces services.

## 1.1. Délégation des modules et des ressources

Les architectures N-tiers se sont répandues rapidement dans le développement des technologies de l'information (IT ou Information Technology) ainsi que dans les domaines du commerce et de l'industrie liés à l'informatique mobile [145]. Comme vu au chapitre 2, ces systèmes respectent le pattern architectural nommé Layer. Il est ainsi convenu de disposer d'une couche de présentation, d'une couche de traitement métier, d'une couche de gestion d'accès aux ressources et enfin d'une couche de gestion de données. Cette séparation par couches de responsabilités sert à découpler au maximum une couche de l'autre afin d'éviter l'impact d'évolutions futures de l'application.

### 1.1.1. Délégation des modules et des ressources

Nous avons déjà mentionné au chapitre 1 les nombreuses limites des périphériques mobiles (section 1.2). L'informatique mobile fait face à de nombreux défis en essayant de fournir les diverses applications résidant dans un seul périphérique avec des ressources limitées telles que la batterie, la mémoire et la bande passante. Nous avons aussi mentionné les besoins au chapitre 1 (section 1.3). Ils laissent apparaître une volonté de gérer au plus juste la consommation en énergie des périphériques d'où le besoin de déléguer des modules de l'application et des ressources consommatrices à des serveurs distants voire en utilisant un Cloud local public ou privé (section 2.1.2). Google offre une solution au niveau PAAS appelée AppEngine [146]. AppEngine permet aux développeurs n'ayant pas une bonne compréhension des limites précédentes ou de la connaissance des infrastructures du Cloud, de déployer des services et d'utiliser le Cloud de Google. Cette plate-forme exécute les services déployés et les expose en tant que services distants. Une telle approche offre une solution à la déportation logicielle d'un module appartenant à une application mobile, dont la consommation d'énergie ne serait pas facilement identifiable en raison de son exécution dans un système embarqué. Désormais, un second aspect est à considérer, il s'agit de l'architecture logicielle d'une application mobile. Il devient désormais crucial de s'appuyer sur une spécification de composant permettant le chargement et le déchargement de composants en cours d'exécution. Cette notion de composant semble un vieux sujet de Génie Logiciel toujours en cours de débat. Pour autant, quelques travaux ont abouti à des spécifications de références telles que OSGi ou SCA [147].

La spécification OSGi autorise de plus la résolution de dépendances à l'exécution. Ainsi, la migration d'un composant peut déclencher le déplacement d'une grappe de composants du périphérique mobile vers une Cloudlet.

### 1.1.2. Niveau d'abstraction et les limites de la virtualisation

Actuellement, les architectures des Clouds mobiles sont basées sur le niveau d'abstractions de services du Cloud Computing (IaaS, PaaS et SaaS) [148]. Ces architectures répondent à la virtualisation et à la

distribution des services déployés. Ainsi une image d'un périphérique construit pour un Cloud $C_1$ n'est pas nécessairement portable pour un Cloud $C_2$.

L'absence de formalisme spécifique pour aborder la virtualisation mobile contribue à l'hétérogénéité des solutions actuelles. En effet, la virtualisation des périphériques et des services se conforme aux architectures des serveurs qui ne sont pas appropriées pour les plates-formes mobiles. En outre, les artefacts des services déployés sont ceux d'un service Web classique. Il n'y a aucune représentation spécifique qui fait abstraction de la déportation des applications et de la gestion de la localisation. Dans ces solutions, les implémentations des services à distance dépendent à la fois de la plate-forme du Cloud et de la capacité en termes de ressources matérielles. Du point de vue développement, cette contrainte implique la non-réutilisation au niveau du client d'un composant logiciel développé comme un service à distance. En outre, les interfaces qui exposent les mêmes services, peuvent être utilisées différemment d'une implémentation à une autre.

## 1.2. Représentation des périphériques mobiles et des capteurs

Plusieurs niveaux d'abstraction sont à prendre en compte au niveau d'une Cloudlet. D'une part, le niveau matériel où un smartphone doit pouvoir être virtualisé, est une image de ce type de smartphone qui doit être construite sur un type de Cloudlet. De plus, les composants d'application mobile qui ont été déportés du smartphone vers la virtualisation de ce dernier doivent aussi être virtualisés. Le niveau logiciel supplémentaire fait apparaître la difficulté de cette approche. Il est naturel qu'un composant ayant accès à une ressource du smartphone telle qu'une connexion réseau, dispose d'une connexion dans le contexte virtualisé. D'où l'initiative de recréer le contexte d'exécution des composants afin que leur chargement et leur usage soient similaires.

Les communications émises par les périphériques sont converties selon les protocoles spécifiques à cette structure. Et les réponses sont stockées sur un cache qui est l'état virtuel du périphérique. Cette représentation agit aussi comme une plate-forme "mobile-friendly" dans le Cloud. En effet, la représentation est construite sur les capacités d'émulation qui offrent un environnement compatible avec le périphérique physique sur lequel la représentation est associée.

Nous distinguons trois types de représentations en fonction de leur association (ou non) avec les périphériques physiques.

a) Le premier type de représentation s'applique aux simples capteurs et actuateurs. C'est la forme de représentation la plus simple qui agit comme un proxy cache avec une interface commune. À titre d'exemple, il est possible de construire une telle représentation pour une caméra ou un microphone ;

b) Le deuxième type de représentation s'applique aux périphériques mobiles tels qu'une tablette ou un smartphone. Ce type offre les capacités de déportation et garde dans leur cache, l'état des différents capteurs et actuateurs disponibles sur le périphérique mobile. Nous considérons ce type de représentation comme une composition de ressources associées à un périphérique mobile. Ce n'est pas une image exacte du périphérique mobile, elle peut être utilisée comme une extension des ressources disponibles localement sur le périphérique mobile ;

c) Le troisième type de représentation n'a aucune association directe avec un périphérique physique. C'est une composition de multiples représentations. Nous la définissons comme une composition de ressources distribuées sur le réseau, qui transforment le périphérique mobile en une sorte de "super périphérique" en éliminant les limitations physiques.

La représentation composite ajoute une intelligence aux périphériques en permettant la composition de plusieurs périphériques d'une manière harmonieuse. Afin de virtualiser la composition d'une caméra et d'un microphone issus de la même pièce, il est possible de construire une telle composition avec ce type de représentation. Dans la section suivante, nous exposons les différents défis à relever au cours de la virtualisation et comment ils sont traités.

# II. Paradigme de la virtualisation dans la Cloudlet

L'usage d'une Cloudlet ou d'un réseau de Cloudlets (section 2.2.3) est apparu naturellement comme une solution de gestion d'énergie en milieu difficile ou hostile. Lorsque le Cloud n'est pas présent, il apparaît utile d'avoir préparé un cache utile de ce Cloud afin de continuer le travail dans de bonnes conditions. D'autres usages sont apparus avec la mobilité informatique (section I du chapitre 1) où si on déporte une partie d'une application mobile, il ne faut pas pour autant interdire la mobilité de son utilisateur final. Ainsi, un réseau de Cloudlets est la solution naturelle pour effectuer le suivi de l'usager mobile et lui offrir une continuité de service pour les composants déportés au cours de ses déplacements. Dans ce cas, il devient indispensable que l'ensemble des Cloudlets partage une même spécification afin de ne pas tomber dans le même écueil que celui des Clouds où l'interopérabilité n'est pas assurée. Nous avons fait le choix d'utiliser un langage de description formel, basé sur une sémantique entièrement définie afin qu'il n'y ait pas d'approximation dans l'interprétation de notre architecture.

Ainsi, il est possible d'envisager un réseau de Cloudlets constitué de plusieurs Cloudlets construites pour différents éditeurs respectant la même spécification formelle.

Dans la suite, nous décrivons les techniques existantes du Cloud qui sont utiles pour le MCC et présentons les formalismes qui sont liés aux différents aspects de la virtualisation. Nous terminons cette section avec une introduction au langage $\pi$-calcul qui est le formalisme utilisé pour notre définition.

## 2.1. Virtualisation et Cloudlet

Dans le cadre d'une Cloudlet, la virtualisation est utilisée pour fournir une architecture matérielle d'accueil à tous les futurs Backend applicatifs une fois la migration effectuée. Bien entendu, cette architecture a des besoins tels que la possibilité d'avoir une connexion point à point avec un autre périphérique virtualisé dans la même Cloudlet ou dans une autre. Comme dans la Figure 2.2, une Cloudlet doit aussi virtualiser un routeur et au moins un switch. D'autres contraintes peuvent être ajoutées au cours de cette étude.

### 2.1.1. Types de virtualisation

Deux sortes de virtualisation sont utilisées pour simuler le matériel de la machine et permettre l'exécution d'un système d'exploitation (SE) invité.

a. Emulation

Ici la machine virtuelle (VM) émule ou simule le matériel complet, si le SE invité est non modifié pour une autre machine, il ne peut pas être exécuté. Il y a certains hyperviseurs spécialisés sur l'émulation comme Bochs, Virtual PC pour Mac et Qemu [105].

b. Totale / Native

Ici la machine virtuelle (VM) simule suffisamment le matériel pour permettre à un SE invité non modifié de s'exécuter de façon isolée. Ce type de virtualisation exige que le même matériel CPU soit utilisé par la machine virtuelle et l'hyperviseur. Cette virtualisation est la plus utilisée grâce à la simplicité de fonctionnement des applications et à la puissance des outils d'administration existants. On retrouve beaucoup d'hyperviseur comme VMware, Workstation, Microsoft Hyper-V.

### 2.1.2. Usage de la virtualisation

Nous pouvons aborder la virtualisation de serveurs et de postes de travail. Étant donné que la virtualisation permet de créer une version virtuelle d'un périphérique ou d'une ressource, de nombreux périphériques informatiques peuvent être virtualisés, y compris un serveur, un poste de travail, une mémoire, un réseau et des systèmes d'exploitation. Ce sont les deux technologies de virtualisation les plus courantes :

a. Virtualisation de serveurs peut se définir comme le fait de faire fonctionner plusieurs serveurs virtuels sur un serveur physique, ce dernier étant alors remplacé par son équivalent virtuel ;
b. Virtualisation de postes de travail ou client consiste à afficher sur un poste, des dizaines, des centaines voire des milliers de postes physiques, une image virtuelle du poste utilisateur qui est en fait réellement exécutée sur un serveur distant.

En utilisant la virtualisation de serveurs, plusieurs instances de la VM contenant les systèmes d'exploitation peuvent fonctionner sur un seul serveur physique ou une seule VM peut utiliser le hardware à partir de plusieurs serveurs physiques, chacune avec un accès aux ressources informatiques du serveur sous-jacent. La virtualisation est utilisée pour absorber les pertes de ressources causées par le fait que les serveurs hôtes fonctionnent à moins de 15 % de leur capacité, conduisant à la prolifération et la complexité des serveurs. Selon les statistiques VMware [149], la virtualisation peut offrir 80 % d'utilisation des ressources sur le serveur ou un meilleur ratio de consolidation du serveur. La virtualisation rend très facile la gestion et l'évolution du système d'information.

Les efforts actuels visent la formalisation des interactions des services du Cloud [150] et l'orchestration [151]. Cependant, ces efforts n'abordent pas l'aspect de la virtualisation de tels systèmes du Cloud Computing.

### 2.1.3. Virtualisation du réseau

La virtualisation du réseau consiste à combiner des ressources réseaux (matérielles et logicielles) dans une seule unité administrative. L'objectif de la virtualisation du réseau est de fournir aux systèmes et utilisateurs un partage efficace, contrôlé et sécurisé des ressources réseaux.

Le résultat de la virtualisation du réseau est un réseau virtuel. Les réseaux virtuels sont classés en deux grandes catégories : externes et internes.

Les réseaux virtuels externes sont composés de plusieurs réseaux locaux administrés par le logiciel comme une entité unique. Les blocs de construction des réseaux virtuels externes standards sont le matériel de commutation et la technologie logicielle VLAN (Virtual Local Area Network ou réseau local virtuel). Les réseaux virtuels externes comprennent par exemple les grands réseaux d'entreprise et des Datacenters.

Un réseau virtuel interne se compose d'un système utilisant des machines virtuelles ou des zones dont les interfaces réseau sont configurées sur au moins une carte réseau (NIC) physique. Les ressources de réseaux virtuels sont également divisées en deux catégories.

- La virtualisation des ressources physiques comme vRouter (routeur) et vSwitch (Switch) ;
- La virtualisation des ressources comme FWA (pare-feu) et LBA (équilibrage de charge).

Cette approche de la virtualisation de réseau est appelée NFV (Network Functions Virtualization) [152].

Les solutions NFV offrent une nouvelle façon pour les CSPs (Communications Service Provider) de concevoir, déployer et gérer des services réseaux. La technologie NFV permet aux CSPs de redéfinir la base de coûts des opérations liées au réseau et de créer les environnements de fourniture de services flexibles dont ils ont besoin pour générer des revenus et réduire les coûts.

En parallèle à des travaux pragmatiques sur la gestion du réseau, il existe un bon nombre de travaux sur la définition du formalisme dédié. U. Montanari et M. Sammartino ont travaillé sur une extension propre

du langage π-calcul [153]. Le calcul du processus résultant fournit à la fois un entrelacement et un réseau concurrent orienté sémantique. Singh A. et al. [154] ont également travaillé sur une extension appelée ω-calcul, qui est le formalisme de modélisation et de raisonnement sur les réseaux mobiles ad hoc sans fil. Ces travaux portent sur le raisonnement et la vérification des protocoles réseaux et ne traitent pas l'aspect de la virtualisation du réseau. Ce manque de formalisme sur la gestion de la virtualisation du réseau motive aussi notre définition de la virtualisation du réseau au haut niveau dans les sections suivantes.

## 2.2. π-calcul

Dans une Cloudlet, les appareils, les services et leurs compositions sont exécutés dans un environnement parallèle et distribué. La représentation de ces périphériques doit suivre un modèle de calcul des systèmes parallèles.

### 2.2.1. Construction du π-calcul

Le π-calcul [155] est une algèbre de processus qui offre une syntaxe pour représenter les processus (comme les représentations des périphériques ou des services), la composition parallèle de processus, la communication synchrone entre les processus par le biais de canaux (ou des noms), la création de nouveaux canaux, la réplication de processus et le non-déterminisme (ou points de choix). Il fournit des primitives pour décrire et analyser un système distribué, qui utilise la migration de processus entre pairs, l'interaction des processus via des canaux de communication dynamiques et privés.

Les deux concepts importants dans la définition du π-calcul [156] sont les processus et les canaux. En effet, un processus est une abstraction d'un processus de contrôle qui peut représenter un périphérique sur la Cloudlet. Un service d'une Cloudlet ou du Cloud Computing comme les fonctions du réseau, ou des business services peuvent migrer des périphériques mobiles à la représentation du périphérique mobile sur la Cloudlet. Un canal est une abstraction de la liaison de communication entre deux processus. Il peut représenter une liaison de service Web entre le périphérique mobile et sa représentation virtuelle en tant que canal de synchronisation. Les canaux peuvent aussi représenter l'interaction de l'envoi et la réception de messages entre un processus local (périphérique mobile) et un autre processus à distance sur le Cloud.

*Tableau 3.1 Construction du Pi-calcul*

| **Composition** | **Dénotation** |
|---|---|
| $P\|Q$ | Processus composé de P et Q exécutés simultanément et en parallèle |
| $P + Q$ | Processus qui se comporte comme P ou Q. (choix non déterministe) |

| $a(x).P$ | Processus qui attend de lire une valeur x du canal a et puis, l'ayant reçue, se comporte comme P. |
|---|---|
| $\bar{a}\langle x\rangle.P$ | Processus qui attend d'envoyer la valeur x via le canal a en premier et puis, se comporte comme P. (communication synchrone) |
| $(\nu\ a)P$ | Restriction : elle garantit que $a$ est un nouveau canal (nom libre) dans P. ($\nu$ se prononce "new") |
| $!\,P$ | Réplication du processus P : c'est une composition infinie ou un nombre infini de copies de P, toute fonctionnant en parallèle. |
| $(\lambda\ a)P$ | Une abstraction de $a$ définie sur le processus P ou substitution par un nom lié |
| P ˆ Q | Chaînage entre processus : le processus P enchaîné à un processus Q par un opérateur combinatoire sur une abstraction binaire. |
| $\vec{a}$ | Un vecteur de noms ou la grandeur du vecteur (où la taille est notée \| a \|) |
| $\emptyset$ | Processus nul, il est utilisé pour terminer un processus ou il s'agit d'un processus qui a terminé son exécution |
| $\tau$ | Une communication interne (action interne, non observable) |

### 2.2.2. Règle du π-calcul : congruence structurelle

La notion de congruence structurelle en π-calcul est utile pour l'étude de comportements et leurs similitudes. Elle permet de prouver l'équivalence sur les constructions de processus. La sémantique opérationnelle en π-calcul est utile pour calculer l'évolution d'un processus et est donnée par un système de transitions étiquetées ou labellisées. Tout comportement concurrent qui a besoin de définir une représentation de notre vision d'une Cloudlet devra être écrit dans les termes de la construction du Tableau 3.1 et des règles du Tableau 3.2 où P et Q désignent des processus.

*Tableau 3.2 Règle du Pi-calcul*

| $P \equiv Q$ | Processus Q obtenu à partir de P en renommant un ou plusieurs noms liés à P. (congruence structurelle) |
|---|---|
| $P \rightarrow P'$ | Processus P' est obtenu après une étape de calcul réalisé par P. (relation de réduction) |
| $P \xrightarrow{\alpha} P'$ | Processus $P'$ est obtenu après une action $\boldsymbol{\alpha}$ (exemple : $\bar{\boldsymbol{a}}\langle\boldsymbol{x}\rangle, \boldsymbol{a}(\boldsymbol{x}), \tau$) réalisée par $P$. (Transition labellisée) |

Il est alors possible de construire un arbre représentant l'évaluation d'un terme π-calcul par l'application des règles de la sémantique opérationnelle. Plusieurs arbres sémantiques (ou arbre de dérivation) peuvent être comparés par l'emploi d'une bisimulation.

# III. Représentation des périphériques virtuels de la Cloudlet

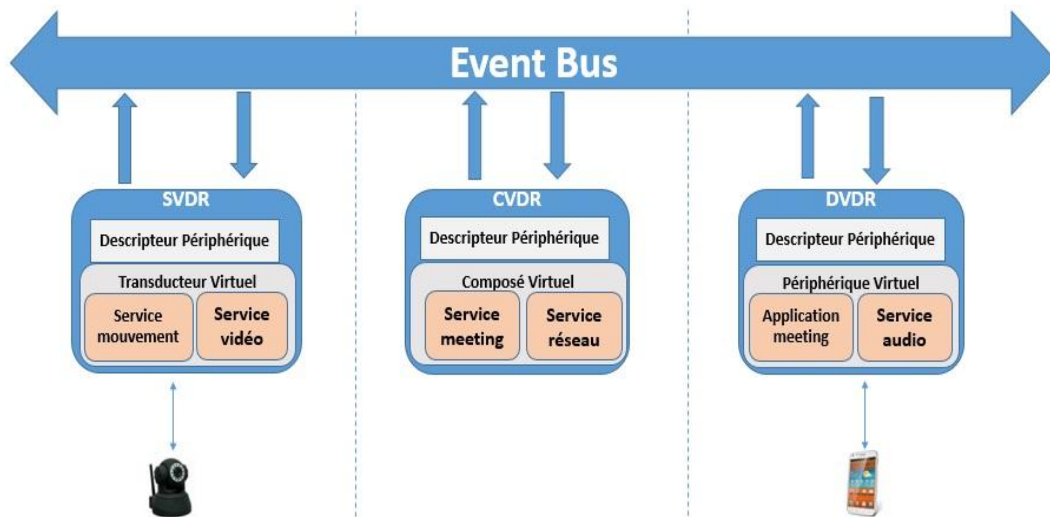
Dans cette section, nous présentons les différents aspects autour de la notion de VDR en spécifiant formellement cette notion en π-calcul d'ordre supérieur (HOπC) [157], soulignant le mécanisme d'orchestration pour la VDR et de la gestion du réseau. Notre choix pour le HOπC est motivé par la nécessité d'exprimer la mobilité des VDRs dans un réseau de Cloudlets, également la mobilité des applications entre les périphériques physiques et ses VDRs. Dans notre définition, nous n'utilisons pas les extensions liées au réseau en π-calcul pour deux raisons : premièrement, ces extensions ne traitent pas le paradigme d'ordre supérieur et deuxièmement elles sont conçues pour exprimer les protocoles de gestion du réseau et non la communication orientée virtualisation.

## 3.1. Composition des VDRs

Dans notre approche, la VDR vise à aborder le paradigme de la virtualisation au sein d'une Cloudlet. Nous utilisons le sigle VDR comme un label générique pour tous les types de représentation dans la Cloudlet.

Il y a beaucoup de différences entre les représentations d'un simple capteur, d'un périphérique mobile, et d'un périphérique composite. D'une part, la capacité de virtualisation diffère d'un type de VDR à l'autre. À titre d'exemple, la VDR d'un capteur ne possède aucune capacité de déportation. Toutefois, la déportation des applications est au cœur de la VDR du périphérique mobile. De l'autre, les connecteurs des périphériques différent également d'un type de VDR à l'autre. Les capteurs pourraient être reliés à leurs VDRs en utilisant certaines normes spécifiques comme 802.15.4 (ZigBee, UWA, ...) [158] qui offrent un taux de transfert acceptable pour synchroniser l'état du capteur avec sa VDR. Pour sa part, le périphérique mobile a besoin d'utiliser des technologies avec un taux de transfert élevé. Cela permet le transfert de données nécessaires à l'application afin de la déporter vers la Cloudlet. La VDR composite n'admet pas de connexion directe au périphérique physique, elle communique avec les périphériques physiques par le biais de leur représentation virtuelle en utilisant un bus d'événement (Event Bus). La Figure 3.1 donne un aperçu des trois différents types de VDR identifiées ainsi que le bus d'événement. De gauche à droite, cette figure montre la liaison d'une caméra et sa virtualisation, celle d'une composition, puis celle d'un périphérique mobile. Le bus d'événement joue le rôle de médiateur entre les VDRs en acheminant l'information à destination de ceux qui peuvent la traiter.

*Figure 3.1 Connexion des VDRs*




En raison de leurs différences, les trois types de VDRs ont une définition formelle spécifique définie par les équations (5), (8) et (12) (voir les pages 84, 85 et 86). Ils ont également des rôles spécifiques dans la Cloudlet :

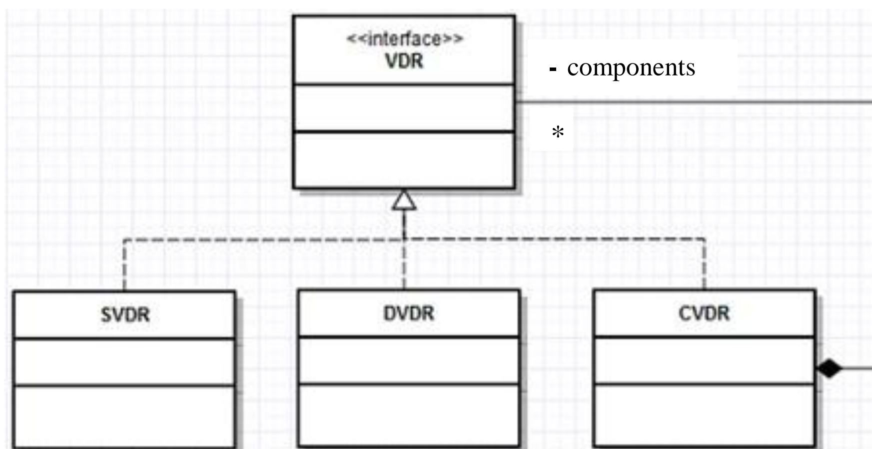
a) VDR du capteur (SVDR ou Sensor VDR) : elle représente un capteur physique ou un actuateur au sein de la Cloudlet, comme une LED ou un capteur de pression ;

b) VDR du périphérique mobile (DVDR ou Device VDR) : elle représente un périphérique mobile physique au sein de la Cloudlet comme un smartphone ;

c) VDR Composite (CVDR) : elle représente une composition de SVDR, de DVDR et des ressources de la Cloudlet comme l'agrégation de toutes les caméras contenues dans une salle de conférence.

La composition des VDRs doit respecter un schéma puisque la SVDR et la DVDR ne peuvent pas être composite. Seule la CVDR peut être en même temps un composite et un composant. La Figure 3.2 illustre les relations de composition et de généralisation entre les différents types VDRs.

*Figure 3.2 Relation de composition entre VDRs*



### 3.1.1. Définition de la VDR

La VDR est définie comme la composition des ressources (CPU, RAM, Mémoire, …), les périphériques mobiles et des capteurs. Elle est un composant logiciel composite qui fournit une émulation du comportement du matériel physique qu'il représente.

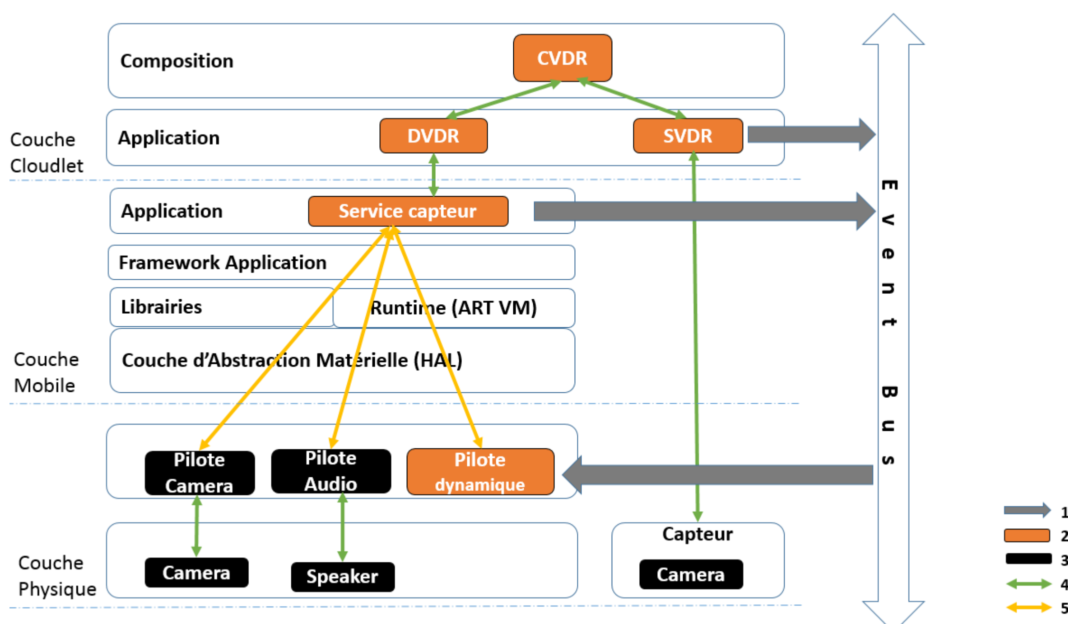
Une VDR est une instance d'une VM utilisée dans la Cloudlet, hébergeant un système d'exploitation mobile et exposant des services de gestion qui émulent un écran d'affichage et/ou un clavier. De même que le périphérique physique de poche qu'elle représente, elle peut exécuter plusieurs types d'applications mobiles (connus sous le nom Apps) et admet une connexion réseau.

Une VDR dispose de deux types d'association sur le périphérique physique. Comme le montre la Figure 3.2, d'une part, il y a l'association directe à un périphérique mobile ou un capteur d'où l'association de généralisation. Nous appelons ce premier type "VDR émulée". D'autre part, on a une agrégation qui décrit la notion de contenance. On admet qu'une VDR ne peut pas avoir une association directe à un périphérique physique. Dans ce cas, on parle de « VDR native ». Dans la Figure 3.2, la CVDR illustre ce type de VDR.

### 3.1.2. Spécification formelle de la composition de la VDR

La VDR fonctionne selon une architecture orientée événements. Toutes les interactions sont initiées par un message envoyé à partir d'un bus ou par un appel de service. Le message survient à la suite de l'activité de détection du matériel. Nous définissons un vecteur d'événements représentant toutes les interfaces d'une VDR. Ce vecteur d'événements contient des canaux qui sont utilisés pour échanger des messages. Le bus d'événements (Event Bus) est au cœur de notre modèle formel.

*Figure 3.3 Rôle de Event Bus (1 : propagation de l'événement, 2 : composants standards, 3 : composants proposés, 4 : communication via la couche application, 5: communication directe)*



Comme illustré sur la Figure 3.3, le bus d'événement est le composant qui permet entre autres la propagation des événements de détection dans un système distribué. En effet, les architectures des périphériques mobiles [159] [160] [161] ne sont pas conçues pour permettre une intégration directe des périphériques distants. Pour permettre ce type d'intégration, nous avons ajouté deux composants dans la couche mobile, appelés le service capteur et le pilote dynamique. Ces composants visent à propager des événements de capteurs embarqués dans les appareils mobiles, à récupérer les événements de détections externes, et intégrer ces événements au niveau du noyau du système d'exploitation mobile (SE).

a) Définition du vecteur d'événements

Le vecteur d'événements n'admet pas une définition formelle statique. Il est défini comme un vecteur qui contient autant de canaux que de capteurs connectés au système. Les événements sont ajoutés au vecteur à la volée tant que les périphériques sont connectés. Un exemple de vecteur d'événements est illustré par l'équation (1) où les canaux sont associés à une caméra, un microphone, un lecteur NFC et un clavier. Tous sont des sources d'événements. Tout au long de ce document, nous utilisons la notation en crochets ([ ]) pour définir les vecteurs.

$$\overrightarrow{evt} \stackrel{\text{def}}{=} [id, c, camera_1, micro_1, micro_2,\ nfc_1, keyboard_1, \ldots] \qquad (1)$$

Le vecteur d'événements contient par défaut deux éléments identifiés par $id$ et $c$. L'élément $id$ est utilisé pour récupérer l'identifiant de la VDR. L'élément $c$ est utilisé pour ajouter une nouvelle VDR à une composition existante. L'équation (13) illustre cette opération d'ajout ou de concaténation. Le vecteur d'événements $\overrightarrow{ev}$ est utilisé uniquement pour les interactions entre les VDRs. Les interactions entre les DVDRs d'une part, des SVDRs et l'appareil physique d'autre part utilisent un service basé sur un canal appelé $ws_i$ qui représente l'échange des services Web.

Nous définissons la généralisation de la VDR comme un choix non-déterministe entre les trois types de VDRs, illustré par l'équation (2).

$$VDR(\overrightarrow{ws}) \stackrel{\text{def}}{=} ((\lambda\, \overrightarrow{ev}\ ws_1)SVDR + (\lambda\, \overrightarrow{ev}\ ws_2)DVDR + (\lambda\, \overrightarrow{ev}\ ws_3)CVDR) \qquad (2)$$

Ce terme est utilisé par l'équation (17) pour instancier une nouvelle VDR spécifique. Le vecteur $\overrightarrow{ws}$ est composé d'une collection de deux éléments $ws_1$ et $ws_2$ qui sont les canaux de services Web associés respectivement à la SVDR et à la DVDR. La CVDR n'admet pas de canal de service Web puisqu'elle n'a aucun lien direct avec un périphérique physique. Cependant, la CVDR a un paramètre d'ordre supérieur représentant la définition précise de la composition comme l'illustre le passage de l'équation (2) à (12).

Le terme $VDR(\overrightarrow{ws})$ admet comme paramètre un vecteur $\overrightarrow{ws}$ de canaux de services Web. Ce vecteur est partagé entre le système du Cloud mobile et les VDRs. Les canaux du $\overrightarrow{ws}$ permettent les échanges avec les périphériques physiques. Le terme VDR crée un nouveau vecteur, appelé $\overrightarrow{ev}$ qui contient les canaux des interfaces VDRs. Nous bénéficions de l'utilisation de l'abstraction dans la définition de la VDR où

le processus $(\lambda\, \overrightarrow{ev}\, ws_1)SVDR$ est écrit de façon naturelle en $SVDR(\overrightarrow{ev}, ws)$, et ainsi de suite pour les deux autres DVDR et CVDR. La VDR spécifique est activée si l'élément correspondant dans le vecteur de $\overrightarrow{ws}$ est un canal valide et non un processus vide $\emptyset$. La syntaxe basée sur l'abstraction permet le passage implicite du vecteur d'événements ($\overrightarrow{ev}$). En effet, un administrateur (humain ou processus automatique) crée un périphérique composite. Lors de la création du périphérique composite, la composition du vecteur d'événements doit être partagée entre les représentations des périphériques qui participent à cette composition. Pour ce faire, le chainage combinatoire de canaux (Voir le Tableau 3.1 et [162]) est utilisé afin de chaîner des canaux d'événements. Une composition doit définir une structure concrète du vecteur d'événements comme illustrée à l'équation (1). Ce vecteur d'événements est utilisé par le chaînage combinatoire (Voir équation (3)) pour activer les canaux nécessaires à l'implémentation concrète des périphériques et des capteurs virtuels.

$$TwoSensors(ws_1, ws_2) \stackrel{\text{def}}{=} VDR(ws_1\hat{}\emptyset)\hat{}\, VDR(ws_2\hat{}\emptyset) \tag{3}$$

$$TwoSensors \equiv (\nu\, \overrightarrow{evt}\,)\, SVDR(ws) \left\{ \overrightarrow{evt} / \overrightarrow{ev} \right\} \Big| (\nu\, \overrightarrow{evt}\,)\, SVDR(ws) \left\{ \overrightarrow{evt} / \overrightarrow{ev} \right\}$$

L'équation (3) montre la combinaison de deux VDRs représentant chacun la virtualisation d'un capteur. Nous pouvons considérer cette représentation de deux capteurs comme une composition parallèle avec interaction sur le canal $ws$.

L'utilisateur final se doit de définir d'une façon plus concrète le comportement des périphériques. Pour illustrer une telle définition, nous donnons dans l'équation (4) un exemple de capteur NFC (Near Field Communication) [163] qui envoie la valeur « $sens$ » sur le canal $nfc_1$.

$$Cnfc(\overrightarrow{ev}, ws) \stackrel{\text{def}}{=} ws(sens).\tau.\overline{ev_{nfc_1}}\langle sens\rangle \tag{4}$$

### b) Définition du capteur virtuel SVDR

La SVDR est activée suite à l'événement de connexion au capteur physique. Une fois connecté, le capteur physique envoie la donnée d'identification (id) au SVDR à travers le canal $ws$. Cette donnée est stockée à l'intérieur de la SVDR à travers le terme $DevId$ défini dans l'équation (7), qui redonne la donnée d'identification sur demande par le canal d'événements.

$$SVDR(\overrightarrow{ev}, ws) \stackrel{\text{def}}{=} ws(id).\tau.(DevId(ev_{id}, id) | VirtualSensor(\overrightarrow{ev}, ws)) \tag{5}$$

Comme illustré dans l'équation (5), le terme $SVDR$ utilise le terme $VirtualSensor$ défini dans l'équation (6) pour envoyer les données perçues par le capteur physique en utilisant le canal d'événements. A ce niveau, nous considérons l'association entre le capteur et le canal de couplage comme une action invisible représentée par $\tau$. Deux comportements possibles peuvent être adoptés par le terme $VirtualSensor$ comme illustré dans l'équation (6) : si une commande Stop de l'équation (18) est reçue, le processus prend fin ; sinon l'action de dispatching est exécutée. La composition parallèle

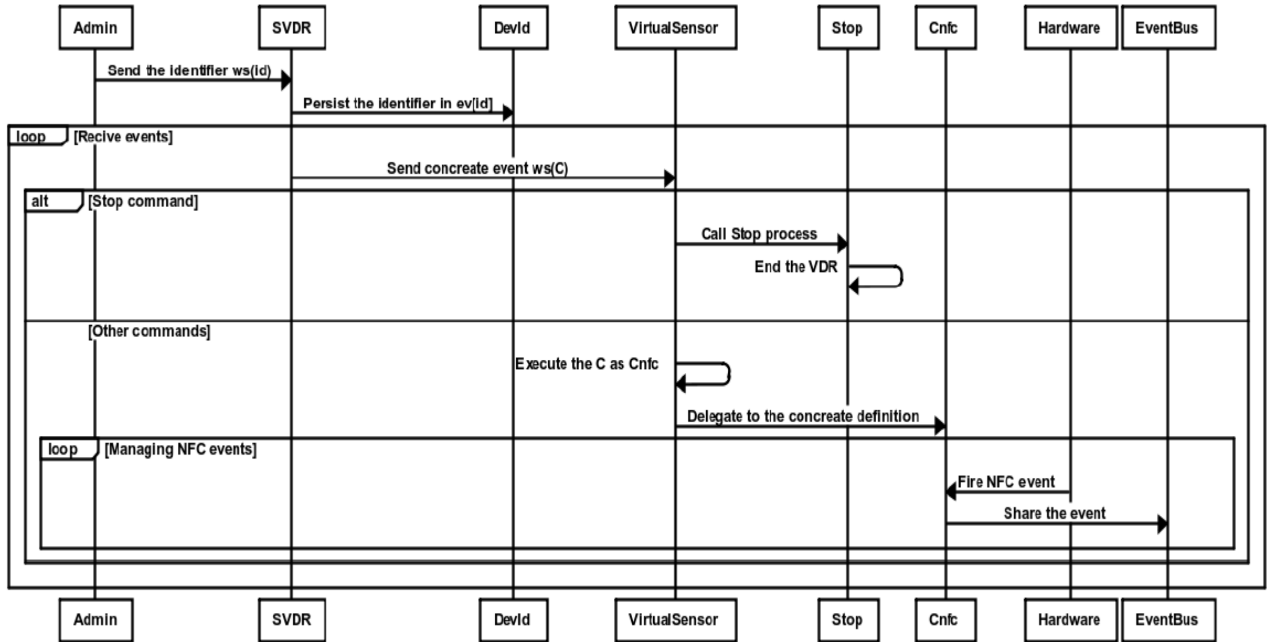
du terme $SVDR$ permet à l'administrateur de récupérer l'identifiant de la VDR en utilisant le canal $ev_{id}$. Cette composition n'influence pas l'exécution du capteur virtuel.

$$VirtualSensor(\overrightarrow{ev}, ws) \stackrel{\text{def}}{=} ws(C).([C = Stop]\ Stop + C(\overrightarrow{ev}, ws).VirtualSensor(\overrightarrow{ev}, ws)) \quad (6)$$

$$DevId(req, id) \stackrel{\text{def}}{=} req(cb).\overline{cb}\langle id\rangle.DevId(req, id) \quad (7)$$

Le diagramme de séquence de la Figure 3.4, illustre l'initialisation d'une SVDR. Les objets sur la première ligne du diagramme de séquence représentent nos termes π-calcul. Cependant, la classe « Hardware » est utilisée pour représenter un capteur physique et la classe « EventBus » est utilisée pour représenter notre bus d'événements $\overrightarrow{evt}$ défini dans l'équation (1). Le « Hardware » (périphérique mobile ou capteur) initie les messages, les envoie via le canal $ws$. Nous ne montrons pas dans ce diagramme les infrastructures réseaux définies dans les équations (22) et (25). Ces infrastructures réseaux sont utilisées pour router les messages d'événements à la cible VDR.

*Figure 3.4 Diagramme de séquence d'initialisation d'une SVDR*




c) Définition du périphérique virtuel DVDR

La spécification d'une DVDR respecte les mêmes fondamentaux que la SVDR. Comme illustrée dans l'équation (8), cette définition est basée sur le terme $DevId$ pour stocker et redonner l'identifiant du périphérique. Elle est basée aussi sur le terme appelé $VirtualDevice$ pour gérer le comportement du périphérique virtuel. Cependant, la DVDR a la capacité d'exécuter des applications au lieu de la SVDR, qui mandate seulement les évènements de capteurs.

$$DVDR(\overrightarrow{ev}, ws) \stackrel{\text{def}}{=} ws(id).\tau.\left(DevId(ev_{id}, id)\middle|VirtualDevice(\overrightarrow{ev}, ws)\right) \quad (8)$$

Nous avons besoin de dissocier les événements de détection envoyés par les capteurs de périphériques embarqués et les demandes d'application déportée. Pour ce faire, nous définissons un type appelé $App$ dans l'équation (9) qui encapsule l'application déportée et agit comme un conteneur d'application.

$$App\big(BackEndProc(ws)\big) \stackrel{\text{def}}{=} BackEndProc(ws) \tag{9}$$

Nous définissons dans l'équation (10) le terme $VirtualDevice$ qui exécute l'application déportée si nécessaire, sinon, il mandate les données de détection.

$$VirtualDevice(\overrightarrow{ev}, ws) \stackrel{\text{def}}{=} ws(C).Com\,.VirtualDevice(\overrightarrow{ev}, ws) \tag{10}$$

$$Com \stackrel{\text{def}}{=} \big([C = Stop]Stop + \big[C = App\big(P(x)\big)\big].P(x) + C(\overrightarrow{ev}, ws)\,\big) \tag{11}$$

La principale préoccupation de ce terme est de faire la distinction entre : l'action de déportation représentée par $App\big(P(x)\big)$, l'action d'arrêt et les actions de détection. Pour ce faire, nous comparons la structure du nom $C$ reçu via le canal de $ws$. Le $VirtualDevice$ exécute le paramètre d'ordre supérieur $P(x)$ au sein de la DVDR sur l'action de déportation, appelle le terme $Stop(\,)$ sur l'action d'arrêt. D'ailleurs, nous transmettons le message au canal d'événements correspondant comme nous l'avons fait pour l'utilisation du terme $C$ concernant la SVDR. Le canal de service utilisé pour la communication entre le périphérique physique et l'application de déportation, est défini comme paramètre $x$ avant l'action de déportation. Ce canal est différent du canal de service qui connecte la DVDR et le périphérique physique.

### d) Définition du périphérique composite CVDR

La CVDR de l'équation (12) ne possède aucune association directe avec un périphérique physique, ses interactions passent à travers une SVDR ou une DVDR. Le terme $CVDR$ est défini comme une agrégation de la SVDR et de la DVDR, qui partage le même vecteur d'événements. Le terme $TwoSensors$ défini dans l'équation (3) est un exemple de définition de la composition qui peut être utilisée comme une première approche.

$$CVDR(\overrightarrow{ev}, C) \stackrel{\text{def}}{=} (\nu\ id)DevId(ev_{id}, id)|CompositeDevice(\overrightarrow{ev})^\frown C \tag{12}$$

Un nouvel identifiant est créé au sein du terme $CVDR$ et remis à la demande en utilisant le canal d'événement $ev_{id}$ du terme $DevId$.

$$CompositeDevice(\overrightarrow{ev}) \stackrel{\text{def}}{=} ev_c(\vec{e}).\Big(CompositeDevice(\overrightarrow{ev})\left\{{}^{\overrightarrow{ev}\wedge\vec{e}}/_{\overrightarrow{ev}}\right\}\Big) \tag{13}$$

Le terme $CompositeDevic$e défini dans l'équation (13) est utilisé pour agréger les canaux d'événements. Le vecteur $\overrightarrow{ev}$ est un canal d'événements associé au terme $CompositeDevice$ actuel tandis que le vecteur $\vec{e}$ est le canal d'événements associé à la VDR pour l'appartenance à cette composition. Cette agrégation est basée sur deux opérateurs : le premier est l'opérateur $\{{}^{x}/_{y}\}$ de changement de nom provenant du langage CCS [164]. Le deuxième est l'opérateur ^ de concaténation.

Le résultat de cette combinaison est l'utilisation de la concaténation de deux vecteurs d'événements $\overrightarrow{ev}\hat{}\vec{e}$ à la place du vecteur d'événements qui a été associé au terme $CompositeDevice$.

## 3.2. Orchestration et gestion du réseau virtuel

Dans un contexte de Cloudlet, l'orchestration est l'automatisation des tâches de gestion et de coordination, des services et des composants. En plus des processus d'interconnexion fonctionnant à travers des systèmes hétérogènes, la localisation des services est une question importante. Les processus et les VDRs doivent traverser plusieurs organisations, systèmes et pare-feu.

L'orchestration de la Cloudlet vise à automatiser la configuration, la coordination, la gestion des VDRs et leurs interactions dans un tel environnement. Le procédé consiste à automatiser les flux de travail nécessaires à la composition des VDRs et à la déportation des applications mobiles. Les tâches impliquées comprennent la gestion de la virtualisation et l'émulation de l'environnement d'exécution des serveurs, le routage du flux de communication des applications aux VDRs et le traitement avec d'éventuelles exceptions à des workflows.

### 3.2.1. Composants de l'orchestrateur

Dans notre approche, l'orchestrateur est composé de trois principaux composants. Comme l'illustre l'équation (14), nous définissons ces trois composants comme des tâches communes d'orchestration :

- Configuration : c'est le composant qui permet à l'orchestrateur du Cloud de gérer le stockage, le calcul et la mise en réseau. Dans cette section, nous ne nous concentrons pas sur l'algorithme d'allocation des ressources (de calcul et de stockage) ; cet aspect sera pris en compte au cours de la prochaine itération de notre projet. Il donnera sûrement de nouveaux résultats sujets à de futures publications. Une spécification de haut niveau du mécanisme de gestion du réseau est présentée dans la section 3.2.2 ;
- Approvisionnement (Provisioning) : il permet à l'orchestrateur du Cloud de gérer les VDRs en fournissant l'exécution, suspension et l'arrêt complet des opérations ;
- Monitoring : c'est le composant qui permet à l'orchestrateur du Cloud de gérer le suivi et le reporting.

Nous avons décrit les détails de l'orchestrateur également sur un document distinct [165] où nous décrivons notre implémentation et des algorithmes détaillés.

$$Orchestrator(\overrightarrow{api}) \stackrel{\text{def}}{=} Configuration(\overrightarrow{api}) \mid \text{Provisioning}(\overrightarrow{api}) \mid (\nu\, \overrightarrow{data})Monitoring(\overrightarrow{api}, \overrightarrow{data}) \quad (14)$$

a) Tâche de configuration

Dans le terme $Configuration$ de l'équation (15), nous illustrons l'usage de la configuration de l'$api$ qui est utilisée pour l'allocation, la désallocation (libération) et la suspension de l'exécution des ressources. L'$api$ est un vecteur dans un espace à deux paramètres $api_j^i$ où l'exposant indique le module cible (exemple $api_a^c$ $\boldsymbol{c}$ signifie $\boldsymbol{Configuration}$) et l'indice indique le service appelé dans le module (exemple : $api_a^c$ où $a$ signifie l'allocation). L'administrateur système utilisera le vecteur $\overrightarrow{api}$ pour configurer l'environnement de déploiement. Nous notons ici que le système d'allocation ou de libération des ressources n'est pas pris en compte. À ce niveau, ces opérations ne sont pas observables. Cependant, nous associons un nouveau nom à chaque demande d'allocation reçue de l'administrateur sur les $api_a^c$.

$$Configuration(\overrightarrow{api}) \stackrel{\text{def}}{=} (api_a^c(allocate).\tau.(\nu\, res)\overline{allocate}\langle res\rangle \mid api_f^c(free).\tau \mid api_s^c(suspend).\tau) \;.Configuration(\overrightarrow{api}) \quad (15)$$

b) Terme Provisioning

Le terme Provisioning de l'équation (16) utilise également une $api$ de demande du module de configuration pour l'allocation des ressources. Une fois attribuée, elle reçoit le terme, exécute le paramètre d'ordre supérieur pour permettre la création de la VDR. Le paramètre est une instance du terme $Run$ défini dans l'équation (17). Nous utilisons l'abstraction des ressources d'information $res$, retournée par le terme $Configuration$, pour communiquer cette information au terme $Run$ qui est préconfiguré avec les deux paramètres avant sa réception à travers le canal $api_r^p$.

$$\begin{aligned}\text{Provisioning}(\overrightarrow{api}) \stackrel{\text{def}}{=} (\,(\,& api_r^p(Run).(\nu\, allocate)\overline{api_a^c}\langle allocate\rangle \\ &\mid allocate(res).(\lambda\, res, ws)Run(\,,)\,) \\ &\mid\; api_s^p(suspend).\overline{api_s^c}\langle suspend\rangle \\ &\quad \mid (\,api_t^p(terminate).terminate(ws)\,.\overline{ws}\langle Stop\rangle.\overline{api_f^c}\langle free\rangle)) \\ &.\,\text{Provisioning}(\overrightarrow{api})\end{aligned} \quad (16)$$

La suspension est déléguée au terme $Configuration$ où elle est représentée comme une action non-observable τ. Le terme Provisioning envoie le terme $Stop$ à la VDR pour terminer son exécution, le canal $ws$ envoyé à travers le canal $terminate$ est utilisé pour cela.

