L'Efficience Énergétique des Services dans les Systèmes Répartis Hétérogènes et Dynamiques : Application à la Maison Numérique

Rémi Druilhe

▶ **To cite this version:**

Rémi Druilhe. L'Efficience Énergétique des Services dans les Systèmes Répartis Hétérogènes et Dynamiques : Application à la Maison Numérique. Modélisation et simulation. Université des Sciences et Technologie de Lille - Lille I, 2013. Français. NNT : . tel-00915321

HAL Id: tel-00915321

https://theses.hal.science/tel-00915321

Submitted on 7 Dec 2013

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.

Université Lille1
Sciences et Technologies

Département de formation doctorale en informatique **École doctorale SPI Lille**
UFR IEEA

L'Efficience Énergétique des Services dans les Systèmes Répartis Hétérogènes et Dynamiques :
Application à la Maison Numérique

THÈSE

présentée et soutenue publiquement le 5 Décembre 2013

pour l'obtention du

Doctorat de l'Université Lille 1 Sciences et Technologies

(spécialité informatique)

par

Rémi Druilhe

Composition du jury

Président :	Jean-Louis Pazat, *INSA, Rennes, France*
Rapporteurs :	Noel De Palma, *Université Joseph Fourier, Grenoble, France* Jean-Marc Menaud, *École des Mines de Nantes, France*
Examinateur :	Laurent Lefevre, *INRIA, Lyon, France*
Directrice :	Laurence Duchien, *Université de Lille I, France*
Co-Directeur :	Lionel Seinturier, *Université de Lille I, France*
Encadrant :	Matthieu Anne, *Orange, France*

Orange
Laboratoire d'Informatique Fondamentale de Lille — UMR USTL/CNRS 8022
INRIA Lille - Nord Europe

orange™ lifl Laboratoire d'Informatique Fondamentale de Lille Inria INVENTEURS DU MONDE NUMÉRIQUE cnrs

Que pensez-vous que l'histoire soit ?
Vous vivez naturellement l'histoire que vous connaissez.
Votre propre histoire.
Par exemple ... disons que cette histoire n'est rien que des mots.
Il n'y a aucun moyen de certifier l'exactitude de ces mots.
Ce n'est pas une histoire sombre, mais une fabulation.
Cette histoire prendra fin avec ce chapitre.
Raccordez-le, ouvrez-le puis fermez-le.

– Katanagatari

Remerciements

Je tiens à remercier en premier lieu mes encadrants académiques **Laurence Duchien** et **Lionel Seinturier** pour m'avoir guidé tout au long de ce travail de trois ans. Je sais que j'ai été râleur, pas très accommodant parfois mais grâce à vous, j'ai progressé aussi bien sur le plan personnel que sur le plan professionnel.

Un grand merci également à mon encadrant industriel officiel **Matthieu Anne**, ainsi qu'à mon encadrant industriel officieux **Jacques Pulou** pour votre soutien, vos conseils et pour le sang et la sueur que vous m'avez forcé à fournir pendant ces trois années. Vos retours au quotidien m'ont grandement éclairé et m'ont permis d'explorer toujours en profondeur mes idées.

Je tiens également à remercier **Noel De Palma** ainsi que **Jean-Marc Menaud** pour avoir accepté de rapporter mon travail. Merci également à **Laurent Lefevre** et **Jean-Louis Pazat** d'avoir accepté d'examiner mon travail et d'assister à la défense de ce doctorat.

Mes remerciements vont également à l'équipe MADE/MAAD/CARE d'Orange qui a changé tellement de fois de nom depuis mon arrivée. En particulier, je remercie mes amis les doctorants, anciens ou nouveaux, **Stéphane**, **Maxime**, **Charbel**, **Loic**, l'autre **Rémi**, **Erick** et **Amal** pour les discussions passionnées que nous avons pu avoir. À ceux qui m'ont accompagné et aidé en parallèle de mon doctorat : **Julien**, **Marc**, **Nordine**, **Anne** et les autres.

Un grand merci à l'équipe ADAM, future SPIRALS, animée par des doctorants et des ingénieurs agréable à côtoyer et passionnés par leurs travaux. Je retiendrai la bonne humeur des réunions d'équipe ou des activités de groupe.

Je ne peux pas terminer sans remercier ma famille et mes amis qui ont cru en moi tout au long de ces trois années.

Enfin, je dédis cette thèse à mes deux grands pères **Papi** et **Papou** qui n'ont pas vu l'aboutissement des mes études. Merci à eux pour leurs leçons de vie.

Résumé

La maison est un environnement dans lequel de plus en plus d'équipements sont interconnectés. Cette multiplication des équipements informatiques et des services associés augmente la consommation d'énergie de la maison numérique. Pour limiter cette tendance, les équipements de l'électronique grand public contrôlent leur consommation d'énergie individuellement, indépendamment les uns des autres. Les environnements répartis offrent de nouvelles opportunités de gestion de la consommation d'énergie des équipements.

Ce travail propose de mettre en place une coordination intelligente entre les équipements de façon à limiter la consommation énergétique de l'ensemble de ces équipements tout en délivrant les mêmes services. Notre approche consiste à déplacer des composants logiciels d'un équipement à l'autre afin de maximiser l'efficience énergétique de la maison numérique. Cependant, ces déplacements sont contraints à la fois par l'environnement, *e.g.,* ressources matérielles disponibles, et par les besoins des composants logiciels, *e.g.,* ressources matérielles requises, présence de l'utilisateur.

Pour concevoir un tel système, il est nécessaire de considérer les propriétés intrinsèques à la maison numérique dont l'hétérogénéité, la dynamicité et la qualité de service. L'hétérogénéité nécessite une modélisation de chaque équipement et de chaque service suivant des critères différenciant, *e.g.,* ressources matérielles, présence de l'utilisateur. La dynamicité requiert de modifier la répartition des composants logiciels lorsqu'un événement significatif survient, *e.g.,* apparition d'un équipement, afin de conserver l'efficience énergétique. Enfin, la mise en place de solutions moins énergivores ne doit pas impacter la qualité de service, *e.g.,* satisfaire les besoins en ressources matérielles des composants logiciels.

Nous proposons une modélisation de ces propriétés. Ce modèle est ensuite considéré par un système décisionnel autonome. Sur l'observation d'événements significatifs, le système prend la décision de modifier la répartition des composants sur les équipements afin d'atteindre l'objectif d'efficience énergétique tout en satisfaisant à la qualité de service.

L'implantation du système décisionnel est réalisée par une architecture elle-même conçue pour être efficiente énergétiquement. Le système décisionnel est considéré comme un service à part entière. Il est construit comme les autres services présents dans l'environnement, et est donc capable de se déplacer d'un équipement à un autre pour s'exécuter sur celui qui est le plus approprié en fonction du contexte.