Le terme $Run$ défini dans l'équation (17) compose un vecteur selon le type de la VDR et en fonction du type demandé en utilisant le vecteur $\overrightarrow{type}$. Après la création de la VDR, il crée et envoie un nouvel identifiant en utilisant le canal $ws$. Cet identifiant permet d'utiliser la nouvelle VDR créée. Les

émissions sur le canal $\overline{ws}\langle id \rangle$ avec le nouvel identifiant créé font la paire avec les équations (5) et (8). Le vecteur $\overrightarrow{type}$ dans l'équation (17) contient les paramètres $type_v$, $type_s$, $type_d$ et $type_c$.

Dans cette équation (17), l'utilisation du paramètre $type_v$ permet de tester le type de la VDR demandée, soit $type_s$ pour la **S**VDR, $type_d$ pour la **D**VDR ou $type_c$ pour la **C**VDR.

$$Run(ws, \overrightarrow{type}) \stackrel{\text{def}}{=} [type_v = type_c](\nu\ id)\overline{ws}\langle id \rangle \tag{17}$$

$$\left| \tau.(\nu\ \overrightarrow{ev}) \begin{pmatrix} [type_v = type_s]\ VDR(ws\hat{}\emptyset\hat{}\emptyset) \\ |[type_v = type_d]\ VDR(\emptyset\hat{}ws\hat{}\emptyset) \\ |[type_v = type_c]\ VDR(\emptyset\hat{}\emptyset\hat{}ws) \end{pmatrix} \right.$$

$$Stop() \stackrel{\text{def}}{=} \emptyset \tag{18}$$

c) Terme Monitoring

Pour conserver le sens de notre définition, nous n'avons pas intégré les communications entre les VDRs et le module de monitoring défini par le terme $Monitoring$. Nous pouvons imaginer qu'après chaque communication sur le canal du vecteur d'événements $\overrightarrow{ev}$, une information doit être transmise au module de monitoring en utilisant le canal $api_{put}^{m}$. Cette information est stockée dans le vecteur de données $\overrightarrow{data}$ à travers l'appel récursif du terme $Monitoring$ défini par l'équation (19). Dans cette notation ( $api_{put}^{m}$), le module cible $m$ désigne le monitoring et le service est désigné par $put$.

$$Monitoring(\overrightarrow{api}, \overrightarrow{data}) \stackrel{\text{def}}{=} (\ api_{put}^{m}(datum).\tau.(\nu\ id)\overline{api_{ret}^{m}}\langle id \rangle \tag{19}$$

$$|\ api_{get}^{m}(id).\ api_{res}^{m}(data_{id}))$$

$$.Monitoring(\overrightarrow{api}, \overrightarrow{data}\hat{}datum)$$

## 3.2.2. Gestion du réseau virtuel

Dans notre approche de la Cloudlet, plusieurs entités peuvent utiliser la même infrastructure physique. La virtualisation du réseau simplifie le multi-tenant. L'infrastructure partagée permet l'indépendance des VDRs vis-à-vis de l'hôte physique sur lequel il est situé. La VDR devrait se déplacer entre les hôtes en fonction de la nécessité. Nous définissons notre spécification du networking pour fournir l'accès aux VDRs à travers deux couches matérielles (couche réseau ou L3 et couche liaison de données ou L2 dans le modèle OSI) différentes dans la même couche de domaine. Ces couches s'intéressent aux parcours des données et leurs adressages.

a) Modélisation du Networking

Le modèle du networking virtuel proposé, permet au module d'approvisionnement (Provisioning), voir l'équation (16), de gérer le composant du réseau virtuel comme une VDR et de cacher la complexité à l'utilisateur. Ce modèle permet également de contourner la limite de 4096 VLAN (Virtual Local Area Network) proposée sur le VXLAN (Virtual Extensible LAN) qui est une standardisation proposée par IETF (Internet Engineering Task Force) RFC 7348 [166]. Notre définition du modèle est composée de deux termes : $vSwitch$ défini dans l'équation (22) et $vRouter$ défini dans l'équation (25).

Pour notre modélisation du réseau, nous définissons la structure du paquet qui transite sur les infrastructures du réseau. Le vecteur $\overrightarrow{ethernet}$ dans l'équation (20) représente la trame de la couche liaison de données du modèle OSI (couche liaison de données ou L2) où les noms $ethernet_{dst}$ et $ethernet_{src}$ sont des canaux correspondant au $ws_x$ utilisé par les VDRs dans l'équation (2). Le terme $ethernet_{ip}$ contient les informations nécessaires pour le terme $vRouter$ et aussi les messages comme $ip_{payload}$. Les noms qui appartiennent à des vecteurs dans les équations (20) et (21) sont des abréviations des champs d'entêtes de paquets tels que décrits dans IETF RFC 791.

$$\overrightarrow{ethernet} \stackrel{\text{def}}{=} [dst, src, tag, type, \overrightarrow{ip}, check] \tag{20}$$

$$\overrightarrow{ip} \stackrel{\text{def}}{=} \begin{bmatrix} version, ihl, tos, len, id, flag, frag, ttl, proto, \\ check, src, dst, opt, payload \end{bmatrix} \tag{21}$$

Étant donné que notre objectif n'est pas d'insister sur les protocoles de réseau, mais de souligner les communications entre les composants virtuels, nous avons fait abstraction de tout comportement réseau qui n'est pas directement lié à la virtualisation. Nous les avons nommés comme des opérations non observables $\tau$.

- **Virtualisation du commutateur**

$$\begin{aligned} vSwitch\left(cntl, \overrightarrow{adr}\right) \stackrel{\text{def}}{=} \; & Control\left(vSwitch\left(cntl, \overrightarrow{adr}\right), cntl, \overrightarrow{adr}\right) \\ & \mid adr_i\left(\overrightarrow{ethernet}\right). \tau. \; \overline{ethernet_{dst}} \langle ethernet_{ip_{payload}} \rangle \\ & . vSwitch\left(cntl, \overrightarrow{adr}\right) \end{aligned} \tag{22}$$

Le terme $vSwitch$ défini dans l'équation (22) représente la virtualisation du commutateur de L2. Il est modélisé comme une congruence entre le terme $Control$ défini dans l'équation (23), qui gère les connexions des VDRs et un pont réseau de L2.

Afin de définir les termes des équations (23) et (24), nous utilisons la notation et la définition des entiers introduite par R. Milner [157] par exemple : $\underline{n}(x,z) \stackrel{\text{def}}{=} (\overline{x}.)^n \, \overline{z}$ . Milner définit le successeur sous la forme :

$$(\nu \; xz)\left(\underline{n}(x,z) \middle| Succ(xz, yw)\right) \approx \underline{n+1}(y,w)$$

Les entiers sont utilisés comme des indices dans le but de gérer dynamiquement le contrôle de la connexion / déconnexion des périphériques.

$$Control\left(Target, cntl, \overrightarrow{adr}\right) \overset{\text{def}}{=} cntl_{connect}(link).Target \mid cntl_{disconnect}(link).(\nu\, i\, z)\left(Disconnect\left(Target, \overrightarrow{adr}, (\nu\, \vec{p}), link, \underline{0}(i,z)\right)\right) \quad (23)$$

Le terme $Control$ prend trois paramètres. Le premier est le paramètre d'ordre supérieur appelé $Target$, qui est utilisé pour transmettre les termes $vSwitch$ et $vRouter$. Le second est appelé $cntl$, il est utilisé comme un canal pour contrôler les connexions des VDRs. Le troisième paramètre est le vecteur contenant les canaux des VDRs connectées.

$$Disconnect\left(Target, \overrightarrow{old}, \overrightarrow{adr}, port, \underline{n}(i,z)\right) \overset{\text{def}}{=} \left[\underline{0+n}(i,z) \approx \underline{0+\|\overrightarrow{old}\|}(y,z)\right] \left(\lambda\ \overrightarrow{adr}\right)Target \mid [old_i = port] \; Disconnect\left(Target, \overrightarrow{old}, \overrightarrow{adr}, port, (\nu\, y)\underline{n+1}\,(y,z)\right) \mid Disconnect\left(Target, \overrightarrow{old}, \overrightarrow{new}\hat{\ }port, port, (\nu\, y)\underline{n+1}\,(y,z)\right) \quad (24)$$

Le paramètre $Target$ est paramétré également dans le terme $Disconnect$ défini dans l'équation (24), nous utilisons l'abstraction λ pour remplacer le vecteur d'adresse $\overrightarrow{adr}$ qui reste un nom libre dans $Target$ quand un périphérique est déconnecté.

- **Virtualisation du routeur par le terme *vRouter***

$$vRouter\left(ipAdr, cntl, \overrightarrow{adr}\right) \overset{\text{def}}{=} Control\left(vRouter\left(ipAdr, cntl, \overrightarrow{adr}\right), cntl, \overrightarrow{adr}\right) \mid adr_i\left(\overrightarrow{ethernet}\right).\tau.\left(\left[ipAdr = ethernet_{ip_{dst}}\right] \overline{ethernet_{dst}}\langle\overrightarrow{ethernet}\rangle + \overline{ethernet_{ip_{dst}}}\langle\overrightarrow{ethernet}\rangle\right).vRouter\left(cntl, \overrightarrow{adr}\hat{\ }link\right) \quad (25)$$

Le terme $vRouter$ défini dans l'équation (25) représente la virtualisation du routage au niveau de la couche réseau ou L3. Il est modélisé comme une concurrence entre un processus Control de l'équation (23) qui gère les commutateurs virtuels ou les connexions VDRs et un pont réseau de L3. Dans ce modèle, nous n'illustrons pas certaines fonctionnalités comme le routage IP afin de rester dans notre contexte de définition.

### b) Gestion des infrastructures réseaux

La gestion de l'infrastructure réseau est exposée dans la partie de l'API Provisioning. Pour ce faire, nous illustrons dans l'équation (26) une extension du terme $Provisioning$ défini initialement dans l'équation

(16). Nous ajoutons à cette définition la possibilité de connecter et de déconnecter des périphériques. En outre, cette extension offre la possibilité de créer un nouveau commutateur au niveau de L2.

$$\text{Provisioning}(\overrightarrow{api}) \stackrel{\text{def}}{=} \left( \begin{array}{c} \dots \\ \dots \\ \dots \\ \left| \begin{array}{l} api^{p}_{vsCreate}(ret).(\nu\ cntl) \\ \quad \left( \begin{array}{l} \tau.(\nu\ \overrightarrow{adr})vSwitch\,(cntl, \overrightarrow{adr}\,) \\ \quad |\ \overline{ret}\langle cntl \rangle \end{array} \right) \end{array} \right. \\ |api^{p}_{vsConnect}(cntl, adr).\tau.\,cntl_{connect}(adr) \\ |api^{p}_{vsDisconnect}(cntl, adr).\tau.\,cntl_{disconnect}(adr) \\ \dots \end{array} \right) \tag{26}$$

$$.Provisioning(\overrightarrow{api})$$

Dans l'équation (26), nous décrivons le commutateur lié à l'API Provisioning, le canal $api^{p}_{vsCreate}$ est utilisé pour créer le commutateur virtuel, ensuite retourner le canal de contrôle $cntl$ à l'initiateur de la demande. L'API Provisioning du routeur est comparable à l'API du Switch, la seule différence est que les canaux $api^{p}_{vs*}$ sont définis comme $api^{p}_{vr*}$ et $api^{p}_{vrCreate}$. Ils sont utilisés pour créer un routeur virtuel. Pour conserver le sens de notre définition, nous avons omis cette partie.

Notre modèle formel vise à définir une nouvelle architecture basée sur la notion de Cloudlet. Les définitions obtenues dans ce travail serviront de socle dans le domaine de la standardisation d'une Cloudlet ou d'un réseau de Cloudlets. Elles pourront être utilisées comme un modèle pour prouver les propriétés liées aux systèmes basés sur la Cloudlet. Dans la section suivante, nous exposons notre architecture proposée pour le réseau de Cloudlets et l'approche que nous utilisons afin d'avoir un réseau de haute performance du Cloud mobile.

# IV. Architecture basée sur la Cloudlet

La spécification présentée dans la section précédente répond à l'ensemble des exigences concernant nos besoins en MCC [167] [168]. Elle aboutit à la définition de la Cloudlet basée sur le MCC. Ces Cloudlets peuvent être considérées comme étant un réseau d'ordinateurs fiables et riches en ressources. Elles offrent des capacités de transition vers Internet. Elles sont disponibles par l'utilisation des périphériques mobiles à proximité via une connexion directe et fiable. Dans cette section, nous décrivons notre architecture orientée Cloudlet en illustrant certains aspects techniques. Vue de façon abstraite dans la définition formelle de la section 3.1.2, nous montrons également le lien entre l'implémentation technique du modèle et le modèle formel correspondant. En outre, nous présentons notre contribution à la configuration du modèle et mettons l'accent sur la projection des propriétés ACID (Atomicité, Cohérence, Isolation, Durabilité) du modèle formel au niveau de l'implémentation du modèle.

## 4.1. Aspects techniques du réseau de Cloudlets

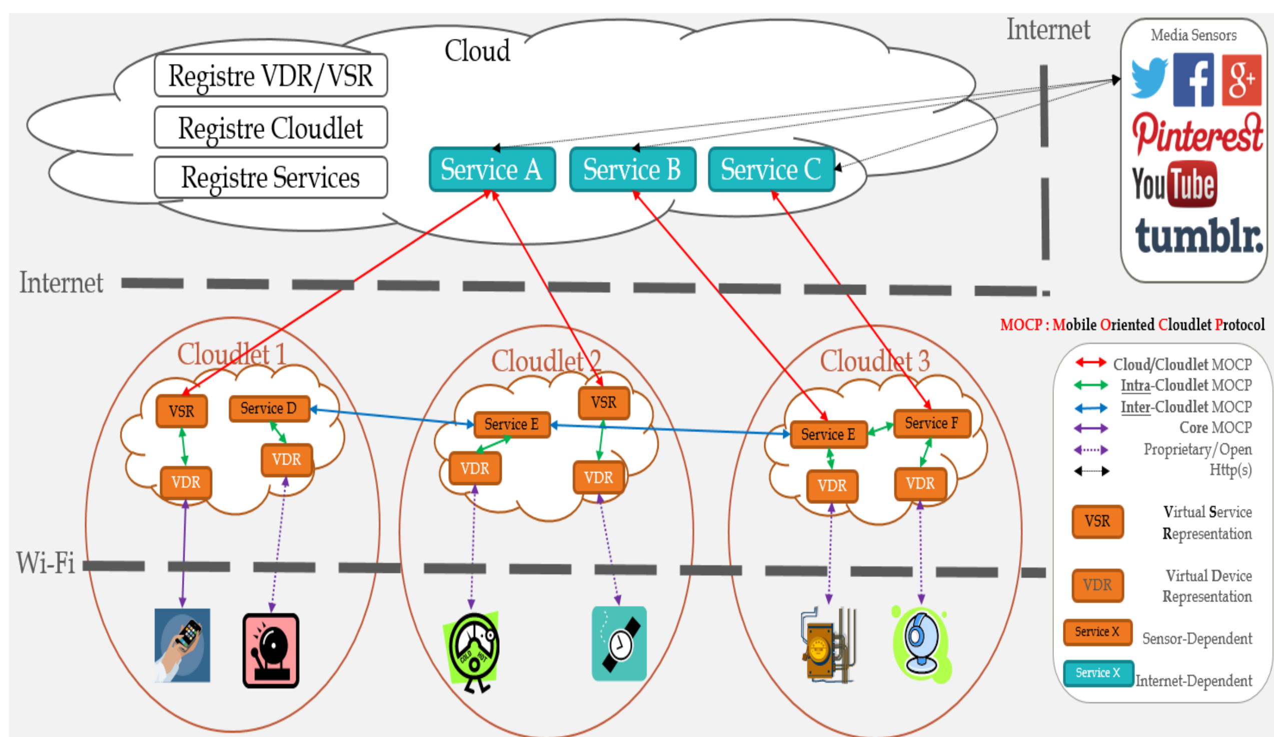
Dans notre approche, nous avons identifié le besoin d'un ensemble de protocoles et de règlements, comme un protocole qui détermine comment les données et les processus sont transmis entre les différents composants du MCC. La Figure 3.5 illustre notre vision du MCC qui est composé de trois couches :

a) La première est la couche du dispositif (DL) composée du capteur physique (Exemple : appareil photo, thermostat, ...) et les périphériques mobiles ;

b) La seconde est la couche de la Cloudlet (CletL) qui est composée à partir du réseau de Cloudlets. Chaque Cloudlet peut contenir des VDRs, des VSRs (Virtual Service Représentation) et des services locaux ;

c) La troisième couche est la couche Internet (IL ou CL) composée par le Cloud central. Le Cloud central contient des services, les registres nécessaires et des services Internet (Par exemple : les capteurs de médias).

*Figure 3.5 Architecture globale*




Dans le cadre de notre vision du système MCC, le protocole MOCP (Mobile Oriented Cloudlet Protocol) est un protocole central qui vise à relier chaque pièce de notre architecture. Les quatre principaux éléments de ce protocole sont illustrés dans la Figure 3.6 et chaque partie est utilisée comme un framework pour la communication entre deux types de composants :

➢ Cloud/Cloudlet MOCP est utilisé pour la communication entre les services du Cloud et les VSRs/services locaux. Il est aussi important pour la gestion des Cloudlets en réseau. Un

annuaire de Cloudlets permet de connaitre une cartographie du réseau pour effectuer le suivi des périphériques ;

- Intra-Cloudlet MOCP est utilisé pour la communication entre les VSRs/services locaux et les VDRs ;
- Inter-Cloudlet MOCP est utilisé pour la communication entre les services locaux des Cloudlets. La gestion des services dans le réseau est assurée par un annuaire de services ;
- Core MOCP est utilisé pour la communication entre les VDRs et les périphériques mobiles.

La définition formelle présentée dans ce mémoire de thèse porte sur les communications utilisées par le Core MOCP (le noyau) pour la migration des applications mobiles (appelées Apps) du périphérique physique au VDR.

## 4.2. Description des composants de notre architecture

Le réseau de Cloudlets permet de mettre en exergue un ensemble de composants interconnectés. Le protocole MOCP comme présenté dans la section 4.1 permet de subdiviser notre architecture en trois niveaux ayant chacun un ensemble de composants ou de ressources.

### 4.2.1. Au niveau des périphériques physiques

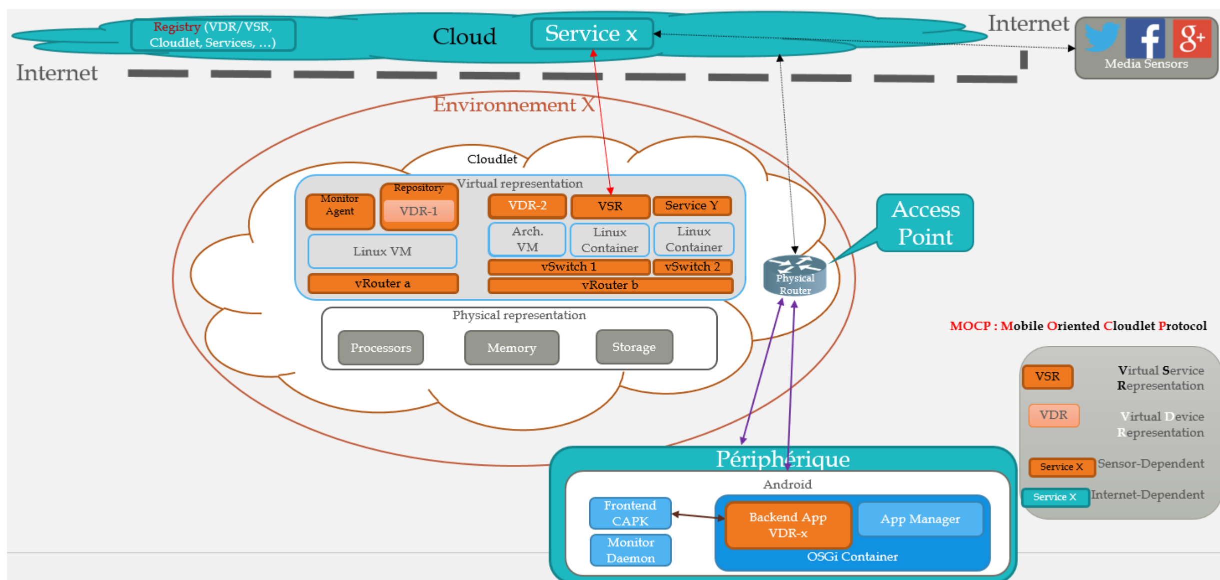
À ce niveau, le protocole MOCP utilise des connexions propriétaires et/ou ouvertes avec le noyau MOCP. La connexion d'un type de périphérique est fonction de trois types de représentation virtuelle (section 3.1). Si le type du périphérique correspond au type de VDR alors il existe une connexion utilisant le MOCP, sinon la connexion est propriétaire ou ouverte. Le périphérique (smartphone, tablettes, capteur, …) est toujours relié à la Cloudlet par sa VDR correspondante. La Figure 2.13 montre de façon détaillée les composants qui se trouvent à l'intérieur d'un périphérique physique et virtuel comme un smartphone. Cette connexion doit être assurée par le périphérique mobile et gérée même lors de ses déplacements. C'est également le cas pour la Figure 2.14 pour les capteurs et actuateurs. Par défaut, nous avons défini une représentation virtuelle composite permettant de représenter tout composant physique dans notre architecture.

### 4.2.2. Au niveau des Cloudlets

Dans la section 4.1, la Figure 3.5 montre l'interconnexion de trois Cloudlets (réseau de Cloudlets). Au niveau de la couche CletL, nous définissons une infrastructure réseau basée sur le NFV. Comme illustré sur la Figure 3.6, le périphérique est relié à la VDR à travers un vRouter défini dans l'équation (25) et un commutateur virtuel défini dans l'équation (22). L'infrastructure réseau est gérée en utilisant l'API

de l’orchestrateur du Cloud. Au chapitre 5, nous utilisons OpenStack [169] qui contient un puissant module de gestion du réseau appelé Neutron. Ce module est basé sur l'Open vSwitch [170]. Cette mise en œuvre fournit une API REST [171] pour la création et la gestion des infrastructures du réseau virtuel fourni. Cela signifie que l’administration d’une Cloudlet peut être effectuée de façon distante via le protocole http. Enfin, cela ouvre la possibilité à de l’autoconfiguration de Cloudlets en réseau par réaction à des événements système par exemple à une panne.

*Figure 3.6 Structure d'une Cloudlet et gestion du réseau*




### 4.2.3. Au niveau du Cloud

Les dispositifs au niveau du Cloud sont fonction des interactions Cloudlet – Cloud. Nous pouvons énumérer les registres du Cloud, registres services, les services du Cloud et les représentations virtuelles de services (VSRs). La Figure 3.6 via sa partie supérieure montre quelques composants du Cloud. Le protocole MOCP Cloud/Cloudlet permet les interactions entre les services fournis par le Cloud et leurs représentations virtuelles au niveau de la Cloudlet. Ils offrent ainsi un cache en cas d’absence de liaison au Cloud. Ces composants sont enregistrés dans un registre de base. En plus des composants propres au Cloud, tout composant de la Cloudlet peut être également au niveau du Cloud.

## 4.3. Projection des propriétés ACID

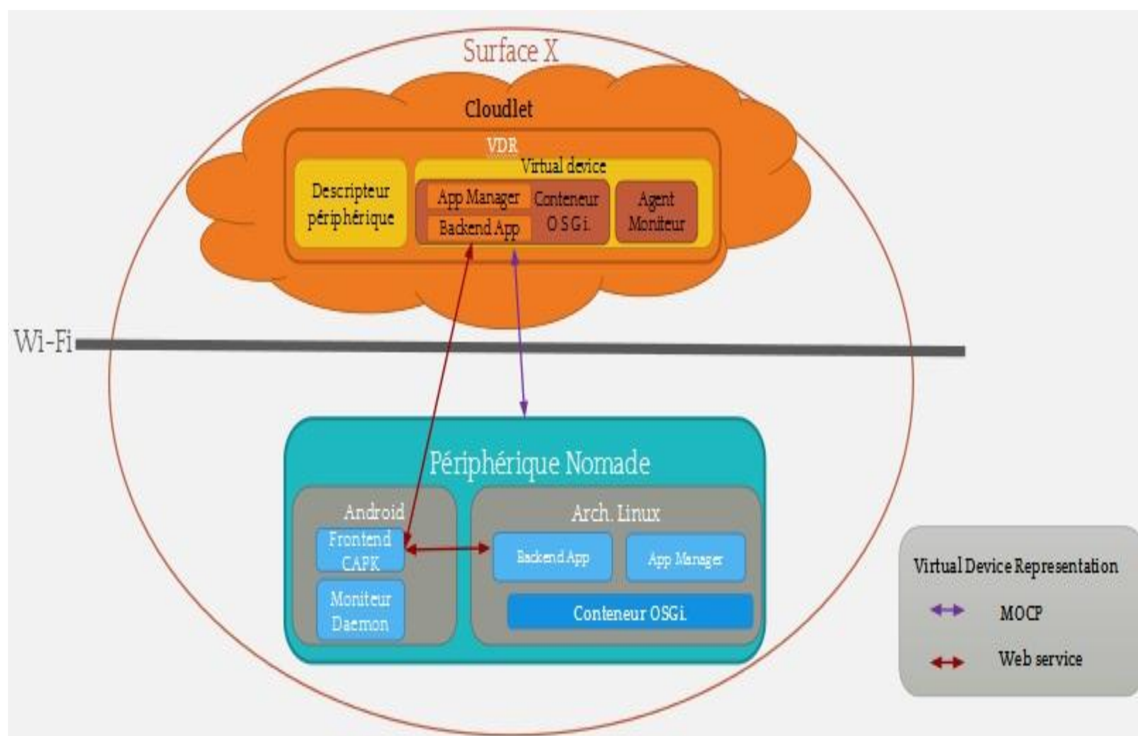
L’implémentation de notre modèle, comme illustrée sur la Figure 3.7, se prolonge dans la définition formelle de l’équation (8) par l'adjonction de détails techniques. Les deux principaux composants de la VDR sont le descripteur du périphérique et le dispositif virtuel. Le descripteur du périphérique est modélisé par le terme $DevId$ dans l’équation (7) et le dispositif virtuel est modélisé dans l’équation (10). L'application backend est modélisée comme un paramètre d'ordre supérieur du terme $BackEndProc$

dans l'équation (9). Les opérations du conteneur App peuvent être considérées comme celles d'un conteneur OSGi (section 3.3). Elles sont considérées comme des opérations non-observables. Notre approche de déportation diffère des approches réelles orientées sur des superpositions [172] qui exécutent des applications de partage automatique par un profilage dynamique. Comme l'illustre la Figure 3.7, cette approche est basée sur un composant appelé Backend App. Ce composant contient la logique métier de l'application mobile. Nous considérons un Backend App comme un service ACID (Atomicité, Cohérence, Isolation et Durabilité) qui peut migrer d'un hôte à un autre en utilisant le terme défini dans l'équation (9).

Notre définition de la DVDR dans l'équation (8) permet, grâce à des propriétés ACID, d'isoler Backend App dans un processus atomique. L'exécution de Backend App impacte de façon durable la cible VDR.

Il apparaît crucial d'imposer une architecture logicielle particulière aux applications mobiles dont un des objectifs est de pouvoir être déportées dans une DVDR. Une telle application doit pouvoir se diviser en un Frontend et un Backend communiquant par l'emploi d'un protocole sans état tel que http.

*Figure 3.7 Modèle d'implémentation de la DVDR*




Ce mécanisme fonctionne avec les services du Backend sans état qui fournissent des réponses à des demandes provenant du Frontend nommé CAPK (Cloudlet Application Android Package), et ne nécessite pas d'autres interactions. Les requêtes subséquentes du Frontend CAPK dépendent du résultat de la première demande. En ce qui concerne le service avec état du Backend, nous nous sommes confrontés à plus de difficultés. Dû au fait qu'une unique action implique plus d'une demande en général. D'où la nécessité d'avoir un autre niveau d'isolement supérieur au conteneur applicatif App. Pour aborder la question de la gestion d'état, nous utilisons en fait un mécanisme proche de la commande

chroot d'ArchLinux qui fournit une couche d'abstraction supplémentaire en utilisant un conteneur applicatif. Celui-ci contrôle les accès réseau et isole son contenu du reste des services.

La section suivante décrit une étude de cas pour une plate-forme MCC faisant apparaître une congruence structurelle entre le système lorsqu'une application mobile est directement en marche sur l'appareil et lorsque cette même application est déportée à une Cloudlet.

# V. Etude de cas

Notre étude de cas vise à établir une similitude des comportements entre un backend applicatif déportée dans une DVDR et le même backend applicatif fonctionnant dans le périphérique mobile. Notre objectif est d'illustrer le fait qu'un backend applicatif (Voir l'équation (9)) qui fonctionne dans une DVDR est identique de façon structurelle, en particulier à la composition parallèle, au backend applicatif qui fonctionne sur un périphérique mobile. Ce résultat est obtenu après la réduction des deux systèmes à un système identique.

## 5.1. Périphérique mobile

Nous définissons d'abord les termes $FrontEnd$ et $BackEnd$. Le terme $FrontEnd$ représente le FrontEnd CAPK et le terme $BackEnd$ représente le Backend applicatif utilisé dans notre étude de cas. Ces termes sont les composants des périphériques mobiles définis dans les équations (29) et (30).

Le terme $FrontEnd$, défini dans l'équation (27), est un modèle d'une "vue Web" qui envoie des messages au $BackEnd$ utilisant le canal $ws$, une fois la réponse reçue par le $BackEnd$ (serveur), le $FrontEnd$ exécute une autre itération de façon récursive. $FrontEnd$ a également le canal $touch$ comme paramètre pour communiquer avec l'utilisateur défini dans l'équation (32).

$$FrontEnd(touch, ws) \stackrel{\text{def}}{=} (\nu\ cb)\ touch(event).\tau.\overline{ws}\langle event, cb\rangle \\ |cb(res).FrontEnd(touch, ws) \tag{27}$$

Le terme $BackEnd$, défini dans l'équation (28), réagit au message envoyé par le $FrontEnd$. Si l'abstraction du canal $intra$ est liée au même canal que le canal $ws$ alors le backend applicatif est exécuté localement sur le périphérique mobile. Sinon, le composant $BackEnd$ envoie un message pour la déportation du backend applicatif vers la DVDR correspondante et termine l'exécution locale. L'exécution du backend applicatif continue dans la DVDR après la déportation.

$$BackEnd(ws) \stackrel{\text{def}}{=} (\lambda\ intra)\ ws(event, cb).\tau.([intra = ws].\overline{intra}\langle\ \ \rangle \\ +\ \overline{ws}\langle App\big((\lambda\ ws)BackEnd(ws)\big)\rangle.\emptyset)\ \big|intra(\ \ ).\tau.(\nu\ res)\overline{cb}\langle res\rangle \tag{28}$$

Nous définissons deux compositions parallèles en tant que modèles pour les périphériques mobiles. Le premier périphérique mobile est défini dans l'équation (29) comme l'exécution en parallèle d'un $FrontEnd$ et d'un $BackEnd$ en local. Le deuxième périphérique mobile est défini dans l'équation (30) comme l'exécution en parallèle d'un $FrontEnd$ et d'un $BackEnd$ qui est configuré pour être déporté dans la DVDR.

$$DeviceLocal(ws, touch) \stackrel{\text{def}}{=} FrontEnd(touch, ws) | (\lambda\, ws) BackEnd(ws) \quad (29)$$

$$DeviceRemote(ws, touch) \stackrel{\text{def}}{=} (\nu\, local)(FrontEnd(touch, local) \; | \; (\lambda\, local) BackEnd(ws)) \quad (30)$$

Pour garder la clarté de notre spécification, nous omettons les détails de la définition du terme $Admin$, nous définissons simplement sa signature dans l'équation (31). Il est important de noter que ce terme envoie tous les messages nécessaires en utilisant le vecteur $\overrightarrow{api}$. Il concerne l'infrastructure du networking et des VDRs.

$$Admin(ws, \overrightarrow{api}) \stackrel{\text{def}}{=} \cdots \quad (31)$$

Le terme $User$ défini dans l'équation (32) représente un utilisateur du périphérique mobile. Il exécute une action unique par l'envoi d'un événement au $FrontEnd$ à travers le canal $touch$ qui représente l'écran tactile du périphérique. Nous avons défini une action simple pour l'utilisateur parce que nous voulons un système qui peut être manuellement réduit par une personne quelconque dans un intervalle de temps raisonnable.

$$User(touch) \stackrel{\text{def}}{=} (\nu\, event)\overline{touch}\langle event\rangle \quad (32)$$

## 5.2. Système

Pour être en mesure de vérifier la congruence structurelle, nous définissons deux systèmes comme une composition parallèle de l'utilisateur mobile, du périphérique mobile, de l'administrateur et de l'orchestrateur. Le terme $SystemMig$ défini dans l'équation (33) représente le système qui conduira à une déportation de $BackEnd$ après quelques réductions.

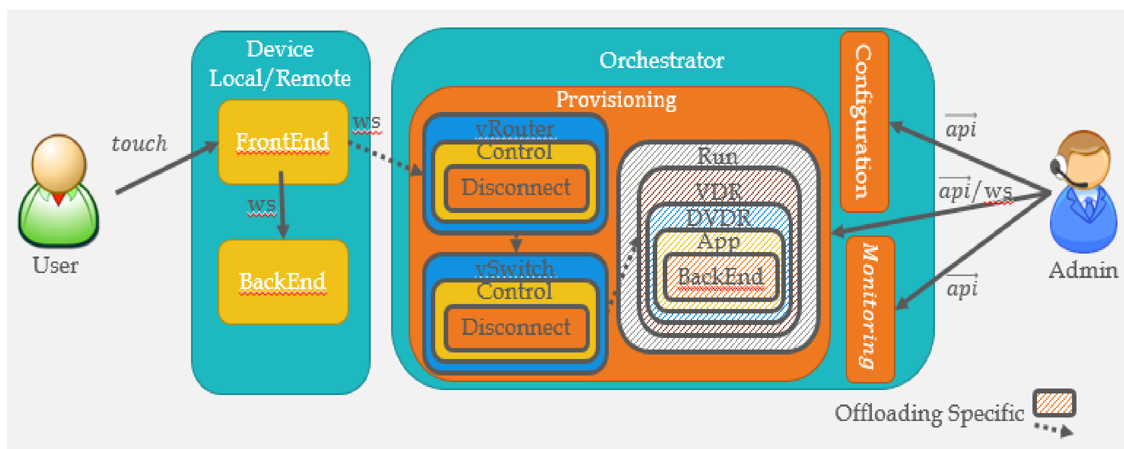
$$SystemMig \stackrel{\text{def}}{=} (\nu\, ws)\begin{pmatrix} (\nu\, touch)\begin{pmatrix} User(touch) \\ |DeviceRemote(ws, touch) \end{pmatrix} \\ \Big| (\nu\, \overrightarrow{api})\begin{pmatrix} Admin(ws, \overrightarrow{api}) \\ |Orchestrator(\overrightarrow{api}) \end{pmatrix} \end{pmatrix} \quad (33)$$

Le terme $SystemLocal$ défini dans l'équation (34) représente le système qui permettra l'élévation d'un $BackEnd$ exécuté localement après une suite de réductions.

$$SystemLocal \stackrel{\text{def}}{=} (\nu\, ws)\left( \begin{array}{c} (\nu\, touch)\left( \begin{array}{c} User(touch) \\ |DeviceLocal(ws, touch) \end{array} \right) \\ \left| (\nu\, \overrightarrow{api})\left( \begin{array}{c} Admin(ws, \overrightarrow{api}) \\ |Orchestrator(\overrightarrow{api}) \end{array} \right) \right. \end{array} \right) \tag{34}$$

Les systèmes (33) et (34) utilisés dans notre étude de cas sont illustrés dans la Figure 3.8 où nous pouvons voir des similitudes entre les deux systèmes. De plus, cette figure illustre les connexions spécifiques de déportation et les composants instanciés. La congruence structurelle que nous soulignons dans la section suivante est basée sur l'hypothèse suivante : après l'exécution des deux systèmes, nous allons trouver deux systèmes réduits $SystemMig'$ et $SystemLocal'$ qui sont structurellement congruents.

*Figure 3.8 Système utilisé*




## 5.3. Congruence structurelle

Nous avons effectué quelques étapes de calculs pour atteindre pleinement un système stable à partir de $SystemMig$. Le calcul commence par l'envoi d'un événement ($event$) tactile via le canal $touch$ ($\overline{touch}\langle event\rangle$) depuis un utilisateur $User$ (32) et se termine par la réception du signal d'arrêt $\overline{ws}\langle Stop\rangle$ sur le périphérique virtuel $VirtualDevice$ (12). Nous appelons $SystemMig'$ cet état stable atteint après ces réductions où :

$$SystemMig \xrightarrow{\overline{touch}\langle event\rangle,\dots} SystemMig' \tag{35}$$

Nous avons appliqué la même opération au terme $SystemLocal$. Toutefois, la réduction de ce système est plus simple, liée au fait de n'avoir aucune réduction de déportation. Ainsi, nous obtenons $SystemLocal'$ :

$$SystemLocal \xrightarrow{\overline{touch}\langle event\rangle,\dots} SystemLocal' \tag{36}$$

La congruence structurelle définie, est une relation de congruence sur un processus respectant les lois suivantes :

Agents (processus) sont identiques s'ils ne diffèrent que par un changement de noms reliés

- $(\mathcal{N}/\equiv,+,\emptyset)$ est un monoïde symétrique
- $(\mathcal{P}/\equiv,|,\emptyset)$ est un monoïde symétrique
- $!P \equiv P \,|\, !P$
- $(\nu\, x)\emptyset \equiv \emptyset, (\nu\, x)(\nu\, y)P \equiv (\nu\, y)(\nu\, x)P$
- si $x \notin fn(P)$ alors $(\nu\, x)(P|Q) \equiv\ P \,|\, (\nu\, x)Q$

Où $fn(P)$ est définie comme l'ensemble des noms libres du processus $P$.

En appliquant le développement des termes $SystemMig$ et $SystemLocal$ qui évoluent en $SystemMig'$ et $SystemLocal'$ suite à l'envoi d'un événement sur le canal *touch* d'un périphérique mobile d'un utilisateur, nous obtenons la dérivation suivante.

$$SystemLocal \xrightarrow{(\nu\, event)\ \overline{touch}\langle event\rangle} (\nu\, ws)\ (\nu\ \text{touch})\ .\, DeviceLocal(ws, touch) \\ |\ (\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ ) \tag{37}$$

Par substitution du terme $DeviceLocal$ de l'équation (29), cela nous conduit à la suivante :

$$\xrightarrow{\nu\ event.\overline{touch}\langle event\rangle} (\nu\, ws)\ \nu\ \text{touch}\ .\,(FrontEnd(touch, ws)|(\lambda\, ws)BackEnd(ws)\ ) \\ |\ (\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ ) \tag{38}$$

Dans le cas d'une exécution locale, le développement et la substitution des termes $FrontEnd$ (27) dans l'équation (39) et $BackEnd$ (28) dans la suite de l'équation (40) ci-dessus donnent :

$$\xrightarrow{(\nu\, event)\ \overline{touch}\langle event\rangle} (\nu\, ws)\ \nu\ \text{touch}\ .\,(\ \big((\nu\, cb)\ touch(event).\tau.\overline{ws}\langle event, cb\rangle\ \big|cb(res)\big) \\ |(\lambda\, ws)BackEnd(ws)) \\ |\ (\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ ) \tag{39}$$

$$\xrightarrow{(\nu\, event)\ \overline{touch}\langle event\rangle} (\nu\, ws)\ (\nu\ \text{touch})\big(\big((\nu\, cb)\ touch(event).\tau.\overline{ws}\langle event, cb\rangle\ \big|cb(res)\big) \\ |((\lambda\, ws)(\lambda\, intra)\ ws(event, cb).\tau.([intra = ws].\overline{intra}\langle\quad\rangle \\ +\ \overline{ws}\langle App\big((\lambda\, ws)BackEnd(ws)\big)\rangle.\emptyset)\ \big|intra(\quad).\tau.(\nu\, res)\overline{cb}\langle res\rangle)) \\ |\ (\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ ) \tag{40}$$

En appliquant une réduction interne par la réception via le canal $touch$, la communication entre l'utilisateur $User$ et la partie $FrontEnd$ du périphérique conduit à la transition labellisée de l'équation (41) :

$$\xrightarrow{(\nu\, x)\ touch(x)} (\nu\, ws)\Big(\big((\nu\, cb).\tau.\overline{ws}\langle x, cb\rangle\ |cb(res)\big)\Big|\{event/x\} \tag{41}$$

$$|((\lambda\, ws)(\lambda\, intra)\ ws(x, cb).\tau.([intra = ws].\overline{intra}\langle\ \ \rangle$$

$$+\overline{ws}\langle App\big((\lambda\, ws)BackEnd(ws)\big)\rangle.\emptyset)$$

$$\Big|intra(\ \ ).\tau.(\nu\, res)\overline{cb}\langle res\rangle))$$

$$|\ (\nu\, \overrightarrow{\overline{api}})(Admin(ws, \overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}})\ )$$

La communication entre le $FrontEnd$ et le $BackEnd$ conduit à une transmission des canaux. La réduction de l'équation (41) par la transition $\overline{ws}\langle x, cb\rangle$ conduit à l'équation (42).

$$\xrightarrow{(\nu\, cb)\ \overline{ws}\langle x,cb\rangle} ((\tau\ |cb(res)\ |\{event/x\}\ |((\lambda\, ws)(\lambda\, intra)\ ws(x, cb).\tau\ ) \tag{42}$$

$$.([intra = ws].\overline{intra}\langle\ \ \rangle$$

$$+\overline{ws}\langle App\big((\lambda\, ws)BackEnd(ws)\big)\rangle.\emptyset)\ \Big|intra(\ \ ).\tau.(\nu\, res)\overline{cb}\langle res\rangle))$$

$$|\ (\nu\, ws)(\nu\, \overrightarrow{\overline{api}})(Admin(ws, \overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}})\ )$$

Le canal $ws$ permet la communication du $FrontEnd$ avec le $BackEnd$. Par réduction et usage des noms liés, la réception via le canal $ws$ conduit à la situation :

$$\xrightarrow{(\nu\, y)\ \mathrm{ws(x,y)}} ((\tau\ |y(res)\ |\{event/x\}|\{cb/y\}|((\lambda\, ws)(\lambda\, intra).\tau) \tag{43}$$

$$.([intra = ws].\overline{intra}\langle\ \ \rangle$$

$$+\ \overline{ws}\langle App\big((\lambda\, ws)BackEnd(ws)\big)\rangle.\emptyset)\ |intra(\ \ ).\tau.(\nu\, res)\bar{y}\langle res\rangle))$$

$$|\ (\nu\, ws)(\nu\, \overrightarrow{\overline{api}})(Admin(ws, \overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}})\ )$$

Comme l'exécution se fait en local (à l'intérieur du périphérique), la condition d'égalité $[intra = ws]$ est vraie et la substitution active $\{intra/ws\}$ est ainsi effectuée. La réduction via la communication à travers le canal $y$ permet d'obtenir l'équation (44).

$$\xrightarrow{(\nu\, res)\ \bar{y}\langle res\rangle} (\tau\ |y(res)\ |\{event/x\}|\{cb/y\}|((\lambda\, intra)\ \tau.\overline{intra}\langle\ \ \rangle)\ |intra(\ \ ).\tau)) \tag{44}$$

$$|\ \{ws/intra\}|(\nu\, intra)(\nu\, \overrightarrow{\overline{api}})(Admin(intra, \overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}})\ )$$

Par réduction et usage des noms liés, la réception via le canal $y$ conduit à la situation :

$$\xrightarrow{y(res)} (\tau\ |\{event/x\}|\{cb/y\}|((\lambda\, intra)\ \tau.\overline{intra}\langle\ \ \rangle)\ |intra(\ \ ).\tau)) \tag{45}$$

$$|\ \{ws/intra\}|(\nu\, intra)(\nu\, \overrightarrow{\overline{api}})(Admin(intra, \overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}})\ )$$

De même, la réduction interne du canal $intra$ donne

$$\xrightarrow{(\lambda\ intra)\ \overline{intra}\langle\ \ \rangle} (\tau\ |\{event/x\}|\{cb/y\}|\ \tau)\ |intra(\ \ ).\tau \tag{46}$$

$$|\{ws/intra\}|(\nu\, intra)(\nu\, \overrightarrow{\overline{api}})(Admin(intra, \overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}})\ )$$

Nous obtenons après réception via le canal $intra$ et réduction, l'équation (47) :

$$\xrightarrow{intra()} \tau |\ (\nu\ intra)(\nu\ \overrightarrow{api})(Admin(intra, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ ) \\ |\ \{ws/intra\}|\{cb/y\}|\{event/y\}|\tau \qquad (47)$$

En définissant le terme $SystemLocal'$ dans l'équation (48), nous obtenons une simplification de l'équation (34) :

$$SystemLocal' \stackrel{\text{def}}{=} \tau |\ (\nu\ intra)(\nu\ \overrightarrow{api})(Admin(intra, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ ) \\ |\ \{ws/intra\}|\{cb/y\}|\{event/y\} \qquad (48)$$

Ainsi, le terme $SystemLocal'$ est obtenu à partir de l'équation du terme $SystemLocal$ en renommant plusieurs noms liés dans l'équation du terme $SystemLocal$, ainsi $SystemLocal \equiv SystemLocal'$.

De même, en appliquant la bisimulation pour le terme $SystemMig$ de l'équation (33), nous obtenons par substitution du terme $User$ l'équation suivante :

$$SystemMig \stackrel{\text{def}}{=} (\nu\ ws)(\ (\nu\ touch)\ ((\nu\ event)\overline{touch}\langle event\rangle \\ |\ DeviceRemote(ws, touch)\ )) \\ |\ (\nu\ \overrightarrow{api})(\ Admin(ws, \overrightarrow{api})\ |Orchestrator(\overrightarrow{api}))) \qquad (49)$$

Dans le cas d'une déportation, la réduction et la substitution du terme $DeviceRemote$ de l'équation (30) dans l'équation (49) donnent :

$$SystemMig \xrightarrow{(\nu\ event)\ \overline{touch}\langle event\rangle} (\nu\ ws)\ ((\nu\ \text{touch})\ .\ DeviceRemonte(ws, touch) \\ |(\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ )) \qquad (50)$$

De cette équation (50), la communication via le canal $touch$ permet en appliquant une réduction d'obtenir l'équation (51) ci-dessous :

$$\xrightarrow{(\nu\ event)\overline{touch}\langle event\rangle} (\nu\ ws)\ ((\nu\ \text{touch})\ .\ (\nu\ local)(FrontEnd(touch, \text{local}) \\ |(\lambda\ local)BackEnd(ws)) \\ |\ (\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ )) \qquad (51)$$

La communication entre $FrontEnd$ et le $BackEnd$ conduit à la réception et l'émission des canaux $event$ et $cb$. En substituant le terme $FrontEnd$ dans l'équation (52), nous obtenons :

$$\xrightarrow{(\nu\ event)\overline{touch}\langle event\rangle} (\nu\ ws)\ ((\nu\ \text{touch})\ .\ (\nu\ local)((\nu\ cb)\ touch(event) \\ .\ \tau.\ \overline{local}\langle event, cb\rangle\ |\ cb(res)|(\lambda\ local)BackEnd(ws)) \\ |\ (\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ )) \qquad (52)$$

La communication entre l'utilisateur $User$ et la partie $FrontEnd$ du périphérique permet la réduction du canal $touch$ ainsi que la réception du canal $event$.

$$\xrightarrow{(\nu\ event)touch(event)} (\nu\ ws)\ ((\nu\ \text{touch})\ (\nu\ local)((\nu\ cb)\ \tau.\overline{local}\langle event, cb\rangle \quad (53)$$
$$|\ cb(res)|(\lambda\ local)BackEnd(ws))$$
$$|\ (\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ ))$$

En substituant le terme $BackEnd$ dans l'équation (53), nous appliquons également une réduction active $\{local/ws\}$.

$$\xrightarrow{(\nu\ event)touch(event)} (\nu\ ws)\ ((\nu\ touch)\ (\nu\ local)((\nu\ cb)\ \tau.\overline{local}\langle event, cb\rangle \quad (54)$$
$$|\ cb(res)\ |\ (\lambda\ intra)\ local(event, cb).\tau$$
$$.([intra = local].\overline{intra}\langle\quad\rangle$$
$$+\ \overline{local}\langle App\big((\lambda\ ws)BackEnd(ws)\big)\rangle.\emptyset)$$
$$|intra(\quad).\tau.(\nu\ res)\overline{cb}\langle res\rangle))\ |\{local/ws\}$$
$$|\ (\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api})\ ))$$

Après réduction du canal $touch$ et réception du canal $event$, l'envoi de ce canal est effectué via le canal $local$. Nous obtenons l'équation (55) suivante

$$\xrightarrow{(\nu\ cb)\overline{local}\langle event,cb\rangle} (\nu\ ws)\ (\ (\nu\ local)(\ \tau\ |cb(res) \quad (55)$$
$$|(\lambda\ intra)\ local(event, cb).\tau.([intra = local].\overline{intra}\langle\quad\rangle$$
$$+\ \overline{local}\langle App\big((\lambda\ ws)BackEnd(ws)\big)\rangle.\emptyset)$$
$$|intra(\quad).\tau.(\nu\ res)\overline{cb}\langle res\rangle)|\{local/ws\}$$
$$|\ (\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api}))\ )$$

Dans ce cas, l'abstraction du canal $intra$ n'est pas liée au même canal que le paramètre $ws$, la partie $BackEnd$ envoie un message pour s'exporter vers la DVDR correspondante. Nous obtenons après une réduction, la suite suivante.

$$\xrightarrow{\text{local}(event,cb)} (\nu\ ws)\ ((\ \tau\ |cb(res)\ |(\lambda\ intra)\tau.(\ \overline{local}\langle App\big((\lambda\ ws)BackEnd(ws)\big)\rangle.\emptyset) \quad (56)$$
$$|intra(\quad).\tau.(\nu\ res)\overline{cb}\langle res\rangle|\{local\ /ws\}$$
$$|\ (\nu\ \overrightarrow{api})(Admin(ws, \overrightarrow{api})|\ Orchestrator(\overrightarrow{api}))\ )$$

La communication via le canal $cb$ permet d'appliquer les règles de réduction successives (émission et réception) sur les équations (57) et (58) :

$$\xrightarrow{(\nu\, res)\overline{cb}\langle res\rangle} (\nu\, ws)\ \Big((\tau\ |cb(res)|(\lambda\ intra)\tau.\,(\ \overline{local}\langle App\big((\lambda\ ws)BackEnd(ws)\big)\rangle.\emptyset\Big)\ |intra(\quad).\tau)\ |\{local/ws\}\ |\ (\nu\ \overrightarrow{\overline{api}})(Admin(ws,\overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}}))\ ) \quad (57)$$

$$\xrightarrow{cb\langle res\rangle} (\nu\, ws)\ (\tau\ |(\lambda\ intra).\tau.\,(\ \overline{local}\langle App\big((\lambda\ ws)BackEnd(ws)\big)\rangle.\emptyset)\ |intra(\quad).\tau)\ |\{local/ws\}|\ (\nu\ \overrightarrow{\overline{api}})(Admin(ws,\overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}}))\ ) \quad (58)$$

La substitution du terme $App$ de l'équation (9) et la déportation donnent :

$$\xrightarrow{cb\langle res\rangle} (\nu\, ws)\ (\tau\ |(\lambda\ intra).\tau.\,\overline{local}\langle BackEnd(ws)\rangle.\emptyset\ |intra(\quad).\tau)\ |\{local/ws\}|\ (\nu\ \overrightarrow{\overline{api}})(Admin(ws,\overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}}))\ ) \quad (59)$$

Le franchissement d'une action interne conduit à la réduction de l'équation (59) comme suit :

$$\xrightarrow{\tau} (\nu\, ws)\ (\tau\ |(\lambda\ intra)\ \overline{local}\langle BackEnd(ws)\rangle.\emptyset\ |intra(\quad).\tau\ |\{local/ws\}|\ (\nu\ \overrightarrow{\overline{api}})(Admin(ws,\overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}}))\ ) \quad (60)$$

Par un changement de nom lié et une réduction active $\{intra/local\}$, nous obtenons l'écriture suivante :

$$\xrightarrow{\lambda\ intra} (\nu\, ws)\ (\tau\ |\ \overline{intra}\langle BackEnd(ws)\rangle.\emptyset\ |intra(\quad).\tau\ |\{intra/local\}\ |\{local/ws\}|\ (\nu\ \overrightarrow{\overline{api}})(Admin(ws,\overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}}))\ ) \quad (61)$$

Ensuite, le franchissement d'une action interne permet d'obtenir une réduction de l'équation (62) nous obtenons :

$$\xrightarrow{\tau} (\nu\, ws)\ (\tau|\{intra/local\}\ |\{local/ws\}\ |\ (\nu\ \overrightarrow{\overline{api}})(Admin(ws,\overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}}))\ ) \quad (62)$$

En définissant ce terme $SystemMig'$, nous obtenons donc l'équation (63) définie comme suit :

$$SystemMig' \overset{\text{def}}{=} (\nu\, ws)\ (\tau|\{intra/local\}\ |\{local/ws\}\ |(\nu\ \overrightarrow{\overline{api}})(Admin(ws,\overrightarrow{\overline{api}})|\ Orchestrator(\overrightarrow{\overline{api}}))\ ) \quad (63)$$

Ainsi, le terme $SystemMig'$ est obtenu à partir du terme $SystemMig$ en renommant une ou plusieurs noms liés à SystemLocal, donc $SystemMig\ \equiv SystemMig'$.

Après la réduction des termes $SystemMig$ et $SystemLocal$, nous avons observé que seuls certains noms liés et les actions non observables composaient la différence entre les deux systèmes réduits. Nous aboutissons à l'équivalence structurelle des deux systèmes :

$$SystemMig'\ \equiv SystemLocal' \quad (64)$$

Par définition, la congruence structurelle est commutative et associative. On peut alors écrire :

Étant donné

$$\begin{aligned} SystemMig &\equiv SystemMig' \\ \text{et}\quad SystemLocal &\equiv SystemLocal' \\ \text{et}\quad SystemMig' &\equiv SystemLocal' \\ \text{Alors}\qquad SystemMig &\equiv SystemLocal \end{aligned} \tag{65}$$

# VI. Bilan

Dans ce chapitre, nous avons présenté une spécification formelle des périphériques virtuels dans une Cloudlet. Une description de notre vision des périphériques virtuels composites a été fournie afin d'élaborer la virtualisation dans un réseau de Cloudlets. Nous avons également fourni un ensemble d'équations constituant notre définition formelle de la représentation des composants d'une Cloudlet, d'un périphérique mobile et d'une représentation virtuelle (SVDR, DVDR et CVDR). Toutes les équations présentées dans ce chapitre, concernent uniquement les interactions entre les composants du réseau de Cloudlets. Cette architecture est dédiée à la réalisation d'une solution MCC. Pour chaque niveau de notre architecture, nous avons énuméré les composants et des services associés. Nous avons défini le protocole de communication MOCP au cœur des interactions entre composants et/ou services durant la migration. Enfin, nous détaillons dans ce chapitre, la projection des propriétés ACID du modèle formel au niveau de l'implémentation du modèle.

Notre étude de cas présente un système distribué en deux parties où la première déclenche la déportation d'une application et la seconde autorise l'exécution locale de la même application. L'évolution d'un tel système est présentée en section V. Cette étude présente une congruence structurelle entre la déportation et l'exécution locale d'une application mobile. Nous aboutissons à la similitude des arbres sémantiques entre le terme local et celui déporté dans un réseau de Cloudlets. L'ensemble des équations fournies dans ce chapitre constitue une définition formelle de référence, que nous utilisons dans la phase de prototypage d'une Cloudlet décrite au chapitre 5.

Dans le chapitre suivant, nous nous intéressons aux comportements temporels d'une Cloudlet au travers d'une modélisation temporelle. Pour cela, à partir de nos spécifications de Cloudlet, d'un périphérique mobile et d'une représentation virtuelle VDR, nous allons construire des automates temporisés. Grâce à l'utilisation d'un outil de model checking, des propriétés temporelles peuvent ainsi être établies. Un grand nombre d'aspects peuvent retenir l'attention telle que la gestion des ressources ou la conservation du contexte d'exécution. Mais l'impact de la mobilité d'application sur un réseau de Cloudlets demeure notre centre d'intérêt tout au long de ce document.

# CHAPITRE 4 : Model-Checking appliqué à un réseau de Cloudlets

L'objectif de ce chapitre est de créer un modèle temporel de notre architecture à base de Cloudlet et de vérifier des propriétés temporelles liées à la migration des Backend applicatifs depuis un périphérique mobile. Pour cela, nous utilisons la logique TCTL* (Timed Computational Tree Logic star). Nous présentons dans un premier temps les apports de cette logique associée à un outil de model-checking. Puis, nous présentons les techniques de spécification en automates et l'outil de vérification. Enfin, nous utilisons la démarche initiée par C. Dumont [173] pour créer un modèle temporel de notre réseau de Cloudlets défini dans le chapitre 3 afin de pouvoir vérifier certaines propriétés temporelles liées à la déportation d'applications mobiles dans une Cloudlet.

## I. Introduction

Aujourd'hui, nous trouvons une grande variété de systèmes réactifs tel un réseau de Cloudlets ou des systèmes embarqués (GPS, contrôle de freinage, commande d'énergie, …). Un réseau de Cloudlet peut être considéré comme un système réactif sous certains aspects, telle que la gestion locale de ressources. Dans notre description, chaque Cloudlet dispose d'une architecture logicielle qui lui assure son autonomie. Elle possède entre autre sa propre gestion temporelle d'événements. Ainsi, la migration d'un périphérique mobile dans une Cloudlet de son voisinage a des effets sur le déroulement de l'application en cours. Il est crucial d'évaluer la portée de ces changements sur les futures exécutions vis-à-vis de l'usage final.

Pour exprimer ces propriétés temporelles, nous utilisons une logique temporelle (TCTL*) dont l'usage est répandu dans notre groupe de travail (Équipe de Spécification et Vérification de Systèmes). Les logiques temporelles sont des formalismes adaptés pour énoncer des propriétés faisant intervenir la notion d'ordonnancement dans le temps, par exemple : Dans un hypermarché, le smartphone d'un usager peut se connecter à une borne de réseau Wi-Fi ouvert du site puis se connecter à une autre borne dont le signal est plus meilleur. Dans cet exemple, les actions s'exécutent suivant un axe de temps, il faut donc une logique qui modélise les expressions du passé et du futur.

Il existe de nombreuses logiques temporelles (CTL[13], LTL[14], TCTL[15], CTL*, …) que nous distinguons principalement en deux classes de logiques :

- Logiques temporelles non-quantitatives comme LTL, CTL, CTL* permettant d'exprimer des propriétés portant sur les arbres d'exécution et sur des chemins individuels (issus de l'état initial) du système ;
- Logiques temporelles temporisées comme TCTL permettant d'exprimer des propriétés portant sur l'ordonnancement temporel des évènements et des contraintes temporelles quantitatives.

Partant de ces deux classes, la logique CTL* ne permet pas d'exprimer des propriétés temporelles quantitatives d'où l'introduction de la logique TCTL. Le model checking est un algorithme utilisé pour faire de la preuve de propriété par l'exploration d'un espace de faits.

Les algorithmes de model-checking sont implémentés par un certain nombre d'outils. Une importante activité de recherche concerne ces outils qui visent à augmenter leur efficacité et à repousser leurs limites. Il existe plusieurs model-checkers temporisés, nous pouvons citer : UPPAAL, ROMEO et TAPAAL. L'outil ROMEO [174] permet la validation et la vérification de systèmes temps réel modélisés par des réseaux de Petri temporels, à chronomètres ou paramétriques. Il a été développé par l'équipe Systèmes Temps Réel de l'IRCCyN de l'Université de Nantes. De même TAPAAL [175] développé au Département d'Informatique de l'Université de Aalborg, permet la modélisation, la simulation et la vérification des réseaux de Petri à arc temporisé. Il fournit un éditeur et un simulateur autonome avec un module de vérification qui transforme les modèles de réseau de Petri à arc chronométré en réseaux d'automates temporisés ayant pour moteur UPPAAL pour l'analyse automatique.

L'outil UPPAAL [176] se démarque des deux autres outils par son interface graphique très conviviale et son évolution constante. Son module de simulation est performant et permet, lors de la phase de modélisation, de faire des tests du modèle pour détecter d'éventuelles erreurs dans la modélisation construite. Il comprend une suite d'outils pour la modélisation, la simulation et la vérification de systèmes réels. Le spécifieur modélise un système réel par un réseau d'automates temporisés et vérifie des propriétés exprimées en logique temporelle arborescente (TCTL). Le format des fichiers utilisés par UPPAAL est le XML.

Dans la suite de ce chapitre, nous introduisons les notions de logiques temporelles puis décrivons la spécification en un système d'automates et sa modélisation (section 2). Nous présentons ensuite notre démarche pour la création d'un modèle temporel et la migration depuis ou vers une VDR dans le modèle

---

[13] CTL (Computational Tree Logic) est le fragment de CTL* restreint de sorte que chaque quantificateur de chemins est associé à un opérateur temporel.

[14] LTL (Linear Tree Logic) est défini uniquement avec les opérateurs temporels.

[15] TCTL (Timed Computational Tree Logic) est une extension de CTL qui intègre des contraintes temporelles quantitatives.

(section 3). Enfin, nous adoptons une démarche de vérification d'exécution locale et d'exécution avec une déportation (section 4).

# II. La spécification formelle en automates

La spécification formelle d'un système en logique temporelle est construite par un système d'automates temporisés. Chacun d'eux fournit une description rigoureuse d'un composant au cours du temps qui respecte des contraintes liées à son contexte d'évaluation. Les travaux passés de Cyril Dumont [174] et Charif Mahmoudi [152] ont montré l'intérêt d'employer une spécification temporelle pour établir les preuves de propriétés au cours du temps. Ainsi, dans les travaux de C. Dumont [174], il prouve des propriétés liées à un système de gestion de cas de calcul numérique sur un jeu de données, tant qu'il existe des tâches à effectuer et des agents « Worker », alors l'état du cas de calcul progresse au sein de l'espace de calcul (voir le chapitre 3 de sa thèse). Quant à Charif Mahmoudi, il établit que le système de gestion d'une orchestration pour la définition d'une route à base d'EIP (Entreprise Integration Pattern) peut s'effectuer sur un nœud ou se poursuivre sur un autre nœud d'évaluation du bus logiciel par migration du contexte d'exécution avec les mêmes résultats.

Basés sur cette expérience, nous faisons le choix de poursuivre notre étude comportementale d'un réseau de Cloudlets. Notre objectif premier est d'établir que le comportement d'une application mobile dont l'architecture logicielle suit nos contraintes présentées au chapitre 3, se comporte de manière similaire pour l'usager final si la partie Backend applicative est migrée dans une Cloudlet voisine.

## 2.1. Motivations

Le chapitre 3 de ce document a fourni une spécification en π-calcul d'ordre supérieur pour un réseau de Cloudlets. Nous avons confronté ce travail à l'épreuve de l'évaluation sémantique de la migration d'un Backend applicatif d'une application mobile dans une Cloudlet. Il est désormais utile de s'intéresser aux évolutions de ce système au cours du temps.

La thèse de C. Dumont [174] a fourni un résultat important pour la construction d'automates temporisés à partir de la spécification π-calcul d'ordre supérieur. Cet ensemble de règles a ensuite été validé au cours des travaux de Charif Mahmoudi [152]. Ces règles ont permis la construction d'automates temporisés pour l'évaluation de patterns d'intégration logicielle qui constituent les éléments de base pour la construction d'orchestration à base d'agents mobiles.

Notre motivation porte sur l'application des règles de construction d'automates pour ensuite rechercher des propriétés temporelles intéressantes à la gestion d'un réseau de Cloudlets. D'un point de vue général, nous souhaitons faire le suivi des applications virtualisées dans une Cloudlet en cas de mobilité de

l'usager final. De plus, la mobilité d'applications entre un périphérique mobile et une Cloudlet est une opération bidirectionnelle. Aussi, une question reste posée sur le retour d'un backend applicatif depuis une Cloudlet vers un périphérique mobile : plus particulièrement les conditions de ce retour.

D'autres motivations sont considérées comme hors de la portée de ce chapitre. Il s'agit par exemple des contraintes de sécurité liées aux périphériques mobiles et aux droits de l'application de retour sur ce périphérique. Doit-on considérer la localisation de départ comme la seule possible localisation de retour ou doit-on envisager que toute localisation utilisée par le même usager puisse être un site d'accueil potentiel pour un retour ?

## 2.2. Systèmes d'automates

Une classe importante de systèmes critiques temps-réel est constituée par les systèmes que l'on appelle temporisés. Ce sont des systèmes équipés d'un ensemble d'horloges qui permettent de calculer et de contrôler le temps écoulé entre les actions. L'absence de temps global à un système distribué est fréquente en informatique distribuée. C'est le cas d'un réseau de Cloudlets et de périphériques mobiles.

### 2.2.1. Gestion du temps

Dans notre contexte de travail, il n'est pas réaliste de considérer que la durée d'un traitement est fixe. Le traitement peut se terminer plus tôt ou plus tard que prévu, sans compter les pannes possibles d'équipements et/ou de périphériques ou encore les pertes de signal provoquant un ralentissement global du système. Deux méthodes de modélisation du temps s'opposent : le temps discret et le temps dense ou continu. Notons que dans cette thèse, nous considérons que le temps s'écoule de façon continue. La notion de temps dense s'oppose à la notion de temps discret dans lequel un grain minimal d'écoulement du temps est défini, c'est-à-dire que rien ne peut se passer dans une période de temps plus petite. Ce point de vue, bien que souvent très proche de la réalité et bien adapté à de nombreux cas, semble une hypothèse forte pour des systèmes distants ne partageant pas d'horloge au sens synchrone.

La sémantique des modèles temporisés s'exprime en termes de systèmes de transitions temporisés (STT) où le domaine de temps, que nous notons $\mathbb{T}$, peut être l'ensemble $\mathbb{N}$ des entiers naturels, l'ensemble $\mathbb{Q} \geq 0$ des rationnels positifs ou nuls, ou l'ensemble $\mathbb{R} \geq 0$ des réels positifs ou nuls. Nous supposons par la suite que le domaine de temps $\mathbb{T}$ est l'ensemble $\mathbb{R} \geq 0$. Nous donnons la définition suivante :

**Définition 4.1 :** Système de transitions temporisé [174]

*Un système de transitions temporisé (STT) est un quadruplet* $\mathcal{T} = (S, s_0, \longrightarrow, \sum)$ *où :* (66)

- ✓ $S$ *est un ensemble d'états de contrôle ;*
- ✓ $s_0$ *est l'état de contrôle initial ;*
- ✓ $\longrightarrow \subseteq S \times (\mathbb{T} \cup \sum) \times S$ *est la relation de transition ;*
- ✓ $\sum$ *est un ensemble d'actions.*

Deux types de transitions sont possibles pour ces systèmes de transitions temporisés :

- Les transitions avec l'action $a$ notées $\overset{a}{\rightarrow}$ , avec $a \in \sum$, correspondent à des actions au sens usuel. Elles sont considérées comme instantanées ;
- les transitions avec l'action $d$, notées $\overset{d}{\rightarrow}$ , avec $d \in \mathbb{T}$, expriment l'écoulement d'une durée $d$. Elles vérifient les conditions particulières suivantes :
  - ***délai nul ou transition vide :*** $s \overset{0}{\rightarrow} s'$ si et seulement si $s' = s$ ;
  - ***additivité :*** si $s \overset{d}{\rightarrow} s'$ et $s' \overset{d'}{\rightarrow} s''$ , alors $s \overset{d+d'}{\longrightarrow} s''$
  - ***déterminisme temporel :*** si $s \overset{d}{\rightarrow} s'$ et $s \overset{d}{\rightarrow} s''$, alors $s' = s''$ ;
  - ***continuité :*** si $s \overset{d}{\rightarrow} s'$ , alors pour toutes actions $d'$ et $d''$ tels que $d = d' + d''$, il existe un état $s''$ tel que $s \overset{d'}{\rightarrow} s'' \overset{d''}{\rightarrow} s'$ .

L'exécution d'un STT est une séquence finie ou infinie de transitions continues et discrètes de $S$. On peut écrire une exécution $\rho$ d'un STT sous la forme suivante :

$$\rho = q_0 \overset{d_0}{\rightarrow} {q'}_0 \overset{a_0}{\rightarrow} q_1 \overset{d_1}{\rightarrow} {q'}_1 \overset{a_1}{\rightarrow} \dots \overset{a_n}{\rightarrow} q_n \overset{d_n}{\rightarrow} {q'}_n$$

### 2.2.2. Automates temporisés

Un automate temporisé est un automate à états finis comportant une représentation du temps continu par l'intermédiaire de variables à valeurs réelles, positives ou nulles, appelées horloges, qui permettent d'exprimer des contraintes temporelles. De façon générale, un automate temporisé est représenté par un graphe où chaque sommet correspond à une location du système et les arcs aux transitions entre ces états. Les contraintes temporelles s'expriment au travers des contraintes d'horloges et peuvent porter sur les états et sur leurs transitions. Chaque automate temporisé possède donc un nombre fini d'états parmi lesquels on distingue un état initial. Dans chaque état, l'écoulement du temps est exprimé par la progression (uniforme) des valeurs d'horloges. Ainsi, dans un état, à tout instant, la valeur d'une horloge x correspond au temps écoulé depuis la dernière remise à zéro de x. A chaque état est associée une contrainte d'horloge, appelée invariant, qui doit être vérifiée pour que le système puisse être dans l'état correspondant. Les transitions entre les états sont instantanées. Elles sont conditionnées par des contraintes d'horloges, appelées gardes, et peuvent remettre certaines horloges à zéro. Elles peuvent aussi porter des étiquettes permettant des synchronisations.

On note par $C(X)$ l'ensemble des contraintes d'horloges sur $X$, c'est également l'ensemble des combinaisons booléennes de contraintes atomiques de la forme $x{\sim}c$ où $x \in X$ est une horloge, $c \in \mathbb{N}$ est une constante et ~ est un opérateur de comparaison ($\sim \in \{=, <, >, \leq, \geq\}$).

Formellement, un automate temporisé est défini comme suit :

**Définition 4.2** : Automate temporisé [174]

*Un automate temporisé $\mathcal{A}$ est un 6-uplet* $\mathcal{A} = (Q, X, q_0, R, Inv, \sum)$ *où :* (67)

- ✓ *$Q$ est un ensemble fini d'états de contrôle ou localités;*

- ✓ *$X$ est un ensemble fini d'horloges;*
- ✓ *$q_0$ est l'état initial ou la localité initiale de l'automate ;*
- ✓ *$T \subseteq Q \times C(X) \times \sum \times 2^X \times Q$ est un ensemble fini de fonction de transitions;*
- ✓ *$Inv: Q \longrightarrow C(X)$ associe un invariant à chaque état ou localité.*
- ✓ *$\sum$ est un ensemble d'actions.*

### 2.2.3. Synchronisation des horloges

Dans tout système d'automates avec des transitions temporisées, l'évaluation d'un tel automate commence par sa configuration initiale : à partir de sa localité ou son état initial $q_0$ avec toutes les horloges à zéro. Ensuite, il effectue successivement des transitions qui peuvent être de deux types sur des localités :

- ➢ Les transitions d'actions qui remettent à zéro certaines horloges, si la valeur des horloges le permet ;
- ➢ Les transitions de temps qui incrémentent toutes les horloges d'une même durée en respectant l'invariant associé à la localité courante.

**Définition 4.3 :** Sémantique des automates temporisés [174]

*La sémantique d'un automate $\mathcal{A} = (Q, X, q_0, T, Inv, \sum)$ est définie par le STT $\mathcal{T}_{\mathcal{A}} = (S, s_0, \longrightarrow, \sum)$ où :* (68)

- ✓ *$S = \{(q, v) \in Q \times \mathbb{R}_{\geq 0}^X \mid v \vDash Inv(q)\}$ est un ensemble fini d'états de contrôle ou localités ;*
- ✓ *$s_0 = (q_0, v_0)$ avec $v_0(x) = 0$ pour tout $x \in X$ ;*
- ✓ *La fonction de transition $\longrightarrow$ correspond à deux types de transitions, à savoir :*
  - *les transitions d'actions définies par : $(q, v) \xrightarrow{a} (q', v')$ si et seulement s'il existe $q \xrightarrow{g,a,r} q' \in T$ tel que $v \vDash g, v' = [r \leftarrow 0]v$ et $v' \vDash Inv(q')$ ;*
  - *les transitions de temps définies par : si $d \in \mathbb{R}_{\geq 0}, (q, v) \xrightarrow{d} (q, v + d)$ si et seulement si $v + d \vDash Inv(q)$.*

Les valeurs des horloges sont modifiées de deux manières :

- ➢ Soit lors d'une transition continue (ou transition de temps). Si un certain délai $d \in \mathbb{T}$ s'écoule, alors les valeurs de toutes les horloges s'incrémentent de $d$. On note par $v + d$ la valuation qui associe à l'horloge $x$ la valeur $(x) + d$. De cette façon, l'automate passe de l'état $(q, v)$ à l'état $(q, v + d)$ ;
- ➢ Soit lors d'une transition discrète (ou transition d'action). Dans ce cas, les mises à jour des horloges sont limitées à des remises à zéro. Pour $r \subseteq X, [r \leftarrow 0]v$ représente la valuation $v'$ définie par : $v'(x) = 0$ pour tout $x \in r$ et $v'(x) = v(x)$ pour $x \in X \backslash r$.

Dans les systèmes d'automates temporisés, tous les automates s'exécutent en parallèle et à la même vitesse. Leurs horloges sont toutes synchronisées sur le temps global unique et le partage d'horloges

entre plusieurs automates du réseau est tout à fait autorisé [174]. On utilise la notation $(\vec{q}, v)$ pour désigner la configuration d'un réseau où $\vec{q}$ est un vecteur de localités et v une fonction associant à chaque horloge du réseau sa valeur à l'instant courant. Le comportement d'un système complexe peut être représenté par un unique automate temporisé qui résulte du produit synchronisé de plusieurs autres (cf. Définition 4.3).

**Définition 4.4 :** Produit synchronisé [174]

*Soient* $\mathcal{A}_1 = (Q_1, X_1, q_{0_1}, T_1, Inv_1, \sum_1)$ *et* $\mathcal{A}_2 = (Q_2, X_2, q_{0_2}, T_2, Inv_2, \sum_2)$ *deux automates temporisés avec* $X_1 \cap X_2 = \emptyset$ *; alors la synchronisation de* $\mathcal{A}_1$ *et* $\mathcal{A}_2$ *est un automate temporisé noté par* $\mathcal{A}_1 \otimes \mathcal{A}_2 = (Q, X, q_0, R, Inv, \sum)$ *où :* (69)

- $Q = Q_1 \times Q_2$
- $X = X_1 \cup X_2$
- $q_0 = (q_{0_1}, q_{0_2})$
- $\sum = \sum_1 \cup \sum_2$
- *Si* $\langle q_1, g_1, a_1, r_1, q'_1 \rangle \in T_1$ *et* $\langle q_2, g_2, a_2, r_2, q'_2 \rangle \in T_2$ *alors T est défini par :*
  - *Si* $a_1 = a_2 = a \in \sum_1 \cup \sum_2$ *alors* $\langle (q_1, q_2), g_1 \wedge g_2, a, r_1 \cup r_2, (q'_1, q'_2) \rangle \in T$
  - *Si* $a_1 \in \sum_1 \backslash \sum_2$ *alors* $\langle (q_1, q_2), g_1, a_1, r_1, (q'_1, q_2) \rangle \in T$
  - *Si* $a_2 \in \sum_2 \backslash \sum_1$ *alors* $\langle (q_1, q_2), g_2, a_2, r_2, (q_1, q'_2) \rangle \in T$
- $\forall (q_1, q_2) \in Q_1 \times Q_2, Inv(q_1, q_2) = Inv_1(q_1) \wedge Inv_2(q_2)$

## 2.3. La logique temporelle

La logique temporelle est une logique modale issue de la logique classique à laquelle ont été intégrées des opérations dédiées au temps, mais également des connecteurs temporels ainsi que des quantificateurs de chemins. Qu'elle soit linéaire ou arborescente, elle permet ainsi d'exprimer des propriétés de correction dont la satisfaction varie au cours de l'exécution du système. Il est par exemple possible de formuler des propriétés telles que « la variable t sera à 0 jusqu'à ce que la variable y soit égale à 0 ». Grâce à la logique temporelle, un raisonnement peut donc être fait dans le temps.

Sachant qu'il existe plusieurs types de modalités, il en découle de plusieurs logiques temporelles. Celles-ci se différencient selon qu'elles sont linéaires ou arborescentes, selon leur expressivité (syntaxique et sémantique), avec ou sans passé et bien sûr selon leur complexité. Parmi elles, les plus couramment utilisées sont la logique temporelle linéaire (LTL) et la logique temporelle arborescente (CTL).

### 2.3.1. Logique temporelle arborescente CTL

La logique CTL (Computation Tree Logic) a été définie au milieu des années 1980. Elle est interprétée sur des structures de Kripke, c'est-à-dire des automates finis dont les états sont étiquetés par des propositions atomiques. Elle permet d'exprimer des propriétés sur ces états.

**Définition 4.5 :** Syntaxe de CTL [174]

*Les formules de CTL sont décrites par la grammaire suivante :* (70)

$$\varphi, \psi ::= P_1 \mid P_2 \mid \cdots \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid EX\varphi \mid EF\varphi \mid EG\varphi \mid E\varphi U\psi \mid AX\varphi \mid AF\varphi \mid AG\varphi \mid A\varphi U\psi$$

*Où $P_i$ sont des propositions atomiques (phrases simples dont on peut déterminer dans un contexte si elles sont vraies ou fausses.) qui parlent des états. Elles donnent pour un état donné une valeur de vérité bien définie.*

En CTL, chaque occurrence d'un combinateur temporel (*U, X, F* ou *G*) doit être immédiatement sous la portée d'un quantificateur de chemin (*A* ou *E*).

Ainsi, par exemple, $E\varphi U\psi$ dénote qu'il existe un chemin partant de l'état courant, et tel que ψ sera vraie dans un état futur, et que tous les états intermédiaires vérifieront $\varphi$. De son coté, $AX\varphi$ dénote que le long de tout chemin partant de l'état courant, le successeur de l'état courant satisfait $\varphi$. En d'autres termes, cela signifie que tous les successeurs de l'état courant satisfont $\varphi$.

$EF\varphi$ dénote qu'il existe un chemin le long duquel $\varphi$ est vérifiée à une certaine position et donc qu'il est possible d'arriver à un état vérifiant $\varphi$. $AF\varphi$ dénote que $\varphi$ est vraie à une certaine position tout le long du chemin, cela signifie que φ est inévitable. $AG\varphi$ signifie que $\varphi$ est toujours vraie pour tout état accessible. Enfin, $EG\varphi$ dénote qu'il existe un chemin le long duquel $\varphi$ est toujours vraie.

Les combinateurs booléens permettent généralement de relier plusieurs sous-formules grâce à la négation ¬, à la conjonction ∧ « et », à la disjonction ∨ « ou » et à l'implication logique ⇒.

### 2.3.2. Logique temporelle linéaire (LTL)

Également appelée logique temporelle de propositions (PTL ou PLTL), la logique LTL est un sous-ensemble de la logique CTL*. Elle permet d'exprimer des propriétés d'exécution en prenant en compte le futur. Il est, par exemple, possible d'exprimer qu'une certaine propriété sera finalement vraie ou bien qu'une condition est vraie jusqu'à ce qu'une autre le devienne. Le domaine d'interprétation d'une formule LTL est un ensemble de chemins tel que chaque chemin est une séquence infinie d'états $s \in S$. L'évaluation de la satisfaction de la propriété se fait donc pour chaque chemin d'exécution indépendamment des autres, sans prendre en compte les entrelacements des différents futurs possibles à un instant donné de l'exécution. Autrement dit, l'évaluation se fait sur un ensemble d'exécutions indépendantes plutôt que sur un arbre des exécutions possibles [177].

**Définition 4.6 :** Syntaxe de LTL [174]

*Une formule de LTL est définie par :* (71)

- ✓ *Un ensemble fini AP de variables de propositions (proportions atomiques)*
- ✓ *Des opérations logiques :* ¬, ∧, ∨, ⇒, ⇔, $vrai$ et $faux$.
- ✓ *Des opérateurs modaux :*
  - ➢ ***X** : neXt (demain) tel que $\boldsymbol{X}_p$ signifie que* $p$ *est vraie dans l'état suivant le long de l'exécution ;*
  - ➢ ***G** : Globally (toujours) tel que $\boldsymbol{G}_p$ signifie que* $p$ *est vraie dans tous les états ;*
  - ➢ ***F** : Finally (un jour) tel que $\boldsymbol{F}_p$ signifie que* $p$ *est vraie plus tard au moins dans un état ;*
  - ➢ ***U** : Until (jusqu'à ce que) tel que $p\boldsymbol{U}q$ signifie que* $p$ *est toujours vraie jusqu'à un état où* $q$ *est vraie ;*
  - ➢ ***R** : Release tel que $p\boldsymbol{R}q$ signifie que* $q$ *est toujours vraie sauf à partir du moment où* $p$ *est vraie, sachant que* $p$ *n'est pas forcément vraie un jour.*

### 2.3.3. CTL* et temporisation de CTL

CTL* est une logique temporelle propositionnelle très générale et expressive qui a pour but d'exprimer des propriétés concernant les exécutions d'un automate $\mathcal{A} = (Q, X, q_0, T, Inv, \sum)$ et dont :

1. les propriétés atomiques sont celles de l'ensemble fini d'horloges $X$ ;
2. les combinateurs logiques sont ceux de la logique propositionnelle : si $\varphi$, $\varphi_1$ et $\varphi_2$ sont des formules, alors $\neg\varphi$, $\varphi_1 \wedge \varphi_2$ et $\varphi_1 \vee \varphi_2$ sont des formules, $\neg\varphi$ étant dite négative, les autres formules étant positives ;
3. les combinateurs temporels permettent de construire des expressions vérifiées dans certains états de l'automate le long d'une exécution donnée ou le long de certaines exécutions de l'automate.

La description d'un système sous la forme d'un réseau d'automates temporisés permet d'énoncer des propriétés avec la logique CTL sur ce système. Ces propriétés sont alors uniquement temporelles et ne mettent pas en jeu les informations quantitatives fournies par les horloges. La possibilité de pouvoir énoncer des propriétés temps-réel devient nécessaire. Pour cela, nous utilisons une logique temporisée qui est une extension d'une logique temporelle par des primitives permettant d'exprimer des conditions sur les durées et les dates. La logique TCTL [178], version temporisée de CTL, fournit ce langage logique pour spécifier des propriétés temporisées adaptées aux systèmes temps-réels.

**Définition 4.7 :** Syntaxe de TCTL [174]

*Les formules de TCTL sont décrites par la grammaire suivante :* (72)

$$\varphi, \psi ::= P_1 \mid P_2 \mid \cdots \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid EX_{(\sim k)}\varphi \mid EF_{(\sim k)}\varphi \mid EG_{(\sim k)}\varphi \mid E\varphi U_{(\sim k)}\psi \mid AX_{(\sim k)}\varphi \mid AF_{(\sim k)}\varphi \mid AG_{(\sim k)}\varphi \mid A\varphi U_{(\sim k)}\psi$$

*Avec* $k \in \mathbb{N}$ *et* $\sim$ *un opérateur de comparaison tel que* $(\sim \in \{=, <, >, \leq, \geq\})$

Il existe différentes familles de propriétés [179] :

- ✓ Les propriétés d'atteignabilité. C'est une propriété qui permet d'énoncer qu'une certaine situation peut être atteinte. En général, il est intéressant d'utiliser surtout la négation d'une telle propriété. Pour exprimer cette propriété en logique temporelle, il est souhaitable d'utiliser le combinateur $EF$ en écrivant $EF\phi$. Ce qui signifie qu'il existe un chemin partant de l'état courant et sur lequel se trouve un état vérifiant $\phi$ ;
- ✓ Les propriétés de sûreté. C'est une propriété qui permet d'énoncer que, sous certaines conditions, quelque chose ne se produira jamais. Dans des cas simples, ces propriétés peuvent être vues comme la négation des propriétés d'atteignabilité. De ce fait, le combinateur $AG$ permet d'exprimer ces propriétés en logique temporelle ;
- ✓ Les propriétés de vivacité. C'est une propriété qui permet d'énoncer que, sous certaines conditions, quelque chose finira par avoir lieu. Cette propriété est plus exigeante que la propriété d'atteignabilité. Malgré cela, cette propriété paraît peu utile, car elle n'apporte aucune information supplémentaire : elle est bien trop abstraite. Les systèmes temporisés apportent la

notion de propriété de vivacité bornée qui énonce un délai maximal avant que la situation souhaitée ne finisse par avoir lieu ;

- ✓ Les propriétés d'équité. Ce type de propriété se rapproche beaucoup du type énoncé pour les propriétés de vivacité. Nous énonçons, dans le cas présent, que sous certaines conditions quelque chose aura lieu (ou pas) une infinité de fois. Le terme « équité » signifie une répartition équitable ou égalitaire.

Nous pouvons signaler une propriété particulière, l'absence de blocage, qui énonce que le système ne se trouve jamais dans une situation où il est impossible de progresser. En logique temporelle, l'absence de blocage s'écrit $AG\ EX\ true$, ce qui signifie : quel que soit l'état atteint, il existera un état successeur immédiat. L'usage d'une telle logique nécessite l'emploi d'un environnement de preuve de propriété, où l'édition, la mise au point et la preuve de propriétés sont assistées.

## 2.4. UPPAAL comme outils de vérification

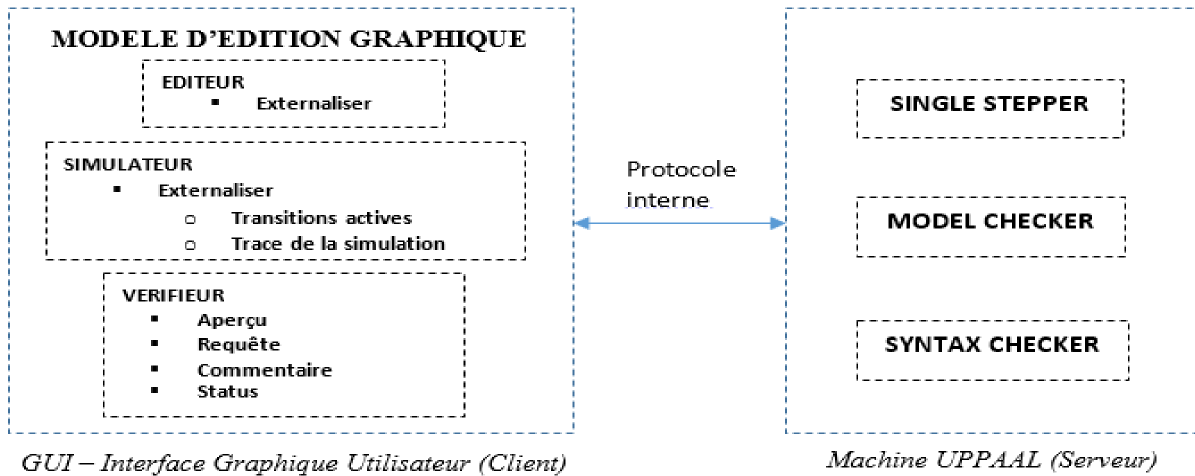
UPPAAL [180] est un logiciel de modélisation et de preuve par model checking appliqué à des automates temporisés. Ce logiciel a été développé par les universités d'Uppsala (Suisse) et d'Aalborg (Danemark). UPPAAL est une suite d'outils pour la modélisation, la simulation et la vérification de systèmes réels. Il permet de modéliser un système réel par un réseau d'automates temporisés et de vérifier que des propriétés exprimées en logique temporelle arborescente (CTL) sont bien respectées. Il fournit également un environnement (graphique) pour modéliser, valider et vérifier les systèmes temps-réels spécifiés comme des réseaux d'automates temporisés étendus par des types de données. Grâce à son interface conviviale, UPPAAL se démarque d'autres outils comme Kronos [181], HYTECH [182] et ROMEO [175]. De plus, son équipe projet reste active et des releases importantes sont disponibles régulièrement, apportant son lot de corrections et de nouvelles fonctionnalités.

L'outil HYTECH développé à l'université de Berkeley par Thomas A. Henziger et Pei-Hsin Ho, vise à vérifier des réseaux d'automates hybrides très généraux. L'outil Kronos, quant à lui, vérifie des automates à l'aide de propriétés exprimées à l'aide de TCTL.

### 2.4.1. Présentation

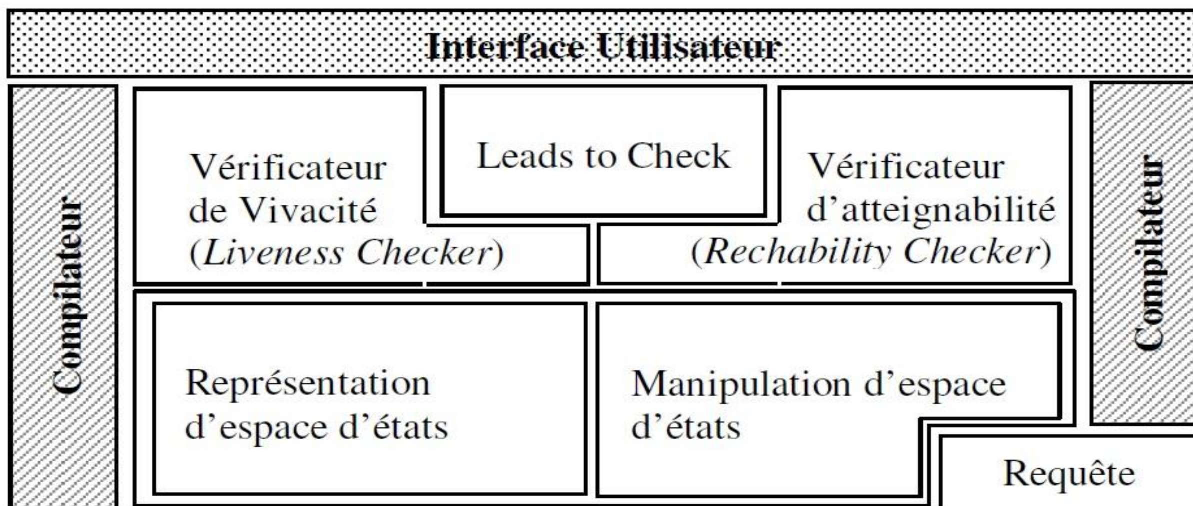
UPPAAL est un ensemble d'outils pour la vérification dynamique et automatique des propriétés formelles comme la sûreté, l'atteignabilité, l'équité et de vivacité bornée, des systèmes temps réel. Son modèle est basé sur une architecture du type client/serveur, où la machine UPPAAL joue le rôle de serveur. UPPAAL est développée en C++ avec une interface coté client en Java. La Figure 4.1 illustre une vue d'ensemble de l'architecture UPPAAL entre le modèle d'édition graphique et la machine UPPAAL via un protocole interne.

*Figure 4.1 Vue d'ensemble d'UPPAAL*




L'interface graphique utilisateur (GUI) est le client, développé en JAVA. La communication s'effectue via un protocole interne basé sur les protocoles TCP et STCP. Ainsi, la conception d'UPPAAL offre la possibilité d'exécuter le serveur et l'interface GUI sur deux machines différentes. La machine UPPAAL est constituée de plusieurs outils tels que checkta (Syntax Checker) et verifyta (Model Checker). L'évolution de l'outil UPPAAL a apporté de nouvelles fonctionnalités, une meilleure rapidité et la flexibilité. La Figure 4.2 illustre l'architecture d'UPPAAL sous forme de modules qui peuvent communiquer entre eux.

*Figure 4.2 Architecture d'UPPAAL*




### 2.4.2. Langage de modélisation

Dans l'outil UPPAAL, un système est décrit par un réseau d'automates temporisés (*TA*) étendus à l'aide de variables entières, de types de données structurés et de la synchronisation de canaux. Lors de la phase de modélisation, la définition du système se décompose en trois parties :

1. La déclaration de variables globales : il est possible de définir des variables globales et des fonctions qui sont accessibles et partagées par tous les TAs. UPPAAL accepte la définition de tableaux de variables, de canaux ou d'horloges ainsi que la définition de nouveaux types. La Figure 4.3 présente un exemple de nos déclarations de variables globales pour un réseau de Cloudlets sous UPPAAL avec la définition d'une horloge $time$, d'un nouveau type Cloudlet (idCL), tel que si $i$ est une variable de type Cloudlet alors $3 \leq i \leq 4$, et d'un tableau de canaux de synchronisation nommé *touch* qui est un tableau à une dimension.

*Figure 4.3 Exemple de déclaration de variables globales pour un réseau de Cloudlets*

Éditeur | Simulateur | Vérifieur

Externaliser

- Projet
  - Déclarations
  - SVDR
  - DevId
  - VirtualSensor
  - DVDR
  - VirtualDevice
  - DeviceLocal
  - DeviceRemote
  - Run
  - Orchestrator
  - Configuration
  - Provisioning
  - Admin
  - CVDR
  - Monitoring
  - User
  - Launcher
  - vRouter
  - CompositeDevice
  - FrontEnd
  - BackEnd
  - Process
  - Composition du système

```
// Place global declarations here.
const int USER = 2;
const int DEVICES = USER;
const int CLOUD = 1;
const int CLOUDLET = 2;
clock time  //temps golobal
typedef int[1, 1+ USER + 1 + CLOUDLET * 2 + CLOUD * 3] agent;
const agent iduser = 1;
typedef int[iduser, iduser] idus;
typedef int[3, CLOUDLET*2] idCL;
const int EVT_SIZE = 20;// vecteur d'evenement
typedef int[0, EVT_SIZE-1] EvtType;
chan evt[idCL][EVT_SIZE];
chan evtvdr[idus][idCL][EVT_SIZE];
chan evt_c[idCL][EVT_SIZE];
const int NB_NAMES = 5;
typedef int[0, NB_NAMES] name;
//gestion de notification et communication
chan touch[idus][name]; //declenchement de la communication
chan notify[idus][name], notify1[idus][idCL][name], endepor[idus][idCL][idCL][name];
chan callback[idus][idCL][name], callback1[idus][name];
chan cChan[idus][idCL][C_SIZE];
chan pChan[idCL][C_SIZE];
```

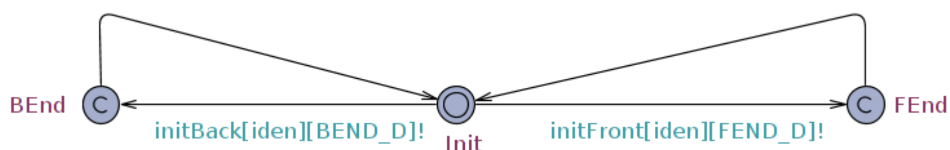
2. La définition de modèles : dans un système UPPAAL, les TAs sont définis comme des instances de modèle. Ces modèles sont indépendants de toute implantation. Dans la définition de ces modèles de TA, il est possible d'ajouter des déclarations de variables locales ou des paramètres. Ces paramètres sont alors considérés comme des variables locales aux instances de TA et sont initialisés lors de la déclaration du système. La Figure 4.4 présente la définition d'un modèle de TA (appelé Template) sous le nouveau format XML d'UPPAAL ;

*Figure 4.4 Modèle d'automate temporisé UPPAAL en XML*

```
<template>
        <name>DeviceLocal</name>
        <parameter>const idus iden</parameter>
        <location id="id27" x="-1700" y="-850">
                <name x="-1760" y="-858">BEnd</name>
                <committed/>
        </location>
        <location id="id28" x="-1207" y="-850">
                <name x="-1190" y="-858">FEnd</name>
                <committed/>
        </location>
        <location id="id29" x="-1471" y="-850">
                <name x="-1479" y="-833">Init</name>
        </location>
        <init ref="id29"/>
        <transition>
                <source ref="id28"/>
                <target ref="id29"/>
                <nail x="-1208" y="-910"/>
        </transition>
        <transition>
                <source ref="id27"/>
                <target ref="id29"/>
                <nail x="-1701" y="-910"/>
        </transition>
        <transition>
                <source ref="id29"/>
                <target ref="id28"/>
                <label kind="synchronisation" x="-1411" y="-841">initFront[iden][FEND_D]!</label>
        </transition>
        <transition>
                <source ref="id29"/>
                <target ref="id27"/>
                <label kind="synchronisation" x="-1666" y="-841">initBack[iden][BEND_D]!</label>
        </transition>
</template>
```

La Figure 4.5 présente l’interface graphique du modèle de TA défini dans la Figure 4.4. Ce TA prend en paramètre une variable du type *idus* qui est *iden* ;

*Figure 4.5 Représentation graphique d'un modèle d'automate temporisé UPPAAL*




3. La déclaration du système : elle consiste en la définition d’un ou plusieurs processus concurrents, chacun modélisé par un TA. Un processus est une instance d’un modèle de TA. L’instanciation d’un modèle d’automate demande de lier une variable à chaque paramètre défini par le modèle. Si le paramètre n’est pas défini par un type borné alors le paramètre doit être lié explicitement. Dans le cas contraire, où le paramètre est d’un type borné, il est possible de laisser UPPAAL lier automatiquement une instance du modèle avec chaque valeur du type défini comme paramètre.