L'approche est validée au travers de son implémentation appelée *HomeNap* et le déroulé de scénarios de la vie courante. Les résultats obtenus montrent des gains énergétiques. Ces résultats montrent également que les gains sont fonctions des usages de la maison numérique. Enfin, ces gains s'accroissent lorsque le nombre d'équipements et de services augmente, ce qui correspond à la tendance actuellement observée.

Mots-clés: Efficience Énergétique, Maison Numérique, Ingénierie Logicielle à base de Composants, Intergiciel

Abstract

The home is a more and more interconnected environment. This proliferation of computer devices and associated services increases the energy consumption of the home network. To limit this tendency, consumer electronic devices control their own energy consumption, independently from each other. Distributed environments offer new opportunities to manage the energy consumption of a set of devices.

This work proposes to set an intelligent coordination between devices in order to limit the energy consumption of the set of devices while providing the same services. Our approach consists, on one hand, in moving software components from one device to another in order to maximize the energy efficiency and, on the other hand, put into low power state unused devices. However, the migrations are constrained by both the environment, *i.e.,* available hardware resources, and the needs from software components, *i.e.,* required hardware resources, user presence.

To design such a system, it is necessary to consider the inherent properties of the digital home such as the heterogeneity, the dynamicity and the quality of service. Heterogeneity needs a model of each device and each service according to different criterion, *e.g.,* hardware resources, user presence. Dynamicity needs to modify the distribution of the software components when a relevant event occurs, *e.g.,* appearance of a device, in order to conserve energy efficiency. Finally, less energy consuming solutions must not impact the quality of service, *e.g.,* satisfying the required hardware resources of the software components.

We propose to model those properties. This model is then considered by an autonomic decision-making system. On the observation of relevant events, the system takes the decision to modify the distribution of the software components on the devices in order to reach the energy efficiency goal while satisfying to the quality of service.

The implementation of the decision-making system is embodied through an energy efficient architecture. The decision-making system is deemed to be a full service. It is designed as any others services deployed in the environment, and is able to move from one device to another to execute itself on the most appropriate one, considering the context.

The approach is validated through its implementation called *HomeNap* et using scenarios from the common life. Results show energetic gains. Results also show that those gains depend on the habits of the user in the digital home. Finally, those gains increase when the number of devices and services grows, which is the current observed tendency.

Keywords: Energy Efficiency, Digital Home, Component-based Software Engineering, Middleware

Table des matières

Chapitre 1 Introduction **1**

1.1 Introduction 2

1.2 La consommation d'énergie électrique dans le monde 2

1.3 Objectif des travaux 7

1.4 Plan du manuscrit 11

Partie I État de l'Art **13**

Chapitre 2 Contexte **15**

2.1 Introduction 16

2.2 Applications réparties 16

2.3 Adaptation autonome des systèmes informatiques 22

2.4 Conclusion 26

Chapitre 3 Gestion énergétique des systèmes informatiques **29**

3.1 Introduction 30

3.2 La gestion énergétique d'un équipement informatique 30

3.3 La gestion énergétique des systèmes répartis 39

3.4 Modification des usages grâce aux systèmes informatiques 47

3.5 Conclusion 47

Chapitre 4 Systèmes adaptatifs et énergie **49**

4.1 Introduction 50

4.2 Systèmes adaptatifs pour la réduction d'énergie 50

4.3 Conclusion 58

Partie II Contribution **61**

Chapitre 5 Modélisation **63**

5.1 Introduction 64

5.2 L'efficience énergétique des équipements informatiques 64

5.3 La maison numérique 70

5.4 Modélisation de la maison numérique 73

5.5 Utilisation des composants 79

5.6 Conclusion 84

Chapitre 6 Architecture **87**

6.1 Introduction 88

6.2 Vue d'ensemble de l'architecture 89

6.3 Adaptation du placement des composants 94

6.4 Contrôle des états énergétiques des équipements 99

6.5 Gestion des pannes 101

6.6 Conclusion 105

Partie III Validation **107**

Chapitre 7 Mise en œuvre d'*HomeNap* **109**

7.1 Introduction 110

7.2 Extension du modèle 111

7.3 Mécanismes de migration 116

7.4 Implémentation d'*HomeNap* 121

7.5 Conclusion 131

Chapitre 8 Évaluation **133**

8.1 Introduction 134

8.2 Évaluation de l'optimisation 135

8.3 Évaluation de l'approche 141

8.4 Conclusion 149

Partie IV Conclusion et Perspectives **153**

Chapitre 9 Conclusion et Perspectives **155**

9.1 Synthèse 156

9.2 Contributions 157

9.3 Perspectives 158

Bibliographie **163**

Annexes **173**

Annexe A Problème d'optimisation **175**

A.1 Le problème de *bin packing* 175

A.2 Résolution du problème de *bin packing* 176

Annexe B Diagrammes d'activité **179**

Liste des figures

1.1 Consommation d'énergie finale . . . 3
1.2 Répartition de la consommation d'électricité par secteur en France . . . 4
1.3 Consommation électrique des secteurs de l'informatique en France . . . 5
1.4 Consommation d'électricité dans une maison en France . . . 8
1.5 Niveaux d'adaptation énergétique . . . 10

2.1 Architecture Orientée Service . . . 21
2.2 Boucle autonome MAPE-K . . . 24

3.1 L'ACPI dans un équipement informatique . . . 32
3.2 Relations entre les états énergétiques de l'ACPI . . . 33
3.3 Efficience énergétique d'un processeur . . . 36
3.4 Réveil de la carte mère . . . 41
3.5 Pîle énergétique d'un système informatique . . . 48

5.1 Processus de transformation de l'énergie en service . . . 66
5.2 Transformation de l'énergie en service final via plusieurs processus . . . 80

6.1 Architecture du système de réduction de la consommation d'énergie . . . 90
6.2 Répartition de la boucle MAPE-K entre le coordinateur et les gestionnaires . . . 95
6.3 Cycle de vie des composants et activités de déploiement . . . 102

7.1 Architecture du coordinateur . . . 127

7.2 Architecture du gestionnaire 128

8.1 Temps de calcul de l'algorithme d'optimisation 136
8.2 Dette énergétique de l'algorithme sur différents équipements 138
8.3 Scénario d'évaluation 143
8.4 Résultats de la simulation 144
8.5 Service de surveillance 145
8.6 Résultats pratiques 146

A.1 Problème de *bin packing* à 2 dimensions 176

B.1 Mécanisme d'élection d'un nouveau coordinateur 180
B.2 Processus de migration 181

Liste des tableaux

4.1 Positionnement de notre approche par rapport à l'existant 59
4.2 Positionnement de notre approche par rapport à l'existant (suite) 60

6.1 Les événements déclenchant une optimisation du plan de répartition 97

8.1 Équipements utilisés pour les évaluations 135
8.2 Équipements utilisés pour la validation théorique 142
8.3 Composants utilisés pour l'évaluation pratique. 145
8.4 Synthèse de l'évaluation pratique avec l'approche 148