Parmi les différents éléments qui composent un système défini dans UPPAAL, nous listons ceux que nous utilisons dans la suite de ce chapitre :

✓ Les actions de synchronisation non-déterministes. Il est possible de définir des actions de synchronisation non-déterministes via la définition d’un tableau de canaux associée à la

déclaration d'une plage d'identifiants propre à une transition. Dans ce cas, une action de synchronisation se situe dans une plage d'actions liée à la plage définie d'identifiants ;

- ✓ Les localités urgentes. Si une localité est déclarée urgente alors le temps ne peut pas s'écouler dans cette localité. Définir une localité *l* urgente est équivalent à définir une horloge h qui est remise à zéro par toutes les transitions arrivant dans la localité *l* et définir un invariant $x \leq 0$ pour la localité *l*. Par la suite, nous modélisons une localité urgente par un cercle avec la lettre majuscule $U$ ($U$ comme Urgent) ;
- ✓ Les localités engagées. Une localité dite committed est une localité urgente qui impose que la prochaine transition du système soit une transition sortante de cette localité (ou d'une autre localité committed). Ce type de localité permet de modéliser une suite de transitions atomiques qui garantit que toutes les transitions sont effectuées sans être interrompues, ou qu'aucune transition n'est effectuée. Par la suite, nous modélisons une localité committed par un cercle avec la lettre majuscule $C$ ($C$ comme Committed).

### 2.4.3. Modèle formel d'un automate UPPAAL

Un automate temporisé dans l'outil UPPAAL est défini formellement comme suit :

**Définition 4.7 : Automate temporisé UPPAAL**

Les formules de TCTL sont décrites par la grammaire suivante :

*Un automate temporisé $\mathcal{A}$ est un 8-uplet $\mathcal{A} = \langle L, l_0, E, V, C, Assign, Inv, K_L \rangle$ où :* (73)

- ✓ *L est un ensemble fini d'états de localités;*
- ✓ *$l_0 \in L$ est la localité initiale;*
- ✓ *$E \subseteq L \times \varsigma(C,V) \times Sync \times Act \times L$ est un ensemble fini de transitions où $\varsigma(C,V)$ est un ensemble de contraintes autorisées dans les gardes, $Sync$ est un ensemble d'actions de synchronisation ou bien internes, $Act$ est un ensemble d'actions d'affectation et de réinitialisation d'horloges ;*
- ✓ *$V$ est un ensemble de variables ;*
- ✓ *$C$ est un ensemble d'horloges où $C \cap V = \emptyset$ ;*
- ✓ *$Assign \subseteq Act$ est un ensemble d'affectations qui assignent des valeurs initiales à des variables ;*
- ✓ *$L \rightarrow Inv(C,V)$ associe un invariant à chaque localité ;*

*$K_L : L \rightarrow \{o, u, c\}$ affecte un type (**o**rdinary, **u**rgent, **c**ommitted) à chaque localité.*

Le tuple $(l_1, Select, Guards, action, Assign, l_2)$ définit une transition entre les localités $l_1$ et $l_2$ où $Select$ est un ensemble de variables locales à la transition, $Guards$ est un ensemble de gardes, $action$ est une action de synchronisation et $Assign$ est un ensemble d'affectations de variables ou d'appels de fonctions. Un réseau d'automates temporisés UPPAAL est défini ainsi qu'il suit :

**Définition 4.8 : Réseau d'automates temporisés UPPAAL**

Un réseau d'automates temporisés noté $\mathcal{A}$ est un 7-uplet $\langle \vec{A}, \vec{l_0}, V_g, C_g, Ch, K_{Ch}, Assign_g \rangle$ où : (74)

- ✓ $\vec{A} = (\mathcal{A}_1, \dots, \mathcal{A}_n)$ est un vecteur de taille $n$ d'automates temporisés où $\mathcal{A}_i = \langle L_i, l_{0_i}, E_i, V_i, C_i, Assign_i, Inv_i, K_{L_i} \rangle$
- ✓ $\vec{l_0} = (l_{0_1}, \dots, l_{0_n})$ est le vecteur de taille $n$ de localités initiales
- ✓ $V_g$ est un ensemble de variables globales partagées par tous les automates temporisés $\mathcal{A}_i$

- ✓ $C_g$ est un ensemble d'horloges globales partagées par tous les automates temporisés $\mathcal{A}_i$ où $C_g \cap V_g = \emptyset$
- ✓ $Ch$ est un ensemble de canaux utilisés par les automates temporisés $\mathcal{A}_i$ pour communiquer entre eux $C_g \cap Ch = \emptyset$ et $Ch \cap V_g = \emptyset$
- ✓ $K_{Ch} : Ch \rightarrow \{o, u\}$ affecte un type (**o**rdinary ou **u**rgent) à chaque canal
- ✓ $Assign_g$ est un ensemble d'affectations qui assignent des valeurs initiales à des variables globales

### 2.4.4. Vérification avec l'outil UPPAAL

L'outil UPPAAL permet de vérifier des propriétés d'atteignabilité sur les réseaux d'automates temporisés :

- $A.e$ exprime que l'automate $A$ est dans l'état $e$
- $v{\sim}n$ ($v$ désigne une variable) ou $x{\sim}n$ ($x$ désigne une horloge) avec $n \in \mathbb{N}$ et $\sim$ est un symbole parmi la suite de symboles de comparaison $=$, $<$, $>$, $\leq$ et $\geq$.

Le langage de requêtes utilisé pour exprimer les propriétés à vérifier via l'outil UPPAAL est un sous-ensemble du langage TCTL (cf. section 2.3). En considérant que les symboles $\psi$ et $\varphi$ sont deux propositions atomiques, nous listons les différentes formes que peut prendre une propriété qui sera vérifiée par l'outil UPPAAL :

- A[ ]$\varphi$ signifie que pour tous les chemins et pour tous les états, $\varphi$ est valide
- E <> $\varphi$ signifie qu'il existe un chemin où éventuellement, $\varphi$ est valide
- A <> $\varphi$ signifie que pour tous les chemins, éventuellement $\varphi$ est valide
- $E$[ ]$\varphi$ signifie qu'il existe un chemin où pour tous les états, $\varphi$ est valide »
- $\psi -\!\!\rightarrow \varphi$ signifie que $\psi$ mène toujours à $\varphi$

Par exemple, nous pouvons tester un automate (DeviceLocal) issu de notre modèle d'automate proposé à la Figure 4.5 à l'aide des propriétés suivantes :

- $A[\,]\ forall\ (i) DeviceLocal(i).Init$ : l'automate $DeviceLocal$ se trouve toujours dans l'état $Init$ pour $1 \leq i \leq 2$ ;
- $A[\,]DeviceLocal(1).BEnd$ imply $E <> DeviceLocal(1).FEnd$ : le fait que l'automate $DeviceLocal(1)$ soit dans l'état $BEnd$ implique que l'automate $DeviceLocal(1)$ était dans l'état $FEnd$.

Notons, pour finir, la possibilité de vérifier qu'il n'existe pas d'état bloquant dans le système avec la propriété $A[\,]\ not\ deadlock$.

# III. Création d'un modèle temporel

Dans cette section, nous proposons une transformation manuelle d'un système de termes écrits en π-calcul polyadique d'ordre supérieur en un système d'automates temporisés. Dans ce but, nous utilisons les résultats de C. Dumont [174] que nous adaptons à notre réseau de Cloudlets.

## 3.1. Démarche

Dans notre vision du réseau de Cloudlets, nous avons défini un protocole appelé MOCP (Mobile Oriented Cloudlet Protocol), centré sur la notion de Cloudlet qui vise à relier tous les éléments du réseau. Nous retrouvons quatre modules :

- ✓ MOCP Core pour la communication entre les VDRs[16] et les périphériques ;
- ✓ MOCP Inter-Cloudlet qui permet la communication entre des éléments locaux d'une Cloudlet [17];
- ✓ MOCP Intra-Cloudlet qui permet la communication entre des éléments d'une Cloudlet à une autre Cloudlet voisine ;
- ✓ MOCP Cloud/Cloudlet qui permet la communication entre le Cloud et les Cloudlets.

Dans le cadre de notre réseau de Cloudlets, la virtualisation est utilisée pour fournir une architecture matérielle à tous les équipements comme un routeur ou un switch. D'autres contraintes peuvent être ajoutées au cours de cette étude.

Un réseau de Cloudlets représente un système modulable, dont nous avons identifié les éléments au chapitre 3 comme la VDR, la Cloudlet, le périphérique mobile, etc. Une des difficultés de la construction du modèle temporisé réside dans la complexité du système, les nombreuses interactions qui engendrent des comportements du Cloud et des Cloudlets. Ainsi, dans une première approche, il peut sembler être souhaitable de créer de façon dynamique des automates temporisés au cours de la simulation d'automates existants. Connaissant la structure de données représentant un automate dans l'outil UPPAAL, il semble possible de mettre en place une telle approche. En revanche, l'outil n'autorise pas la prise en compte des nouveaux automates en cours de simulation, ce qui interdit tout avenir à une approche dynamique.

Or cette dynamicité est intrinsèque à notre spécification formelle du chapitre 3, où une orchestration est décrite par un terme qui permet l'automatisation des tâches de gestion des Cloudlets, la configuration, la coordination des services et des composants, et leurs interactions dans un réseau de Cloudlets. Le terme Orchestrator permet de ressortir trois principaux composants (Configuration, Provisioning, Monitoring). La simulation de ce terme ne peut donc être faite que si l'automate et ses composants sont déjà présents au départ de la simulation. La construction du système d'automates est donc totalement statique. D'une part, une étude de propriété porte sur l'état des Cloudlets, c'est-à-dire l'interopérabilité et l'adaptabilité. D'autre part, une autre étude de propriété est dédiée à la déportation des applications dans une Cloudlet.

[16] Référence à la définition de VDR du chapitre 3, section 3.1.1.
[17] Référence à l'architecture globale au chapitre 3, figure 3.6.

### 3.1.1. Abstraction depuis la spécification initiale

Initialement, dans un processus de spécification formelle, nous définissons des processus composants notre système. Dans la partie de la déclaration globale au niveau de l'en-tête de notre fichier de spécification, nous avons défini un ensemble de vecteur d'événements et de canaux de communication. Dans notre spécification, nous avons externalisé une vingtaine de templates ou d'automates. Dans cette configuration, les automates User et DeviceLocal permettent le déclenchement de notre modèle dans le simulateur. Chacun de ces modèles déclare un ou deux paramètres typés comme défini pour le modèle d'automate DeviceLocal. Ces paramètres permettent d'identifier le Device et/ou la Cloudlet correspondant. La modélisation d'une Cloudlet est représentée par un ensemble d'automates tel que : Orchestrator, Run, Configuration, Provisioning et Launcher. Les représentations virtuelles du périphérique mobile ou du Device sont définies par les modèles d'automates tels que la SVDR, la DVDR et la CVDR. Ces modèles ont des sous-modèles en commun tel que la DevId.

### 3.1.2. Introduction d'horloges dans le modèle

Dans notre modèle, la notion d'horloges est utile lors de la communication entre les Cloudlets surtout lorsqu'un usager se déplace d'une Cloudlet à une autre. L'absence d'horloge dans la partie locale montre la non synchronisation de certains processus et ne traite pas l'aspect de la virtualisation du réseau. Ce manque de formalisme sur la gestion de la virtualisation du réseau motive aussi notre définition de la virtualisation du réseau au haut niveau dans les sections suivantes.

## 3.2. Construction du modèle

Dans la construction de notre réseau de Cloudlet, en appliquant les opérations de transformation décrites par Cyril Dumont [174], nous obtenons un réseau d'automates dans lequel chaque automate est une instance d'un modèle d'automate. La réduction du nombre de termes composant notre spécification permet de simplifier la lecture du réseau d'automates temporisés résultant de la transformation. En effet, la transformation de tous les termes définis dans le chapitre 3 rendrait la lecture de la suite de ce chapitre laborieuse, car les figures présentées seraient alors plus complexes et de ce fait moins lisibles.

De plus, les limites mémoires du processus UPPAAL sont rapidement atteintes. L'expérience avec cet outil nous a appris à anticiper ce type de problème coûteux en temps. En effet, cette limite apparaît lors de la première simulation, c'est-à-dire lorsque tous les modèles d'automates sont construits, saisis. Ajouter de la mémoire au processus n'est pas une solution, car une limite explicite est fixée dans le code de l'application. Il ne reste donc qu'à reformuler la spécification formelle de sorte que la taille du résultat ait une empreinte mémoire inférieure.

Dans cette section, nous présentons le résultat de la transformation en cinq parties : la structure des données partagées, la déclaration du système UPPAAL et des composants, le modèle de Cloudlet, le modèle d'un périphérique mobile, et enfin le modèle d'une représentation virtuelle VDR.

### 3.2.1. Structures de données partagées

Dans la modélisation de notre réseau de Cloudlets, nous avons défini une structure de données permettant de sauvegarder l'identifiant du périphérique mobile et le type de périphérique. Certains canaux de communication ont été définis pour les interactions entre processus. Le canal *touch*[idus][name] représente l'écran du périphérique mobile identifié par idus et permet l'interaction entre l'usager et ce périphérique mobile. Certains canaux admettent pour paramètre un identifiant de Cloudlet. L'abstraction des infrastructures de communication de services Web est gérée par les canaux *evt*[idCL][EVT_SIZE], *ws*[idCL][WS_SIZE], *P*[WS_SIZE] et *api*[idCL][API_SIZE][API_SIZE]. La Figure 4.6 illustre les variables globales ainsi que les structures de données utilisées dans le modèle de notre réseau de Cloudlets.

*Figure 4.6 Déclaration de variables globales*

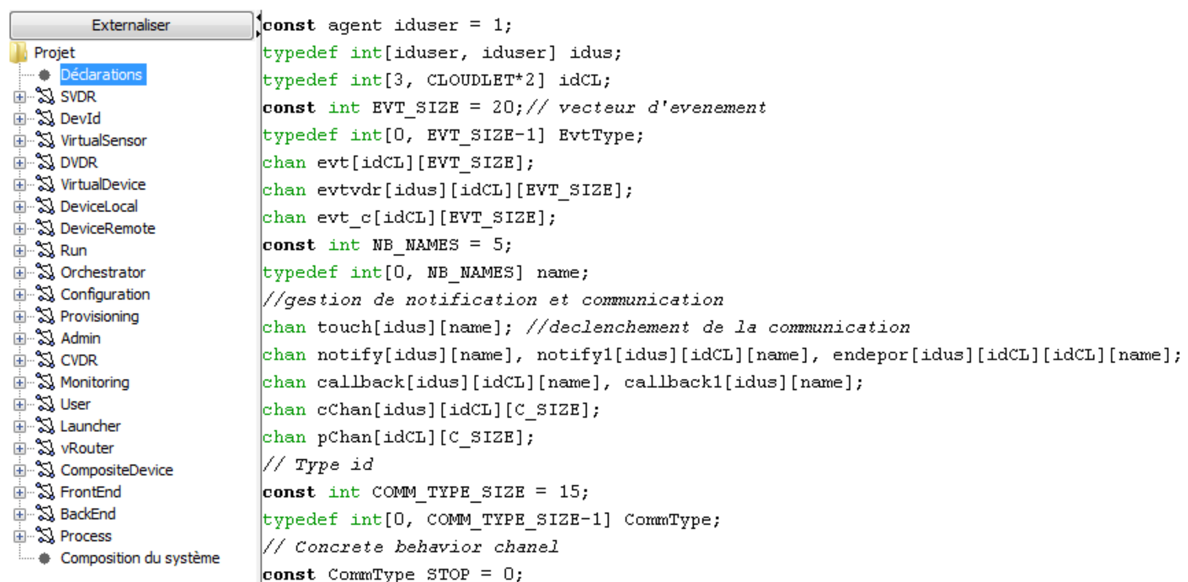
```
chan evt[idCL][EVT_SIZE];
chan evtvdr[idus][idCL][EVT_SIZE];
chan evt_c[idCL][EVT_SIZE];
name data, res;
idus iden;

chan stop[idus][idCL][name], stop1[idus][name]; // canal de terminaison pour l'arret
chan migrate[idus][idCL][idCL]; //canal pour la migration vers une autre Cloudlet
//gestion de notification et communication
chan notify[idus][name], notify1[idus][idCL][name], endepor[idus][idCL][idCL][name], callback[idus][idCL][name]
chan cChan[idus][idCL][C_SIZE];
chan pChan[idCL][C_SIZE];
chan touch[idus][name]; //declenchement de la communication
chan devchan[idus][DEV_SIZE][DEV_SIZE];//chan intra[DEV_SIZE][DEV_SIZE];
chan deporchan[idus][idCL][DEV_SIZE];
chan virDevChan[idus][idCL][VIR_DEV_SIZE];// Virtual device
chan devIdchan[idus][idCL][VIR_DEV_SIZE];
chan ws[idCL][WS_SIZE]; // web service (ws)
chan P[WS_SIZE];
chan cb[idCL][name];

chan initUser[idus][USERS];
chan initFront[idus][USERS];
chan initBack[idus][USERS];
chan initOrChan[ORCH];
chan alloc[idCL][name];//type de service de configuratiion
chan terminate[idCL][name];
chan initRun[idus][idCL][exe];
// les différents services
ApiConf all, free, susp;
ApiProv su, ter, run;
ApiMoni ret, get, put;
chan api[idCL][API_SIZE][API_SIZE];
chan initApiChan[idCL][API_SIZE];
```

### 3.2.2. Déclaration du système de composants

Le module de construction représente le premier sous-système entre les quatre sous-systèmes que nous décrivons dans notre réseau de Cloudlets. Le deuxième concerne le modèle de la Cloudlet. Il comprend des composants d'orchestration, d'approvisionnement, de configuration et l'abstraction des fonctions de virtualisation du réseau et est décrit à la sous-section 3.2.2. Le troisième concerne le modèle d'un périphérique mobile composé d'un composant frontal et d'un composant arrière-plan (backend). Elle correspond à la sous-section 3.2.4. Le quatrième concerne le modèle d'une représentation virtuelle VDR composé de SVDR pour les capteurs, de DVDR pour les périphériques et la CVDR composée, est décrite dans la sous-section 3.2.5. La Figure 4.7 illustre la déclaration du système de composants pour notre réseau de Cloudlets.

La déclaration du système de composants consiste en une instanciation des modèles intervenants dans la construction du réseau de la Cloudlet. Par exemple, les modèles VirtualDevice et BackEnd utilisent respectivement des canaux *notify* et *notify1* pour signaler une exécution distante et un retour du backend applicatif dans le Device où réside le FrontEnd. Dans les sous-sections suivantes, nous allons définir chaque composant intervenant dans notre réseau de Cloudlets.

*Figure 4.7 Déclaration du système et de ses composants*




### 3.2.3. Modèle de Cloudlet

Le modèle temporel de Cloudlet est construit à partir de sa définition donnée en section 4 du chapitre 3. Notre modèle est une composition parallèle de plusieurs autres composants. Cette modélisation consiste en une instanciation des automates intervenants dans la Cloudlet. Pour simplifier la lecture de notre modèle de Cloudlet, nous supprimons les termes modélisant les propriétés et l'accès aux équipements réseaux. La modélisation de l'orchestrateur est au cœur de la Cloudlet et vise à modéliser les tâches de

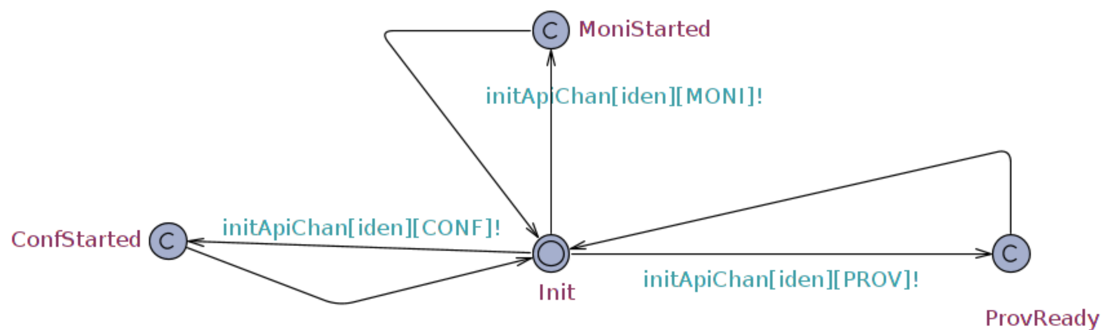
configuration, d'approvisionnement et de monitoring. Nous allons décrire dans la suite les composants ainsi que les automates associés.

a) Composant Orchestration

L'équation (14) décrite dans la section 3.2 du chapitre 3 définit un orchestrateur comme étant un système composé de trois termes. Son objectif est d'automatiser la configuration, la coordination, la gestion des VDRs et leurs interactions dans un tel environnement. La Figure 4.8 présente le résultat de la π-transformation du terme Orchestrator de l'équation (14). Le modèle d'automate « Orchestrator » est composé de quatre états et trois principales transitions :

- ✓ L'état Init représente l'état initial, cet état permet l'activation des différents composants de l'orchestrateur ;
- ✓ L'état ProvReady est atteint suite à l'activation du composant Provisioning sur le canal initApiChan. Le paramètre iden permet d'instancier ce composant ;
- ✓ L'état ConfStarted est atteint suite à l'activation du composant Configuration sur le canal initApiChan ayant iden comme paramètre d'instanciation ;
- ✓ L'état MoniStarted est atteint suite à l'activation du composant Monitoring sur le canal initApiChan. Le paramètre iden permet d'instancier ce composant.

*Figure 4.8 Modèle d'automate Orchestrator*




Ce modèle est évalué plusieurs fois par la simulation du système. Il communique durant cette simulation sur le canal *initApiChan* afin de dépiler sa définition. Il fait appel à d'autres composants (Provisioning, Configuration et Monitoring) par l'émission sur le canal *initApiChan.* Les états de ce modèle sont définis comme committed, cela veut dire que l'application du Template Orchestrator est une étape atomique où l'horloge n'avance pas. Cela suppose que les états de ce modèle sont urgents et les émissions sur les canaux *initApiChan* sont prioritaires par rapport aux autres émissions effectuées. Le paramètre *iden* du modèle d'automate Orchestrator est de type idCL et prend ses valeurs dans l'intervalle [3, 4] (nous avons restreint notre simulation sur un réseau de deux Cloudlets).

b) Composant Provisioning

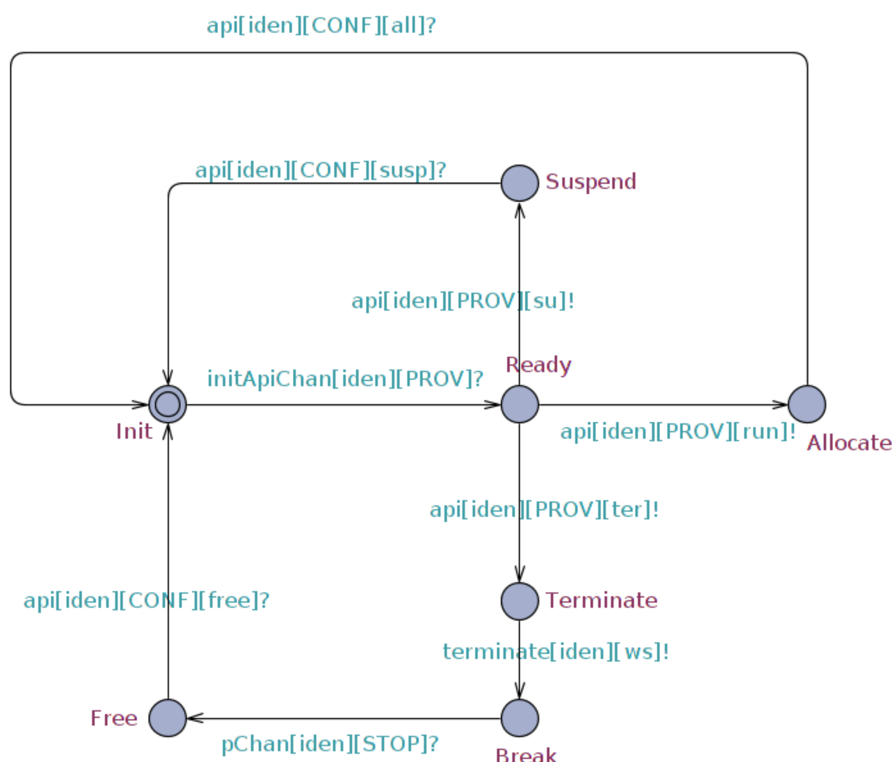
La π-transformation du terme Provisioning donne le modèle d'automate Provisioning (cf. Figure 4.9) utilisant le canal *api* en fonction de la demande du module de configuration et l'exécution de la création

de la VDR. Les différents états de cet automate désignent les services de l'orchestrateur. Le modèle d'automate « Provisioning » est composé de sept états et huit principales transitions :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Ready marque l'activation du composant Provisioning à partir du composant Orchestrator à travers le canal *initApiChan*. Cet état permet l'activation des différents services tels que la demande d'allocation, l'arrêt ou la suspension de l'exécution de la VDR et bien d'autres ;
- ✓ L'état Suspend est atteint suite à une demande de suspension sur le canal *api*. Cette demande de suspension est prise en compte par le composant Configuration ;
- ✓ L'état Allocate marque l'activation du démarrage du processus Run et une demande d'allocation de ressources auprès du composant Configuration via le canal *api* ;
- ✓ L'état Terminate marque le début d'un processus de terminaison qui commence par une demande via le canal *api* ;
- ✓ L'état Break est atteint pour chaque transfert du canal *ws* à travers le canal *terminate* pour l'arrêt d'une exécution ;
- ✓ L'état Free marque la fin d'une exécution de la VDR par la réception du terme *STOP* grâce au canal *pChan* et une demande de libération de ressources allouées ;

La transition entre les états Ready et Allocate permet d'envoyer un paramètre d'ordre supérieur au composant Launcher pour la création de la VDR. Ce paramètre est une instance du terme Run qui est cependant activé à travers le composant Launcher.

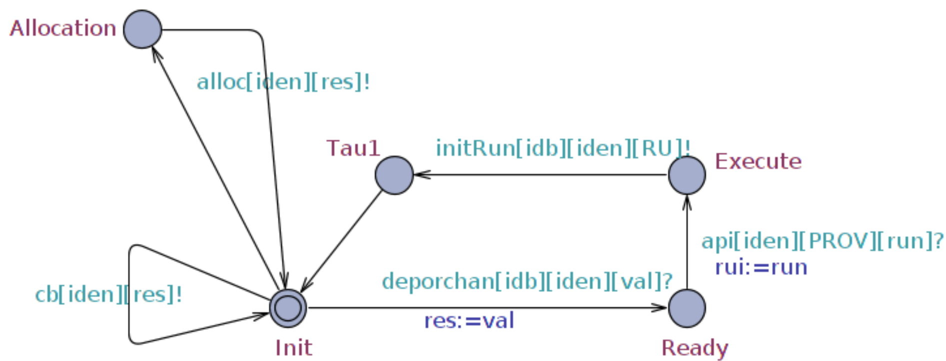
*Figure 4.9 Modèle d'automate Provisioning*



Le modèle d'automate Launcher (Figure 4.10) est construit pour permettre la déportation du backend applicatif au niveau de la Cloudlet. Son objectif est de recevoir une demande de migration et de création de la VDR associée, puis d'activer le composant Run pour la création. Le modèle d'automate « Launcher » est composé de cinq états et cinq principales transitions :

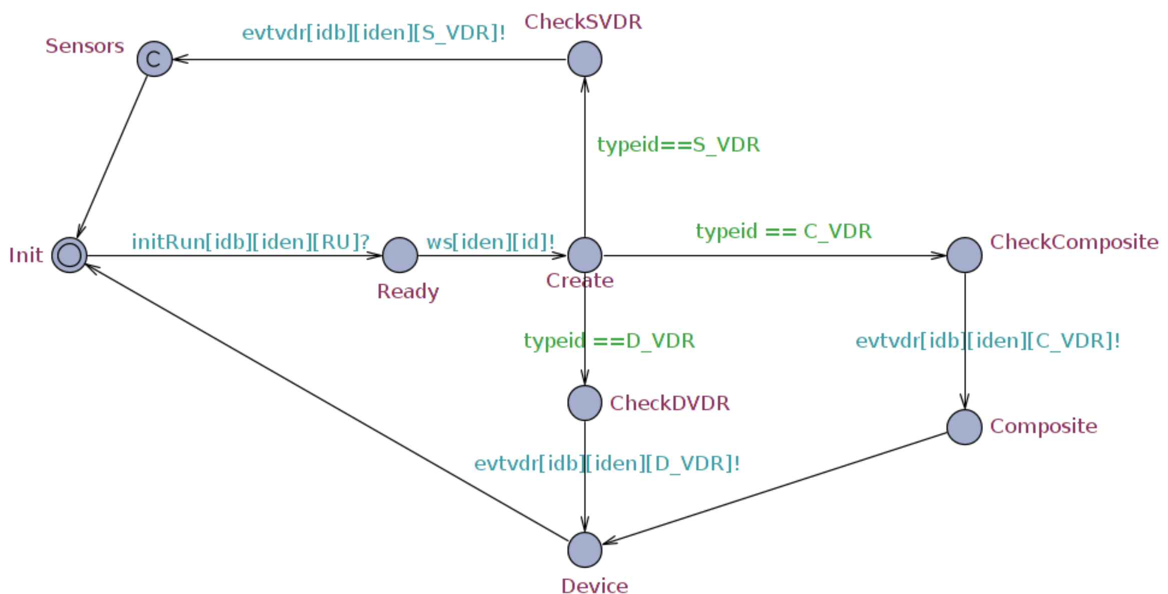
- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Ready marque l'activation du composant Launcher à partir du composant BackEnd à travers le canal *deporchan* et la déportation du backend applicatif au niveau de la Cloudlet ;
- ✓ L'état Execute est atteint par une demande d'exécution du composant Run venant du composant Provisioning ;
- ✓ L'état Tau1 est atteint suite à l'activation du composant Run sur le canal *initRun*. Il représente l'activation du backend et la création de la VDR associée ;
- ✓ L'état Allocation est atteint suite à une demande d'allocation de ressource de la part du composant Configuration.

*Figure 4.10 Modèle de l'automate Launcher*




La Figure 4.11 présente le résultat de la π-transformation du terme Run défini par l'équation (17)

*Figure 4.11 Modèle d'automate Run*



Le modèle d'automate « Run » est composé de neuf états et huit principales transitions :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Ready marque l'activation du composant Run à partir du composant Launcher après une réception sur le canal *initRun* ayant l'identifiant (iden) de la Cloudlet concernée comme paramètre ;
- ✓ L'état Create est atteint suite à l'émission de l'identifiant (id) de la VDR sur le canal *ws* ;
- ✓ Les états CheckSVDR, CheckComposite et CheckDVDR représentent les états marquants le type de VDR à créer en fonction du type demandé de Device ;
- ✓ L'état Sensors est atteint suite à une demande de création de la SVDR. Cet état marque également l'activation du composant SVDR suite à l'émission sur le canal *evtvdr* ;
- ✓ L'état Composite est atteint suite à une demande de création de la CVDR. Cet état marque également l'activation du composant Composite suite à l'émission sur le canal *evtvdr* ;
- ✓ L'état Device est atteint suite à la demande de création de la DVDR. Cet état marque également l'activation du composant DVDR suite à l'émission sur le canal *evtvdr*.

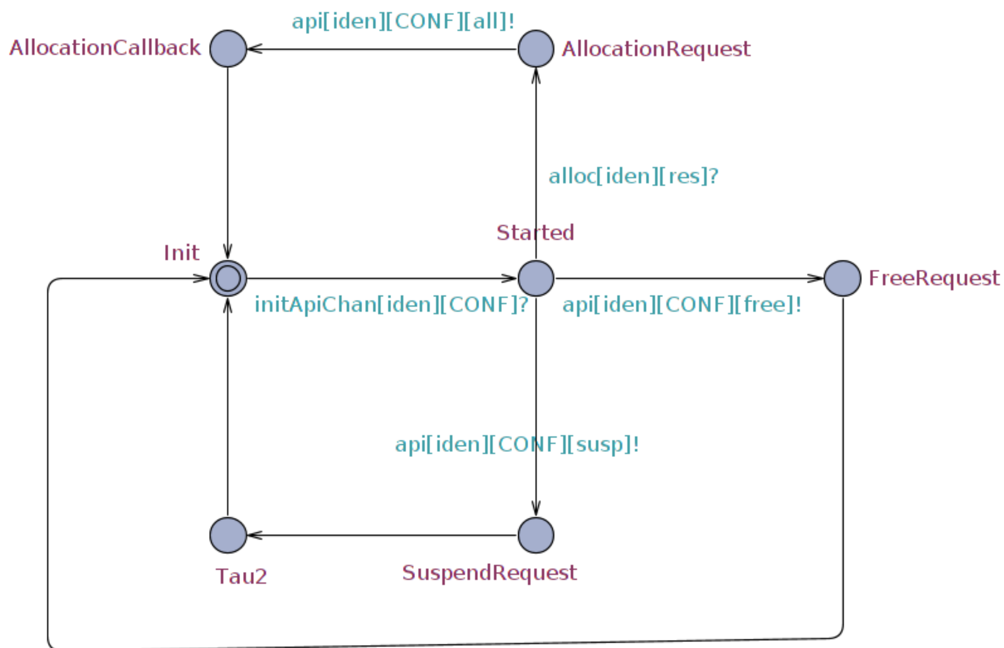
c) Composant de Configuration

La π-transformation du terme Configuration défini à l'équation (15) donne le modèle d'automate Configuration (cf. Figure 4.12). Ses états sont atteints suite à des demandes d'allocation, de libération et de suspension de l'exécution des ressources émanant d'autres composants à travers le canal *api*.

Ce modèle est composé de sept états et cinq principales transitions :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Started marque l'activation du composant Configuration à partir du composant Orchestrator après une réception sur le canal *initApiChan*. Cet état permet l'activation des différents services configurés par l'administrateur ;
- ✓ L'état FreeRequest est atteint suite à une libération de ressources après une émission sur le canal *api* à destination de l'automate Provisioning ;
- ✓ L'état SuspendRequest est atteint suite à une suspension de l'exécution de ressources après une réception sur le canal *api* venant de l'automate Provisioning ;
- ✓ L'état AllocationRequest est atteint après une réception d'une demande d'allocation sur le canal *alloc* venant de l'automate Provisioning ;
- ✓ L'état AllocationCallback marque l'allocation de ressources suite à une émission sur le canal *api* à destination de l'automate Provisioning.

*Figure 4.12 Modèle d'automate Configuration*

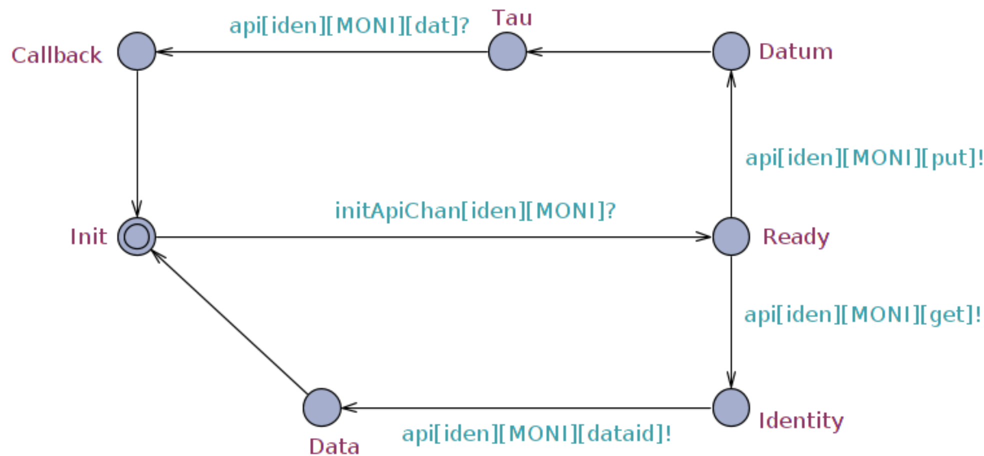



d) Composant Monitoring

La $\pi$-transformation du terme Monitoring défini à l'équation (19) donne le modèle d'automate Monitoring (cf. Figure 4.13). Ce modèle est composé de sept états et cinq principales transitions :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Ready marque l'activation du composant Monitoring à partir du composant Orchestrator après une réception sur le canal *initApiChan* ;
- ✓ L'état Identity est atteint suite à une émission sur le canal *api* concernant une demande d'information sur l'identifiant du backend applicatif ;
- ✓ L'état Datum est atteint suite à une émission sur le canal *api* concernant une transmission de l'identifiant du backend applicatif ;
- ✓ L'état Data est atteint suite à l'émission de l'identifiant de l'information sur le canal *api* ;
- ✓ L'état Callback est atteint après une réception d'une information sur le canal *api* venant de l'automate Administrateur.

*Figure 4.13 Modèle d'automate Monitoring*




e) Abstraction des fonctions de virtualisation réseau

Comme nous avons mentionné au chapitre 3, section 3.2.2, plusieurs entités peuvent utiliser la même infrastructure physique. Pour respecter les limitations de l'usage mémoire d'UPPAAL nous n'avons pas instancié la gestion de l'infrastructure réseau exposée dans le modèle d'automate Provisioning (voir l'équation 26). Nous avons fait abstraction des composants du réseau virtuel. Cette limitation a pour but de limiter la taille des automates construits dans cette section et de conserver le sens de notre définition. Les termes vSwitch, vRouter, Control et Disconnect n'ont pas été pris en compte dans nos différents modèles d'automates temporisés.

### 3.2.4. Modèle d'un périphérique mobile

Dans notre modèle d'un périphérique mobile, deux termes (FrontEnd et le BackEnd) sont utilisés pour définir son architecture logicielle. Ils représentent respectivement un modèle d'une vue Web et le Backend applicatif. Nous décrivons dans les sous-sections suivantes les automates utilisés dans la définition du périphérique mobile.

a) Composant frontal

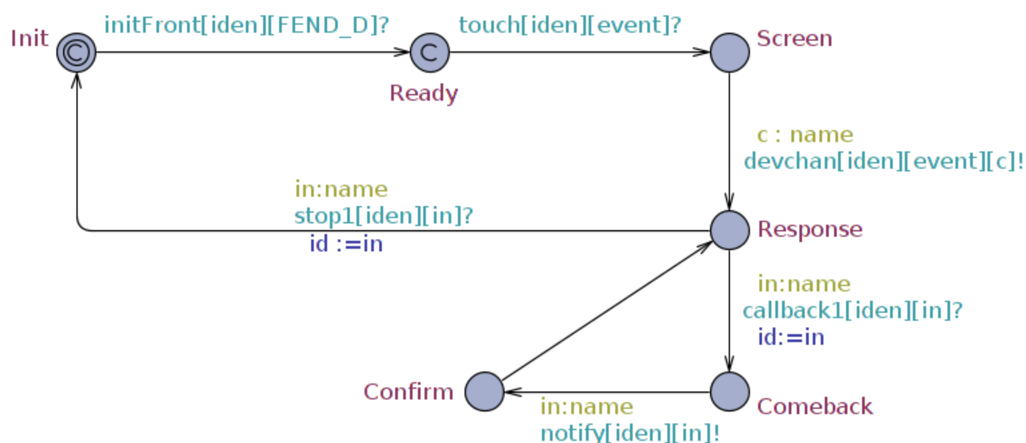
Le composant frontal est représenté par le terme FrontEnd (cf. Equation 27) ayant comme paramètres les canaux *touch* et *devchan* pour la communication avec l'utilisateur et avec le BackEnd. La $\pi$-transformation du terme FrontEnd donne le modèle d'automate *FrontEnd* (cf. Figure 4.14).

Ce modèle est composé de six états et six principales transitions :

- ✓ L'état Init représente l'état initial ou d'inactivité. C'est un état urgent et l'émission sur le canal *initFront* est prioritaire par rapport aux autres émissions effectuées ;
- ✓ L'état Ready marque l'activation du composant FrontEnd à partir du composant Device après une réception sur le canal *initFront*. Cet état committed montre la priorité d'une émission sur le canal *touch* par rapport aux autres émissions effectuées sur les autres canaux ;

- ✓ L'état Screen représente l'interaction entre l'usager et le périphérique mobile. Il est atteint suite à une réception d'une action sur le canal *touch* ;
- ✓ L'état Response est atteint suite à l'action de réception sur le canal *devchan*. Il marque la première communication entre les composants BackEnd et FrontEnd ;
- ✓ L'état Comeback représente la déportation du backend applicatif vers la Cloudlet. Cet état est atteint suite à l'action de réception sur le canal *callback1*, sinon nous observons un retour du backend applicatif dans l'état Init pour marquer la fin du processus ;
- ✓ L'état Confirm est atteint suite à l'émission d'une notification au niveau du composant BackEnd pour signaler le retour effectif du backend applicatif.

*Figure 4.14 Modèle d'automate FrontEnd*




b) Composant BackEnd

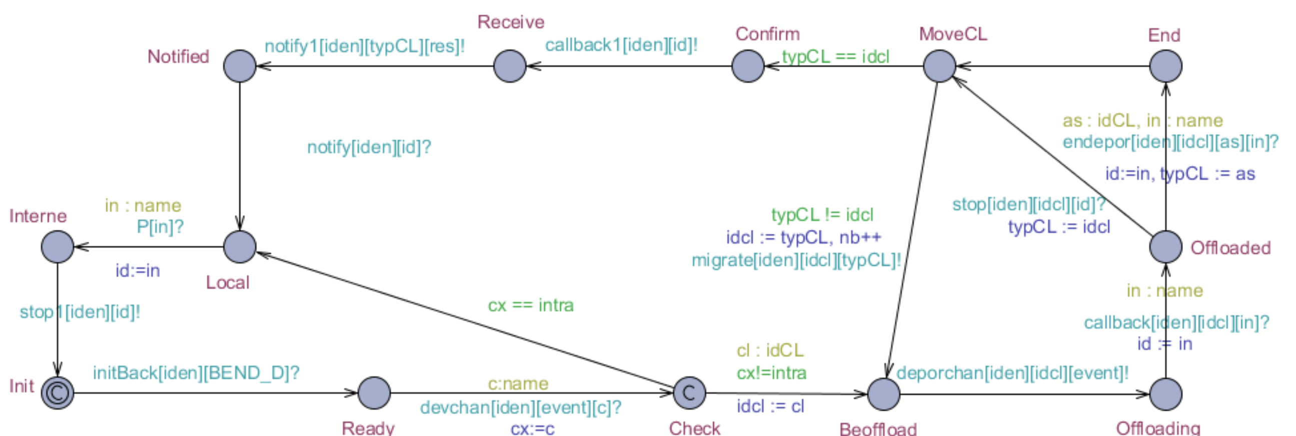
Le composant modélisant le métier d'une application mobile est représenté par le terme BackEnd (cf. Equation 28) ayant comme paramètre le canal *devchan* pour la communication avec le composant FrontEnd. La π-transformation du terme BackEnd donne le modèle d'automate BackEnd (cf. Figure 4.15).

Ce modèle est composé de treize états et quinze principales transitions :

- ✓ L'état Init représente l'état initial ou d'inactivité. Il est défini comme committed, cela veut dire que cet état est urgent et atomique ;
- ✓ L'état Ready marque l'activation du composant BackEnd à partir du composant Device après une réception sur le canal *initBack* ;
- ✓ L'état Check représente l'interaction avec le composant FrontEnd. Il est considéré comme committed pour marquer la priorité et l'urgence sur les autres états. C'est dans cet état que le backend applicatif décide d'effectuer une exécution locale ou distante. Cet état est atteint suite à une réception d'une action sur le canal *devchan* ;

- ✓ L'état Beoffload représente une situation d'exécution distante. La variable cx reçue à travers le canal *devchan* permet de connaître le traitement (*cx* != *intra* alors le traitement est locale sinon distant) ;
- ✓ L'état Local représente une situation d'exécution locale. Dans cette situation si la variable cx est différente de la variable intra alors il est à noter le retour du backend applicatif de la Cloudlet sinon l'exécution est faite totalement en local ;
- ✓ L'état Interne est atteint suite à une exécution locale du backend applicatif. À cet état, on émet un signal de fin d'exécution au niveau du composant FrontEnd avec l'identifiant de l'application sur le canal *stop1* ;
- ✓ L'état Offloading est atteint suite à l'émission d'une demande de déportation sur le canal *deporchan*. Cet état marque le début de la déportation et le lancement de la VDR correspondante est fait par la transition *callback[idcl][in]?* entre l'état Offloading et l'état Offloaded ;
- ✓ L'état Offloaded est atteint suite à une réception sur le canal *callback* de l'identifiant de la VDR au niveau de la Cloudlet ;
- ✓ L'état End marque la fin de la déportation suite à une réception sur le canal *endepor*. Dans cet état, nous recevons également une variable (as) qui permet de spécifier une migration possible dans une Cloudlet voisine. Cette information est stockée dans la variable *typCL* qui nous permet de savoir si la Cloudlet de départ est identique à la Cloudlet de migration ;
- ✓ L'état MoveCL permet de vérifier le déplacement de l'usager vers une Cloudlet voisine grâce au contenu de la variable *typCL* et celui du paramètre de la Cloudlet de départ. Le contenu de la variable *nb* permet de confirmer la migration de l'application vers la Cloudlet voisine lorsque sa valeur est supérieure à 0 et de retourner à l'état Beoffload.

*Figure 4.15 Modèle d'automate BackEnd*



### 3.2.5. Modèle d'une représentation virtuelle VDR

La notion de représentation virtuelle VDR s'applique à deux types sur le périphérique physique, soit à un périphérique mobile soit à un capteur. Pour généraliser cette représentation, nous avons défini à l'équation (2) un choix non-déterministe entre les différentes VDRs. Dans cette définition, le composant DevId est commun aux différentes VDRs, il permet d'émettre le canal qui servirait à la réception de l'identifiant de la VDR. Dans la suite de cette sous-section, nous donnons les différents modèles d'automates liés aux composants de la VDR.

a) SVDR pour les capteurs

Le modèle d'une représentation virtuelle d'un capteur (SVDR) est construit à partir de l'équation (5). La π-transformation du terme SVDR donne le modèle d'automate SVDR (cf. Figure 4.16) et cet automate admet quatre états et quatre transitions :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état CreateSVDR marque l'activation du composant SVDR à partir du composant Run et est atteint suite à une réception sur le canal *evtvdr* ;
- ✓ L'état Identity représente la sauvegarde de l'identifiant de la VDR créée au niveau du composant Admin. Cet état est atteint suite à une émission de l'identifiant sur le canal *ws* ;
- ✓ L'état IdRegistered permet l'activation du composant VirtualSensor à travers une émission sur le canal *virDevChan* et marque la fin de l'activation du composant DevId. Cet état est atteint suite à l'émission sur le canal *devIdchan*.

La figure 4.16 illustre l'automate de la SVDR. Cet automate instancie deux autres automates qui sont la VirtualSensor et la DevId.

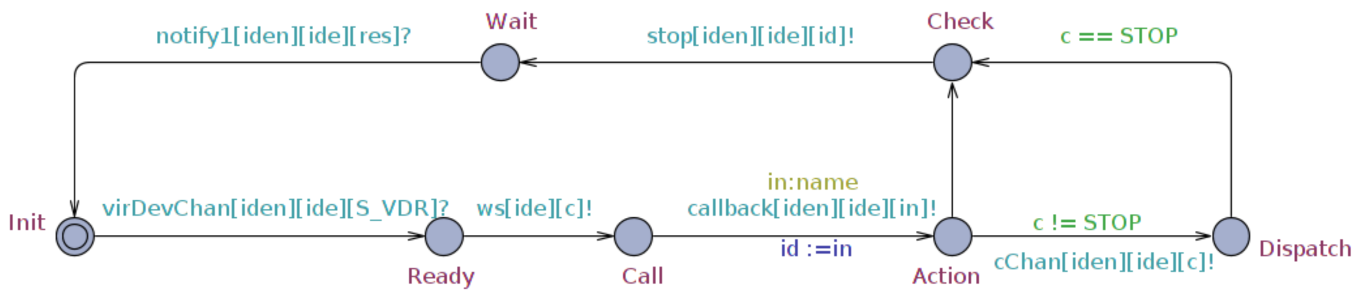
*Figure 4.16 Modèle d'automate de la SVDR*

La π-transformation du terme VirtualSensor donne le modèle d'automate VirtualSensor (cf. Figure 4.17). Cet automate est construit à partir de l'équation (6) et admet sept états :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Ready marque l'activation du composant VirtualSensor à partir du composant SVDR et est atteint suite à une réception sur le canal *virDevChan* ;

- ✓ L'état Call représente la sauvegarde de l'identifiant de la VDR créée au niveau du composant Admin. Cet état est atteint suite à une émission de l'identifiant sur le canal *ws* ;
- ✓ L'état Action permet de transiter à l'état Check si la commande reçue est un STOP. Sinon, cette action permet de se retrouver dans l'état Dispatch ;
- ✓ L'état Check est atteint suite au choix opéré par la commande c. La transition entre l'état Check et l'état Wait est une émission sur le canal *stop* pour spécifier la fin de l'exécution du processus ;
- ✓ L'état Wait est atteint suite à l'émission sur le canal *stop*.

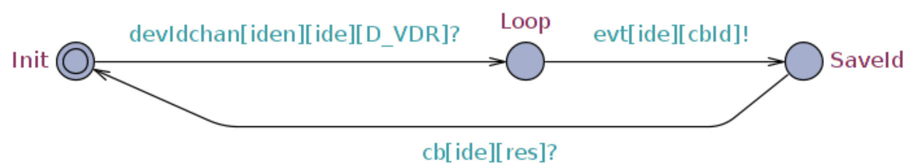
*Figure 4.17 Modèle d'automate VirtualSensor*




La π-transformation du terme DevId défini à l'équation (7) donne le modèle d'automate DevId (cf. Figure 4.18). Ce modèle est composé de trois états et trois principales transitions :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Loop marque l'activation du composant DevId à partir du composant SVDR ou DVDR après une réception de l'identifiant sur le canal *devIdchan* ;
- ✓ L'état SaveId est atteint suite à une émission sur le canal *evt* et permet de retourner à l'état initial par une émission de l'identifiant reçu sur le canal *cb*.

*Figure 4.18 Modèle d'automate DevId*




b) DVDR pour les périphériques

Le modèle d'une représentation virtuelle d'un périphérique (DVDR) est construit à partir de l'équation (8). La π-transformation du terme DVDR donne le modèle d'automate ci-dessous (cf. Figure 4.19) et cet automate admet quatre états et est similaire à celui du composant SVDR :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Ready marque l'activation du composant DVDR à partir du composant Run et est atteint suite à une réception sur le canal *evtvdr* ;

- ✓ L'état Tau représente la sauvegarde de l'identifiant de la VDR créée au niveau du composant Admin. Cet état est atteint suite à une émission de l'identifiant sur le canal *ws* ;
- ✓ L'état Identity permet l'activation du composant VirtualDevice à travers une émission sur le canal *virDevChan* et marque la fin de l'activation du composant DevId. Cet état est atteint suite à l'émission sur le canal *devIdchan*.

*Figure 4.19 Modèle d'automate de la DVDR*

Cet automate fait appel à d'autres composants (VirtualDevice et DevId) par l'émission sur les canaux *virDevChan* et *devIdchan.*

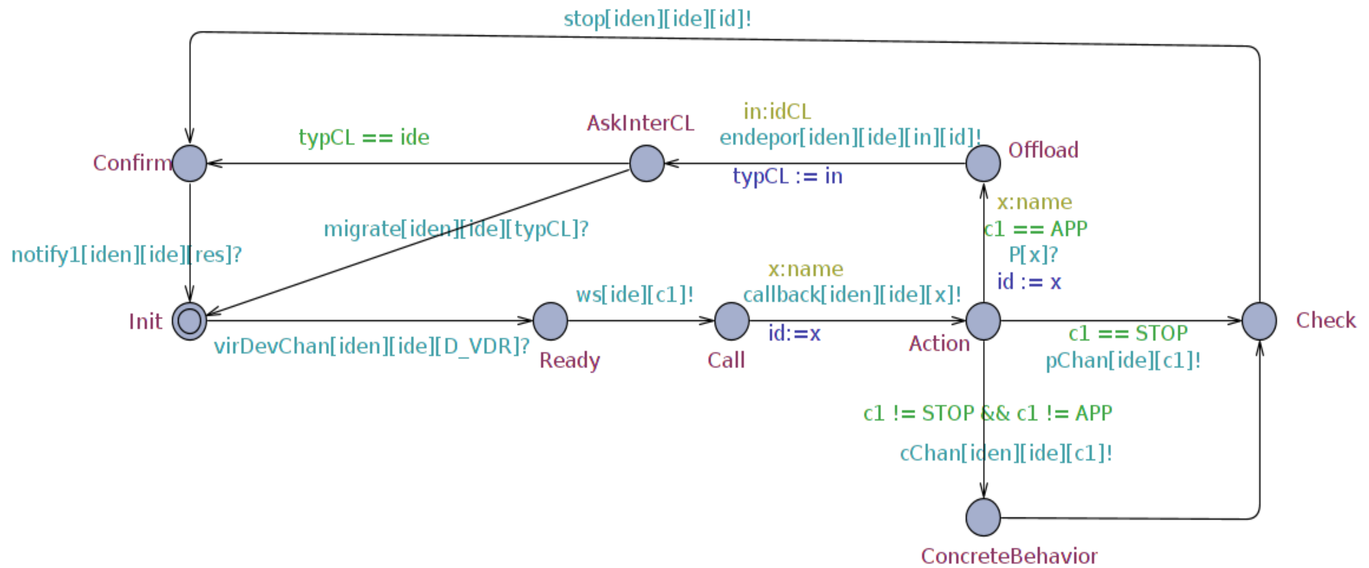
La $\pi$-transformation du terme VirtualDevice donne le modèle d'automate (cf. Figure 4.20). Cet automate est construit à partir de l'équation (10) et admet neuf états :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Ready marque l'activation du composant VirtualDevice à partir du composant DVDR et est atteint suite à une réception sur le canal *virDevChan* ;
- ✓ L'état Call représente la sauvegarde de l'identifiant de la VDR créée au niveau du composant Admin. Cet état est atteint suite à une émission de l'identifiant sur le canal *ws* ;
- ✓ L'état Action permet de transiter à l'état Check si la commande reçue est un STOP ou d'exécuter le paramètre d'ordre supérieur *P(x)* sur l'action de déportation. Dans le cas contraire, cet état permet de transiter dans l'état ConcreteBehavior suite à une émission sur le canal *cChan* ;
- ✓ L'état Check est atteint suite au choix opéré par la commande c et une émission sur le canal *pChan* à destination de l'automate Administrateur ;
- ✓ L'état Offload est atteint suite à l'exécution du paramètre d'ordre supérieur P(x) et au stockage de son identifiant dans la variable *id* ;
- ✓ L'état AskInterCL marque la fin de la déportation au niveau de la Cloudlet suite à une émission sur le canal *endepor* et la création de la VDR proprement dite. Dans cet état, le contenu de la variable *typCL* nous permet soit de migrer vers une Cloudlet voisine lorsque ce contenu est différent de celui du paramètre ide, soit de continuer l'exécution dans la même Cloudlet dans le

cas contraire. La transition entre l'état AskInterCL et l'état Init montre la migration vers une Cloudlet voisine ;

- ✓ L'état Confirm est atteint suite à une émission sur le canal *stop* ou marque la fin de la déportation sur la Cloudlet de départ. La transition de cet état à l'état Init montre la réception de la notification du composant BackEnd sur le canal *notify1* pour marquer le retour de l'application sur le BackEnd (c'est-à-dire le périphérique mobile).

*Figure 4.20 Modèle d'automate VirtualDevice*

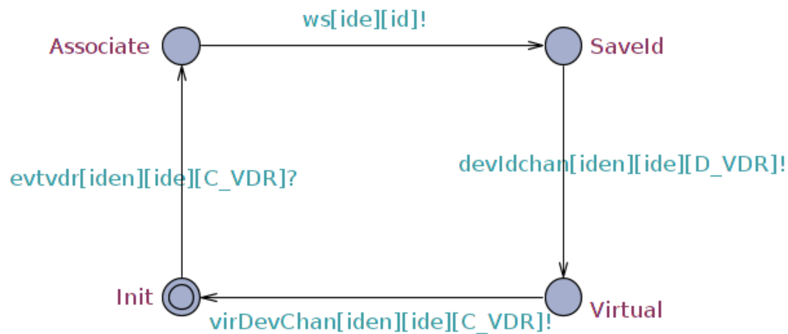



c) CVDR composite

Le modèle d'une représentation virtuelle composite (CVDR) est construit à partir de l'équation (12). La π-transformation du terme CVDR donne le modèle d'automate ci-après (cf. Figure 4.21) et cet automate admet quatre états et quatre transitions :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Associate marque l'activation du composant CVDR à partir du composant Run et est atteint suite à une réception de l'identifiant de la CVDR sur le canal *evtvdr* ;
- ✓ L'état SaveId représente la sauvegarde de l'identifiant de la VDR créée au niveau du composant Admin. Cet état est atteint suite à une émission de l'identifiant de la DVDR sur le canal *ws* ;
- ✓ L'état Virtual permet l'activation du composant CompositeDevice à travers une émission de son identifiant sur le canal *virDevChan* et marque la fin de l'activation du composant DevId. Cet état est atteint suite à l'émission sur le canal *devIdchan*.

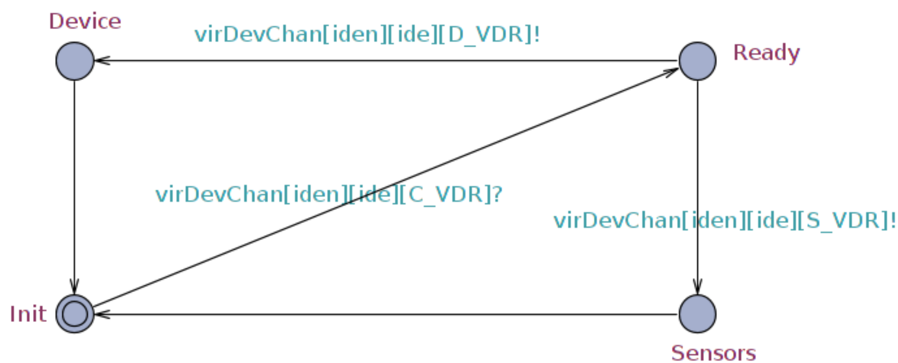
*Figure 4.21 Modèle d'automate de la CVDR*




La π-transformation du terme CompositeDevice donne le modèle d'automate CompositeDevice (cf. Figure 4.22). Cet automate est construit à partir de l'équation (13) et admet quatre états :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Ready marque l'activation du composant CompositeDevice à partir du composant CVDR et est atteint suite à une réception de l'identifiant de la CVDR sur le canal *virDevChan* ;
- ✓ L'état Sensors marque l'activation du composant VirtualSensor à partir du composant CompositeDevice et est atteint suite à une émission de l'identifiant de la SVDR sur le canal *virDevChan* ;
- ✓ L'état Device marque l'activation du composant VirtualDevice à partir du composant CVDR et est atteint suite à une émission de l'identifiant de la DVDR sur le canal *virDevChan* à destination du composant VirtualDevice.

*Figure 4.22 Modèle d'automate CompositeDevice*

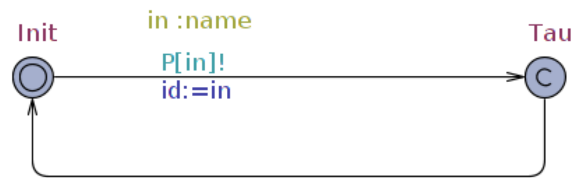



La π-transformation d'ordre supérieur du terme App considéré comme un conteneur d'application donne le modèle d'automate Process (cf. Figure 4.23). Cet automate est construit à partir de l'équation (9) et admet deux états :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Tau marque l'encapsulation de l'application déportée qui est définie comme paramètre avant l'action de déportation. Cet état défini comme committed montre l'urgence et la

priorisation de la déportation. La transition entre l'état Tau et l'état Init montre la réalisation d'une action interne.

*Figure 4.23 Modèle d'automate Process*




L'ensemble des automates temporisés présentés dans cette section sont à la base de nos simulations. Nous nous sommes focalisés sur les opérations liées à la mobilité du backend applicatif vers les Cloudlets.

## 3.3. Migration de VDR dans le modèle

Pour valider notre modélisation de Cloudlets, nous avons étudié le comportement d'un backend applicatif déporté dans une DVDR et celui résidant sur un périphérique mobile. Pour cela, nous avons spécifié certains composants impliquant le protocole MOCP, liés aux échanges entre les périphériques et une Cloudlet. Dans cette section, nous mettons en exergue les échanges entre l'usager, la synchronisation et l'interaction durant la migration.

### 3.3.1. Simulation des rôles humains

L'interaction humaine se fait à deux niveaux : via le rôle Administrateur, via le rôle User. Dans la suite, nous modélisons les composants Administrateur et User.

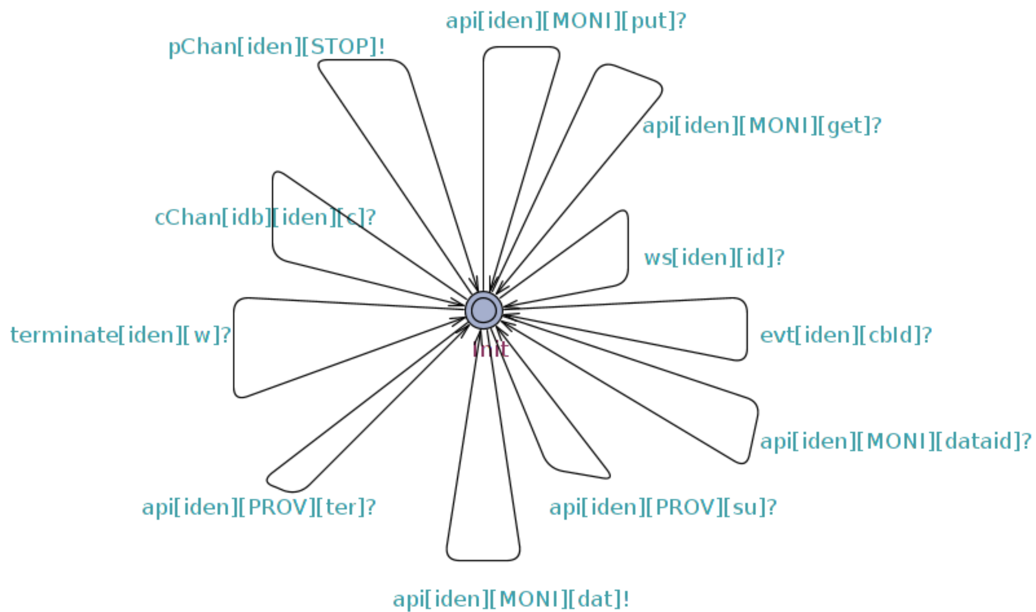
a) Composant Administrateur

Au niveau de l'administrateur réseau et système, le composant Administrateur est défini par le terme *Admin*. Nous avons défini sa signature dans l'équation (31) pour garder la clarté de nos spécifications et travailler sur une abstraction du système. Son objectif est d'envoyer tous les messages nécessaires en utilisant le vecteur $\overrightarrow{api}$.

Le modèle d'automate du composant Admin illustré par la Figure 4.24 admet un état considéré comme état puit et plusieurs transitions :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ La transition *ws[iden][id]?* permet la sauvegarde de l'identifiant de la VDR ;
- ✓ La transition *api[iden][][]?* permet de recevoir tous les messages provenant des composants Monitoring et Provisioning.

*Figure 4.24 Modèle d'automate Administrateur*




b) Composant User

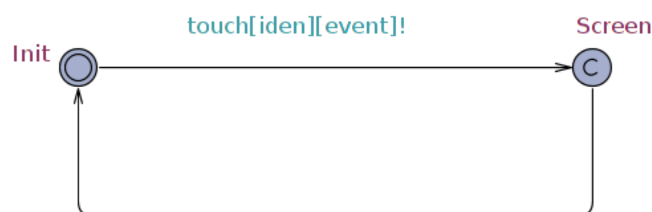
Le modèle d'automate User est construit à partir de la définition de l'équation (32) et sa π-transformation (cf. Figure 4.25). Le terme User représente un usager du périphérique mobile. Son objectif est de communiquer avec le périphérique via sa partie frontale appelée FrontEnd à travers l'écran tactile du périphérique.

Ce modèle est composé de deux états et une principale transition :

- ✓ L'état Init représente l'état initial ou d'inactivité ;
- ✓ L'état Screen représente l'interaction entre l'usager et le périphérique mobile. Il est atteint suite à une émission sur le canal *touch*. Cet état défini comme committed, montre l'urgence et la priorité de l'émission sur les canaux *touch*.

La transition entre l'état Screen et l'état Init montre la réalisation d'une action interne.

*Figure 4.25 Modèle d'automate User*




Ces automates décrivent le langage d'actions que peut faire l'user ou l'administrateur. Il est ainsi envisageable de vérifier si une action peut être faite avec un backend local au périphérique ou exporté dans une Cloudlet.

# IV. Vérification

Nous avons mis en valeur des propriétés liées à notre réseau de Cloudlets à partir des spécifications écrites au chapitre 3. Ces propriétés doivent être préservées dans l'implantation qui est faite au chapitre 5. Dans cette section, nous nous intéressons à leur preuve par model-checking grâce au spécifieur d'UPPAAL.

## 4.1. Démarche

Les propriétés qui nous intéressent sont des propriétés liées à la déportation, aux communications dans le réseau de Cloudlets et à l'exclusion mutuelle des VDRs. Nous avons séparé les propriétés en deux parties, car nous distinguons deux cas d'exécution (en locale et dans la Cloudlet). Dans chaque partie, il existe des propriétés liées à la communication que nous prouvons dans les sections suivantes.

Nous énonçons trois propriétés liées à l'exécution que nous vérifions :

- ✓ Propriété 1 : un BackEnd peut toujours migrer dans une Cloudlet et continuer son exécution ;
- ✓ Propriété 2 : un BackEnd peut toujours revenir sur le périphérique mobile s'il est en relation avec la Cloudlet sur laquelle a migré le BackEnd initialement ;
- ✓ Propriété 3 : l'intercommunication entre deux Cloudlets ou le suivi d'une Cloudlet à une autre n'interrompt pas l'interaction avec le FrontEnd.

Les deux premières propriétés mettent en jeu la communication d'un périphérique mobile avec une Cloudlet, une exécution distante et une exécution locale. Ces propriétés montrent que la déportation d'une application dans la Cloudlet ne nuit pas à l'interactivité de l'utilisateur. La troisième propriété montre le suivi sur un réseau de Cloudlets, c'est-à-dire la migration d'un backend applicatif d'une Cloudlet à une autre. Ainsi, un user lance sur son périphérique une application A1 composée d'un FrontEnd et d'un BackEnd. Il déporte le Backend sur la Cloudlet C1 de son voisinage puis se déplace de sorte qu'il part du voisinage de la Cloudlet C1 où il était pour être pris en charge par la Cloudlet C2 voisine tout en assurant la continuité de service.

La continuité de service dans notre réseau de Cloudlets est vérifiée par la propriété suivante : A[] *not deadlock*. Tous les composants s'exécutent de manière infinie donc pas d'interblocage dans le système et nous notons une interopérabilité du système.

## 4.2. Vérification d'exécution avec déportation et migration

Dans un contexte d'exécution avec migration, le périphérique mobile et l'user sont en interaction avec la Cloudlet à travers le composant VirtualDevice. Dans cette sous-section, nous détaillons la preuve de propriétés liée à la déportation et à la communication.

### 4.2.1. Propriété de déportation

Ramenée à notre contexte de simulation, la déportation signifie que le terme VirtualDevice se trouve dans un état Offload (Cf. Figure 4.20), la condition de migration est vérifiée (*BackEnd(i).cx != BackEnd(i).intra*) et l'exécution s'est effectuée au niveau de la Cloudlet. Ce qui conduit à la situation suivante : le terme BackEnd se trouve dans l'état Offloaded (Cf. Figure 4.15) et le FrontEnd dans l'état Response (Cf. Figure 4.14). Pour établir cette propriété, nous la décomposons en deux sous-propriétés pour montrer l'existence et l'atteignabilité des états.

$$E <> \ forall(i : \ idus)\ forall(j : idCL)\ VirtualDevice(i,j).Offload\ \&\&\ Procs.Tau \\ \&\&\ \left(BackEnd(i).cx\ ! = \ BackEnd(i).intra\right) \quad (75)$$

La sous-propriété de l'équation (75) montre que ces états sont atteignables pour tout périphérique ou toute Cloudlet. Le fait que le terme Procs se trouve dans l'état Tau montre une exécution du backend applicatif dans la Cloudlet. La création de la VDR est justifiée par la présence du terme VirtualDevice à l'état Offload.

$$E <> \ forall(i : idus)\ BackEnd(i).Offloaded\ \&\&\ FrontEnd(i).Response \quad (76)$$

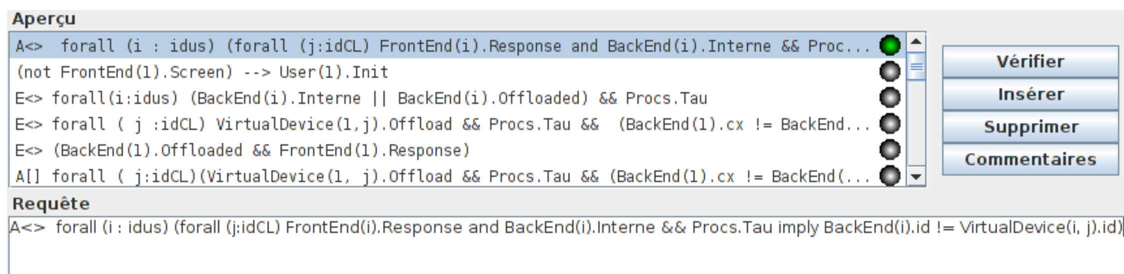
La sous-propriété de l'équation (76) montre l'existence des termes BackEnd et FrontEnd respectivement dans les états Offloaded (Cf. Figure 4.15) et Response (Cf. Figure 4.14).

Au final, pour établir la propriété 1, nous établissons une implication entre les deux sous-propriétés pour toutes les exécutions :

$$A <> \ forall(i : idus)\ forall(j : idCL)\ (VirtualDevice(i,j).Offload\ \&\&\ Procs.tau\ \&\& \\ \left(BackEnd(i).cx\ ! = \ BackEnd(i).intra\right) \\ imply\ E <> \ BackEnd(i).Offloaded\ \&\&\ FrontEnd(i).Response) \quad (77)$$

L'automate VirtualDevice dans l'état Offload (Cf. Figure 4.20) et l'automate Procs dans l'état Tau (Cf. Figure 4.23) montrent une exécution dans la Cloudlet après une migration du backend applicatif dans la Cloudlet.

*Figure 4.26 Aperçu de la vérification de la propriété 1*




La propriété 1 de l'équation (77) a été vérifiée par model checking via le système de vérification UPPAAL.

### 4.2.2. Propriété de communication

Nous avons identifié une propriété relative à la communication dans notre réseau de Cloudlets. Cette propriété permet de valider les échanges ou les appels entre le périphérique mobile et la Cloudlet. La propriété 2 énoncée concerne les automates BackEnd, FrontEnd, Procs et VirtualDevice. Pour vérifier qu'un backend applicatif peut revenir sur le Device et fonctionner avec son FrontEnd, nous allons procéder en trois étapes :

- ✓ La création d'un canal pour la communication du BackEnd et du FrontEnd avec l'application ;
- ✓ La transmission de l'identifiant du backend applicatif jusqu'au composant Procs d'où la création d'un nombre unique que nous retournons au composant FrontEnd et sur la chaîne de migration ;
- ✓ La simulation de la synchronisation d'état entre le BackEnd et le FrontEnd.

Chaque étape a été simulée et vérifiée sur notre modèle. Nous détaillons par la suite les différents composants permettant d'obtenir cette propriété.

$$E <> forall(i : idus) forall(j : idCL)((BackEnd(i).Offloaded \\ \&\& FrontEnd(i).Response \&\& VirtualDevice(i,j).Offload \\ \&\& (Procs.id == BackEnd(i).id == VirtualDevice(i,j).id)) \tag{78}$$

La propriété de l'équation (78) exprime la transmission de l'identifiant du backend applicatif et la migration de ce dernier dans la Cloudlet.

$$E <> forall(i : idus)\ forall(j : idCL)\ (BackEnd(i).Interne \\ \&\& not\ VirtualDevice(i,j).Offload\ \&\& Procs.Tau \&\& FrontEnd(i).Response \\ \&\& (Procs.id = FrontEnd(i).id == BackEnd(i).id\ != VirtualDevice(i,j).id)) \tag{79}$$

Cette propriété de l'équation (79) exprime le retour du backend applicatif sur un périphérique mobile et la synchronisation d'état entre le BackEnd(i) et le FrontEnd(i).

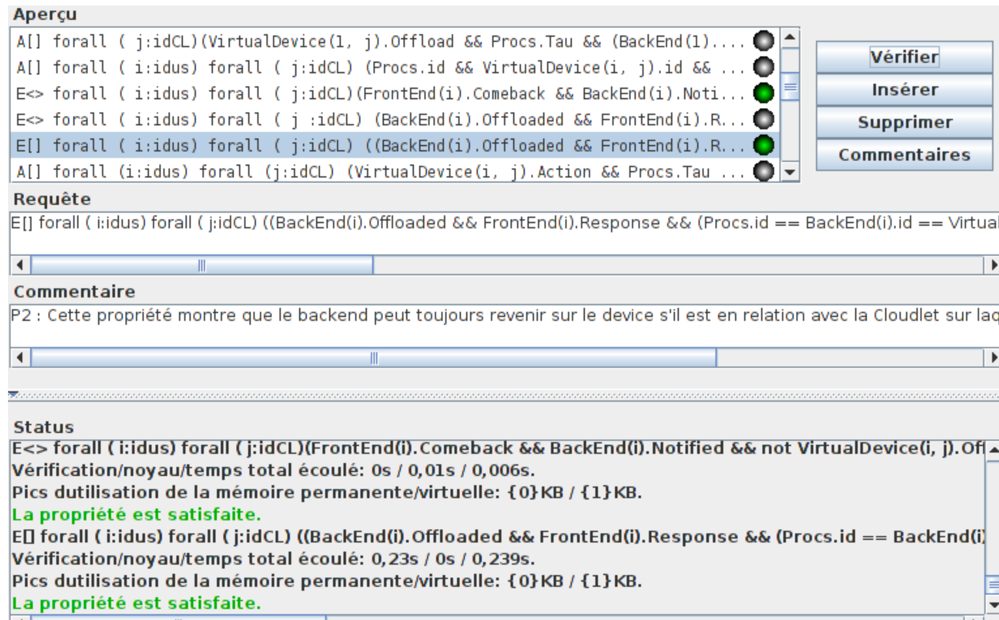
Ces deux propriétés (des équations (78) et (79)) montrent la création d'un canal de communication entre le BackEnd et le FrontEnd. L'existence de ces expressions a été vérifiée dans le système de vérification UPPAAL. Ce qui conduit à la propriété 2 ci-dessous :

$$E\ [\,]\ forall(i : idus)\ forall(j : idCL)\ ((BackEnd(i).Offloaded \\ \&\& FrontEnd(i).Response \&\& VirtualDevice(i,j).Offload \\ \&\& (Procs.id == BackEnd(i).id == VirtualDevice(i,j).id)) \\ imply\ (BackEnd(i).Interne \&\& not\ VirtualDevice(i,j).Offload \&\& Procs.Tau \\ \&\& (Procs.id = FrontEnd(i).id == BackEnd(i).id\ != VirtualDevice(i,j).id)) \tag{80}$$

L'équation (80) exprime le fait que lorsque l'identifiant du backend applicatif est le même dans les composants BackEnd, VirtualDevice et Process, nous observons une migration de ce backend applicatif

dans la Cloudlet. De même, il est à noter le retour de cette application au niveau du Device, car l'automate VirtualDevice se trouvant dans l'état Offload (Cf. Figure 4.20), l'automate FrontEnd dans l'état Response (Cf. Figure 4.14) et l'automate BackEnd dans l'état Offloaded (Cf. Figure 4.15). Ce retour se justifie également par le fait que l'identifiant n'est plus le même après une exécution du backend applicatif au niveau local.

*Figure 4.27 Aperçu de la vérification de la propriété 2*




La propriété 2 de l'équation (80) a été vérifiée par model checking via le système de vérification UPPAAL.

### 4.2.3. Propriété de migration

Notre réseau de Cloudlets offre la possibilité de suivi d'une Cloudlet à une autre. Dans notre contexte de simulation, nous avons déclaré deux Cloudlets et deux usagers (Users). Chaque Cloudlet possède ses propres composants, de même pour les différents périphériques mobiles ou capteurs.

La migration d'un backend applicatif d'une Cloudlet à une autre, nous amène à définir deux variables (nb et typCL) et un canal *endepor[idus][x][y][name]* pour simuler cette migration (*x* est l'identifiant de la Cloudlet de départ et *y* est l'identifiant de la Cloudlet d'arrivée).

Ce qui conduit à la situation suivante : le terme BackEnd se trouve dans l'état *Init* (Cf. Figure 4.15) et le FrontEnd dans l'état Init (Cf. Figure 4.14). Pour établir cette propriété, nous positionnons la variable *nb* à 0 ; lorsqu'il y a une migration dans une autre Cloudlet, cette valeur s'incrémente.

Pour établir cette propriété, nous établissons que :

$$E <> \ forall(i: idus)\ FrontEnd(i).Init \ \&\&\ BackEnd(i).Init \ \&\&\ BackEnd(i).nb > 0 \tag{81}$$

Cette propriété a un niveau d'abstraction supérieur et démontre une situation de déportation suivie d'une possibilité de migration vers la Cloudlet voisine.

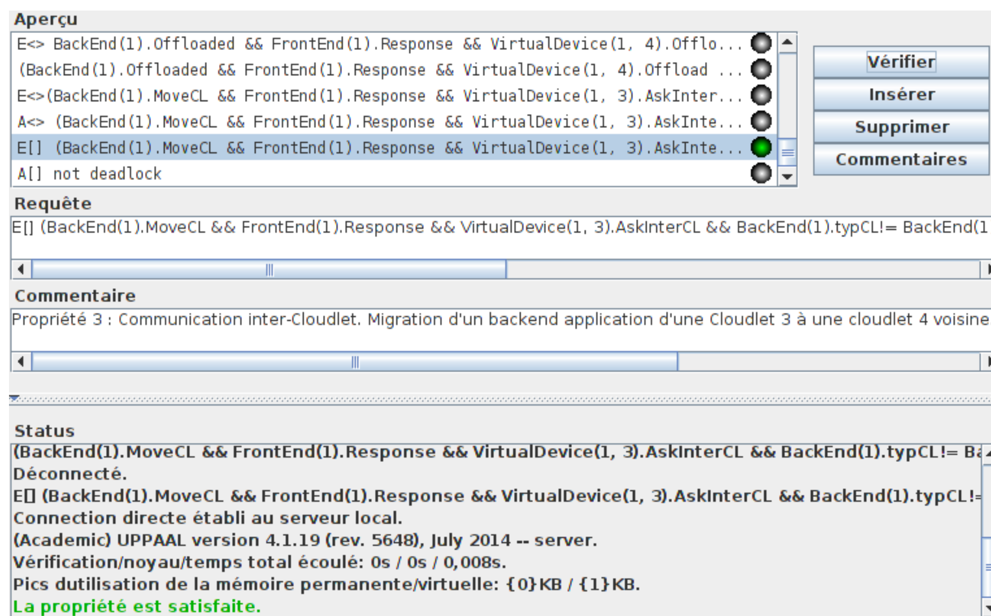
De même, nous montrons l'atteignabilité des états dans l'équation (82) décrite ci-dessous. Dans cette équation, nous avons une migration du backend applicatif de la Cloudlet (3) à la Cloudlet (4).

$$E[\,]\ (BackEnd(1).MoveCL\ \&\&\ FrontEnd(1).Response\ \&\&\ VirtualDevice(1,3).AskInterCL \quad (82)$$

$$\&\&\ BackEnd(1).typCL\ != \ BackEnd(1).idcl)\ imply\ (BackEnd(1).Offloaded$$

$$\&\&\ FrontEnd(1).Response\ \&\&\ VirtualDevice(1,4).Offload\ \&\&\ Procs.Tau\ \&\&\ Run(4).Init)$$

*Figure 4.28 Aperçu de la vérification de la propriété 3*




Cette propriété 3 de l'équation (82) a été vérifiée par model checking via le système de vérification UPPAAL.

### 4.2.4. Propriété de curation

Nous avons identifié une propriété relative à la réinitialisation du système suite à la construction de notre réseau de Cloudlets. Cette propriété concerne la réinitialisation de l'automate User lorsque l'automate FrontEnd se trouve dans un état autre que l'état Screen (Cf. Figure 4.14) de notre modèle. Elle est définie comme suit :

$$(not\ FrontEnd(1).Screen)\ -->\ (User(1).Init\ ||\ User(2).Init) \qquad (83)$$

Cette propriété (équation (83)) a été vérifiée par model checking via le système de vérification UPPAAL. Elle exprime le fait que pour toutes les exécutions, l'automate User se trouve toujours dans l'état Init tant que l'automate FrontEnd est dans un état autre que l'état Screen.

## 4.3. Vérification d'exécution locale

Dans notre réseau de Cloudlets, nous observons deux contextes d'exécution qui conduisent à l'exécution du backend applicatif dans le Device. Le premier contexte montre qu'il n'y a pas de déportation du backend applicatif au niveau de la Cloudlet. Le deuxième contexte montre une déportation et/ou un retour du backend applicatif sur le Device et l'exécution sur ce dernier.

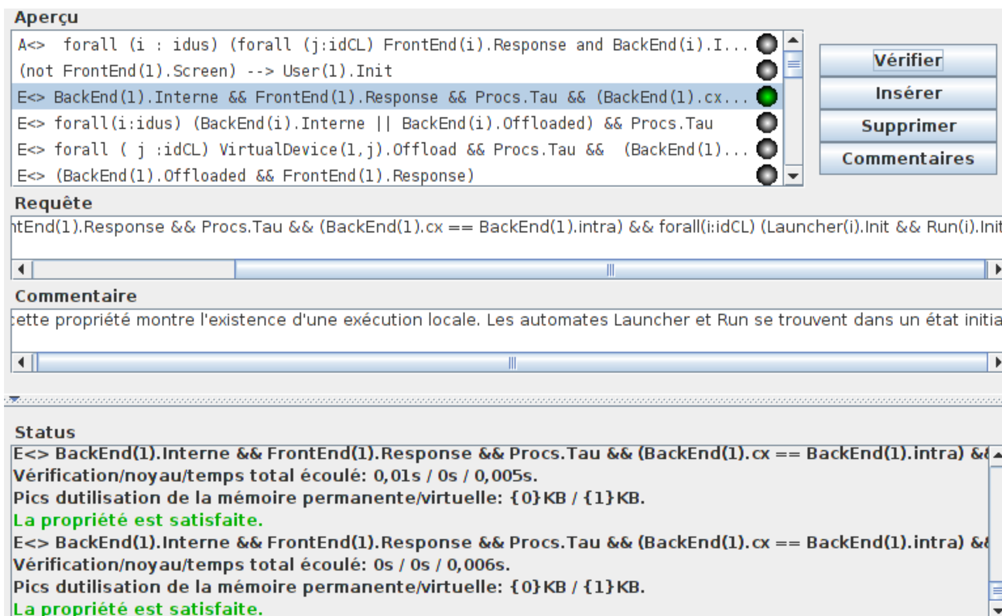
Dans cette section, nous vérifions uniquement le premier contexte d'exécution vu que le deuxième contexte a déjà été vérifié grâce aux propriétés 2 et 3.

Dans ce contexte, une exécution se caractérise par le fait que le contenu de la variable *intra* est égal à celui de la variable *cx*.

$$E <> BackEnd(1).Interne \;\&\&\; Procs.Tau \;\&\&\; FrontEnd(1).Response \\ \&\& \; (BackEnd(1).cx \; == \; BackEnd(1).intra) \\ \&\& \; forall(i: idCL) \; (Launcher(i).Init \;\&\&\; Run(i).Init) \tag{84}$$

Cette propriété exprime l'existence d'une exécution locale. Le fait que les automates Launcher et Run restent dans leur état d'inactivité (Init), montre que la VDR n'a pas été créée. Les automates BackEnd dans l'état Interne, Procs dans l'état Tau et FrontEnd dans l'état Response montrent une exécution du backend applicatif. L'égalité des variables *cx* et *intra* de l'automate BackEnd et l'exécution du backend application justifient que cette exécution est au sein du Device.

*Figure 4.29 Aperçu de la vérification d'une exécution locale sans déportation*



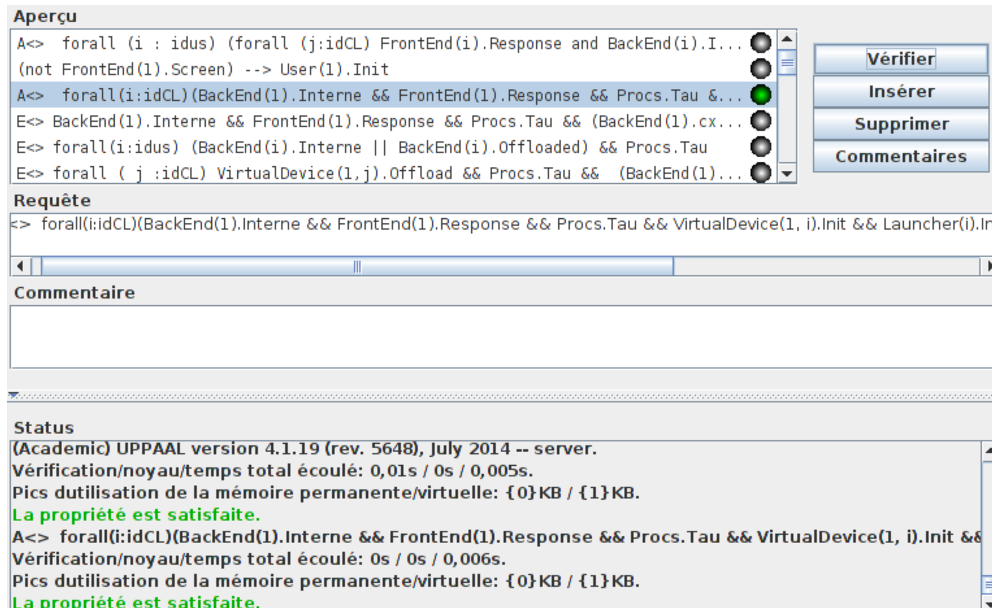

La propriété (équation (84)) a été vérifiée par model checking via le système de vérification UPPAAL et montre l'existence d'une exécution locale.

$$A <> forall(i: idCL)(BackEnd(1).Interne \;\&\&\; Procs.Tau \;\&\&\; FrontEnd(1).Response \\ \&\& \; VirtualDevice.Init \;\&\& \; (Launcher(i).Init \;\&\&\; Run(i)).Init \tag{85}$$

$$imply\ BackEnd(1).cx\ ==\ BackEnd(1).intra$$

Cette équation est une réécriture de la propriété précédente. Elle justifie une exécution locale en utilisant l'opérateur d'implication (imply). Elle exprime le fait que pour toutes les exécutions où les automates VirtualDevice, Launcher et Run sont dans leur état initial et qu'il y a exécution d'un backend applicatif, cette exécution a lieu dans le Device.

*Figure 4.30 Aperçu de la vérification d'une exécution locale pour intra = cx*




La propriété (équation (85)) montre une exécution s'effectue en local (dans le Device) lorsque la valeur de la variable *intra* est identique à celle de la variable *cx* dans un Backend. De même, il existe toujours une communication entre le FrontEnd et le BackEnd avant toute exécution de l'application.

# V. Bilan

Nous avons abordé dans ce chapitre la transformation de notre spécification construite au chapitre 3 en systèmes d'automates temporisés. Une première étape représente la construction d'une structure de données partagées et la déclaration du système et de ses composants. La deuxième étape modélise la Cloudlet en définissant des sous-systèmes ou composants, un périphérique mobile ainsi qu'une représentation virtuelle VDR. La troisième étape porte sur la simulation de la migration de VDR dans le modèle de réseau de Cloudlets de même les interactions durant la migration ou la déportation.

Nous nous sommes intéressés à des propriétés liées à deux contextes d'exécution : local au périphérique et distant.

Dans le cadre d'une exécution locale, nous avons prouvé des propriétés de communication et de curation. Les propriétés de déportation, d'exclusion mutuelle et de migration ont été établies pour une exécution distante.

Ces preuves de propriétés ont été établies en conservant une totale indépendance avec une implantation spécifique de notre réseau de Cloudlets. Ces informations enrichissent notre modèle et ajoutent des propriétés à préserver dans notre implémentation. Le suivi des applications virtualisées dans une Cloudlet a été fait en cas de mobilité de l'usager final. Le réseau de Cloudlets est évolutif et les interactions des usagers doivent se faire dans le respect d'un ensemble de propriétés minimal dont celles de ce chapitre. La préservation de ces propriétés au niveau de l'implémentation est essentielle.

Dans le chapitre suivant, nous nous appuyons sur les résultats des chapitres 2, 3 et 4 pour construire une implémentation d'un prototype de notre réseau de Cloudlets en respectant les spécifications et les propriétés étudiées.

# CHAPITRE 5 : Prototypage et résultats

L'objectif de ce chapitre est de présenter notre implémentation de Cloudlet basée sur nos résultats des chapitres 2, 3 et 4. Pour répondre aux exigences précédemment citées, des choix techniques sont nécessaires pour assurer entre autres la mobilité de composants depuis une plate-forme mobile vers une Cloudlet et de façon générale l'ensemble de son cycle de vie.

Pour construire une implémentation du protocole MOCP (core, intra/inter Cloudlet et Cloud), d'autres contraintes techniques sont à satisfaire afin de permettre l'indépendance des composants, le chargement dynamique de ressources et l'interopérabilité entre composants. Nous présentons en premier nos contraintes d'implémentation associées à leur rôle logiciel et notre démarche de conception basée sur de l'open documentation. Puis nous abordons le déploiement logiciel des composants, les services et leur structure. Enfin, nous traitons un scénario de mobilité applicative entre un périphérique mobile et un réseau de Cloudlets. Au cours de celui-ci, des mesures sont faites pour évaluer l'impact de la mobilité sur le déroulement de l'exécution. Cette analyse est élargie par les résultats de monitoring des Cloudlets.

## I. Introduction

Les applications logicielles deviennent de plus en plus complexes, une approche louable aujourd'hui pour maîtriser cette complexité consiste à concevoir ces applications en termes d'assemblages de modules ou de composants logiciels. Les composants logiciels peuvent être vus comme des unités indépendantes, réutilisables et remplaçables destinées à remplir une fonction définie au sein d'une application logicielle [183]. L'architecture logicielle traite des composants logiciels et des interactions entre ces composants. Comme exemple de base, nous pouvons citer, une architecture client/serveur qui est appliquée dans de très nombreux contextes tels que les applications Web ou des applications Big Data, etc. Une telle architecture est une simple abstraction cachant certains détails du système par encapsulation. Il est ainsi plus facile d'identifier les propriétés d'un serveur et d'un client [184]. Des exemples plus complexes se rencontrent avec les applications JavaEE[18] distribuées. Ils contiennent plusieurs niveaux d'abstraction, chacun avec sa propre architecture. Les modèles architecturaux sont utilisés sur le principe de modèle en couche [185]. Les principes de ce modèle forment la base de l'architecture recommandée par les directives de conception JavaEE BluePrints et les patterns JavaEE.

[18] JavaEE : Java Entreprise Edition

Nous proposons dans cette première section notre vision de l'architecture d'une Cloudlet et du réseau auquel elle appartient.

## 1.1. Serveur de composants dynamiques

Notre but est de proposer pour chaque nœud de déploiement, un modèle de composants ainsi qu'un modèle de communication inter-composant. Plusieurs choix ont été présentés au chapitre 2 (section IV).

Une approche à base de composants implique une stratégie appropriée pour gérer les composants dans la durée. Dans un contexte de développement d'applications mobiles ou de mobilité de composants, le déploiement à chaud est une propriété non-fonctionnelle qui est essentielle pour les nouveaux logiciels même s'il s'agit d'un périphérique intégré. Plusieurs stratégies existent déjà dans les serveurs d'applications Java. Par exemple, la norme OSGi (Open Services Gateway initiative) est souvent mise en œuvre dans les serveurs qui gèrent le cycle de vie des composants, leur dynamisme et les services qu'ils apportent.

La norme OSGi possède diverses implémentations adhérant aux contraintes de sa spécification. En dehors de l'implémentation de référence basique fournie sur le site officiel, nous pouvons citer d'autres serveurs applicatifs qui ont pris en compte cette norme :

- ✓ Apache Felix : framework implémentant OSGi Release 6 (R6) intégré dans d'autres projets sous la licence Apache tels que Apache Karaf ;
- ✓ JBoss OSGi : framework OSGi R5 inclus dans le serveur d'application WildFly. WildFly est basé sur les modules JBoss pour fournir une plate-forme de chargement d'application JavaEE ;
- ✓ Apache ServiceMix : framework d'intégration open source basé sur OSGi fournissant un ESB qui implémente la spécification JBI pour Java Business Interface JSR-208 avec des objectifs de connectivité et de flexibilité.

La spécification OSGi propose de partager la machine virtuelle à l'exécution entre applications dans une architecture modulaire en isolant les applications les unes des autres. La notion de bundle correspond alors à une unité de déploiement sur un serveur OSGi. Ce nouveau type de composant, nommé bundle dispose d'une déclaration explicite de dépendances qui autorisent une gestion fine de son cycle de vie à l'exécution.