Chapitre 1

Introduction

La Force est ce qui donne au Jedi son pouvoir. C'est un champ d'énergie créé par tous les êtres vivants. Elle nous entoure et nous pénètre. C'est ce qui lie la galaxie en un tout uni.
– Obi-Wan Kenobi, *Star Wars*

Sommaire

1.1 Introduction **2**
1.2 La consommation d'énergie électrique dans le monde **2**
1.2.1 Progression de la consommation mondiale 3
1.2.2 Palliatifs à ce problème 4
1.2.3 L'énergie électrique dans l'informatique 5
1.3 Objectif des travaux **7**
1.3.1 La maison numérique 8
1.3.2 Les challenges 9
1.3.3 Présentation de l'approche 9
1.3.4 Contexte des travaux 11
1.4 Plan du manuscrit **11**

1.1 Introduction

L'énergie tire son nom du grec ancien *enérgeia* qui signifie « force en action ». Il s'agit d'un travail qui amène à une transformation du système soumis à cette force. L'énergie est présente sous diverses formes, *e.g.*, énergie solaire, énergie thermique, énergie chimique, énergie de masse. Elle ne peut être ni créée, ni détruite. Une fois dépensée elle réapparaît sous une autre forme.

Depuis toujours, l'Homme a su tirer profit de certaines de ces formes d'énergie pour ses propres besoins. Ainsi, il a très vite utilisé cette force pour se nourrir (*e.g.*, énergie thermique pour la cuisson), pour se déplacer (*e.g.*, énergie mécanique pour se mouvoir), pour se protéger (*e.g.*, énergie thermique pour se chauffer) ou pour accumuler d'autres forces de travail (*e.g.*, élevage d'animaux, esclavage).

Depuis la révolution industrielle, l'énergie a pris une place prépondérante dans la société moderne. Initialement utilisée dans l'industrie comme nouvelle source de production de biens communs, l'énergie s'est progressivement installée dans les environnements domestiques grâce à l'électricité.

Ce document traite uniquement de l'énergie électrique comme force de travail. Aussi le terme « énergie » utilisé tout au long de ce document est assimilé au terme « énergie électrique ».

Structure du chapitre

Ce chapitre s'articule comme suit : la Section 1.2 introduit l'énergie et son utilisation à travers le monde et dans le secteur de l'informatique. Section 1.3 présente les enjeux liés à l'énergie dans l'environnement domestique et introduit les challenges traités et l'approche proposée. Enfin, la Section 1.4 présente le plan de ce manuscrit.

1.2 La consommation d'énergie électrique dans le monde

L'énergie électrique est une énergie couramment utilisée dans la société moderne. Elle alimente nos éclairages, nos transports (*e.g.*, métro, train grande vitesse) ou encore nos systèmes informatiques (*e.g.*, ordinateur, routeur, antennes-relais de téléphonie mobile). L'électricité est omniprésente dans notre quotidien. Et pourtant, l'électricité brute n'est pas récupérée directement de la Nature, *e.g.*, via la foudre.

L'énergie électrique que nous consommons est le résultat d'une transformation d'une énergie primaire en énergie finale, *i.e.*, sa forme utile. L'énergie primaire est une énergie présente dans la Nature et que l'Homme adapte à son besoin. Cela comprend les énergies renouvelables (*e.g.*, énergie éolienne, énergie hydraulique), les énergies nucléaires (*e.g.*, uranium) et les énergies fossiles (*e.g.*, gaz, charbon, pétrole).

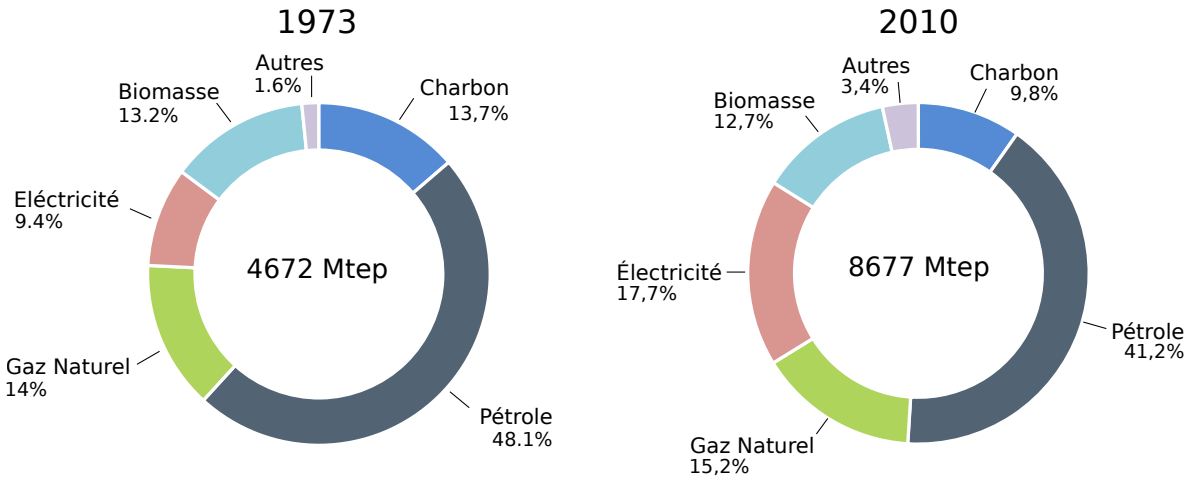

FIGURE 1.1 – **Consommation d'énergie finale.** Entre 1973 et 2010, l'énergie finale ayant eu la plus forte progression est l'électricité. Le consommation d'électricité a triplé en une quarantaine d'année [iae12]. L'unité d'énergie « tep » désigne « tonne équivalent pétrole ».

Toutes les énergies primaires ne sont pas converties en énergie électrique. Par exemple, le pétrole est également transformé en énergie mécanique pour faire avancer les voitures, le gaz est transformé en énergie thermique pour chauffer les logements. Toutefois, l'énergie électrique a su s'imposer comme une énergie d'avenir (cf. Figure 1.1). Il s'agit d'une énergie facile et rapide à transporter une fois produite. Toutefois, son stockage pose encore des problèmes d'où la nécessité d'équilibrer à tout instant la production et la consommation.

1.2.1 Progression de la consommation mondiale

La demande en électricité mondiale croit d'environ 3 % par an [Fur12]. Cette augmentation est principalement due aux pays en voie de développement de plus en plus gourmand en énergie afin de satisfaire les besoins des populations. Mais les pays développés participent également à l'accroissement de la demande en électricité, à cause du changement des usages.

Dans le cas de la France, le secteur résidentiel-tertiaire est devenu le principal consommateur d'électricité aux alentours des années 1980 (cf. Figure 1.2).

- Dans le secteur résidentiel, l'électricité est utilisée pour se chauffer, pour cuisiner, pour s'éclairer ou encore pour se divertir.
- Dans le secteur tertiaire, l'électricité alimente les climatiseurs des bureaux, les centres de traitement de données, ou encore les équipements de télécommunication.

Ainsi, la demande en énergie électrique continue de croître. Or l'énergie électrique que nous utilisons est issue de la transformation d'une énergie primaire. Aussi pour continuer

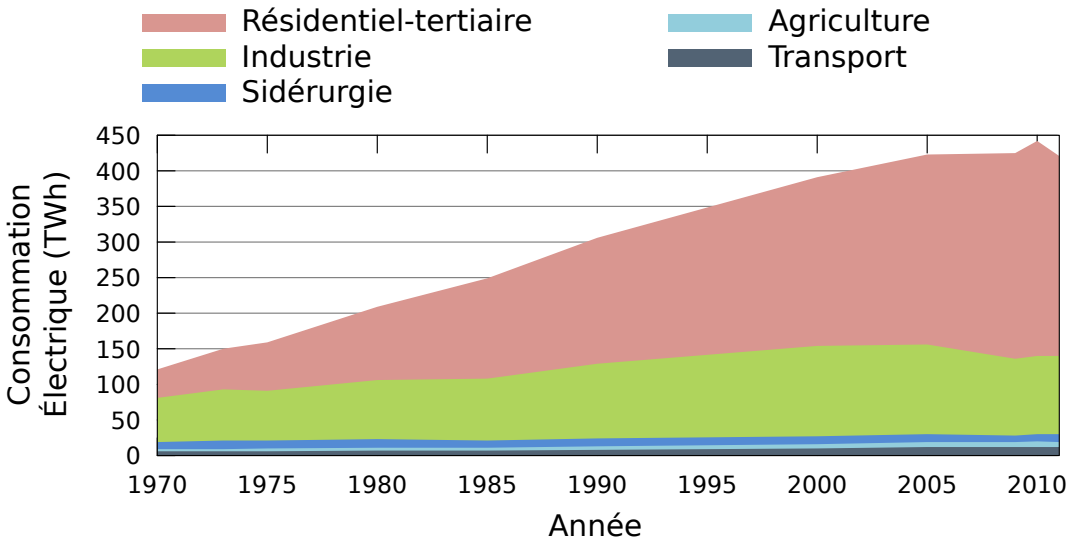

FIGURE 1.2 – **Répartition de la consommation d'électricité par secteur en France.** Entre 1970 et 2010, la part de consommation d'électricité dans le secteur résidentiel-tertiaire a bondi de 700% en France [CGD12].

à suivre le rythme d'accroissement de 3 % d'électricité consommée chaque année, il faut transformer toujours plus d'énergie primaire en électricité.

1.2.2 Palliatifs à ce problème

A l'heure actuelle, il existe deux solutions pour palier le problème de la demande grandissante en énergie électrique : la construction de nouveaux moyens de production d'électricité ou la réduction de la demande en électricité.

La première solution nécessite la construction de nouvelles usines électriques. Ces nouvelles usines accroissent la demande en énergie primaire afin de produire plus d'électricité. Toutefois, cette solution a pour inconvénient d'accroître l'empreinte environnementale, *e.g.*, déforestation, rejet de carbone dans l'atmosphère.

La deuxième solution nécessite de réduire la demande en électricité en modifiant les usages. Des initiatives comme « une heure sans lumière »[1] visent à sensibiliser les utilisateurs à leur usage de l'énergie électrique. Toutefois, cette solution est difficile à faire accepter aux utilisateurs malgré une prise de conscience grandissante [dev06].

Un compromis consiste à réduire la consommation d'énergie tout en fournissant le même niveau de confort et de qualité de service. Il s'agit de l'« efficience énergétique ». Cette réduction de la consommation électrique se fait en modifiant le fonctionnement des outils consommant ces énergies tout en restant transparent pour l'utilisateur. C'est cette solution que nous mettons en avant dans ce travail.

1. http://earthhour.fr (2013)

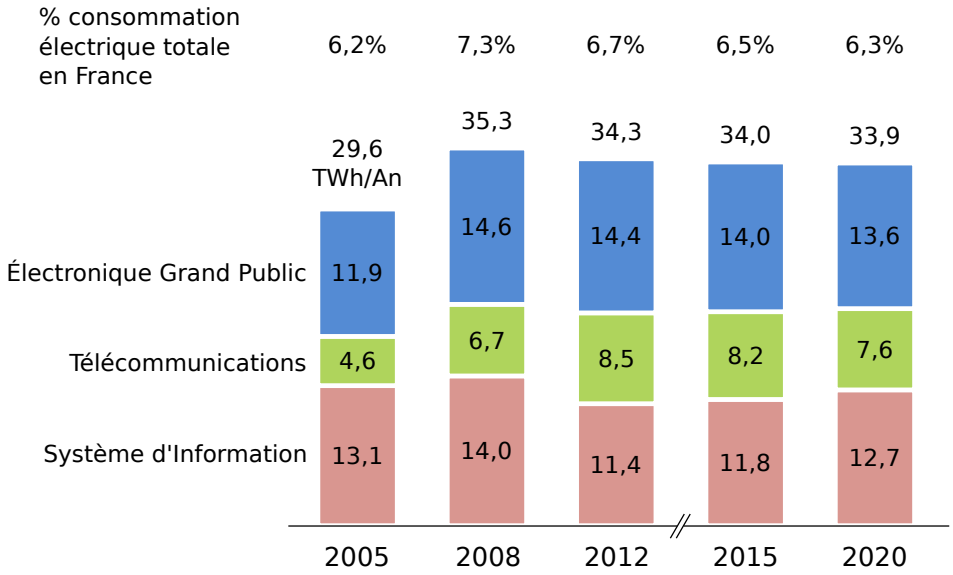

FIGURE 1.3 – **Consommation électrique des secteurs de l'informatique en France en France.** Le consommation électrique de l'ensemble des équipements grand public atteint celui des systèmes d'informations [Sal10]. De plus, dans les années à venir (2015-2020), cette consommation tend à se stabiliser tous domaines confondus sous l'effet des politiques d'efficience énergétique.

1.2.3 L'énergie électrique dans l'informatique

Le secteur informatique est un grand consommateur d'électricité. L'électricité est d'ailleurs la ressource sur laquelle se base ce secteur pour fonctionner. La croissance en électricité de ce secteur est d'environ 6,6 % par an [LLVH+13, PVD+08]. La généralisation et l'accroissement des équipements électriques ou électroniques permettant de produire, transporter ou consommer des services entraîne une demande en électricité en conséquence.

Dans l'industrie, l'informatique mobilise de l'électricité en grande quantité pour qu'un client puisse profiter d'un service. Il y a trois domaines industriels grands consommateurs d'électricité en informatique (cf. Figure 1.3) :

Les systèmes d'information. Ce terme désigne toutes les infrastructures informatiques dans les entreprises, soit pour son fonctionnement interne, soit pour sa production. Par exemple, cela concerne les acteurs fournissant des services comme OVH, Google ou Yahoo. Ces services sont fournis grâce à des centres de traitement de données. L'électricité alimente les serveurs et la climatisation de ces derniers.