Il apparaît naturel que l'architecture d'une Cloudlet doive respecter la norme OSGi afin d'autoriser le chargement dynamique de composants. Enfin, le déchargement de composant est de même importance. Il est l'étape initiale de la déportation d'application. Aussi, nous faisons le choix logiciel de structurer toute application mobile sujette à la déportation, autour d'un serveur OSGi léger tel qu'Apache Felix.

Les contrats des composants OSGi sont orientés dépendances vers d'autres composants, mais ont été étendus dans des projets tels que SpringDM[19] ou Blueprint[20] (Apache Ariès) pour définir des contrats avec des dépendances dynamiques, c'est-à-dire dont l'évaluation est faite lors du chargement du composant. De plus, la spécification a été étendue avec DOSGi (Distributed-OSGi) [186] dans les systèmes distribués ou DSW comme référence générique à tout type de protocole et de format de données capable d'invoquer des services distants (c'est-à-dire dans une autre JVM ou espace d'adressage). La plate-forme DOSGi comprend un service de découverte et aussi une extension de métadonnées SCA[21] pour la configuration de plusieurs composants distribués.

La spécification OSGi met en avant un modèle collaboratif à base de composants. Cette architecture permet un couplage faible entre bundles. Cela favorise la migration d'un bundle entre deux nœuds de déploiement sans entrainer nécessairement la migration des dépendances. Enfin, cela remplace une communication locale entre deux bundles par une communication distante tout en préservant le système de dépendances dynamiques entre bundles via DOSGi. De plus, un suivi de service à l'exécution est géré grâce aux notifications d'événements d'apparition et de disparition de service. Cela peut ressembler pour certains aspects à un modèle SOA entre JVMs.

D'un point de vue qualité de service, la plate-forme d'exécution se doit d'être disponible, en particulier s'il s'agit d'une Cloudlet, afin d'être prête pour toute opération d'offloading. La spécification OSGi décrit un cycle de vie précis pour chaque bundle afin d'éviter le redémarrage, de façon indépendante de la plate-forme d'exécution. Il est alors possible d'installer, de désinstaller, de mettre à jour un bundle et ainsi d'effectuer la migration d'un composant applicatif, de le démarrer sans redémarrer la plate-forme d'exécution (Cloudlet). Il en est de même de la gestion des configurations et de la sécurité qu'il est possible de faire évoluer à chaud.

Lors du suivi de l'usager en cours d'utilisation d'une application, un réseau de Cloudlets effectue en fonction du déplacement de l'usager, le déplacement du Backend applicatif de Cloudlet en Cloudlet. L'accès d'une Cloudlet à une autre amène une diminution du nombre de bundles chargés lors du déplacement d'une application métier et offre ainsi une baisse de la charge du poste mobile de l'usager. Le périphérique de l'usager est ainsi disponible pour d'autres applications mobiles concurrentes.

Une mise en œuvre directe de ces principes est fournie par une API de type REST lorsqu'une nouvelle application est déportée sur une Cloudlet. Les événements de notification assurent une prise en compte fine de la découverte de Cloudlets dans le voisinage du périphérique mobile de l'utilisateur lorsque celui-ci est en cours de déplacement. De manière plus uniforme, c'est l'ensemble des dispositifs qui est équipé de tels services.

---

[19] http://www.springsource.org/osgi

[20] http://aries.apache.org/modules/blueprint.html

[21] SCA (Service Component Architecture) est un standard ayant pour objectif de rapprocher les architectures à service et à composant, proposé par l'OSOA (Open Service Oriented Architecture)

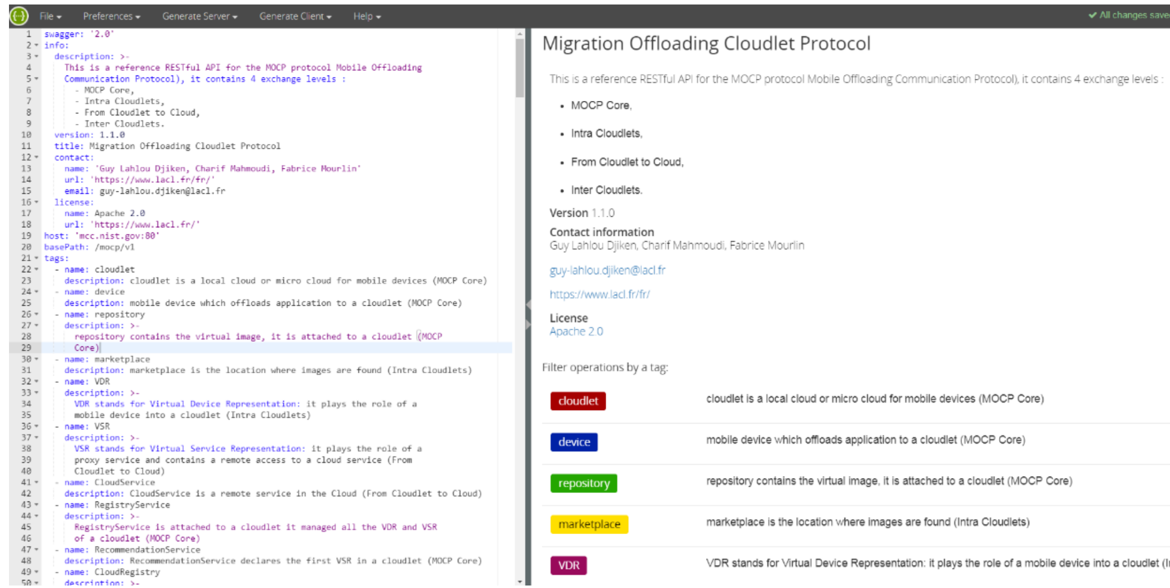
## 1.2. Définition du protocole MOCP

Nous avons défini le protocole MOCP pour les échanges entre un périphérique mobile et une Cloudlet ou entre Cloudlets ou encore entre Cloudlet et un Cloud. Nous ne souhaitons pas imposer une uniformité d'implémentation des Cloudlets ou celle de périphériques mobiles. Ainsi, il faut que ces échanges distants exploitent non seulement la puissance de DOSGi, mais aussi l'indépendance du protocole d'échange sous-jacent. Dans notre cas, le protocole http est notre choix de véhicule des échanges de messages entre les différents nœuds de déploiement. Le saut entre l'analyse faite aux chapitres 2 et 3, et un prototype doit être fidèle. Dans ce but, un processus de conception par raffinement jusqu'à une API Restful est mis en place. Actuellement, cela s'apparente à l'utilisation d'une API de documentation. Laquelle doit être enrichie suffisamment pour qu'un squelette de code puisse être généré. Nous avons choisi un standard appelé OpenAPI [187], anciennement connu sous le nom de Swagger RESTful API qui permet d'avoir des spécifications simples par une documentation exhaustive. OpenAPI désigne un ensemble de spécifications permettant de documenter à un format convenu les APIs REST des composants de notre architecture. Dans notre documentation de MOCP, tous les composants doivent être équipés d'une interface REST. Ainsi, toute Cloudlet doit exposer certains de ses services en REST tout comme le backend applicatif de nos applications mobiles. Dans ce but, nous avons construit une description des échanges pour chaque composant d'une Cloudlet ou d'une application mobile. Par l'emploi de l'outil Swagger UI, nous avons accès à une documentation interactive qui permet de tester chaque composant unitairement par le biais de son interface REST. Les spécifications actuelles du protocole MOCP permettent de créer une représentation JSON ou YAML qui décrit l'ensemble des éléments de nos APIs. Ces éléments commencent par un modèle de données pour chaque composant auquel s'ajoute :

- ✓ URL des ressources accessibles de manière distante ;
- ✓ Code statut des réponses aux requêtes reçues ;
- ✓ Verbes http utilisés ;
- ✓ Paramètres des requêtes ou invocations ;
- ✓ Conteneur des réponses ;
- ✓ Informations pour les futurs tests unitaires ;
- ✓ Informations optionnelles à propos de la sécurité, l'usage de transactions ;
- ✓ Etc.

À titre d'exemple, la Figure 5.1 fournit une vue partielle de notre documentation de MOCP. Cette représentation interactive est un premier support de test des services décrits par notre API.

*Figure 5.1 Documentation du protocole MOCP*




Dans le cadre de ce travail, nous avons évalué la suite logicielle offerte par Swagger comme la plus adaptée pour une description systématique de chaque échange de message. Les types des messages sont décrits dans le but de définir les transformations de format depuis le client vers le service ou inversement et assurer l'interopérabilité entre les composants. De même, la définition d'une association ou mapping entre un échange http et un verbe de ce protocole plus une définition fine des entêtes http représentent une progression vers la définition d'une API REST pour chaque composant. Si les bonnes pratiques REST sont toujours en consolidation, notre démarche s'appuie sur deux piliers : notre spécification formelle du chapitre 3 ainsi que notre étude de propriétés temporelles du chapitre 4.

Nous avons ajouté quelques préceptes naturels tels que la simplicité de l'API REST et l'auto description afin que tout utilisateur potentiel ait accès à une documentation pouvant faire office de contrat de service, telle une suggestion d'utilisation. Enfin, notre documentation de MOCP autorise la génération d'un squelette d'implémentation dans différents langages et frameworks par l'emploi de Swagger Codegen. Nous avons choisi le framework Spring Boot pour les implémentations de services côté Cloudlet et le framework Restlet pour les implémentations de services appartenant à l'application mobile. Une démarche analogue est utilisée pour la génération des parties clientes de chacun des services invoqués. Le choix du framework d'implémentation côté service est sans effet sur celui des clients, par respect du principe d'interopérabilité. En revanche, pour un nœud donné, nous avons pris soin de minimiser le nombre de frameworks utilisés pour éviter les conflits de librairies.

# II. Architecture logicielle pour une application embarquée mobile

Haeng-Kon Kim a présenté l'architecture logicielle pour une application mobile comme un nouveau domaine [188]. Aujourd'hui, une application mobile envoie des requêtes soit à un serveur, soit à un poste de travail, soit à un périphérique mobile, mais est aussi prête à en recevoir. Ceci est particulièrement important dans notre étude de cas sur le transfert d'applications mobiles sur un réseau de Cloudlets. En raison des performances actuelles de la plate-forme Android, il est possible de choisir des communications http et de concevoir un composant de liaison basé sur un mécanisme de déportation. En outre, nous avons validé une approche à base de service REST pour les échanges. Nous avons choisi le framework Restlet, car il autorise le développement de services REST sur mobile de manière portable à celui d'une plate-forme JavaEE. Il a été utilisé pour tout échange entre les périphériques mobiles et contribue à une architecture orienté ressource (ROA).

Dans cette section, nous présentons les composants qui interviennent dans la construction de notre architecture logicielle pour une application mobile. Nous commençons par expliquer le rôle de la norme OSGi pour la mobilité de composants. Ensuite, nous proposons une construction d'applications autour du serveur OSGi pour la plate-forme Android. Puis nous détaillons les scénarios principaux du protocole MOCP pour mieux identifier le rôle de chaque composant.

## 2.1. Application à base de composants OSGi

La plate-forme de services OSGi se divise en deux parties : le framework OSGi d'une part et les services standards OSGi d'autre part. Les composants sont conditionnés dans un format spécifique. Cette sous-section décrit les principes OSGi utiles à la mobilité de composants ainsi qu'une architecture OSGi d'une application mobile.

### 2.1.1. Principes OSGi utiles à notre prototypage

La spécification OSGi [189] est entre autre basée sur l'emploi d'un registre de services qui permet aux fournisseurs de services de les déclarer explicitement avec leurs propriétés. D'autre part, les utilisateurs de ces services ont la possibilité de connaître les propriétés qui les intéressent. Ainsi, pour une interface de service, la version de celui-ci correspond à un ensemble d'opérations utiles aux clients. Il devient alors aisé de publier plusieurs versions d'un même service pour des clients différents. Certains souhaitent conserver une version passée pour des raisons de stabilité logicielle.

De plus des mécanismes de notifications associés à des événements du cycle de vie des composants permettent aux utilisateurs de réagir en s'intéressant aux changements d'une implémentation ou à la

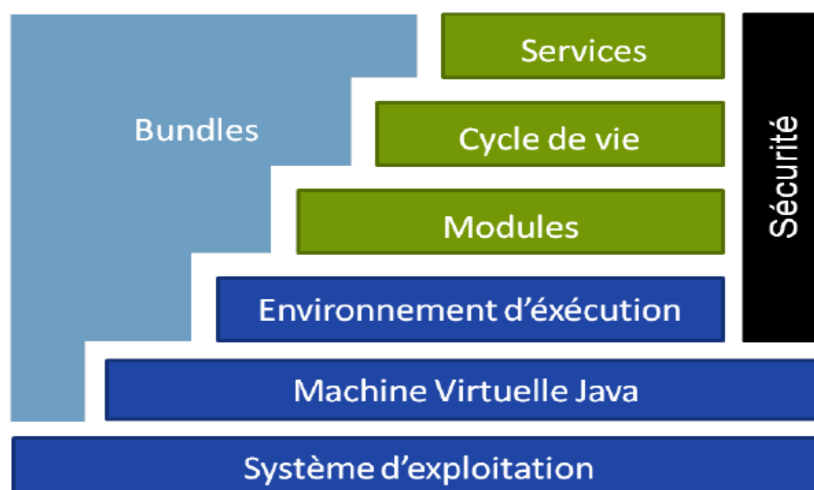
publication d'une nouvelle version de service. Ainsi, une application à base de composants OSGi peut offrir une palette de services différents au cours du temps. Par exemple, une Cloudlet peut permettre la déportation d'applications provenant de certains types de périphériques nomades, voire de certains types de backends applicatifs alors que la prise en compte d'autres types n'est pas encore acceptée.

La Figure 5.2 extraite de la documentation OSGi™ Alliance, présente les différentes couches logicielles et les entités OSGi comme suit :

- ✓ Les bundles sont les unités de déploiement de composants OSGi fournissant des services ;
- ✓ Les services mettent en relation les bundles suivant l'approche orientée service ;
- ✓ Le cycle de vie fournit les APIs pour installer, désinstaller, démarrer et arrêter les bundles ;
- ✓ Les modules définissent le chargement et la politique de partage des classes Java ; c'est-à-dire les classes privées aux bundles et les classes partagées avec d'autres bundles ;
- ✓ La sécurité gère les aspects non-fonctionnels liés à la sécurité ;
- ✓ L'environnement d'exécution définit quelles sont les méthodes et classes disponibles dans la plate-forme d'exécution.

On note sur ce diagramme que l'environnement OSGi s'appuie sur une machine virtuelle Java. Que celle-ci soit traditionnelle (du type JVM d'Oracle) ou embarquée (du type ART d'Android), il est indépendant des concepts mis en avant.

*Figure 5.2 Couches OSGi (source : OSGi Alliance, 2007)*

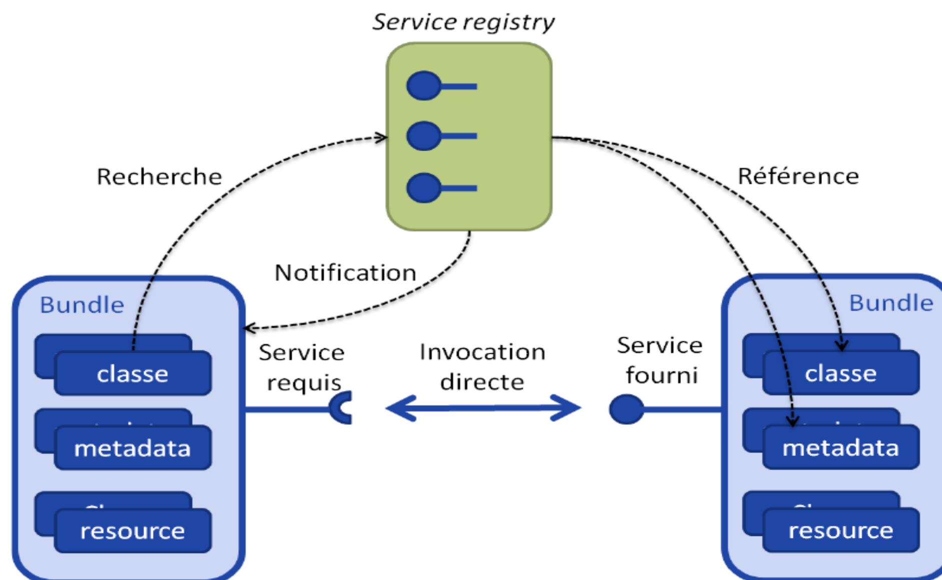



À la lecture de la Figure 5.2, nous notons qu'un bundle (ou composant OSGi) peut contenir des modules, des listeners d'événements OSGi ou encore des services OSGi qui dans notre cadre de travail seront aussi des services REST. Il est donc possible de dynamiquement exposer en REST un composant de notre architecture ou de stopper cette visibilité.

OSGi est basé sur une description de services entre deux entités et cette description est publiée dans un annuaire de service, appelé service Registry. Ainsi toute application structurée autour d'un serveur OSGi

tel qu'Apache Felix effectue des recherches dans son annuaire pour trouver un service à partir de son interface d'appel (voir Figure 5.3). Ainsi, un service utilisé peut changer au cours du temps.

*Figure 5.3 Approche orientée service dynamique OSGi*




Un tel schéma de recherche est particulièrement simple à mettre en œuvre pour la recherche d'un service par un client sur une même plate-forme (par exemple un périphérique mobile). En revanche, cette même recherche devient plus délicate lorsque le client et le service sont distribués sur deux plate-formes différentes. Par exemple, le client est sur un périphérique mobile et le service recherché est déployé sur une Cloudlet. Dans ce cas, il est indispensable d'équiper le serveur OSGi de la plate-forme supportant le service recherché, par la capacité (ou feature) D-OSGi. Cela signifie résoudre les dépendances entre client et services distants lors de l'exécution, ce qui revient à rendre accessible le service Registry du serveur (Figure 5.3) depuis le client distant.

### 2.1.2. Architecture OSGi d'une application mobile

Nous avons formalisé au chapitre 3 la structure d'une application mobile (équations 26 et 27) divisée en *FrontEnd* et *BackEnd* (applicatif). Ces deux composants représentent naturellement la partie IHM (Interface Homme Machine) et la partie applicative, candidate à la migration. Nous avons matérialisé les échanges entre ces parties via des Web services (canal *ws*) (équation 28).

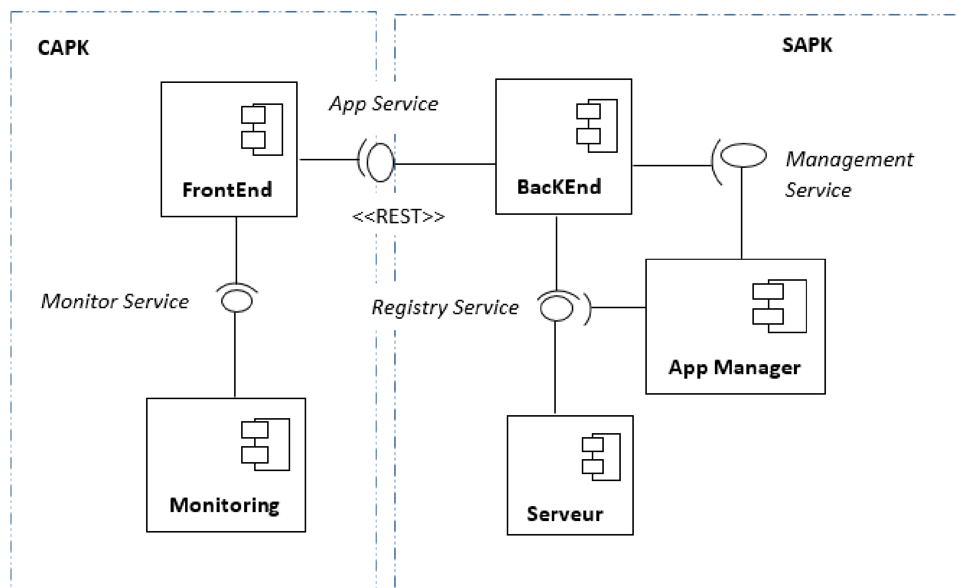
Ces deux composants respectent naturellement la spécification OSGi où la dépendance logicielle apparaît naturellement de manière déclarative. Mais ce ne sont pas les seuls composants présents. Nous avons défini le terme CAPK comme la partie résidente figée sur le périphérique nomade. Elle comprend le composant IHM pour les interactions avec l'usager ainsi qu'un composant de monitoring pour l'analyse de l'activité.

Nous avons défini le terme de SAPK comme la partie résidente sur le périphérique nomade gestionnaire de la déportation de backend applicatif. Elle comprend initialement un backend applicatif, un composant

gestionnaire de la déportation/importation de ce backend (nommé AppManager), ainsi qu'un serveur OSGi propre à notre application.

Le diagramme de composant ci-dessous décrit notre architecture logicielle pour une application mobile (Figure 5.4). Tous les composants mentionnés sont chargés au lancement de l'application mobile par le composant Serveur qui est une version du serveur Apache Felix personnalisé pour nos besoins.

*Figure 5.4 Architecture logicielle d'une application mobile*




L'appel REST entre FrontEnd et BackEnd est un appel local au périphérique mobile tant que le backend applicatif réside dans le périphérique. Il devient un appel distant après la demande de déportation. Cette requête est émise par le composant AppManager à une Cloudlet appartenant au voisinage de l'usager. Cette requête est matérialisée par un appel REST aussi vers un service de la Cloudlet décrit dans le protocole MOCP. Le composant AppManager est aussi impliqué lors du rapatriement d'un Backend depuis une Cloudlet vers le périphérique mobile. Ce composant expose aussi son API REST afin d'intervenir de manière concertée dans le protocole MOCP.

## 2.2. Implémentation mobile du protocole MOCP

Le protocole MOCP a pour but la mise en place de tous les échanges entre les différentes entités présentes dans notre architecture distribuée (chapitre 3, section IV).

À partir de l'analyse des échanges définis au chapitre 3 entre le périphérique mobile, les Cloudlets et le Cloud, nous avons procédé à plusieurs raffinements pour ajouter tous les détails techniques nécessaires à nos échanges REST.

### 2.2.1. Architecture globale avec les niveaux d'échanges

Notre architecture distribuée est un système de composants/services qui communiquent entre eux par échanges de messages. Il est donc nécessaire pour déclencher, réguler, inhiber le service ou déporter un composant d'offrir un interfaçage pertinent. Pour mieux aborder les communications dans notre architecture, le protocole MOCP implémente deux modes de communication à savoir une communication de type REST (via le protocole http) pour les échanges essentiellement avec d'autres nœuds du réseau et une communication de type Bus de message (via le bus Event) pour des échanges internes à une Cloudlet. Ce protocole implémente la programmation orientée par composant (POC).

Au niveau architectural, les échanges se font à quatre grands niveaux via le protocole MOCP. La communication par bus de message est adaptée à des sous-niveaux de notre architecture telle que les échanges entre une Cloudlet et les VDRs qui sont démarrées sur cette plate-forme. Une implémentation telle que RabbitMQ joue le rôle de médiateur entre les différents composants d'une Cloudlet. Le bus reçoit des requêtes que d'autres composants et/ou services viennent traiter. Le monitoring à ce niveau permet d'avoir une vue globale et permanente sur les composants/services par rapport à l'aspect asynchronisme et découplé du système. Notre framework admet un Dashboard permettant de détecter rapidement les problèmes.

Dans la sous-section suivante, nous allons décrire chacun des échanges entre les couples de composants et les propriétés associées.

### 2.2.2. Echanges entre les composants et propriétés du protocole MOCP

La Figure 3.5 et la Figure 2.13 montrent respectivement une architecture à trois niveaux et les composants qu'on retrouve sur nos périphériques mobiles. Les applications mobiles peuvent être créées ou déployées en utilisant plusieurs composants connectés l'un à l'autre chacun fournissant une fonctionnalité spécifique et permettant des interactions sans état. L'une des fonctionnalités qui nous intéresse ici est l'échange de composants, de messages ou de services. En outre, la gestion de composants doit être appliquée même si les composants sont sur un système embarqué.

Ces propriétés sont cruciales et assurent un meilleur cycle de vie pour les applications des utilisateurs. Le protocole http est utilisé pour véhiculer les messages sous un format textuel tel que JSON.

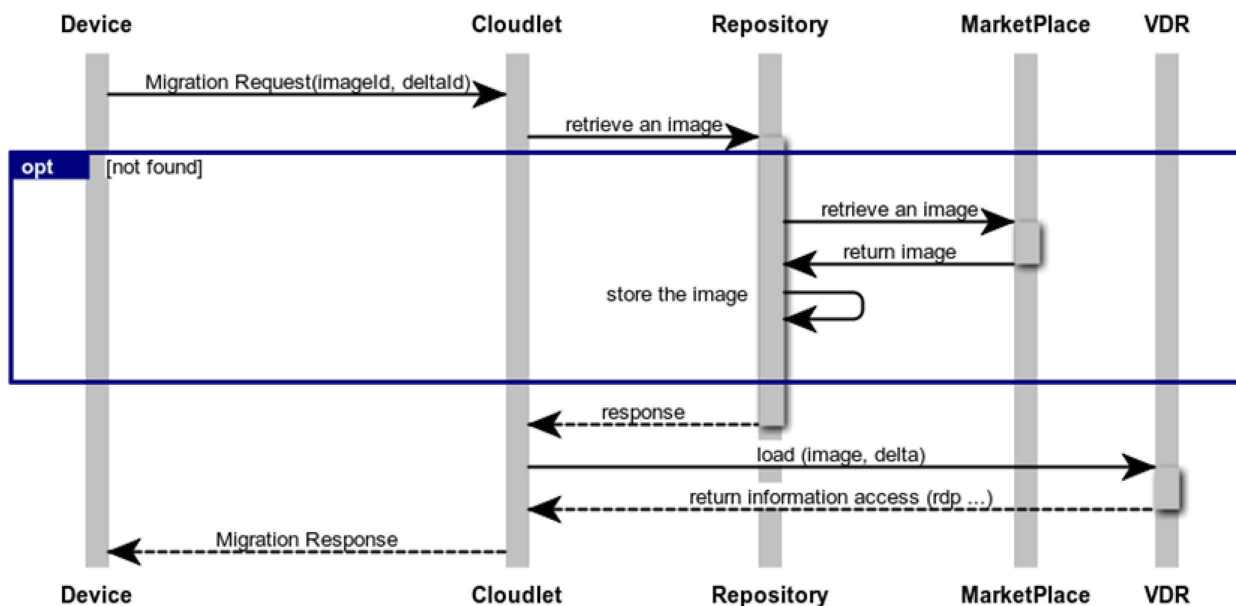
#### a) Echanges entre un périphérique mobile et une Cloudlet

Le noyau du protocole MOCP est au cœur des échanges entre le périphérique et la Cloudlet. Supposons qu'une connexion soit établie entre un périphérique mobile et une Cloudlet et que tous les échanges suivent le protocole MOCP. Pour cela, nous distinguons trois phases à savoir :

- ✓ Phase 1 : identification du périphérique mobile sur une Cloudlet et la mise à jour du Backend applicatif au niveau de la Cloudlet ;
- ✓ Phase 2 : virtualisation du périphérique mobile ou création de la VDR ;
- ✓ Phase 3 : migration de l'application mobile.

Le diagramme de séquence montre une demande de migration du périphérique mobile vers une Cloudlet.

*Figure 5.5 Demande de migration d'une application*




Comme nous pouvons le voir sur ce diagramme de séquence (Figure 5.5), les phases 1 et 2 sont déjà réalisées et certains composants communiquent par des requêtes http. Le périphérique (FrontEnd et BackEnd) envoie des requêtes de migration ou de mise à jour via une url depuis une application mobile. Les requêtes permettent d'obtenir une description des services proposés. Sur ce diagramme de séquence, à la demande de migration provenant d'AppManager (s'exécutant sur un périphérique mobile), la Cloudlet déclenche la recherche de l'image à instancier afin de recevoir le Backend applicatif. Les images déjà connues au sein d'une Cloudlet sont gérées par le composant Repository (ou Référentiel en Français). Suite à une réponse affirmative du Repository, l'image est chargée par une VDR existante répondant aux caractéristiques du périphérique mobile. Cette VDR est une machine virtuelle ou une virtuelle représentation du Device mobile. En revanche, si l'image n'est pas présente au sein du Repository alors, il faut la charger depuis un MarketPlace (ou dépôt publique d'images) présent dans le Cloud. Ce MarketPlace est un site de référence que l'on peut considérer comme un Docker Hub spécifique. Enfin une fois l'image chargée et démarrée, une réponse à la demande de migration est retournée à l'AppManager qui l'autorise à effectuer le transfert de l'état du composant BackEnd déporté dans la Cloudlet.

### b) Echanges intra-Cloudlet

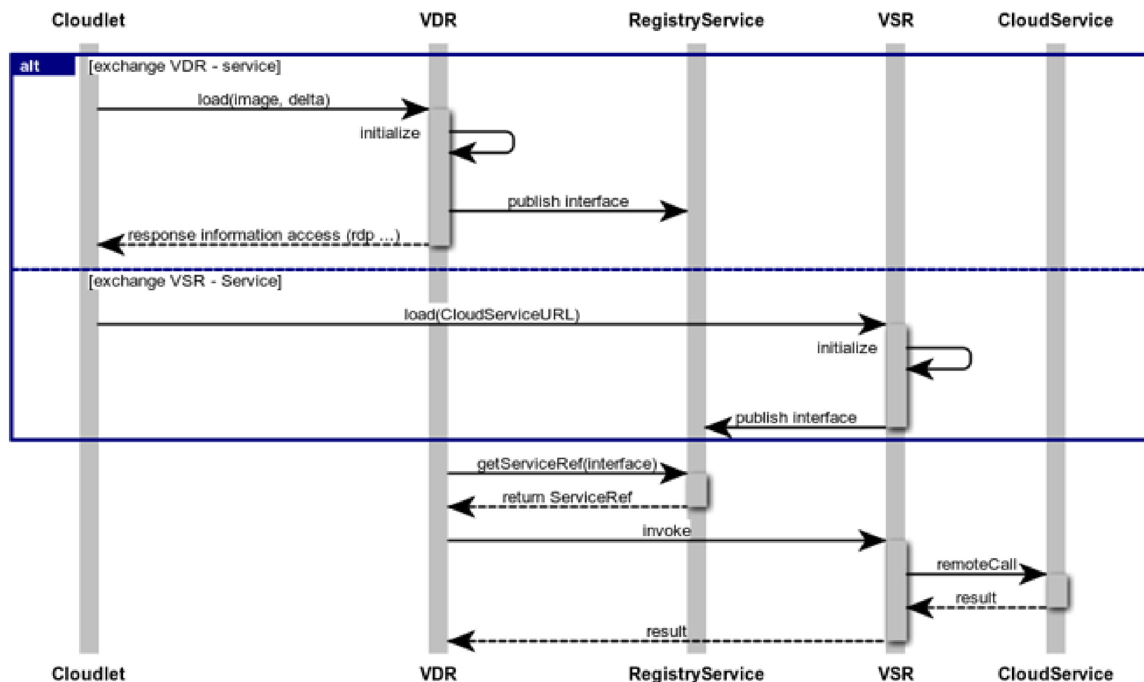
Les conséquences principales d'une migration d'application sont le chargement d'une image de l'application dans une VDR et l'utilisation du protocole RDP (Remote Desktop Protocol) à partir du périphérique mobile. Nous retenons deux phases :

- ✓ Phase 1 : invocation locale à une Cloudlet entre une VDR et une VSR ;
- ✓ Phase 2 : création de la VSR et son cycle de vie

Des interactions se produisent entre les composants d'une Cloudlet. Le backend d'une application mobile dans une VDR peut invoquer un service. Celui-ci peut être locale à la Cloudlet ou présent dans le Cloud. Alors cet appel s'effectue vers une VSR.

Le diagramme de séquence (Figure 5.6) montre le chargement d'une VSR lorsqu'une dépendance VDR est découverte. Le protocole http est utilisé de façon spécifique pour l'invocation de la VSR par la VDR.

*Figure 5.6 Interaction entre une VDR et une VSR*




Que l'invocation s'effectue entre deux VDRs ou depuis une VDR vers une VSR, l'approche est semblable. La Figure 5.6 illustre le chargement de la représentation virtuelle et sa publication dans le composant RegistryService de la Cloudlet. Une référence au service recherché est fournie à la VDR appelante. Dans ce scénario, la VSR joue le rôle de proxy d'un service du Cloud. Après l'invocation, un résultat est retourné à la VDR appelante.

Toute Cloudlet contient un service de recommandation qui peut être considéré comme un générateur de VSR. À la fin de l'interaction, la VSR publie son interface de service dans le composant RegistryService local à la Cloudlet actuelle et publie également sa référence dans le composant CloudRegistry du Cloud pour les autres Cloudlets.

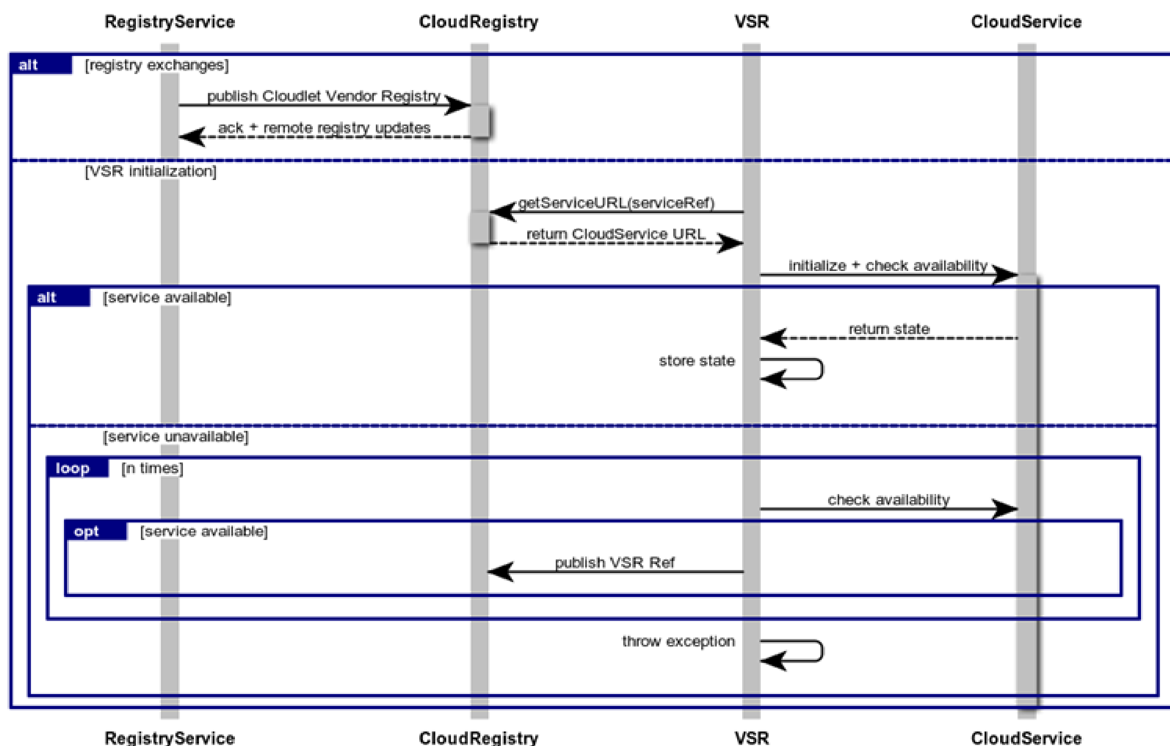
### c) Echanges entre une Cloudlet et un Cloud

Les échanges entre une Cloudlet et un Cloud surviennent lorsque :

- ✓ Une interface VDR est publiée dans un Cloud de sorte qu'une autre Cloudlet puisse mettre à jour son référentiel ;
- ✓ Une VSR locale à une Cloudlet appelle un service du Cloud ;
- ✓ Une VSR publie l'interface d'un service du Cloud dans un registre du Cloud.

Le diagramme de séquence représenté par la Figure 5.7 décrit les trois cas d'interactions. Le composant RegistryService d'une Cloudlet publie sa référence dans le composant CloudRegistry d'un Cloud. Ce dernier doit être accessible par toutes les Cloudlets du réseau afin que des échanges aient lieu. Dans le cas d'une VSR (Figure 5.7), la publication de sa référence dans le CloudRegistry est similaire. Mais l'usage d'une VSR au sein d'une Cloudlet est de permettre l'accès à un service du Cloud lorsque ce dernier est disponible (service available sur la Figure 5.7). En revanche, quand l'accès au service n'est pas possible alors si un résultat en cache est disponible ; il est retourné sinon un test d'accessibilité est effectué à plusieurs reprises avant le déclenchement d'une exception (service unavailable, la Figure 5.7).

*Figure 5.7 Interaction entre une Cloudlet et un Cloud*

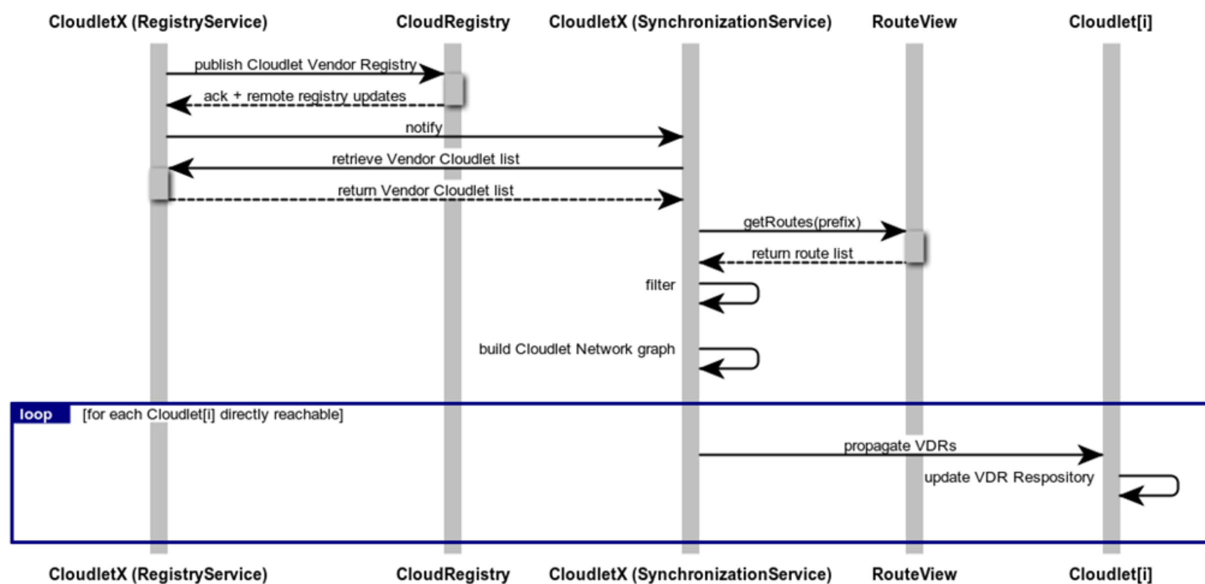



### d) Echanges entre les Cloudlets

Les échanges entre Cloudlets sont cruciaux lorsqu'un périphérique mobile se déplace d'une Cloudlet à une autre. Le diagramme de séquence représenté par la Figure 5.8 montre les interactions entre deux

Cloudlets. C’est la migration d’une VDR initiée par une Cloudlet. Le composant RegistryService de cette Cloudlet publie son interface au composant CloudRegistry du Cloud.

*Figure 5.8 Migration d'une VDR entre deux Cloudlets*




Les Cloudlets d’un réseau sont géographiquement déployées afin d’assurer la meilleure couverture pour les utilisateurs se déployant dans cette même zone géographique et donc certaines applications sont pour partie déportées dans une Cloudlet. À l’issue du déploiement des Cloudlets, chacune connait son voisinage. Comme observé sur la Figure 5.8, le composant RegistryService publie sa référence dans le composant CloudRegistry du Cloud. En réponse, il reçoit non seulement un acquittement de cette opération, mais aussi toutes les références des composants RegistryService des Cloudlets du réseau. Ces informations sont conservées par le composant SynchronisationService de la Cloudlet. Ces références de services permettent de construire un graphe des composants RegistryServices. Ensuite, l’étape suivante consiste à diffuser à chaque Cloudlet du voisinage immédiat, les références des VDRs actuellement actives sur la Cloudlet courante. Ceci permet de déclencher le chargement des images des VDRs par les Cloudlets voisines avant le déplacement de l’usager au sein du réseau. Quand celui-ci aura lieu, seul l’état de la VDR sera à transmettre.

Cette étape d’analyse du protocole achevée, la conception des APIs est à poursuivre. Comme évoqué en section 1, nous avons choisi l’approche Open Documentation pour chaque composant intervenant dans les différents scénarios.

# III. Architecture logicielle pour une Cloudlet

Les spécifications écrites au chapitre 3 ont mis en évidence la structure d'une Cloudlet. Afin de respecter ces exigences, nous avons choisi de construire une Cloudlet tel un système de composants OSGi. Cela nous autorise une grande dynamicité pour le chargement et le déchargement d'images de représentations (VDR, VSR, etc.). Une autre contrainte exige que les interfaces entre les composants soient déclaratives, stables, tel que le propose les fonctionnalités SCR (Service Component Runtime). Des couches du système peuvent être changées sans affecter le reste du système. C. Chih-Hung fournit son expérience sur la relation entre ces modèles de conception et la qualité du développement du service [190]. Dans la suite de cette section, nous expliquons l'apport de la spécification OSGi, pendant les exécutions et les configurations. Nous expliquons ensuite l'importation d'un composant depuis un autre composant appelé Repository et la construction du protocole MOCP via la suite logicielle Swagger.

## 3.1. Architecture basée sur un serveur OSGi Karaf

L'utilisation d'un framework OSGi au niveau des serveurs (serveurs d'application, bus, …) est une tendance forte pour les systèmes hautement dynamiques. Le développement d'applications autour d'un noyau OSGi améliore considérablement la qualité de service, la robustesse, la forte cohérence des modules grâce aux services au niveau de la conception et un faible couplage.

De plus, le framework OSGi est partagé par un grand nombre d'applications ayant une architecture en couches, composants et services. De plus en plus léger, OSGi nécessitant peu de ressources, n'influence pas sur les actions des applications (utilisation CPU, empreintes mémoire, etc.). Actuellement, des travaux sont mémés pour rendre accessible la technologie OSGi sous d'autres plateformes. Les conteneurs / framework OSGi sont implémentés par plusieurs projets Open source tels qu'Equinox, Oscar, Felix, etc.

En se basant sur les travaux de S. Bouzefrane [191] et sur notre structure de Cloudlet définie dans le chapitre 3, section IV, nous adjoignons trois autres éléments essentiels à notre architecture à savoir :

- ✓ Un serveur de composants OSGi ;
- ✓ Un composant permettant de résoudre les dépendances en respectant la spécification D-OSGi ;
- ✓ Un Repository de composants OSGi prêt à être chargés.

Le package DOSGi (OSGi à distance) permet de gérer les services OSGi à distance et aussi les interactions entre les frameworks OSGi répartis entre les périphériques mobiles et les Cloudlets. La spécification D-OSGi est définie en étant complètement indépendante du transport et du protocole.

Notre architecture de réseau de Cloudlet peut se décrire en une architecture à trois niveaux. Un serveur OSGi est installé sur chacun des nœuds (périphérique mobile – Cloudlet – Cloud). Au sein d'une Cloudlet, un serveur OSGi peut s'exécuter sur plusieurs VMs et ainsi masquer la chaine supportée par les demandes de déportation d'applications. Ainsi, l'élasticité, comme une des propriétés intrinsèques de la Cloudlet et du Cloud, est garantie.

Karaf[22] est un serveur d'application basé sur les spécifications OSGi. Il est fréquemment utilisé comme socle logiciel pour des projets plus volumineux comme ServiceMix.

Karaf est composé, entre autre d'un module appelé « Blueprint ». Blueprint fournit un framework d'injection de dépendances pour OSGi et a été standardisé par l'alliance OSGi. Il consiste à travailler avec la nature dynamique des composants OSGi, où les services peuvent devenir disponibles ou indisponibles à n'importe quel moment. Blueprint permet, via un container, de superviser l'état des composants dans le framework et d'accomplir des actions en fonction de leur état. Cette aptitude est particulièrement adaptée à la surveillance de Cloudlets.

Un composant est dit Blueprint à partir du moment où son descripteur respecte le schéma XML Blueprint. Ces fichiers XML sont localisés sous le répertoire OSGI-INF/blueprint. Ainsi, le serveur crée un container Blueprint basé sur le nom de ce composant. Ce container Blueprint est responsable :

- ✓ d'analyser les fichiers XML Blueprint ;
- ✓ d'instancier les composants ;
- ✓ de relier les composants entre eux ;
- ✓ d'enregistrer les services ;
- ✓ de chercher les références des autres services.

De même, iPOJO [192] est un projet inclus dans la pile logicielle d'Apache Karaf et dont le but est d'offrir un modèle de composant au-dessus du modèle de service de la plate-forme OSGi. Ce projet comble le manque de description de contrat de composant d'OSGi en proposant un rapprochement des paradigmes services et de composants. Ainsi, les fonctionnalités déployées sur une plate-forme OSGi sont encapsulées dans une notion de composant, dont les ports, décrits sous forme de service, sont liés par extension au mécanisme d'abonnement de service d'OSGi.

Le modèle de composant iPOJO résultant de cette association se focalise sur le développement d'une plate-forme pour la phase d'exécution Java. Pour cela, le modèle de développement inclut une notion étendue de résolution de service. Ainsi, il est possible d'étendre la façon dont la plate-forme lie les composants les uns aux autres afin de respecter les contraintes définies dans les descripteurs. Ce concept est essentiel pour ne pas migrer un Backend et toutes ses dépendances d'une plate-forme mobile à une Cloudlet, mais seulement le composant Backend.

[22] http://felix.apache.org/site/apache-felix-ipojo.html

## 3.2. Démarche logicielle pour la construction de composants OSGi

À partir de l'ensemble des scénarios dont les principaux ont été présentés en section 3.1, nous avons choisi une étape de raffinement à l'aide du langage Swagger. Cette étape peut se voir comme un pas en direction d'une réalisation par l'usage de Web Services REST.

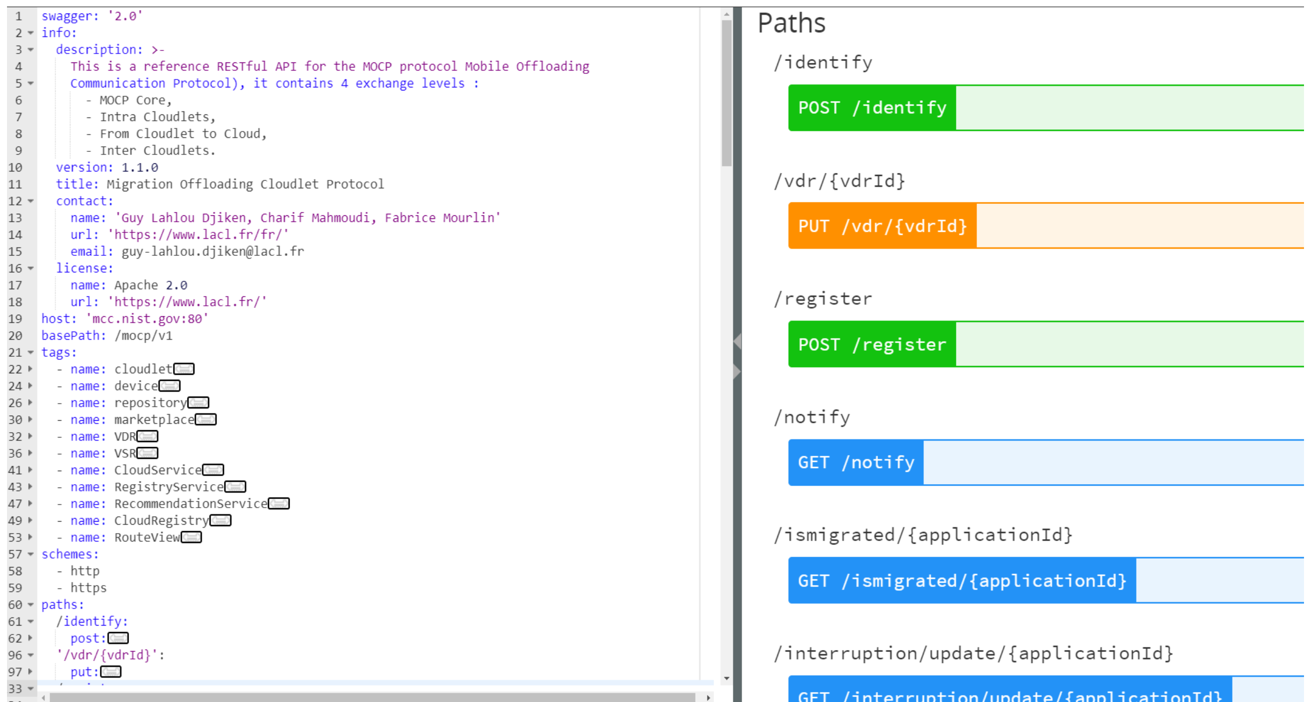
Dans ce but, nous avons relu l'ensemble des scénarios pour en assurer la cohésion. Puis nous avons débuté la réécriture de notre spécification MOCP en Swagger par la définition du modèle de données. Celle-ci décrit essentiellement les types de données échangées lors des interactions ainsi que les cas d'erreur. Ensuite, chaque interaction donne lieu à la définition d'une opération d'un Web service REST appartenant à la classe de l'objet recevant cette requête.