Les télécommunications. Ce terme désigne l'ensemble des équipements reliant les fournisseurs de services à ceux qui les consomment. Les télécommunications regroupent entre autres les fournisseurs d'accès Internet ou mobile comme Orange, AT&T ou encore

Vodafone. L'électricité alimente les équipements comme les routeurs ou encore les antennes relais.

L'Électronique Grand Public (EGP). Ce terme désigne l'ensemble des équipements terminaux utilisés par les consommateurs de services. L'EGP regroupe entre autres les tablettes, les téléphones intelligents ou encore les ordinateurs personnels qui sont alimentés par l'électricité.

En plus de ces domaines, la recherche en définit d'autres où la problématique de l'énergie est importante. Pour simplifier, nous ne citons que les deux domaines qui sont les plus étudiés à l'heure actuelle et qui diffèrent de ceux présentés auparavant :

Les réseaux de capteur sans fil. Un réseau de capteurs sans fil est constitué d'une multitudes de systèmes embarqués comprenant un ou plusieurs capteurs, un module de communication et une batterie. Ces systèmes embarqués communiquent entre eux pour remonter les informations qu'ils recueillent grâce à leurs capteurs. Une fois déployés, les systèmes embarqués n'ont pas vocation à se déplacer.

Les réseaux mobiles *ad hoc*. Les réseaux mobiles *ad hoc*, ou *MANets*[2], sont composés d'équipements alimentés par batterie reliés par un réseau *ad hoc*. Ces équipements sont amenés à se déplacer, nécessitant ainsi une reconfiguration du réseau. Par exemple, il s'agit de réseaux de téléphones portables.

L'ensemble de ces domaines, autant dans l'industrie que dans la recherche, est concerné par la consommation d'énergie mais pour des raisons qui peuvent différer :

Financier. Cet enjeu est important pour les domaines où les équipements sont branchés sur le secteur, *i.e.*, une source d'électricité ininterrompue. Cela concerne les centres de traitement de données afin que l'industriel ne dépense pas des fortunes pour fournir des services. Cela concerne également les télécommunications afin que le coût de transfert des données soit le plus faible possible. Enfin, cela concerne l'EGP car les clients finaux sont souvent des ménages qui cherchent à contrôler leurs économies.

Autonomie. Cet enjeu est important pour tous les domaines employant des équipements ayant une batterie et donc une durée de vie de l'équipement définie par sa consommation d'énergie. Cela concerne une partie de l'EGP avec les téléphones intelligents ou les tablettes qui fonctionne sur batterie pour pouvoir consommer des services plus longtemps. Cela concerne également les réseaux de capteurs sans fil afin de pouvoir remonter des données plus longtemps. Enfin, cela concerne aussi les réseaux mobiles *ad hoc* afin de conserver plus longtemps le réseau de communication.

2. Mobile Ad hoc Networks

Écologique. Enfin dans une moindre mesure puisqu'il ne s'agit souvent pas de la raison première, chaque domaine cherche à faire des économies d'énergie car cela leur permet d'avoir une bonne image auprès du grand public.

Dans les réseaux mobiles *ad hoc*, les réseaux de capteurs sans fil ou les centres de traitement de données, la recherche a largement contribué à réduire leur consommation en électricité. Elle propose notamment des solutions où les équipements cherchent à se coordonner entre eux afin de diminuer la consommation globale de l'environnement. Cela peut se faire au détriment d'un équipement si cela réduit la consommation globale.

La particularité de ces domaines est qu'il sont homogènes en ressources matérielles, *i.e.*, un équipement en vaut un autre. Aussi, il est assez facile de le substituer par un autre équipement une fois que le premier n'est plus alimenté. Mais cette solution n'est pas transposable dans l'EGP où les équipements sont très différents les uns des autres.

Dans l'EGP, il existe des solutions limitant leur consommation d'énergie. Toutefois, ces solutions sont souvent propres à l'équipement, *e.g.*, variation de la fréquence du processeur en fonction du besoin. Elles ne considèrent pas un ensemble d'équipements collaborant entre eux afin de réduire leur consommation d'énergie.

1.3 Objectif des travaux

La maison est le parfait exemple d'un environnement hétérogène en équipements où il n'existe aucune coordination entre eux afin de réduire la consommation globale d'énergie. À l'origine, chaque équipement était destiné à fournir un service particulier. Avec le temps, les services sont restés mais les équipements ont évolué et sont devenus plus polyvalents permettant ainsi de fournir un nombre toujours plus important de services. Ainsi, un téléphone intelligent permet de regarder la télévision, d'écouter la radio et accessoirement de téléphoner. Il en va de même avec l'ordinateur portable se trouvant chez les ménages.

La maison est également un bon cas d'étude car sa consommation en électricité tend à augmenter avec l'arrivée de nouveaux équipements informatiques [Lav11]. Chercher à réduire leur impact sur la consommation électrique des ménages est une nécessité pour accélérer leur adoption. Il est tout de même important de relativiser cette hausse. Pour l'heure, les principaux postes consommateurs d'électricité demeurent le chauffage et l'eau chaude sanitaire (cf. Figure 1.4). Mais cette hausse est tout de même présente et tend à se poursuivre.

Ainsi l'objectif des travaux présentés dans ce manuscrit vise à faire des économies d'énergie dans une environnement hétérogène comme celui de la maison. Toutefois, nous nous limitons à ce que nous appelons la « maison numérique » que nous décrivons ci-après.

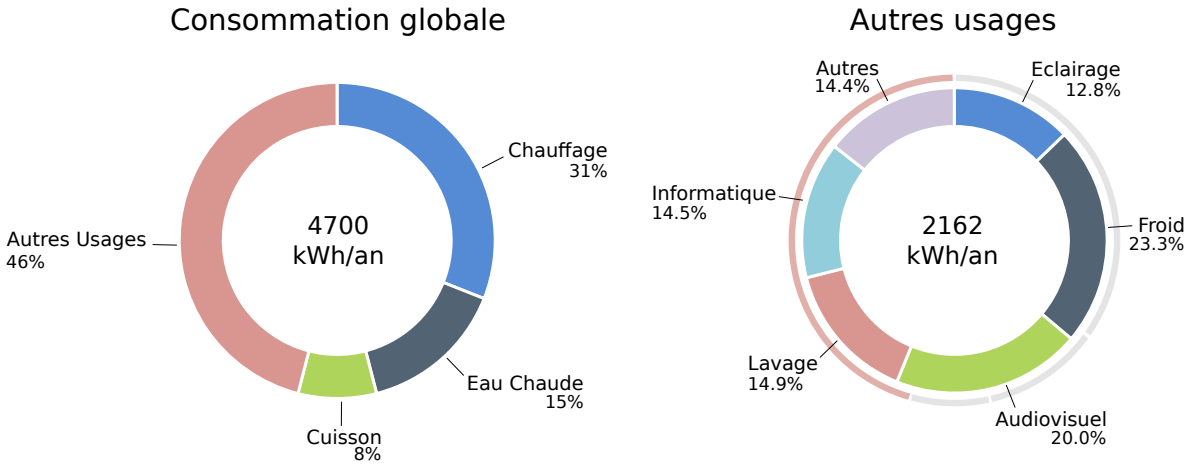

FIGURE 1.4 – **Consommation d'électricité dans une maison en France.** Dans une maison alimentée uniquement par l'électricité, le chauffage demeure le principal consommateur d'énergie [Rem08]. Toutefois, le poste informatique et le poste audiovisuel tendent à prendre une part toujours plus conséquente dans cette consommation [Lav11].

1.3.1 La maison numérique

Nous appelons « maison numérique » l'ensemble des équipements informatiques communiquant et hébergeant un environnement d'exécution qui composent l'environnement domestique. Il s'agit donc des ordinateurs personnels, des téléphones intelligents, des passerelles résidentiels, des boîtiers décodeur, etc. Les écrans, les capteurs ou le réfrigérateur ne font pas partis des équipements considérés car ils ne peuvent pas communiquer avec d'autres équipements ou ne peuvent pas héberger de nouvelles applications.

La maison numérique se distingue des autres domaines par les cinq propriétés la caractérisant. Ces propriétés sont discutées plus longuement dans le Chapitre 5.

L'hétérogénéité. Les différences qui caractérisent au moins deux équipements, *e.g.*, systèmes d'exploitation, ressources matérielles, consommations d'énergie.

La volatilité. L'apparition ou la disparition imprévisible d'équipements ou de services, *e.g.*, départ d'un téléphone portable, arrêt d'un film.

La répartition. La fourniture d'un service au travers de plusieurs équipements, *e.g.*, diffusion en flux d'un film depuis un serveur de stockage en réseau vers une télévision.

L'ouverture aux tiers. La capacité d'intégrer et de gérer de nouveaux équipements ou services par des entreprises tiers, *i.e.*, qui ne sont pas celles ayant participer à la conception ou à la vente de l'équipement.

La qualité de service. La satisfaction des critères requis par un utilisateur et traduits dans une spécification par un architecte, *e.g.*, résolution de l'image.

1.3.2 Les challenges

Réduire la consommation d'énergie de la maison numérique doit se faire en considérant ces propriétés. Ainsi, les challenges que nous adressons sont au nombre de quatre :

Prise en compte de l'hétérogénéité. Proposer une approche s'appliquant à un environnement hétérogène en équipements. L'hétérogénéité de cet environnement ne se borne pas à des systèmes d'exploitation différents, mais également à des consommations d'énergie différentes, des ressources matérielles différentes et des usages différents sur les équipements.

Prise en compte de la volatilité. Proposer une approche s'adaptant aux événements imprévisibles de la maison numérique. Cela comprend l'apparition et le disparition des équipements et des services en fonction des habitudes des utilisateurs.

Conservation de la qualité de service. Conserver le même niveau de service pour les utilisateurs. Les usages de l'utilisateur ne doivent pas être modifiés ou dégradés sinon, l'approche risque de ne pas être acceptée. Aussi, les besoins des services doivent être spécifié pour être satisfaits lorsque l'utilisateur consomme le service.

Amélioration de l'efficience énergétique. Réduire la consommation d'énergie de la maison numérique en conservant la qualité de service et en prenant en compte la volatilité et l'hétérogénéité de la maison numérique. La mise en œuvre de la solution doit également se considérer dans la recherche d'une solution.

Dans ce travail, nous prenons l'hypothèse que le challenge de l'ouverture au tiers est résolu. Aussi, nous supposons que nous avons la possibilité de déployer une application sur un équipement conçu par un tiers.

Nous prenons également l'hypothèse que le challenge de la répartition est résolu. Notre approche se base sur cette propriété pour réduire la consommation d'énergie de la maison numérique.

1.3.3 Présentation de l'approche

Les services présents dans la maison numérique ne sont pas toujours liés à un équipement spécifique. Par exemple, l'encodage d'un film est faisable sur différents différents équipements. Il en va de même pour le téléchargement d'un fichier. Aussi, il existe des services, ou des parties d'un service, qui peuvent s'exécuter indépendamment de l'équipement sur lequel ils sont déployés.

Toutefois, tous les services ne peuvent pas être fournis par tous les équipements. Par exemple, un film peut être regardé sur différents équipements mais doit *in fine* s'afficher sur

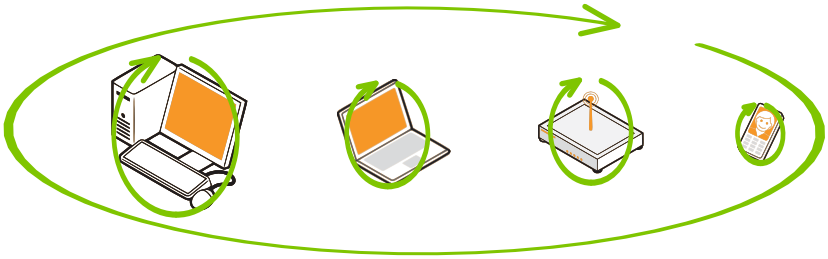

FIGURE 1.5 – **Niveaux d'adaptation énergétique.** Il y a deux niveaux d'adaptation énergétique. Le premier est une adaptation au niveau service afin de que les services soient fournit par les équipements les plus adaptés. Cette adaptation nécessite une coordination entre tous les équipements. Le deuxième est une adaptation au niveau des équipements où chaque équipement inutilisé suit sa politique d'économie d'énergie.

un écran devant l'utilisateur. Aussi, si l'utilisateur consomme ce service sur l'équipement le plus énergivore, alors cet équipement ne peut pas être éteint afin d'économiser de l'énergie.

L'approche consiste alors à migrer les services qui peuvent l'être sur l'équipement actuellement utilisé pour regarder le film tout en veillant à conserver la qualité de service pour ne pas déranger l'utilisateur. Les autres équipements, devenant inutilisés suite à cette migration, sont placés dans un état de basse consommation.

Aussi, notre approche est double. Chercher à faire des économies d'énergie en plaçant les services sur les équipements les plus adaptés en fonction de l'environnement et ensuite suivre la politique énergétique des équipements lorsqu'ils ne sont plus utilisés (cf. Figure 1.5). Lorsqu'un équipement est inutilisé sa politique énergétique lui dicte le plus souvent de passer dans un état de basse consommation. C'est ainsi que les économies d'énergie sont faites.

Par la suite, lorsqu'un événement significatif survient, *e.g.*, apparition d'un nouvel équipement, la même opération de répartition est répétée afin d'avoir un environnement consommant toujours le moins d'énergie possible. Cette opération tient toujours compte des propriétés de la maison numérique que nous considérons. Si nécessaire, un équipement est réveillé afin d'héberger des services.