Dans cette sous-section, nous expliquons notre implémentation du protocole MOCP via la suite logicielle Swagger et du principe d'importation de composants depuis un Repository.

### 3.2.1. Conception du protocole MOCP via l'outil Swagger

L'outil Open source Swagger nous a permis de concevoir, construire, documenter et consommer nos API REST. Il est composé de trois outils (Swagger Editor – Swagger UI – Swagger Codegen) permettant le développement sur l'ensemble du cycle de vie du protocole MOCP, de la conception et de la documentation, au test et au déploiement. L'outil permet la génération d'une couche de services REST qui appelle des composants Karaf qui sont le cœur de la Cloudlet. Nous pouvons citer quelques-uns de ces composants : CloudRepository écrit via l'API OpenStack pour la gestion des images, ServiceRepository qui emploie l'API OpenStack, AppManager pour la gestion des applications déportées. La Figure 5.9 montre un état de notre spécification OpenAPI version 2.0 ayant les informations de description de notre API, la version 1.1.0, le titre, les contacts et bien d'autres en partie gauche et une liste de Paths (chemins) en partie droite.

*Figure 5.9 Initialisation du protocole dans Swagger UI*




Il s'agit du cartouche initial pour la construction de notre API REST MOCP. En d'autres termes, c'est l'API RESTful de référence pour le protocole MOCP. Il fournit la définition du protocole support (http), url de base pour chaque composant ainsi que des précisions sur le format des échanges (application/json) ainsi que la version courante de ce travail (1.1.0) et les sources de référence.

Nous avons annoncé comme contrainte que notre API devait être RESTful afin de faciliter l'usage des utilisateurs dans la découverte des différentes URI d'appel. Aussi, une documentation peut sembler superflue pour les futurs utilisateurs. Mais dans notre contexte, cette documentation interactive a pour but de guider la génération d'un squelette de code ainsi que le passage des tests unitaires. Nous pouvons conclure que cette réécriture de spécification intervient davantage dans notre démarche logicielle qu'à des fins de documentation utilisateur.

La Figure 5.9 illustre la première partie de ce travail de réécriture avec la définition des types de données échangées, ainsi que des messages d'erreurs. Ils forment la base pour la génération ultérieure des DTO (Data Transfer Object), autrement dit des classes qui effectuent la sérialisation des données en JSON dans l'usage du protocole MOCP. Associé à la notion de tag Swagger, il est alors naturel de définir un tag par composant de notre architecture de Cloudlet afin d'obtenir une interface de service par composant.

Ensuite, nous décrivons en détail chaque URI d'échange ainsi que les verbes associés. Par exemple, la première méthode (*POST/identify*) effectue une identification d'un périphérique mobile dans une Cloudlet ayant comme paramètre l'IMEI (International Mobile Equipement Identity) du périphérique. La sous-section 2.2.2 explique les échanges entre un périphérique mobile et la Cloudlet et nous définissons ensuite la payload de la requête ainsi que les réponses possibles en cas de succès ou en cas

d'échec. Toutes ces informations sont fournies au format YAML (Yet Another Markup Language). Le typage des éléments présents dans la requête et dans les réponses potentielles provient des définitions de types de données construites initialement.

*Figure 5.10 Méthode POST /identify*

La Figure 5.10 illustre non seulement l'aspect documentation, mais aussi test avec la possibilité de soumettre via un bloc de données au format JSON à l'opérateur d'identification d'un périphérique mobile.

Dans la version actuelle, nous n'avons pas utilisé de token de sécurisation dans les entêtes http avec les propriétés X-Auth-Token et security definition qui permet de configurer la méthode d'authentification sans passer par des attributs lisibles dans le corps de la requête. Cela apparait indispensable pour les échanges entre périphériques et Cloudlets afin d'assurer l'indépendance des échanges. Ce sera l'objet d'une prochaine version de notre API RESTful.

Toutes les interactions sont décrites avec la même précision. Nous ne présentons dans ce document que quelques exemples, tels que :

- ✓ la méthode (*GET/notify*) qui permet la notification de la Cloudlet d'une modification par le composant Repository. La Figure 5.11 illustre cette méthode avec les différents paramètres présents dans l'URL par respect de la norme http.

*Figure 5.11 Méthode GET /notify*

```yaml
/notify:
  get:
    tags:
      - cloudlet
    summary: The repository notifies the cloudlet about a change
    description: >-
      The repository notifies the cloudlet about the new delta, it just
      received (sequence diagram 1, stimulus 4)
    consumes: []
    produces:
      - application/json
    parameters: []
    responses:
      '200':
        description: The cloudlet is now in notified by the repository
        schema:
          required:
            - applicationId
          properties:
            applicationId:
              type: string
'/ismigrated/{applicationId}':
  get:
    tags:
      - cloudlet
    summary: The cloudlet tests whether a mobile application has moved.
    description: >-
      The cloudlet tests whether a mobile application has moved from a mob
      device (sequence diagram 1, stimulus 5)
    consumes: []
    produces:
      - application/json
    parameters:
      - name: applicationId
        in: path
        description: >-
          The identifier of the mobile application which has been updated
          previously
        required: true
        type: string
    responses:
      '200':
        description: The cloudlet is now in notified by the repository
        schema:
          required:
```

cloudlet ×

Paths

/identify

POST /identify

/register

POST /register

/notify

GET /notify

cloudlet

Summary

The repository notifies the cloudlet about a change

Description

The repository notifies the cloudlet about the new delta, it just received (sequence diagram 1, stimulus 4)

Responses

| Code | Description | Schema |
|---|---|---|
| 200 | The cloudlet is now in notified by the repository | { applicationId: string * $folded: } |

Try this operation

/ismigrated/{applicationId}

GET /ismigrated/{applicationId}

/vdr/{vdrId}/migrate

- ✓ la méthode POST/vdr/{vdrId}/migrate initie la migration de Backend applicatif d’un périphérique mobile vers une VDR d’une Cloudlet ayant en paramètre l’identifiant de la VDR où une application mobile sera virtualisée. La Figure 5.12 donne la documentation liée à cette méthode.

*Figure 5.12 Méthode POST /vdr/{vdrId}/migrate*

```yaml
'/vdr/{vdrId}/migrate':
  post:
    tags:
      - cloudlet
    summary: >-
      it performs an application migration from a mobile device into a virtual
      device representation (VDR) in a Cloudlet
    description: >-
      it performs an application migration from a mobile device into a virtual
      device representation (VDR) in a Cloudlet (sequence diagram 2, stimulus
      1)
    consumes:
      - application/json
    produces: []
    parameters:
      - name: vdrId
        in: path
        description: >-
          This is the identifier of the vdr where a mobile application will be
          virtualize.
        required: true
        type: string
      - name: body
        in: body
        required: false
        schema:
          $ref: '#/definitions/ApplicationMigration'
    responses:
      '201':
        description: >-
          The request has been fulfilled and resulted in a new resource being
          created.
        schema:
          $ref: '#/definitions/ApplicationAccess'
      '204':
        description: >-
          Indicates that the resource has not been modified since the version
          specified by the request headers If-Modified-Since or If-None-Match.
          This means that there is no need to retransmit the resource, since
          the client still has a previously-downloaded copy.
        schema:
          $ref: '#/definitions/ApplicationAccess'
      '404':
        description: >-
          The requested resource could not be found but may be available again
          in the future. Subsequent requests by the client are permissible.
        schema:
          $ref: '#/definitions/Error'
/virtualize:
```

- ✓ la méthode *POST/virtualize* permet le chargement d'une VDR et son démarrage dans la Cloudlet réceptrice de la requête.

Notre spécification est également enrichie de raffinements concernant la génération du squelette de code ainsi que des tests et de leur organisation. Ainsi la zone « info » de la cartouche initiale comprend une liste de propriétés values afin de piloter la création du squelette de code par Swagger CodeGen appartenant à la suite logicielle Swagger. À titre d'exemple, nous citons les propriétés suivantes :

- ✓ useControllerPostfix : à la valeur « true », le nommage des classes qui jouent le rôle de contrôleur dans le design pattern MVC sont suffixées par Controller ;
- ✓ collectParameters : à la valeur « true », les paramètres de toute requête http sont passés sous la forme d'une collection, une map pour un objet passé en paramètre par exemple ;
- ✓ enableAdditionnalModelProperties : à la valeur « true », des propriétés supplémentaires sont ajoutées à la réponse JSON telle que les nouvelles URIs accessibles suite à la requête MOCP reçue. Celles-ci sont regroupées derrière la clé « link ».

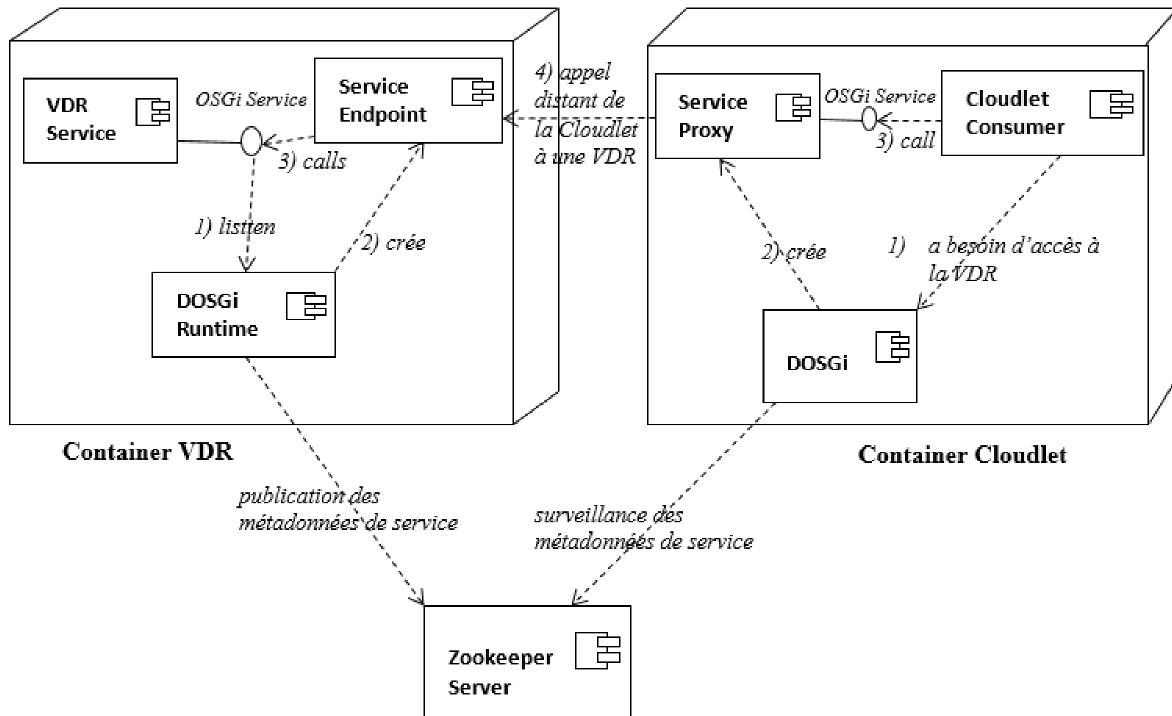
### 3.2.2. Squelette de code issu de la description Swagger

L'outil graphique ne permettant pas de configurer aisément la phase de génération de code, il est nécessaire d'utiliser un outil d'automatisation de tâches tel qu'Apache Maven accompagné du plugin Swagger. Par cette démarche, nous avons obtenu un squelette de code utilisant le framework Spring Boot pour la partie implémentation.

Le format de ce squelette généré est aussi un projet Maven configuré pour que la création des modules OSGi soit aisée. Ainsi, l'évaluation de la phase package de ce projet Maven généré fournit un livrable (ou jar file) prêt à être déployé. À ce projet, nous avons ajouté les requêtes afin que l'API RESTful de MOCP puisse invoquer les composants OSGi techniques de MOCP. Afin de ne pas être en situation de modifier manuellement du code généré, nous procédons par la création de sous-classes et la redéfinition de méthodes afin que celles-ci soient ensuite invoquées par polymorphisme lors de l'exécution.

L'architecture d'une Cloudlet est basée sur le serveur Apache Karaf illustré à la Figure 5.13.

*Figure 5.13 Diagramme de composant illustrant une interaction de la figure 5.6*




Cette figure représente une partie de notre architecture mise en place dans une Cloudlet. Elle illustre l'interaction entre une Cloudlet et une de ses VDRs telle que décrit à la Figure 5.6 pour la partie MOCP au sein d'une Cloudlet.

La structure d'une Cloudlet est naturellement distribuée. Nous avons mis en place les composants DOSGi au sein de serveurs Karaf. Ainsi quand un composant tel que la « VDR Service » est déployé dans un container Karaf, il émet ses métadonnées à destination du composant DOSGi local qui les publie dans un annuaire distribué géré par un agent ZooKeeper. Le composant DOSGi démarre un composant « Service Endpoint » dans le but d'attendre les futures requêtes qui seront propagées vers la « VDR Service ». Pour rappel, ce dernier comprend une partie du squelette généré par Swagger Codegen et modifié pour déclencher des composants techniques utilisant l'API OpenStack afin de lancer les machines virtuelles (image de périphérique mobile).

De même lorsque le composant « Cloudlet Consumer » est déployé dans un autre container, un système analogue de composants est installé. Dans le cas du scénario de la Figure 5.6, ce composant invoque le composant « VDR Service », il y a donc une étape de résolution de dépendance ou déploiement du composant « Cloudlet Consumer », car il possède la dépendance dans son descripteur de déploiement. Cette dépendance est reçue par le composant DOSGi local qui la propage auprès de l'annuaire distribué géré par ZooKeeper. Après avoir reçu les métadonnées du service recherché (« VDR Service » dans notre exemple), le composant DOSGi crée un service Proxy ou requêteur idéal. Celui-ci reçoit la requête du composant « Cloudlet Consumer », l'adopte pour la propager au composant « EndPoint Service ». Ces deux composants techniques s'occupent du routage de message entre les VDR et Cloudlet.

De même que précédemment, le composant « Cloudlet Consumer » contient initialement le code généré de la spécification Swagger permettant d'émettre des requêtes à destination d'une VDR. À ce code, ont été injectées les références afin que les URLs ne soient pas figées dans le code source de ce composant. Un principe analogue a été mis en place pour tous les composants du scénario de la Figure 5.6, nous permettant ainsi d'effectuer des tests d'intégration pour chaque scénario.

### 3.2.3. Validation du déploiement des composants d'une Cloudlet

Nous avons construit des scénarios pour chaque module du protocole MOCP. Ainsi, l'étape de validation suit la démarche de construction de nos scénarios. Afin d'adopter une approche de test d'intégration automatisée, nous avons fait le choix de l'outil SoapUI (SmartBear) qui autorise les requêtages en SOAP et REST. Ainsi, tout déclenchement d'un scénario s'effectue par l'émission d'un paquet JSON à l'URI indiquée dans la documentation Swagger. Pour conclure, l'éditeur de documentation interactive de Swagger autorise la soumission d'une requête avec évaluation d'assertions sur la réponse (propice aux tests unitaires), alors que SoapUI autorise l'écriture de scénarios de test (propice aux tests d'intégration).

Lorsqu'une campagne de test est effectuée, l'analyse approfondie des résultats est basée en premier sur le rapport des tests avant d'approfondir par l'analyse de fichiers de log en cas d'anomalies. Les premières métriques sont la couverture de test indiquant comment faire évoluer les données d'entrée pour parcourir un pourcentage de code supérieur. Des plugins supplémentaires sont alors mis en place comme celui de sonarQube pour l'étude de la couverture de test ou PMD pour la qualité du logiciel en phase de test.

a) Validation unitaire

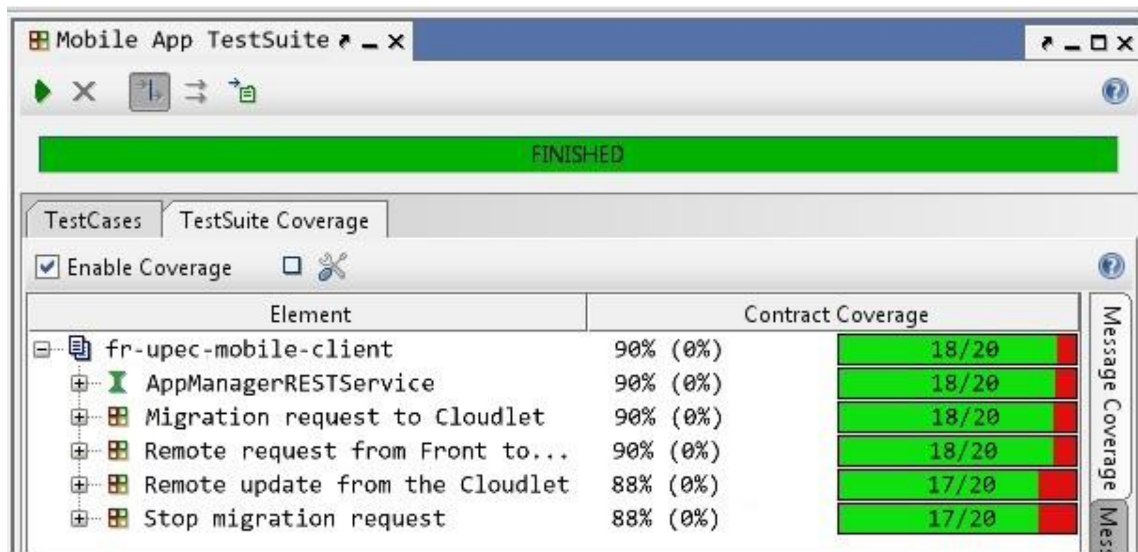
À titre d'exemple, nous décrivons ci-après le test unitaire d'une opération REST, illustrée sur la figure 5.5 au cours d'une demande de migration d'une application mobile. L'application manager présent sur le périphérique mobile émet une requête Post à destination de sa Cloudlet, à l'URI /vdr/{vdrId}/migrate avec en paramètre un ensemble d'informations utiles pour que le Backend de l'application puisse continuer après cette étape de migration. La description des données est fournie dans la spécification Swagger. Mais dans le cadre de test unitaire, nous avons besoin de comparer la réponse réelle de cette requête avec la requête attendue dans le cas de notre étude. Afin d'automatiser cette demande comparative entre le réel et l'escompté, nous avons ajouté à notre spécification Swagger, des informations pour le passage des tests (attributs préfixés par 'x-') voir Tableau 5.1. Un ensemble d'extension existe au travers de l'outil de validation http nommé Gavel [193]. Dans notre exemple d'échange entre l'AppManager et la Cloudlet, nous avons ajouté à notre spécification au niveau de l'opération POST associée à l'URI précédente, une définition de notre test unitaire simplifié pour ce document.

*Tableau 5.1 Test unitaire simplifié pour l'opération POST*

```
"x-unitTests" :[{
                "request" : {
                  "method" : "POST",
                  "uri":"/vdr/18541/migrate"
                },
            "expectedResponse":{
              "statusCode":"201",
              "headers":[{
                  "content-type":"application/json"
                }],
                "body":{
                    "applicationName":"mobile-test",
                    "deltaIdentifier":"41021"
                    }
                }
            "x-testName": "demande de migration",
            "x-testEnabled":"true",
            "x-testShouldPass": "true",
            "x-testDescription":"Post une demande de migration
                                pour un Backend applicatif de l'application mobile-test"

    }],
```

De cette manière, plusieurs objets peuvent être spécifiés dans ce tableau afin de préparer plusieurs cas de test. Parmi ceux-ci certains peuvent être considérés comme plus importants que d'autres. Des propriétés Gavel autorisent à nommer et à décrire les cas de test afin de disposer au final d'un rapport plus riche. Le passage de ces tests unitaires fournit des rapports individuels qu'il est possible d'agréger lors d'une suite de tests. SoapUI permet de visualiser ces tests unitaires, de représenter graphiquement les résultats pour ainsi naviguer au plus près des scénarios qui ont levé des anomalies (Figure 5.14).

*Figure 5.14 Simple test Suite pour agréger les cas de test*




Le bilan de cette phase de test montre qu'un pourcentage élevé de code des divers composants a été couvert. De plus, les tests annotés prioritaires ont tous validés le code que nous avions déployé sur le serveur Karaf.

b) Validation de l’intégration des composants

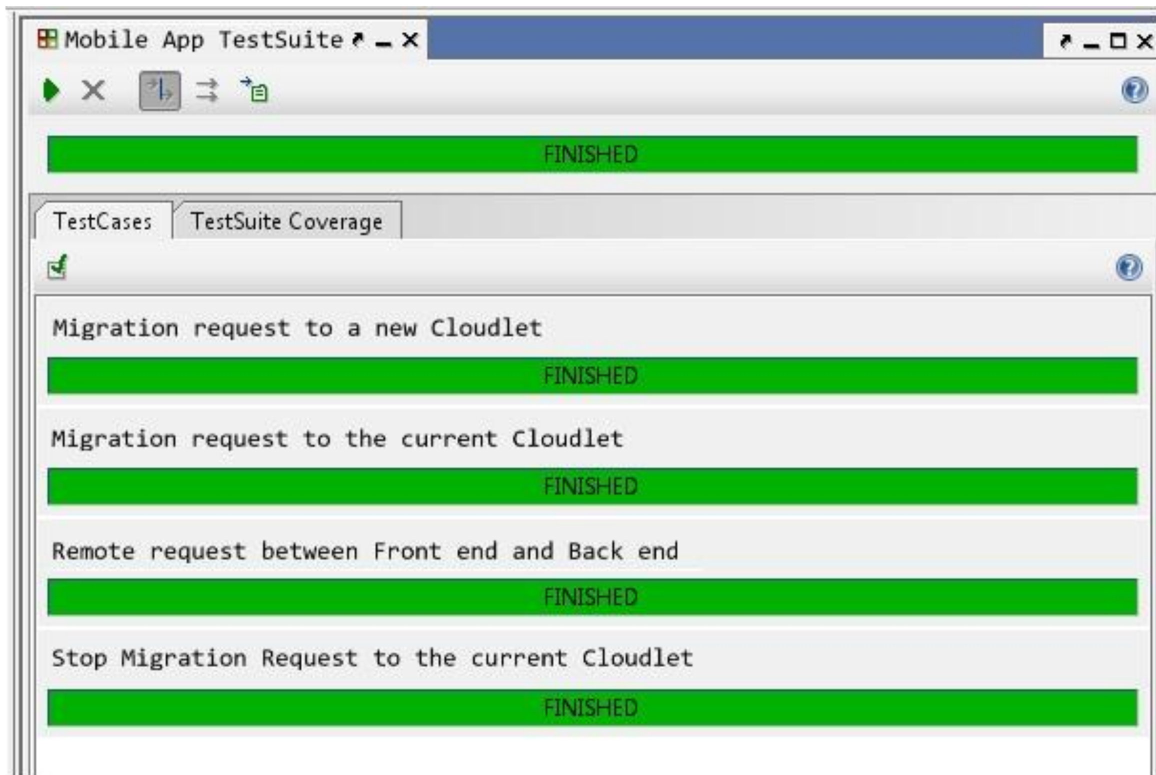
Nous avons utilisé l’outil SoapUI pour plusieurs aspects tels que la construction de suite de test, la construction de mock de service pour isoler certains composants lors des tests unitaires, mais aussi la construction de test d’intégration. L’écriture du script de test est proche de la construction de mock par programmation. Pour rappel un mock de service contient un nombre quelconque d’opérations. Celles-ci peuvent simuler des opérations de différents services REST. Chaque mock opération contient aussi un nombre quelconque de messages qui sont retournés à l’appelant lorsque l’opération en question est invoquée. Ce message, à renvoyer est nommé mock réponse. Il est choisi de plusieurs manières : à partir d’une séquence de messages prédéterminés, d’un choix aléatoire sur cette même séquence ou par évaluation d’un script. Ainsi, l’évaluation d’un script permet de démarrer l’ensemble des composants utiles à un scénario de test d’intégration, de configurer chacun d’eux afin qu’il puisse répondre de manière convenue, puis de lancer la première requête qui déclenchera l’ensemble des interactions. Bien entendues, les réponses retournées dépendent des requêtes reçues et le langage de script de SoapUI (Groovy) simplifie l’accès aux différentes parties d’une requête pour constituer la réponse attendue. Par exemple le mot-clé requestContext est utilisé pour stocker toutes les données propres à la requête, sous la forme de couples clé, valeur. Il est alors aisé d’y accéder telle une table afin de constituer une réponse. Pour accéder au contenu de la requête elle-même, il est souvent utile d’ajouter un langage de requêtage tel que XPath afin d’extraire une partie du body de la requête POST. Ainsi, il devient aisé de sélectionner une réponse à une requête à partir de celle-ci et de compléter cette réponse grâce aux informations.

À partir du scénario défini à la Figure 5.5, nous avons défini un script de test utilisant l’ensemble des composants appartenant à une Cloudlet. Ce script, écrit en Groovy, respecte une écriture classique de langage à objets. Un script est composé d’étapes, nous appliquons cette structure à notre scénario où une flèche représente un appel REST. Chaque étape a accès à trois variables déjà initialisées par SoapUI : log pour tracer les événements du scénario, context pour la gestion des données lors de l’exécution et plus particulièrement les informations provenant des étapes et testRunner qui est le moteur de test déclenchant chaque cas test. Ainsi, notre scénario d’intégration se découpe en étapes :

- ✓ Lire des informations de description du test (information sur le périphérique, sur la VDR employée, sur l’image à lancer dans la Cloudlet, …) issues d’un fichier csv ;
- ✓ Invoquer le composant AppManager afin qu’il émette une requête POST à destination du composant Cloudlet pour qu’elle traite la demande de migration */vdr/18541/migrate* auquel s’ajoute un paquet JSON ;
- ✓ Surveiller le composant Cloudlet afin d’observer l’émission d’une requête GET à destination du composant Repository pour une recherche d’image */search /18541* auquel s’ajoute une entête http spécifique ;
- ✓ Récupérer la réponse du composant Repository afin de constituer la réponse du composant Cloudlet au composant AppManager (représenté sur le scénario par Device) ;

- ✓ Surveiller le composant AppManager afin d'analyser la réponse finale et poursuivre cette demande de migration par le transfert de l'état du backend applicatif.

*Figure 5.15 Bilan de Test*




Si l'écriture de tels scripts représente une étape fastidieuse à cause de la mise au point des tests d'intégration, cela reste accessible par l'usage de nombreux patterns tels que MockObject. Chaque script valide la cohérence d'une fonctionnalité dans son ensemble sur une partie de notre architecture. Dans notre exemple, il s'agit du nœud où la Cloudlet est déployée. L'implémentation des API REST pour chaque composant entraîne de traverser toutes les couches de l'application. Le but n'est pas de tester les frameworks utilisés, mais l'utilisation qui en est faite au sein de chaque scénario. Ainsi, dans le test d'intégration précédent, l'observation du succès de son déroulement montre non seulement l'enchaînement des requêtes qui a été présenté initialement mais la recherche d'une VDR par son identification, le test de son état et le lancement d'une image applicative. L'ensemble de ces observations est automatisé grâce au langage de script employé.

Nous avons automatisé le passage de ces tests par l'emploi de l'outil Maven et la définition de profil. Il est ainsi possible d'exécuter les tests d'intégration avec un profil développeur pour le regroupement des composants sur une même plate-forme avec une priorité à la rapidité d'exécution de ces tests. Un autre profil nommé recette effectue les tests sur une architecture réellement distribuée pour la Cloudlet. Ce second profil est adopté à la collecte de mesures pour évaluer les coûts des différentes opérations du protocole MOCP et plus particulièrement l'étape de migration.

Un rapport de test d'intégration (format textuel) fournit les principaux résultats de validations du protocole MOCP. Un exemple d'une extraction de ce rapport est donné dans la Figure 5.15, il indique les étapes MOCP franchies et remplit une partie des résultats de la matrice d'intégration.

# IV. Cas d'étude

La diffusion des données est un problème commun dans le monde de la programmation mobile. Ce problème peut se considérer de manière orientée, c'est-à-dire soit d'un serveur d'information vers les périphériques concernés : c'est le cas des mises à jour par le biais de notification. À l'inverse, il peut s'agir de remontées d'informations depuis les périphériques mobiles vers un serveur : c'est le cas d'applications de notation qui s'intéressent à l'avis des utilisateurs nomades sur un sujet précis tel que la restauration ou le trafic routier. Notre étude de cas vise à montrer l'impact de la migration d'une application et/ou des données d'un périphérique mobile et/ou d'une Cloudlet à une Cloudlet et/ou un périphérique et sa gestion dans un périphérique mobile.

Partant d'un réseau de Cloudlets dans une entreprise, la question est celle-ci : comment permettre aux personnels d'utiliser leurs applications en mode déporté au sein de l'entreprise ? Éventuellement réimporter l'application qui avait été déportée. On souhaite illustrer les propriétés temporelles décrites au chapitre 4, mais aussi dimensionner les coûts des opérations au sein du réseau de Cloudlets.

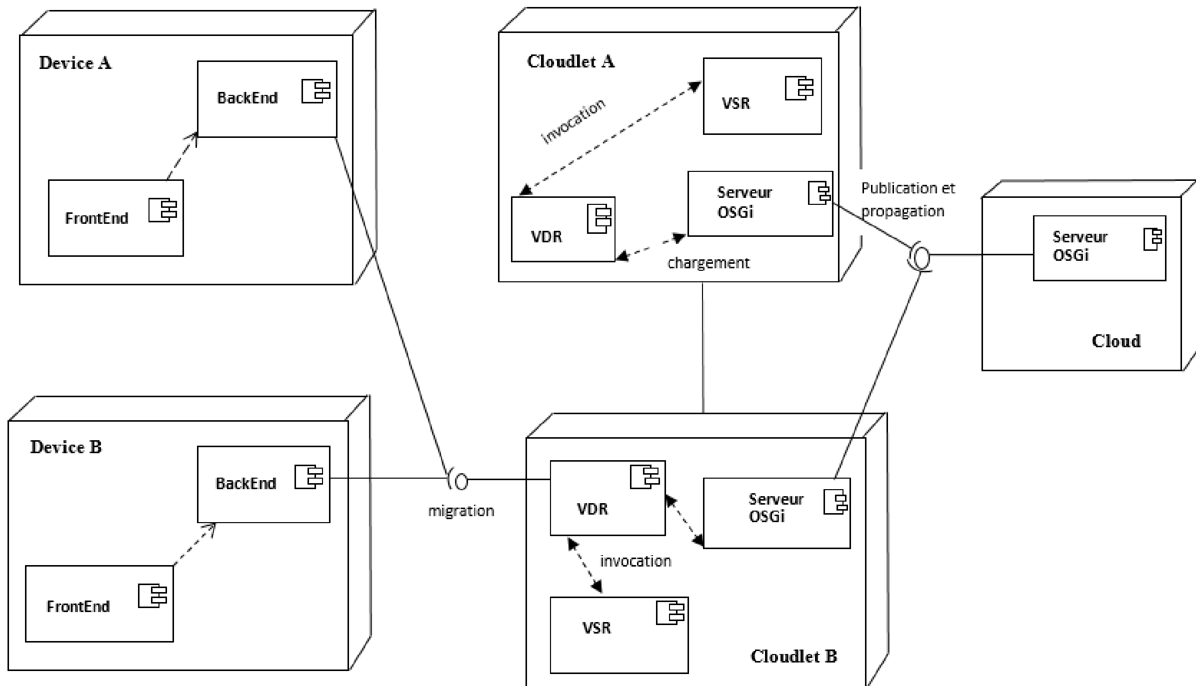
## 4.1. Présentation de la configuration

Dans notre cas d'étude, nous considérons des employés d'une entreprise qui utilisent leurs périphériques mobiles pour effectuer des opérations sur des photos prises depuis leur périphérique ou simplement visualisées sur celui-ci. Ces opérations sont de l'ordre de la gestion simple (un CRUD) ou de l'ajout de méta-informations (un avis sur l'image ou son utilisation).

### 4.1.1. Phase de déploiement

Nous avons illustré la structure de notre système distribué à plusieurs reprises. Dans cette étude, nous allons utiliser notre prototypage. Ainsi, la première phase consiste à déployer l'ensemble des artefacts sur les périphériques mobiles et le réseau représentant celui de l'entreprise. La majorité est construite en tant que bundle OSGi ou comporte un serveur OSGi embarqué. Ensuite, les services sont démarrés afin que le protocole MOCP soit actif.

*Figure 5.16 Diagramme de déploiement de notre cas d'étude*




La Figure 5.16 fournit un diagramme de déploiement simplifié où figurent deux périphériques mobiles, deux Cloudlets et un nœud représentant un Cloud partagé par les deux Cloudlets.

De plus, les deux périphériques mobiles sont dans le voisinage de la même Cloudlet B. Une telle configuration autorise le scénario suivant :

- ✓ Chaque périphérique mobile supporte l'application nommée ImageScore permettant à un usager de fournir son avis sur les images de son périphérique ;
- ✓ Chaque Cloudlet déploie derrière un serveur Apache Karaf l'ensemble des bundles vus précédemment pour la gestion de son Repository, ses images, etc.

La liste des bundles déployés n'est pas représentée sur ce diagramme afin qu'il puisse être lisible sur une page. À la suite de ce déploiement, les nœuds sont connectés au réseau et partagent ainsi une zone d'adressage commune. Une première validation de la phase de déploiement s'obtient en observant depuis chaque périphérique mobile que les deux Cloudlets sont visibles dans le voisinage réseau.

Une seconde validation s'obtient en listant les services disponibles sur chaque Cloudlet, par une requête au serveur OSGi sous-jacent.

### 4.1.2. Description du matériel employé pour les tests

Les périphériques mobiles utilisés par les usagers testeurs pour prendre et modifier des photos sont sous le système d'exploitation Android 7.1 et Debian Kit pour avoir une compatibilité avec le système de base sur Android et le système Debian. Ceci permet l'exécution d'une implémentation d'Apache Felix déxifiée sur la plate-forme Android. Dues aux contraintes de déxification, la version minimaliste

d'Apache Felix utilisée est 1.4.0. Les versions les plus récentes d'Apache Felix utilisent certaines classes qui ne sont pas compatibles avec la machine virtuelle Dalvik. Cette incompatibilité a été résolue par Google en définissant la machine virtuelle ART.

L'application ImageScore a été déployée sur les plate-formes de test depuis un serveur de développement avec un APK signé. Il contient dans un répertoire assets un serveur OSGi qui est indispensable pour le découpage logiciel BackEnd, FrontEnd et la gestion dynamique des composants imposés par l'AppManager.

Les plate-formes support des Cloudlets sont des machines sous Linux 14.0 Ubuntu. Chacune supporte des librairies de virtualisation telles que celles fournies par OpenStack afin de disposer d'une infrastructure de Cloud Computing. Les services Nova, Swift sont disponibles pour le contrôle des ressources des machines virtuelles.

### 4.1.3. Outils de mesures ou de monitoring

Les outils de monitoring sont fonction du système d'exploitation utilisé et laissé au libre choix de l'administration (voir chapitre 2, section 2 et 3). Pour notre cas d'étude, nous avons utilisé le composant Sensu pour la partie Cloudlet et l'orchestrateur OpenStack++ avec ses composants pour le monitoring et l'interopérabilité. Notre première étape consiste à se connecter au réseau du laboratoire. L'utilisation de notre framework pour la surveillance d'un réseau de Cloudlets décrit au chapitre 2 (section 4.1), permet la collecte de données et fournit les informations se déroulant à l'intérieur des périphériques en temps réel. Au niveau du périphérique mobile, nous utilisons les journaux propres à Android, ainsi que nos logs applicatifs.

## 4.2. Déroulement du scénario nominal

Chaque périphérique mobile est utilisé par un usager différent. Les applications mobiles démarrées par chaque usager n'entrent pas dans ce scénario, excepté ImageScore. Une fois démarrée, elle permet à chaque usager de noter les images, d'en assurer la gestion, d'enrichir les descriptions de ces images. Une copie est fournie (Figure 5.17), une image est affichée, le menu « overflow » permet l'accès aux opérations de gestion et de scoring. Celles-ci ont un impact sur le répertoire de travail de l'application. Les images proviennent de la galerie photo de l'application. L'usager a la possibilité de les importer depuis différentes sources : la galerie du périphérique, une carte additionnelle. Il est important dans cette version de notre application que l'ensemble de ressources utiles soient clairement identifiées avec leur mode d'accès (local ou distant). Le mode local stipule que la ressource est liée à l'application mobile et fera partie de la migration du Backend si nécessaire. Alors que le mode distant indique que la ressource est liée au périphérique et ne participera pas à une étape de migration de son Backend si l'usager la demande. Elle est alors accessible au travers d'un service REST.

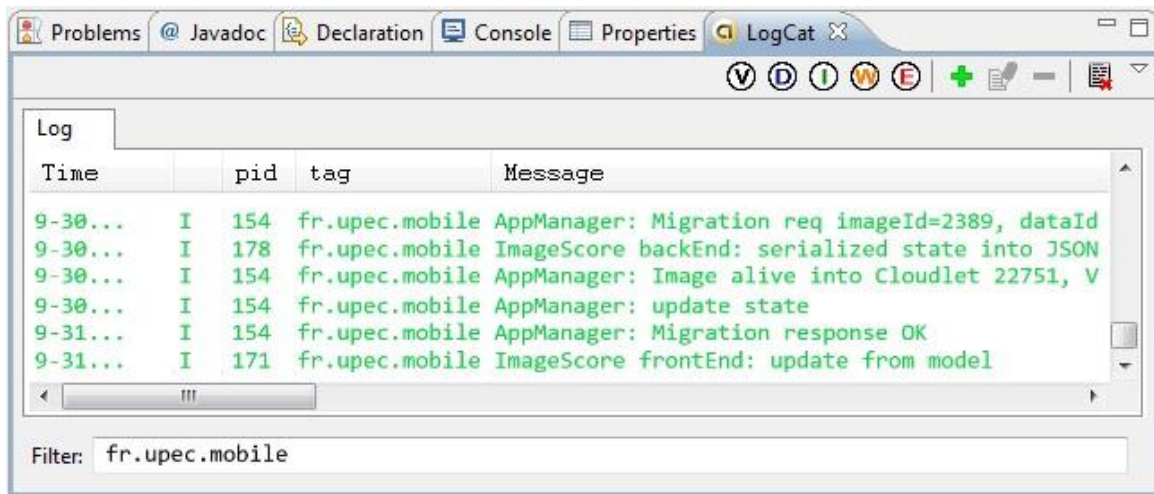
*Figure 5.17 Support de la phase d’évaluation : application mobile ImageScore*

La suite de l’interface graphique porte sur la gestion du backend et plus largement du cycle de vie de l’application. Ainsi, le bouton Move Application (Figure 5.17) déclenche auprès de l’AppManager de l’application ImageScore, une demande de migration auprès de la Cloudlet sélectionnée. Une fois l’opération effectuée, un bouton nommé Back Application remplace le bouton Move Application, interdisant ainsi toute mauvaise opération. Le comportement de cet autre bouton déclenche auprès de l’AppManager une demande de rapatriement du backend applicatif.

Lorsque l’image est notée et taggée par des remarques de l’usager alors une autre image est chargée pour permettre à l’usager d’avancer dans sa session de travail.

Les détails des scénarios de migration ont été décrits (Figure 5.5), l’implémentation suit rigoureusement ces séquences d’interactions. Plus précisément, les API REST que nous avons construites via la suite Swagger, interviennent tout au long des échanges afin d’effectuer la migration du backend. Chaque composant dispose de fichiers, de logs afin de tracer les méthodes appelées. La Figure 5.18 est une extraction de logs issus du périphérique mobile en relation avec ceux de la Cloudlet en relation.

*Figure 5.18 Log du périphérique mobile*




Ces traces apportent un premier suivi du comportement de la migration depuis un périphérique mobile ou vers celui-ci. Mais ces informations ne sont pas suffisantes pour une analyse quantitative de cette gestion d'application. Il est nécessaire de définir de premières métriques afin d'estimer les coûts en temps et en mémoire, présentés précédemment.

## 4.3. Description des mesures et analyse critique

Nous avons décrit au chapitre 2, section 3 un modèle d'architecture pour le monitoring d'une Cloudlet qui permet la collecte d'informations. Bien entendu, les volumes de données sont importants et un filtrage adapté aux métriques à calculer est indispensable. Parmi les observations caractéristiques de la migration, nous nous intéressons à la performance de la migration de composants. Dans cette étude, ces composants sont les backends applicatifs d'applications mobiles. Lors du déplacement d'un tel composant ou une Cloudlet, il y a plusieurs facteurs qui participent à ce coût. La charge de la Cloudlet cible est un facteur clé sur la disponibilité du composant sur la Cloudlet.

Des facteurs secondaires portent sur les images des machines virtuelles à charger, voire des images de containers à charger dans ces VMs. Il apparaît évident que les performances sont meilleures lorsque les images demandées par l'AppManager du périphérique mobile sont déjà résidantes sur la Cloudlet cible. Nous nous plaçons dans le cadre de ces mesures dans une situation où les images sont déjà présentes dans le Repository de la Cloudlet support, avec les identifiants publiés au niveau du ServiceRegistry.

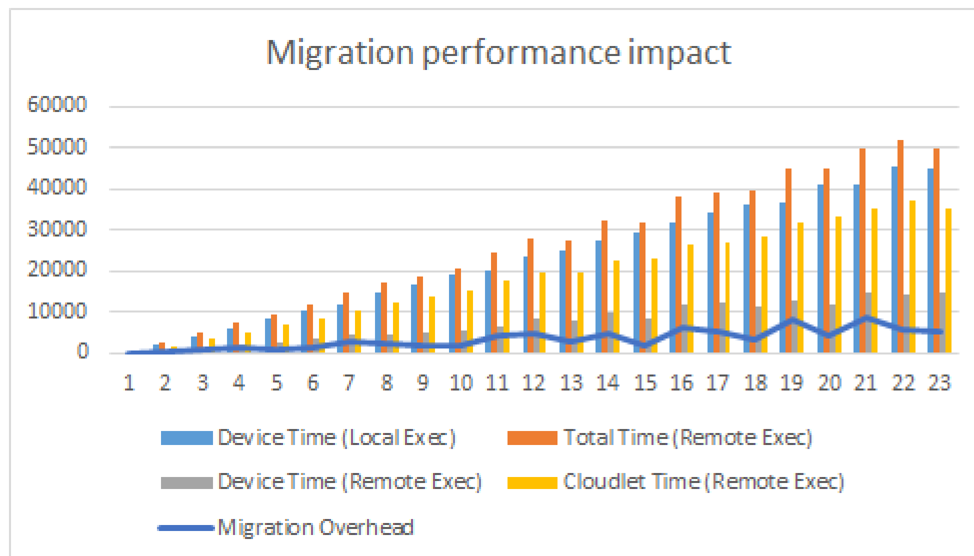
D'autre part, notre architecture de monitoring prend en compte les échanges entre Cloudlets. Cela signifie des échanges de données pour préparer la mobilité de l'usager du périphérique mobile. Cette gestion du voisinage d'une Cloudlet fait partie aussi des facteurs secondaires, car plus ce voisinage est vaste plus l'impact de cette propagation d'information est visible sur la prise en compte des composants. Le protocole MOCP couvre quatre aspects de communication avec une Cloudlet, il semble naturel que chaque aspect de notre protocole puisse influer sur la partie Core.

Pour effectuer les prises de mesure, nous avons déployé sur la Cloudlet les outils de mesures et de logs présentés au chapitre 2 pour les traces et mesure de temps. De plus, grâce à la chaîne logicielle basée sur (Sensu, RabbitMQ, Redis, Carbon Cache), les données sont collectées et stockées en vue d'être filtrées et présentées.

Pour les observations sur les périphériques mobiles, nous avons configuré la machine virtuelle ART ainsi que les outils présents sur les périphériques tels que « Monitoring Tool ». Ainsi, nous disposons de mesures de temps pour les processus, voire les applications embarquées, les accès réseau, l'usage des capteurs, la localisation de données générées.

À l'issue de plusieurs vagues de tests, nous avons traité les données afin de représenter le coût de la migration (Figure 5.19). La figure représente en abscisse le nombre d'applications déjà présentes sur une image de périphérique virtualisé dans une Cloudlet, en ordonnée est représenté le temps d'exécution du test en milliseconde. Cette figure illustre que le coût de la migration croit avec le nombre d'applications dont le backend a déjà migré sur la Cloudlet. Mais cela montre aussi la proportion du temps d'exécution impacté par l'accès réseau. Nous avons calculé l'overhead amené par la déportation d'applications sur une Cloudlet, qui s'obtient par comparaison entre le temps total d'exécution sur le périphérique et le temps d'exécution sur la Cloudlet. Il apparaît que ce coût est de l'ordre de 10 % de temps global sachant qu'il est presque constant en fonction du nombre d'applications déportées. À partir de ces premiers résultats, ce coût apparaît acceptable au regard des gains obtenus grâce à cette migration (chapitre 2, section 3.2).

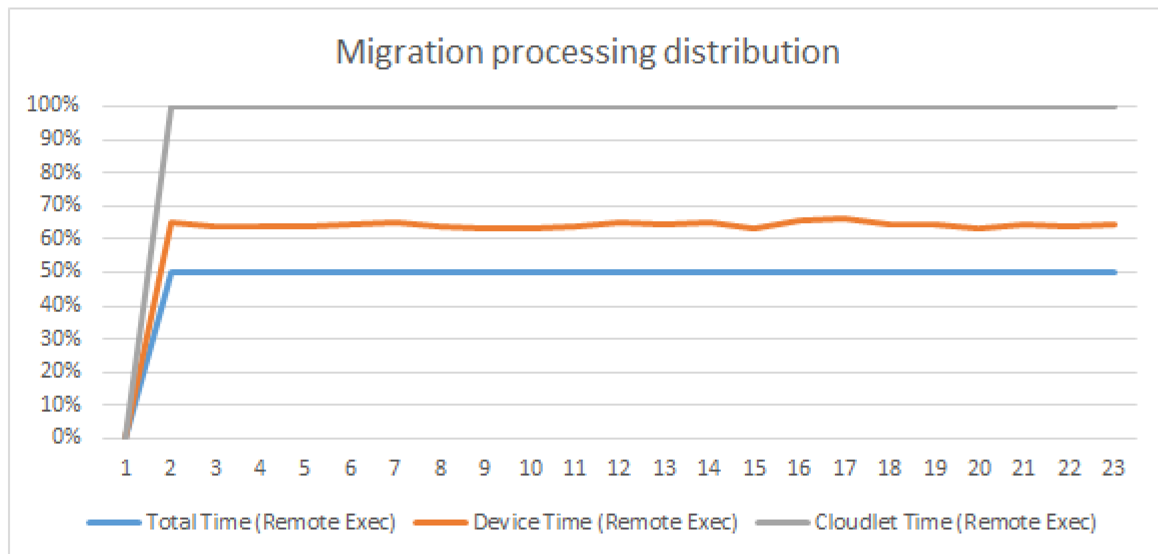
*Figure 5.19 Impact de la migration sur la performance*




Dans une seconde analyse, nous avons calculé les ratios entre les temps d'exécution sur le périphérique mobile et sur la Cloudlet pour observer l'impact de notre architecture logicielle d'une application « migrable ». Ainsi au chapitre 3, nous avons structuré une telle application entre FrontEnd et un BackEnd dont les échanges s'effectuent par le biais de Web Services REST. De tels échanges permettent ainsi la migration d'une partie de ce couple logiciel. Mais nous voulions connaître l'impact de la migration au

cours de l'exécution avec la répartition des coûts entre périphérique et Cloudlet. Même avec des valeurs approchées, nous constatons que plus de la moitié du temps d'exécution de notre application ImageScore est pris en charge par la Cloudlet alors que 40 % voire moins est consacré au périphérique mobile (Figure 5.20). Nous pouvons en conclure que la structure de notre application est effectivement bien adaptée pour être déployée d'une part sur un périphérique mobile et d'autre part sur une Cloudlet.