Pour parvenir à ce résultat, nous supposons que les services modernes sont fournis par des applications conçues à base de composants. L'usage de ces applications augmente les possibilités de répartition des applications dans l'environnement et augmente ainsi l'efficience énergétique.

Notre approche est basée sur une modélisation de la maison numérique. Cette modélisation rend compte de l'hétérogénéité, notamment les ressources matérielles, la consommation d'énergie et la présence de l'utilisateur. Cette modélisation considère également la volatilité de l'environnement au travers de la représentation d'un ensemble d'événements significatifs et d'un ensemble d'actions que le système peut prendre.

Cette approche s'inscrit également au travers d'un système autonome. Ce système se considère dans la maximisation de l'efficience énergétique. Ce système est réactif, *i.e.*, il effectue des migrations d'applications uniquement lorsque des événements significatifs surviennent afin de limiter sa consommation d'énergie. Enfin, l'algorithme décidant sur quel équipement une application va être hébergée respecte la contrainte de conservation de la qualité de service.

Notre système est mis en œuvre au travers du prototype *HomeNap*. Les technologies utilisées se retrouvent dans les équipements modernes qui peuplent la maison numérique, *e.g.*, UPnP, Java ou qui sont en passe d'être déployées dans cet environnement, *e.g.*, OSGi. La faisabilité et la possibilité de son intégration dans cet environnement sont ainsi validés.

La validation évalue également les performances d'*HomeNap*. Les résultats montrent que lorsque *HomeNap* est actif, des gains énergétiques sont constatés et que plus il y a de services et d'équipements, plus les gains sont importants. Toutefois, ces évaluations montrent également que les habitudes des utilisateurs ont un impact important sur les gains énergétiques générés par *HomeNap*. Aussi, nous proposons en conclusion quelques pistes d'amélioration de notre système.

1.3.4 Contexte des travaux

Ces travaux sont le résultat d'une thèse CIFRE, effectuée au travers du projet DigiHome, une collaboration entre Orange et l'équipe-projet ADAM de l'Inria [3] et du LIFL [4]. Ces travaux s'inscrivent dans le cadre du projet FUI EconHome [5]. Ce projet vise à réduire de 70 % la consommation électrique de la maison numérique.

1.4 Plan du manuscrit

Ce document est divisé en quatre parties. La première partie offre une base de connaissance afin de comprendre la suite du manuscrit. Elle propose également un état de l'art des approches existantes réduisant la consommation d'énergie d'un système informatique ainsi qu'un positionnement par rapport à ces travaux. La deuxième partie détaille la contribution au travers du modèle et de l'architecture. La troisième partie valide l'approche au travers de son implémentation et de ses performances. Enfin, la quatrième partie conclut.

Partie 1 : État de l'art.

Chapitre 2: Contexte. Ce chapitre détaille la base de connaissances nécessaire à la compréhension de notre contribution, à savoir les applications réparties à base de composants orientés services. Ce chapitre traite également des systèmes autonome et de leur adaptation à un environnement changeant.

3. Institut National de Recherche en Informatique et en Automatique
4. Laboratoire d'Informatique Fondamentale de Lille
5. http://www.systematic-paris-region.org/fr/projets/econhome-v2 (2013)

Chapitre 3: Gestion énergétique des systèmes informatiques. Ce chapitre décrit les approches réduisant la consommation d'énergie d'un équipement, d'un ensemble d'équipements informatique ou de modifier le comportement des utilisateurs.

Chapitre 4: Systèmes adaptatifs et énergie Ce chapitre présente les systèmes autonomes existants réduisant la consommation d'énergie d'un ensemble d'équipements. Chaque système utilise plusieurs approches de réduction de la consommation d'énergie décrites dans le chapitre précédent. Nous positionnons notre approche par rapport à ces travaux.

Partie 2 : Contribution.

Chapitre 5: Modélisation. Ce chapitre présente le modélisation des propriétés que nous considérons : l'hétérogénéité, la volatilité, la qualité de service et l'efficience énergétique. Nous présentons également l'avantage procuré par les composants logiciels pour améliorer l'efficience énergétique de la maison numérique.

Chapitre 6: Architecture. Ce chapitre décrit l'architecture logicielle prenant en compte le modèle et proposant une adaptation en fonction des événements significatifs que nous traitons. Il décrit également les mécanismes utilisés afin que notre architecture soit également efficiente énergétiquement.

Partie 3 : Validation.

Chapitre 7: Mise en œuvre d'*HomeNap*. Ce chapitre présente la mise en œuvre du modèle et de l'architecture au travers de l'intergiciel *HomeNap*. Nous discutons certains aspects de notre approche qui ne sont pas traités dans la contribution mais qui sont nécessaires à son implémentation.

Chapitre 8: Évaluation. Ce chapitre évalue notre approche au travers d'*HomeNap*. Ces évaluations portent sur les performances de l'algorithme et de l'architecture au travers d'une série d'expériences basées sur des scénarios de la vie courante.

Partie 4 : Conclusion et Perspectives.

Chapitre 9: Conclusion et Perspectives. Ce chapitre conclut notre travail en rappelant les challenges considérés. Il présente également les perspectives à court et à long terme.

Première partie

État de l'Art

Chapitre 2

Contexte

L'E2PZ extrait l'énergie produite depuis une région artificielle de sous-espace-temps au moment où elle atteint son entropie maximale
– Dr. Radek Zelenka, *Stargate*

Sommaire

2.1 Introduction 16
2.2 Applications réparties 16
2.2.1 Approche à composants 17
2.2.2 Approche orientée services 19
2.2.3 Composants orientés services 21
2.3 Adaptation autonome des systèmes informatiques 22
2.3.1 Boucle autonome MAPE-K 24
2.3.2 Politiques de décision 25
2.4 Conclusion 26

2.1 Introduction

Notre approche cherche à modifier la répartition des services à l'exécution afin de toujours consommer le minimum d'énergie. Pour cela, nous avons besoin que le système informatique s'adapte à l'environnement. Cette adaptation à l'exécution d'un système informatique doit être pensée à sa conception. La connaissance des approches facilitant la modification structurelle d'une architecture logicielle est nécessaire afin de comprendre le travail présenté dans ce document.

Ce chapitre introduit les concepts généraux sur lesquels notre travail s'appuie. Il présente d'abord les approches permettant de manipuler une architecture logicielle. Pour cela, nous nous basons sur l'approche à composants orientés services qui marie l'approche à composant et l'approche orientée services. Cela nous permet de proposer des architectures de services flexibles, *i.e.*, dont l'architecture est modifiable sans altérer l'objectif fonctionnel.

Ce chapitre présente également les systèmes autonomes, capable de s'adapter à un environnement changeant. Ces systèmes permettent de manipuler les architectures de services flexibles en vue d'atteindre un objectif non fonctionnel, *i.e.*, dans notre cas, celui de l'efficience énergétique.