*Figure 5.20 Migration du processus de distribution*




D'autres mesures ont été faites par rapport au rapatriement du backend en cours d'exécution ou encore par rapport à l'usage de ressources locales au périphérique ou de ressources distantes sur la Cloudlet. Afin de ne pas surcharger cette section, nous nous limitons à ces premiers résultats qui attestent de la capacité de ce nouveau type d'architecture logicielle, spécialement pour l'offloading (ou déportation logicielle) dans un réseau de Cloudlets.

# V. Bilan

Ce chapitre a permis de présenter notre implémentation de Cloudlet respectant la spécification des chapitres précédents et les propriétés liées, et d'aborder des outils que nous avons utilisés pour évaluer notre solution.

Nous avons présenté nos contraintes d'implémentation associées à leur rôle logiciel. Notre architecture à base de composants est adoptée pour la mise en œuvre de notre modèle de communication à base de service REST. L'apport de la modularité ajoute une meilleure administration de notre architecture logicielle. Elle permet de définir le plan d'évolution des composants de notre application grâce à l'utilisation de composants OSGi.

Nous avons donné une réalité à la notion de Cloudlet en prototypant tout un écosystème autour de machines virtuelles, contenant entre autre un serveur OSGi pour l'accueil dynamique de composants.

De plus, nous avons mis en place une démarche logicielle pour la définition du protocole d'échange qui est supporté par cette architecture. Avec un égal souci de réalisme, nous avons outillé cette approche avec une phase de conception technique basée sur la suite Swagger. Nous conservons la précision et la rigueur de la spécification du chapitre 3, section II. Non seulement la documentation interactive du protocole est un vecteur de communication avec les utilisateurs, mais la validation par les tests des APIs REST offre un premier résultat probant sur le respect de notre implémentation à l'égard de nos choix initiaux.

Enfin, nous avons fourni quelques mesures qui donnent une estimation du coût de notre approche de la déportation de code dans une Cloudlet. Cette section pourrait être détaillée davantage pour faire apparaître de nouveaux critères clés pour la migration de code. Cette étude élargie fait partie des tâches planifiées pour le futur de nos recherches et plus particulièrement l'automatisation du déploiement des outils de mesures avant la campagne de test de charge, ainsi que la collecte de données après cette même campagne de test. Si l'agrégation des données reste manuelle, la mise en valeur des métriques recherchées et la représentation de celles-ci est un travail essentiel pour la compréhension des résultats et leur prise en compte pour de futures versions.

# CHAPITRE 6 : Conclusions et perspectives

Dans les chapitres précédents, nous avons présenté une architecture logicielle pour répondre aux besoins et limites de la mobilité applicative dans un cadre nomade. Cela s'accompagne de contraintes logicielles telles que la résolution de la déportation ou migration des applications dans un réseau de Cloudlets. Notre démarche respecte un cycle de vie où nous avons formalisé notre architecture logicielle grâce à l'écriture d'une spécification formelle. Celle-ci a guidé d'une part la construction d'un modèle temporel pour la définition de propriétés temporelles. D'autre part, elle est la référence pour la phase de prototypage. L'étape finale a été la prise de mesures applicatives au cours de tests d'intégration.

Dans ce chapitre, nous faisons une synthèse de notre contribution et des propositions faites, nous analysons ensuite quels sont les axes de recherches à poursuivre après ce travail.

## I. Contributions

Ce travail s'est déroulé sur une longue période entre les universités UPEC et Yaoundé I. Il a permis de répondre aux problèmes énoncés dans le chapitre I et de présenter nos résultats dans ce document. Dans cette section, nous décrivons les résultats majeurs de cette thèse de doctorat sur trois aspects tout en soulignant le rôle essentiel du nouveau concept de la Cloudlet.

### 1.1. Définition formelle d'une architecture logicielle

Partant d'un constat sur les multiples usages des périphériques mobiles, nous avons défini dans un premier temps une architecture de virtualisation des périphériques mobiles dans un réseau de Cloudlets ainsi que les contraintes techniques associées. Les défis lors du monitoring de ces périphériques ont été relevés afin d'aborder la conception d'un framework à base de composants.

Ce framework en mode passif est un outil clé de détection d'anomalies par rapport à d'autres solutions classiques comme Capsa (section 5.3, chapitre 2). Il a pour avantage l'omniprésence des périphériques mobiles et permet de collecter à chaud les données pour fournir les informations à remonter, issues d'une exécution dans un périphérique mobile.

Notre définition d'un réseau de Cloudlets repose sur notre besoin d'effectuer le suivi des usagers d'applications mobiles dans leurs déplacements. Ainsi, une Cloudlet est considérée dans une première lecture comme un Cloud local ou le cache d'un Cloud avec ses fonctionnalités essentielles telle que la virtualisation de périphériques et d'applications.

Nous avons fait le choix d'une définition formelle de Cloudlet écrite en $\pi$-calcul. Pour mieux spécifier tous ces périphériques, nous avons décrit la notion de périphérique virtuel composite. Cette opération autorise la composition de périphériques avec d'autres accessoires. Cette définition permet d'élaborer la virtualisation dans un réseau de Cloudlets et les interactions entre composants. Afin d'accepter d'autres implémentations de Cloudlets que la nôtre, nous avons défini l'interface d'échange autour d'une Cloudlet. Le protocole MOCP est la concrétisation de cette démarche. La validation a été faite pour mettre en évidence la similitude entre l'exécution locale d'une application mobile et sa déportation dans un réseau de Cloudlets. Nos objectifs de migration d'application depuis un périphérique mobile vers une Cloudlet et son retour sont alors atteints.

## 1.2. Développement du framework à base de composants

Le réseau de Cloudlets permet de mettre en exergue un ensemble de composants interconnectés. Cela rend possible le développement d'un framework à base de composants. Notre framework propose un ensemble d'outils intégrant également le monitoring et la migration des applications mobiles dans notre réseau de Cloudlets. L'architecture logicielle proposée pour notre framework traite des composants logiciels et des interactions communes entre ces composants. Elle apporte d'autres techniques pour l'indépendance et l'interopérabilité entre les composants comme OpenStack++, RabbitMQ ou Sensu, et l'implémentation du protocole MOCP. Etant donné que chaque infrastructure utilise une des clés spécifiques pour sécuriser la connexion des composants de RabbitMQ, nous avons modifié les paramètres de sorte à avoir un certificat de sécurité unique pour ces infrastructures. Ce correctif de sécurité a permis la cohabitation de tous les composants qui utilisent le même routeur. De même, le composant Sensu est réparti à travers de multiples périphériques connectés et est au cœur de notre solution de surveillance. L'apport d'un framework OSGi pour l'architecture des serveurs Apache Felix, Apache Karaf permet l'importation de composants dans un repository. Celui-ci nous a permis de mettre en œuvre le principe de migration vers une Cloudlet voisine. Ainsi, la migration d'un composant peut déclencher le déplacement d'une grappe de composants du périphérique mobile vers une Cloudlet.

## 1.3. Développement d'outils de mesure

Pour réaliser l'évaluation de notre framework, nos propres outils de mesures ont été développés dans le but de réduire au maximum les effets des perturbations sécuritaires sur le code source des différentes Cloudlets et de périphériques mobiles. D'autres outils ont permis de mesurer l'impact de la migration et le coût d'usage d'une Cloudlet afin d'assurer un gain aux usagers et de promouvoir son application. Nous avons également présenté des études de cas sur la cyber-sécurité portant sur les chevaux de Troie et traitant la sécurité connexe. Le développement de notre solution Trojunnel a permis de créer un tunnel

de connexion entre l'interface WiFi connecté à un réseau de sécurité critique et un réseau mobile connecté à Internet. Ce programme nous a permis de faire une étude comparative sur la vulnérabilité des systèmes d'exploitation des périphériques mobiles et les moyens de sécurisation. De plus, en se basant sur les outils existants, notre solution donne lieu à un prototypage basé sur l'emploi du serveur OSGi, d'une architecture logicielle d'application mobile et le principe de migration.

Faire le point sur les apports d'un travail de plusieurs années n'a d'intérêt que dans l'étude des perspectives qu'il offre sur l'avenir. Celles-ci sont nombreuses, car nos travaux passés sont multi-facettes au carrefour du Mobile Cloud Computing, de la Cloudlet, du Big Data, de la virtualisation, de la sécurité dans un réseau de Cloudlets et du génie logiciel.

# II. Perspectives

Pour illustrer les premières perspectives de recherche issues de nos travaux, nous proposons des voies à explorer en fonction d'applications directes.

## 2.1. Optimisation du temps lors de la déportation

Les échanges entre les périphériques mobiles et/ou les Cloudlets sont effectués via le protocole MOCP. Ainsi, il serait souhaitable d'enrichir ce protocole entre les Clouds mobiles par l'ajout d'informations pour mettre sur pied un ensemble de règles et de références afin d'estimer si le temps de déportation d'une application est coûteux. Cette gestion de la déportation applicative a des aspects proches de la gestion de la charge sur un cluster de serveurs. Dans notre cas, elle permettrait de choisir une Cloudlet parmi celles accessibles dans le voisinage de l'usager.

Cet enrichissement du protocole peut être une référence ou constituer un effort de normalisation pour le Mobile Cloud Computing et la déportation d'applications mobiles d'un périphérique mobile vers une Cloudlet et vice-versa. L'évolution de ce protocole doit spécifier les nouveaux services pour chacune des catégories suivantes : le noyau de MOCP ou MOCP Core, la MOCP Intra-Cloudlet et Inter-Cloudlet et la MOCP de la Cloudlet vers le Cloud.

## 2.2. Etude de nouvelles propriétés

L'une des perspectives porte sur l'écriture et la preuve de nouvelles propriétés liées à la transparence de la mobilité, de la déportation et ainsi montrer que les ressources telles que les données ou des applications mobiles peuvent se déplacer sans qu'un utilisateur en ait connaissance. Nos simplifications apportées à la spécification du chapitre 3 ont eu pour objectif de rendre lisible l'application de notre opérateur de transformation afin que les automates résultant soient de taille à être présentés dans cette thèse. En conservant la spécification telle qu'elle a été écrite au chapitre 3, nous pouvons dans une étude

de propriété plus ambitieuse, mettre en valeur cette propriété de mobilité et de déportation. L'utilisateur ne sera plus l'observateur conscient de la mobilité, de la déportation de l'application mobile ou de données, mais l'utilisateur averti que des propriétés non-fonctionnelles de transparence assurent la gestion de sa déportation. Le résultat qu'il obtient en fin d'exécution est le fruit de la gestion de l'ensemble des ressources sur un réseau de Cloudlets.

Cette évolution a plus de conséquences que ne le laisse présager ce propos. L'enrichissement de nos automates temporisés a une incidence forte sur les capacités de mémoire et de calcul de notre plate-forme de model checking. D'où la nécessité de repenser certains aspects opératoires.

## 2.3. Mesure de l'impact de la migration et de l'usage d'une Cloudlet

Les chapitres précédents ont permis d'établir les bases solides pour la déportation et la migration des applications mobiles ; des questions restent sur l'impact de cette migration ou déportation en termes de coûts et de temps. Les stratégies d'ordonnancement des applications mobiles doivent être définies pour une réduction de temps de réponse moyen et du coût global d'utilisation des ressources du réseau de Cloudlets. La qualité de service est un facteur important dans une décision de migration ou de déportation. L'usage d'une Cloudlet dans une entreprise aboutit au besoin de sécurité des données et des infrastructures. Autrement dit, qu'en est-il de la propriété des données provenant d'une application mobile initialement exécutée sur un terminal nomade. Si l'on souhaite gérer finement ces droits, il devient essentiel de choisir les Cloudlets de déportation en fonction des règles de gestions de droits qu'elles supportent. Par exemple, une application commerciale virtualisée dans une Cloudlet publique doit-elle rendre public toutes les données générées après sa déportation ? Si la réponse n'est pas immédiate, il est aisé de se rendre compte qu'il faut définir une politique de gestion des droits sur les données et les applications au cours du déplacement de l'usager.

Les perspectives ont l'intérêt de proposer des pistes d'avenir à un travail existant. Bien entendu, il ne faut pas omettre les rencontres, les partenariats, voire les futures affectations qui transforment le prévisible en inattendu. L'essentiel reste que cette recherche soit vivante et se poursuive par nos soins et tous ceux qui souhaiteront en profiter.

# LISTE DE PUBLICATIONS

## Articles de Conférences

1. Mahmoudi C., Mourlin F. and Djiken G. L. : Mobile Agent for Nomadic Devices. In The Third International Conference on Mobile Services, Resources and Users, Mobility 2013, page 59-68, Portugal (November 2013).
2. Djiken G. L., Asnae M. and Mourlin F. : Design of Mobile Services for Embedded Platforms, ICSEA 2014, The Ninth International Conference on Software Engineering Advances, page 354-361, France (October 2014).
3. Mahmoudi C., Mourlin F. and Djiken G. L. : Vers une définition formelle du Cloud Computing Mobile, CAL'2016, Conférence francophone sur les Architectures Logicielles, page 64-79, France (Juin 2016).

## Article de Journal

1. Djiken G. L., Asnae M. and Mourlin F. : Design of Mobile Services for Embedded Device, IARIA Journals, 2017.

# BIBLIOGRAPHIE


[1] R. Couillet et M. Debbah, «Le téléphone du futur : plus intelligent pour une exploitation optimale des fréquences,» *Revue de l'électricité et de l'électronique,* 2010.

[2] N. Lane, M. Emiliano et L. Hong, «A survey of mobile phone sensing,» *IEEE Communications Magazine,* vol. 48, n° 19, sept 2010.

[3] «Services mobiles,» 25 Octobre 2016. [En ligne]. Available: http://www.servicesmobiles.fr. [Accès le 4 Novembre 2016].

[4] «Technos Medias,» Fevrier 2015. [En ligne]. Available: http://www.latribune.fr. [Accès le 4 Novembre 2016].

[5] K. olivier, R. Kleinschmager et C. Cauvin, «Modélisation et représentations spatio-temporelles des déplacements quotidiens urbains,» *CiteUlike,* p. 27, Juin 2007.

[6] T. Gobert, «Mobilité, portabilité, transfert, migrations et navigation numériques : un nomadisme ?,» *Mobilités numériques, Ax-les-Thermes,* 2011.

[7] P.-J. Barlatier, «Management de l'innovation et nouvelle ère numérique : enjeux et perspectives,» *Revue Française de Gestion,* vol. 42, n° 1254, pp. 55-63, 2016.

[8] M. Faouzia, «Dispositif de E-learning : quels usages pour améliorer la formation au maroc ?,» CEDoc - sciences de l'Ingénieurs, Casablanca, 2013.

[9] M. Savoure et J. FRAYSSINET, «La telephonie mobile : technologies, acteurs et usages,» IREDIC, MARSEILLE, 2006.

[10] T. That et Dai-Hai, «Gestion efficace et partage sécurisé des traces de la mobilité,» Thèse de Doctorat, Paris Saclay, 2016.

[11] N. Muller, Mobile Telecommunications Factbook, New york: Mc Graw Hill, 2003.

[12] L. Claire, F. Creplet et U. Galadrielle, «Technologies de la mobilité,» *documentaliste-Sciences de l'Information,* vol. 49, n° 13, pp. 26-41, 2012.

[13] V. Sergère, «Normes Wi-Fi 802.11a/b/g/n/ac : quelles fifférences dans la partique ?,» 14 septembre 2014. [En ligne]. Available: http://www.frandroid.com. [Accès le 05 novembre 2016].

[14] S. Priyanka et J. Makrariya, «A review on heterogeneous 5G architecture,» *International Research Journal of Engineering and Technology (IRJET),* vol. 3, n° 15, pp. 2342-2346, 2016.

[15] M. Gallisot, J. Caelen, F. Jambon et B. Meillon, «Une plate-forme usage pour l'intégration de l'informatique ambiante dans l'habitat : Domus,» *Techniques de Système d'Information,* pp. 1-22, 2013.

[16] H. Korth et I. Tomasz, «Introduction to Mobile Computing,» *The Kluwer International Series in Engineering and Computer Science,* vol. 353, pp. 1-43, 1996.

[17] G. Pavithra et K. Sundram, «Verilog module for on the Go Implementation,» *International Conference on Energy Efficient Technologies for Sustainability (ICEETS),* 2016.

[18] F. Douglis et R. Caceres, «Storage Alternatives for mobile computers,» *AT&T Bell Laboratories,* 1996.

[19] L. Horellou, G. Gouault et Al., «Du tangible au digital,» Communication graphique, Strasbourg, 2014.

[20] S. Rafik, E. Anquetil et A. Delaye, commandes gestuelles interactives avec prédiction de trajectoires de gestes graphiques, Rennes, 2010.

[21] C. Menrath, A. Gonord et Al., Mobiles attitude : ce que les protocoles ont changé dans nos vies, Paris: Hachette, 2005.

[22] L. Spiegel, «La télévision portable : enquête sur les voyages dans l'espace domestique,» *In la television du téléphonoscope à Youtube,* pp. 249-271, 2009.

[23] J. Coutaz et Al., «Quand les surfaces deviennent interactives …,» *LCN,* vol. 3, n° 14, pp. 101-126, 2002.

[24] K. Sloan, OpenGL Special Effects, Apress, Springer, 2016.

[25] S. Ratabouil, Android NDK : beginner's guide, birmingham: Packt Publishing Ltd., 2015.

[26] S. Hout et E. Lecolinet, «Archmenu et thumbmen : contrôler son dispositif sur le pouce,» *In IHM 07, Proceedings of the 19th International Conference of the Association Francophone d'Interaction Homme-Machine,* pp. 107-110, 2007.

[27] M. Ebling, «Can Cognitive Assistants Disappear ?,» *IEEE Pervasive Computing,* vol. 15, n° 13, pp. 4-6, 4 Avril 2016.

[28] B. Asma et M. Hadj, Réseau de capteurs sans fil comportementaux pour l'aide au maintien à domicile par la surveillance en habitat intelligent, Toulouse: Ordiateur et société, 2015.

[29] P. Charith, A. Zaslavskyy et Al., «Capturing Sensor Data from Mobile Phones using Global Sensor Network Middleware,» *23rd International Symposium on Personal Indoor and Mobile radio Communications (PIMRC),* 2012.

[30] N. Logre, S. Mosser et Al., «Visualisation de donnes en provenance de capteurs vers une visualisation adaptable à l'usage,» *5 journée du GDR, GPL,* Avril 2013.

[31] L. Anne-marie, «Détecter et monitorer les séismes grâce aux capteurs embarqués dans les smartphones,» *Congrès INFORSID,* n° 134, 2016.

[32] M. Botts, G. Percivall et Al., «Sencor Web Enablement : Overview and High level architecture,» *Proceedings of the 5th International ISCRAM Conference,* 2008.

[33] M. Popa, «Considerations regarding the cross-platform mobile application development process,» *Academy of economics studies, economyInformatics,* vol. 1, n° 113, pp. 40-52, 2013.

[34] R. Ghatol, P. Yogesh et Al., Beginning PhoneGap : Mobile Web Framework for JavaScript and HTML5, New York: Springer Science, 2012.

[35] J. Richard-Foy, «Ingénierie des applications Web : réduire la complexité sans diminuer le contrôle,» Génie logiciel, Rennes, 2015.

[36] J. Solanky, P. Kajal et P. Gayatri, «Resemblance of PhoneGap and Titanium for Mobile Application Development,» *International Journal of Computer Applications,* vol. 144, n° 110, pp. 1-10, Mai 2016.

[37] R. Jean-Pierre et A. Dupont, «Analyse du développement mobile multiplate-forme avec Xamarin,» Haute Ecole de gestion & Tourisme, 2015.

[38] E. Ozcan, «Conception et implantation d'un environnement de développement de logiciel à base de composants, applications aux systèmes multiprocesseurs sur puce,» Réseaux et télécommunications, Grenoble, 2007.

[39] D. Popovic, «Gestion du contexte pour des applications mobiles dédiées aux transports,» Sciences de l'ingénieur, Lille, 2013.

[40] M. Vandana, S. Krishnan et R. Sumit, «On optimal hotspot selection and offloading,» *International Conference on Communications (ICC),* 7 Janvier 2016.

[41] M. Buffa, A. Giboin et T. Beregeron, Etat de l'art sur les techniques de transfert data/Audio/video basées Web, Projet AZKAR, 2015.

[42] A. Bener et M. Adacal, «Mobile Web services : a new agent based framework,» *IEEE Internet computing,* vol. 10, n° 13, pp. 58-65, 2006.

[43] L. Johnrud et D. Hadzic, «Efficient Web Services in Mobile Networks,» *European Conference on Web Services (ECOWS),* 2008.

[44] N. Dnyaneshwari et S. Ghumbre, «The Web-Telecom Capsule : Bridging heterogeneous technologies for telephony services,» *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* vol. 5, n° 15, 2016.

[45] Ericsson, «Cellular Networks for massive IoT,» Ericsson White Paper, 2016.

[46] A. Medded, «Internet of things standards : who stands out from the crowd?,» *IEEE Communications Magazine,* vol. 57, n° 17, pp. 40-47, 2016.

[47] R. Woodings et D. Joos, «Rapid Heterogeneous Connection Establishment : Accelerating Bluetooth Inquiry Using IrDA,» *Department of Computer Scienc, Brigham Young University,* vol. 1, pp. 342-349, 2002.

[48] A. Akbari-Moghanjoughi et S. Aduwati, «Tightly-Coupled Integrated LR-WPAN/WiMAX Network: Architecture and Performance Modelling,» *IERI Procedia,* vol. 4, pp. 59-67, 2013.

[49] A. Bisognin, «Evaluation de technologies organiques faibles pertes et d'impression plastique 3D afin de contribuer au développement de solutions antennaires innovantes dans la bande 60GHz-140GHz,» Thèse de doctorat, Nice, 2015.

[50] R. Philippe et M. Dalmau, «Gestion de la Qualité de Service par Reconfiguration Dynamique dans les Applications Interactives Multimédia,» *Journées ALP,* pp. 17-18, 2015.

[51] H. Yang, H. Luo et Y. Fan, «Security in mobile ad hoc networks : challenges and solutions,» *IEEE Wireless Communications,* vol. 11, n° 11, 2014.

[52] J. Bellardo et S. Savage, «802.11 denial-of-service attacks : Real vulnerabilities and practical solutions,» *USENIX Security,* pp. 15-28, 2003.

[53] T. Sobh, «Wired and wireless intrusion detection system: Classifications, good characteristics and state-of-the-art,» *Computer Standards & Interfaces,* vol. 28, n° 16, pp. 670-694, 2006.

[54] A. Beach, M. Gartrell et Al., «Solutions to security and privacy issues in mobile social networking,» *In Proc. IEEE International Conference on Computational Science and Engineering,* vol. 4, pp. 1036-1042, 2009.

[55] D. Geneiatakis, T. Dagiuklas et G. Kambourakis, «Survey of security vulnerabilities in session initiation protocol,» *IEEE Communications Surveys and Tutorials,* vol. 8, n° 11-4, pp. 68-81, 2006.

[56] S. Schuckers, «Spoofing and anti-spoofing measures,» *Information Security Technical Report,* vol. 7, n° 14, pp. 56-62, 2002.

[57] W. Stallings, «Network Security Essentials : Applications and Standards,» *Pearson Education India,* 2007.

[58] S. Cereola et J. Cereola, «Breach of data at TJX: An instructional case used to study COSO and COBIT, with a focus on computer controls, data security, and privacy legislation,» *Issues in Accounting Education,* vol. 26, n° 13, pp. 521-545.

[59] J. Rensburg et V. Irwin, «Wireless security tools,» *Computer Science,,* vol. 83, n° 1944, p. 3924, 2006.

[60] T. Hardjono et J. Seberry, «Information Security Issues in Mobile Computing,» *In Information Security—the Next Decade. Springer US,* pp. 143-151, 1995.

[61] M. Decker, «A security model for mobile processes,» *In Proc. 7th International IEEE Conference on Mobile Business,* pp. 211-220, 2008.

[62] G. Forman et A. Zahorjan, «The challenges of mobile computing,» *Computer,* vol. 27, n° 14, pp. 38-47, 1994.

[63] Z. DURUMERIC, J. KASTEN, D. ADRIAN et Al., «The matter of heartbleed,» *Proceedings of the 2014 Conference on Internet Measurement Conference. ACM,* pp. 475-488, 29 Avril 2014.

[64] F. Fitzek, J. Widmer et Al., «Survey on Energy Consumption Entities on the Smartphone Platform,» *Conference Paper IEEE Xplore,* 2011.

[65] F. Shearer, «Power management in mobile devices,» *chapter Batteries and Displays for Mobile Devices,* pp. 149-180, 2008.

[66] D. Stober, Nanowire battery can hold 10 times the charge of existing lithium-ion battery, Stanford: Stanford technical report, 2007.

[67] V. Naing, J. Hoffer, D. Weber, A. Kuo et Al., «Biomechanical energy harvesting : Generating electricity during walking with minimal user effort,» *Science,* vol. 319, n° 15864:807, 2008.

[68] P. Perrucci, P. Fitzek et V. Petersen, «Energy Saving Aspects for Mobile Device Exploiting Heterogeneous Wireless Access Networks,» *Architectures and Protocols : Springer,* pp. 277-304, 2008.

[69] F. Shearer, «Chapter Hierarchical View of Energy Conservation,» chez *Power management in mobile devices*, Newnes, 2008, pp. 32-75.

[70] K. Kumari, «Challenging Issues and Limitations of Mobile Computing,» *An international journal of advanced computer technology,* vol. 3, n° 12, 2014.

[71] Q. Hammouri, A. Manasrah et E. Abu-Shanab, «Examining the impact of privacy, security and legal framework on trust in mobile computing in busines environment : An exploratory study,» *IT College,* 2016.

[72] S. Pelurson et L. Nigay, «Visualisation bifocale sur supports mobiles : une étude empirique,» *27 ème conférence francophonesur l'Interaction Homme-Machine,* pp. 1-20, Octobre 2015.

[73] I. Mavridis et G. Pangalos, «Security issues in a mobile computing paradigm,» *Communications and Multimedia Security, Springer,* vol. 3, p. 61, 1997.

[74] F. Heikkila, «Encryption : Security considerations for portable media devices,» *IEEE Security & Privacy,* vol. 5, n° 14, pp. 22-24, 2007.

[75] S. Schwiderski-Grosche et H. Knospe, «Secure mobile commerce,» *Electronics & Communication Engineering Journal,* vol. 14, n° 15, pp. 228-238, 2002.

[76] J. Friedman et D. Hoffman, «Protecting data on mobile device : A toxonomy of security threats to mobile computing and review of applicable defenses,» *Information, Knowledge, Systems Management,* vol. 7, n° 11, pp. 159-180, 2008.

[77] J.-J. SCHWARTZMANN, O. GRUSON et G. GOURMELEN, «Authentification forte par opérateur de réseau mobile à l'usage des utilisateurs de services Web,» *7ème Conférence sur la Sécurité des Architectures Réseaux et Systèmes d'Information (SAR SSI),* 2012.

[78] B. Rajkuma, S. YEO-Chee et S. VENUGOPAL, «Market-oriented cloud computing : Vision, hype, and reality for delivering it services as computing utilities,» *High Performance Computing and Communications, HPCC'08. 10th IEEE International Conference,* pp. 5-13, 2008.

[79] CNIL, Recommandations pour les entreprises qui-envisagent de souscrire à des services de Cloud Computing, Paris, 2012.

[80] A. Eric et Lozano, Executive's Guide to Cloud Computing, John Wiley & Sons, 2010.

[81] M. Vaquero, L. Rodero-Merino, J. Caceres et M. Lindner, «A Break in the Clouds : Towards a Cloud Definition,» *ACM SIGCOMM Computer Communication Review,* vol. 39, n° 11, pp. 50-55, 2009.

[82] F. Gens, R. MAHOWALD, R. VILLARS et Al., «Cloud computing 2010 : An IDC update,» *International Data Corporation,* 2010.

[83] P. Mell et T. Grance, «The NIST definition of cloud computing,» *Communications of the ACM,* vol. 53, n° 16, p. 50, 2010.

[84] J. Mounet, Tout ce que vous devez savoir sur l'informatique dans le nuage, Paris: Syntech informatique, 2010.

[85] P. Hofmann et D. Woods, «Cloud Computing : The Limits of Public Clouds for Business Applications,» *IEEE Internet Computing,* vol. 14, n° 16, p. 90–93, 2010.

[86] D. Milojicic et R. Wolski, «Eucalyptus : Delivering a Private Cloud,» *Computer,* vol. 44, n° 14, p. 102–104, 2011.

[87] Y. JADEJA et K. MODI, «Cloud computing-concepts, architecture and challenges,» *International Conference Computing Electronics and Electrical Technologies (ICCEET),* pp. 877-880, 04 10 2012.

[88] D. Kovachev, Y. Cao et R. Klamma, «Mobile Cloud Computing : A Comparison of Application Models,» *arXiv preprint arXiv : 1107.4940,* 2011.

[89] M. Satyanarayanan, Z. Chen, W. Richter et P. Pillai, «Cloudlets : at the leading edge of mobile-cloud convergence,» *inProc.ofMobiCASE2014,* vol. 1, n° 12, pp. 1-7, 2014.

[90] M. Satyanarayanan, G. Lewis, E. Morris, S. Simanta et J. Boleng, «The role of cloudlets in hostile environments,» *IEEE Pervasive Computing,* vol. 12, n° 14, pp. 40-49, 2013.

[91] S. Davy, J. Famaey, J. Serrat-Fernandez et Al., «Challenges to support edge-as-a-service,» *IEEE Communications Magazine,* vol. 52, n° 11, pp. 132-139, 2014.

[92] F. Bonomi, R. Milito, J. Zhu et S. Addepalli, «Fog computing and its role in the internet of things,» *in Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM,* pp. 13-16, 2012.

[93] Z. Becvar, J. Plachy et P. Mach, «Path selection using handover in mobile networks with cloud-enabled small cells,» *in Proc. of IEEE PIMRC 201,* 2014.

[94] T. Taleb et A. Ksentini, «Follow me cloud : interworking federated clouds and distributed mobile networks,» *IEEE Network,* vol. 27, n° 15, pp. 12-19, 2013.

[95] S. Wang, T. Guan-Hua et Al., «Mobile Micro-Cloud : Application Classification, Mapping, and Deployment,» *Proc. Annual Fall Meeting of ITA (AMITA),* 2013.

[96] M. SATYANARAYANAN, R. SCHUSTER, M. EBLING et Al., «An open ecosystem for mobile-cloud convergence,» *IEEE Communications Magazine,* vol. 53, n° 13, pp. 63-70, 2015.

[97] I. Foster, Y. Zhao et I. Raicu, «Cloud computing and grid computing 360 degree compared,» *Grid Computing Environments Workshop, IEEE,* pp. 1-10, 2008.

[98] R. Jesup, S. Loreto et M. Tuexen, «WebRTC data channels,» *IETF, Standards Track, draft-ietf-rtcWeb-data-channel-13. txt,* 05 06 2014.

[99] F. Roy, «REST : Architectural Styles and the Design of Network-based Software Architectures,» Université de California, California, 2000.

[100] O. Liskin, L. Singer et K. Schneide, «Teaching old services new tricks : adding HATEOAS support as an afterthought,» *Proceedings of the Second International Workshop on RESTful Design. ACM,* pp. 3-10, 2011.

[101] Y. Allendes, «Spring for Android,» hèse de doctorat. Haute école de gestion de Genève., Genève, 2012.

[102] G. Hohpe et B. Woolf, Enterprise Integration Patterns : Designing, Building and Deploying Messaging Solutions, Addison Wesley Professional, 2003.

[103] G. Nebuloni et V. Srikumar, «Les entreprises de l'EMEA face à la virtualisation des clients en 2011,» *Vague du changement dans l'informatique d'entreprise,* septembre 2011.

[104] D. Bartholomew, «Qemu a multihost multitarget emulator,» *Linux Journal,* n° 1145, pp. 1-3, 2006.

[105] P. Pradeep, Z. Xiaoyun et W. Zhikui, «Performance Evaluation of Virtualization Technologies for ServerConsolidation,» *HPL,* 2007.

[106] N. Ruest et D. Ruest, Virtualization, A Beginner's Guide, Mc Graw-Hill, 2009.

[107] B. Sotomayor, R. Montero et I. Lorente, «Virtual infrastructure management in private and hybrid clouds,» *IEEE Internet computing,* vol. 13, n° 15, pp. 14-22, 2009.

[108] A. Darabseh, A. Al-Ayyoub et Y. Jararweh, «Sdstorage : a software defined storage experimental framework,» *IC2E,* 2015.

[109] R. Ayoub, S. Sharifi et S. Tajana, «Gentlecool : Cooling aware proactive workload scheduling in multi-machine systems,» *Proceedings of the Conference on Design, Automation and Test in Europe,* pp. 295-298, 2010.

[110] J. Tate, F. Enders, T. Jensen et Al., Implementing the IBM System Storage SAN Volume Controller V7, IBM Redbooks, 2015.

[111] A. Blenk, A. Basta, M. Reisslein et W. Kellerer, «Survey on network virtualization hypervisors for software defined networking,» *IEEE Communications Surveys & Tutorials,* vol. 18, n° 11, pp. 655-685, 2016.

[112] V. Autefage et D. Magoni, «Virtualisation de réseau avec Network Emulator et application à l'évaluation d'un réseau recouvrant,» *16e Colloque Francophone sur l'Ingénierie des protocoles,* pp. 153-160, 2012.

[113] F. O'donncha, E. Ragnoli et S. Venugopal, «On the Efficiency of Executing Hydro-environmental Models on Cloud,» *Procedia Engineering,* vol. 154, pp. 199-206, 2016.

[114] M. Rosenblum et T. Garfinkel, «Virtual machine monitors : Current technology and future trends,» *Computer,* vol. 38, n° 15, pp. 39-47, 2005.

[115] R. Creasy, «The origin of the vm/370 time-sharing system,» *IBM J. Res. Dev.,* vol. 25, n° 15, pp. 483-490, 1981.

[116] E. Bugnion, D. Scott, G. Kinshuk et M. Rosenblum, «Disco : Running commodity operating systems on scalable multiprocessors,» *ACM Trans. Comput. Syst.,* vol. 15, n° 14, p. 412–447, 1997.

[117] P.-B. Galvin, «VMware vSphere Vs. Microsoft Hyper-V : A Technical Analysis,» *Corporate Technologies, CTI Strategy White Paper,* 2009.

[118] G. Rich-Uhlig, D. Rodgers, M. Bennett, F. Leung et S. Larry, «Intel virtualization technology,» *Computer,* vol. 38, n° 15, pp. 48-56, 2005.

[119] A. Paging, «Technical report,» *Advanced Micro Devices,* 2008.

[120] «Mobile Cloud Computing,» NIST, [En ligne]. Available: https://www2.nist.gov/programs-projects/mobile-cloud-computing.

[121] L. Lamport, «Time, clocks, and the ordering of events in a distributed in a distributed system,» *Communication of the ACM,* vol. 21, n° 17, pp. 558-565, 1978.

[122] B. Gregg, «Linux Performance,» 21 11 2014. [En ligne]. Available: http://www.brendangregg.com/linuxperf.html. [Accès le 29 12 2014].

[123] H. Darren, «Linux Performance Monitoring,» *Linux world conference & expo,* 2008.

[124] C. Havee, S. Mongkolluksame et Al., «Using Nagios as a groundwork for developing a better network monitoring system,» *Technology Management for Emerging Technologies,* pp. 2771-2777, 2012.

[125] A. Kora, Nagios Based Enhanced IT Management System, a. preprint, Éd., arXiv:1206.1611, 2012.

[126] D. Josephsen, Building a monitoring infrastructure with Nagios, Prentice Hall PTR, 2007.

[127] A. Stanley et R. Holman, «The history and technical capabilities of Argus,» *Coastal Engineering,* vol. 6, n° 154, pp. 477- 491, 2007.

[128] M. Badger, Zenoss Core Network and System Monitoring, Packt Publishing Ltd, 2008.

[129] S. Porter, «Sensu, Heavy Water Operations,» [En ligne]. Available: http://sensuapp.org/. [Accès le 2 3 2015].

[130] L. Andrews, Where's Waldo ? : Geolocation, Mobile Apps, and Privacy, SciTech Lawyer, 2013.

[131] E. Lee, «NFC hacking : The easy way,» *DefCon hacking conference,* vol. 20, pp. 63-74, 2012.

[132] H. Suhas et K. Mahima, «Android based mobile application development and its security,» *International journal of computer trends and technology,* vol. 31, n° 13, pp. 486-490, 2012.

[133] W. DeBorger, W. Joosen et B. Lagaisse, «A generic and reflective debugging architecture to support runtime visibility and traceability of aspects,» *Proceedings of the 8th ACM international conference on Aspect-oriented software development,* pp. 173-184, 2009.

[134] K. Kail, Reprogrammable remote sensor monitoring system, vol. 225, US Patent 6, 2001.

[135] J. Polo et J. Higuera, «Contribution toward interoperability of wireless sensor networks based on IEEE1451 in environmental monitoring applications,» *Barcelona Forum on Ph. D. Research in Communications,* 2010.

[136] L. Kang et E. Song, «Wireless sensor network based on IEEE 1451.0 and IEEE 1451.5-802.11,» *IEEE 8th International Conference on Electronic Measurement and Instruments,* 2007.

[137] H. Dinh, D. Niyato et P. Wang, «A survey of mobile cloud computing: architecture, applications, and approaches,» *Wireless communications and mobile computing,* vol. 13, n° 118, pp. 1587-1611, 2013.

[138] Y. Jararweh et A. Fadi, «Resource efficient mobile computing using cloudlet infrastructure,» *IEEE Ninth International Conference on Mobile Ad-hoc and Sensor Networks (MSN),* pp. 373-377, 2013.

[139] D. Georgakopoulos, K. Mitra et R. Rajiv, «MediaWise cloud content orchestrator,» *MediaWise cloud content orchestrator,* vol. 4, n° 11, pp. 1-14, 2013.

[140] A. Ghodsi, B. Hindman, A. Konwinski et S. Shenker, «Dominant Resource Fairness : Fair Allocation of Multiple Resource Types,» *USENIX Symposium on Networked Systems Design and Implementation,* vol. 11, pp. 24-24, 2011.

[141] Velte, V. Toby et Al., Microsoft Virtualization with Hyper-V, McGraw-Hill, Inc., 2009.

[142] R. Anshul, B. Ranjita et G. Saikat, Generalized Resource Allocation for the Cloud, Proceedings of the 3rd Symposium on Cloud Computing (SOCC), 2012.

[143] G. Genqiang, L. Qingchun, X. Wen et Al., «Comparison of open-source cloud management platforms: OpenStack and OpenNebula,» *9th International Conference on Systems and Knowledge Discovery (FSKD),* pp. 2457-2461, 2012.

[144] L. McAfee, «www.mcafee.com/fr/mcafee-labs.aspx,» [En ligne]. Available: www.mcafee.com/fr. [Accès le 01 Janvier 2017].

[145] C. Kyung-Yong, J. Yoo et K. Kim, «Recent trends on mobile computing and future networks,» *Personal and Ubiquitous Computing,* vol. 18, n° 13, pp. 489-491, 2014.

[146] D. Sanderson, Programming google app engine: build and run scalable Web apps on google's infrastructure, O'Reilly Media, Inc., 2009.

[147] M. Mohammad-Abu, R. Mizouni et S. Alzahmi, «Towards software product lines based cloud architectures,» *Cloud Engineering (IC2E), 2014 IEEE International Conference on. IEEE,* pp. 117-126, 2014.

[148] M. Armbrust, «A view of cloud computing,» *Communications of the ACM,* vol. 53, n° 14, pp. 50-58, 2010.

[149] B. Walters, «VMware virtual platform,» *Linux journal,* vol. 63, p. 6, 1999.

[150] J. Jiulei, L. Jiajin, H. Feng et Al., «Formalizing Cloud Service Interactions,» *Journal of Convergence Information Technology,* vol. 7, n° 113, p. 2012, 2012.

[151] C. Mahmoudi, «Orchestration d'agents mobiles en communauté,» Creteil, 2014.

[152] R. Boutaba et K. Chowdhuryr, «A survey of network virtualization,» *Computer Networks,* vol. 54, n° 15, pp. 862-876, 2010.

[153] M. Sammartino et U. Montanari, Network conscious pi-calculus, Pisa : Universita di Pisa, 2012.

[154] A. Singh, C. Ramakrishnan et A. Smolka, «A process calculus for mobile ad hoc networks,» *Science of Computer Programming,* vol. 75, n° 16, pp. 440-469, 2010.

[155] D. Walker et D. Sangiorgi, The pi-calculus : a Theory of Mobile Processes, Cambridge: Cambridge university press, 2003.

[156] R. Milner, Communicating and mobile systems : the pi calculus, Cambridge university press, 1999.

[157] M. Kumar et R. Napier, iOS 7 Programming Pushing the Limits: Develop Advance Applications for Apple iPhone, iPad, and iPod Touch, John Wiley & Sons, 2014.

[158] R. Milner et P. Joachim, «A calculus of mobile processes,» *Information and computation,* vol. 100, n° 11, pp. 1-40, 1992.

[159] J. Annuzzi, L. Darcey et S. Conder, Advanced Android Application Development, Addison-Wesley Professional, Addison-Wesley Professional, 2014.

[160] J. Wingfield, Developing a Windows Phone Application, Cardiff Metropolitan University, 2014.

[161] R. Milner, The polyadic π-calculus: a tutorial, Berlin Heidelberg, Springer, 1993.

[162] A. Stroud et G. Milette, Professional Android sensor programming, John Wiley & Sons, 2012.

[163] R. Milner, A calculus of communicating systems, Springer, 1980.

[164] C. Mahmoudi, F. Mourlin et G.-L. Djiken, «Vers une définition formelle du Cloud Computing Mobile,» *Conférence francophone sur les Architectures Logicielles,* pp. 64-79, 2016.

[165] T. Sridhar, L. Kreeger, D. Dutt et Al., Virtual eXtensible Local Area Network (VXLAN) : A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks, IETF, 2014.

[166] H. Dinh, C. Lee, D. Niyato et Al., «A survey of mobile cloud computing : architecture, applications, and approaches,» *Wireless communications and mobile computing,* vol. 13, n° 118, pp. 1587-1611, 2013.

[167] E. Cuervo et A. Balasubramanian, «MAUI : making smartphones last longer with code offload,» *Proceedings of the 8th international conference on Mobile systems, applications, and services,* pp. 49-62.

[168] A. Corradi, M. Fanelli et L. Foschini, «VM consolidation: A real case based on OpenStack Cloud,» *Future Generation Computer Systems,* vol. 32, pp. 118-127, 2014.

[169] B. Pfaff, K. Amidon et Al, Extending Networking into the Virtualization Layer, Hotnets, 2009.

[170] R. Fielding, «Architectural Styles and the Design of Netowork-based Software Architecture,» pp. 76-85, 2000.

[171] Alliance, «OSGi, Osgi service platform,» chez *IOS Press*, 2003.

[172] Felix, «Apache Felix-welcome,» 12 03 2015. [En ligne]. Available: http://felix.apache.org. [Accès le 26 03 2015].

[173] C. DUMONT, «Système d'agents mobiles pour les architectures de calculs auto-adaptatifs,» Thèse de doctorat, Paris Est-Creteil, 2014.

[174] D. LIME, O. ROUX, C. SEIDNER et Al, «Romeo: A parametric model-checker for Petri nets with stopwatches,» *International Conference on Tools and Algorithms for the Construction and Analysis of Systems,* pp. 54-57, 2009.

[175] B. Joakim, K. Jorgensen et J. SRBA, «TAPAAL : Editor, simulator and verifier of timed-arc Petri nets,» *International Symposium on Automated Technology for Verification and Analysis,* pp. 84-89, 2009.

[176] J. Bengtsson, K. Larsen et F. Larsson, UPPAAL a tool suite for automatic verification of real-time systems, Berlin: Heidelberg: Springer, 1996.

[177] M. Ben-ari, «Mathematical Logic for Computer Science, Technion-Israel Institute of Technology.,» *Prentice Hall International (UK) Ltd,* 1993.

[178] G. Hwang et F. Wang, «TCTL inevitability analysis of dense-time systems,» *Implementation and Application of Automata,* pp. 176-186, 2003.

[179] D. Dill, C. Courcoubetis et R. Alur, «Model-checking for real-time systems,» *Logic in Computer Science,* pp. 414-425, 1990.

[180] G. Behrmann, D. David et D. Larsen, «A tutorial on uppaal,» *Formal methods for the design of real-time systems,* pp. 200-236, 2004.

[181] S. Yovine, «Kronos : A verification tool for real-time systems,» *International Journal on Software Tools for Technology Transfer (STTT),* vol. 1, n° 111, pp. 123-133, 1997.

[182] T. Henzinger et H. Pei-Hsin, «HyTech : The Cornell hybrid technology tool,» *International Hybrid Systems Workshop,* pp. 265-293, 1994.

[183] H. Roussain et G. Frédéric, «Déploiement de composants logiciels sur des équipements mobiles comminicants : une approche coopérative,» *JC'05, Le Croisic, France,* pp. 83-88, 2008.

[184] M. Shaw, «Toward higher-level abstractions for software systems. Data & Knowledge Engineering,» vol. 5, pp. 119-128, 1990.

[185] P. Avgeriou et Z. Uwe, «Architectural patterns revisited : a pattern language,» *10th European Conference on Pattern Languages of Programs,* 2005.

[186] J. Domaschaka, H. Schmidt et F. Hauck, «D-OSGi : An architecture for instant replication,» *Supplement Proceedings of the 39th Annual IEEE/IFIP international Conference on Dependale Systems and Networks,* 2009.

[187] M. Sneps-Sneppe et D. Namiot, «M2M Applications and Open API : What Could Be Next ?,» *Internet of Things and Smart Spaces,* vol. 7469, pp. 429-439, 2012.

[188] K. Haeng-Kon, «Architecture for Adaptive Mobile Application,» *International Journal of Bio-Science and Bio-Technology,* vol. 5, n° 15, pp. 197-210, 2013.

[189] D. Morand, «Cilia : un framework pour le développement d'applications de médiation autonomiques,» Université Joseph Fourier, Grenoble I, 2013.

[190] C.-H. Chang, L. Chih-Wei, L. Chih-Hao et Al, «Experience of Applying Pattern-based Software Framework to Improve the Quality of Software Development : 4. The Design and Implementation of OS2F,» *Journal Software Engineering Studies,* vol. 2, n° 16, pp. 185-194, 2008.

[191] S. Bouzefrane, «Service architecture for multi-environment mobile cloud services,» *International Journal of High Performance Computing and Networking,* vol. 9, n° 14, pp. 1-15, 2016.

[192] C. Escoffier, R.-S. Hall et P. Lalanda, «iPOJO : An extensible service-oriented component framework,» *Services Computing, IEEE International Conference,* pp. 474-481, 2007.

[193] GitHub, GitHub, [En ligne]. Available: https://github.com/apiaryio/gavel-spec. [Accès le 24 Janvier 2018].