Structure du chapitre

Ce chapitre s'articule comme suit : la Section 2.2 présente les concepts généraux pour la conception d'application répartie. Elle discute notamment de l'approche à composants, de l'approche orientée services et des composants orientés services. La Section 2.3 présente les mécanismes d'adaptation architecturaux et décisionnels. Enfin la Section 2.4 conclut ce chapitre.

2.2 Applications réparties

La création d'une application par un architecte est dictée par la nécessité de satisfaire le besoin d'un client. Comme le client n'est pas nécessairement du domaine informatique, l'architecte traduit les besoins du client en besoins fonctionnels. Ces besoins fonctionnels sont ensuite décrits dans une spécification fonctionnelle qui sert de modèle pour la réalisation de l'application.

Ce document décrit également les besoins non fonctionnels portés par différents acteurs utilisant l'application. Par exemple, il peut s'agir de besoins de fiabilité ou d'efficience énergétique du service. Ces besoins non fonctionnels contraignent l'architecte lors de la conception de l'application. Par exemple, cela lui impose des choix architecturaux ou des technologies particulières. Les applications réparties répondent à certains besoins non fonctionnels qu'un architecte peut rencontrer.

Une application répartie exécute des modules différents sur un ensemble d'équipements informatiques interconnectés par un réseau. Les modules sont des blocs fonctionnels reliés les uns aux autres. Chaque bloc fonctionnel peut être déployé sur des équipements différents, conduisant à la répartition de l'application. Cette répartition est cachée aux yeux de l'utilisateur qui ne voit qu'un seul et unique point d'entrée fournissant un service.

L'approche à composants et l'approche orientée services permettent la conception d'applications réparties. L'approche orientée services diffère de l'approche à composant en produisant des services depuis des composants autonome et externe à l'application les consommant. Ainsi, l'ingénierie logicielle orientée service est un type d'ingénierie à composant utilisant une notion de composants plus simple [Som10].

Dans cette section, nous détaillons les propriétés correspondantes à ces deux approches de conception d'applications réparties. Enfin, nous présentons également les composants orientés services, mixant les avantages des deux approches précédemment citées.

2.2.1 Approche à composants

L'approche à composants est apparue à la fin des années 1960 [MBNR68]. Les composants permettent de s'abstraire des langages de programmation et de considérer seulement les services qu'ils fournissent. Cette approche vise à faciliter la spécification et la conception d'une application. Un composant logiciel est défini comme suit :

Définition 1 : Composant logiciel

Un composant est une unité de composition avec des interfaces définies contractuellement et des dépendances de contexte. Un composant logiciel peut être déployé indépendamment et est sujet à composition par des tiers [SGM02].

L'approche à composants répond aux propriétés suivantes.

Modularité et composants. La modularité logicielle est un principe de conception d'applications dans lequel les différents services sont fournis par des composants distincts. Cette séparation des fonctions permet de diviser le travail de conception entre plusieurs équipes de développement indépendantes. Ainsi un composant fournit un service indépendant, séparé et complet vis-à-vis d'autres composants. Cela implique qu'un composant ne peut pas être déployé partiellement.

Composition de composants. Un service est fourni par un composant. Toutefois, un service peut également être l'œuvre de plusieurs composants fonctionnant de concert. Ce service est alors une composition de services issus de composant et peut être vu comme issu d'un seul composant, englobant les composants participant à le rendre.

Facilité de conception. En séparant les services dans des composants, la conception des applications s'en trouve facilitée. Un architecte se retrouve avec des composants qu'il faut lier ensemble afin de produire une application plus générale. Cette liaison est opérée soit par composition, soit en les reliant afin qu'ils communiquent entre eux. En s'abstrayant de l'implémentation, il est possible de décrire les relations entre les composants, *i.e.*, composition ou liaison.

Séparation des préoccupations. La séparation des préoccupations consiste à considérer l'ensemble des points de vue des acteurs travaillant sur la spécification de l'application. La spécification s'occupe de séparer le besoin fonctionnel, *i.e.*, ce que l'application doit faire, des besoins non fonctionnels portés par les acteurs, *i.e.*, qui ne concerne pas le besoin fonctionnel mais qui est nécessaire pour être utilisé par un acteur.

Par exemple, dans une application de stockage de données, le besoin fonctionnel est de stocker les données. Un acteur souhaite mettre en avant la sécurité tandis qu'un autre souhaite que l'application soit fiable. Dans ce cas, la sécurité et la fiabilité sont des besoins non fonctionnels.

Cycle de vie. Le cycle de vie des composants fait partie intégrante de l'approche à composants. Le cycle de vie d'un composant définit les états dans lesquels peuvent se retrouver les composants, depuis leur déploiement, en passant par leur exécution, jusqu'à leur remplacement. Chacun des états décrit un ensemble de tâches nécessaires au déploiement d'un composant.

Par exemple, dans le cycle de vie décrit dans [CFH+98], l'état d'installation prévoit une tâche de transfert du composant sur l'équipement qui va l'exécuter puis une tâche de configuration de ce composant. Par la suite, l'état d'activation s'occupe de la tâche de démarrage du composant.

Réflexivité. Dans les modèles à composants évolués, la séparation des préoccupations est mise en œuvre grâce à la séparation entre le code de base, *i.e.*, répondant au besoin fonctionnel, et le code de contrôle, *i.e.*, répondant aux besoins non fonctionnels [BCL+06]. Dans ce genre de modèle, il est possible de récupérer à l'exécution des informations sur les composants (*e.g.*, liaisons, propriétés, état) grâce du code de contrôle, voir à les modifier.

Par exemple, lorsqu'un composant démarre, un code de contrôle le relie à un autre composant. Par introspection, il est possible de connaître à quel composant il est relié. Et par intercession, il est possible de modifier ce lien afin de le relier à un autre composant fournissant le même service.

Réutilisation. Un composant est défini au travers d'une spécification plus ou moins détaillée. Dans le cas où le niveau de détail de la spécification est faible, *i.e.*, boite noire, seules les préconditions et les postconditions sont accessibles à un développeur tiers. Dans le cas où la spécification est très détaillée et où tous les détails sont accessibles à un développeur tiers, il s'agit de boite blanche.

Enfin, un niveau entre les deux correspond à des boites grises, ne détaillant pas trop le fonctionnement interne du composant mais suffisamment pour que le développeur tiers puisse se faire une idée rapide du composant qu'il utilise [BW97]. Le niveau de détail participe à la réutilisation des composants. Ainsi, un développeur tiers passe plus ou moins de temps à choisir les composants qu'il utilise dans son architecture afin d'éviter de les concevoir de zéro.

2.2.2 Approche orientée services

L'approche orientée services fait suite, chronologiquement, à l'approche à composant, mais diffère dans la manière dont l'application est conçue [Pap03]. L'approche orientée service part du paradigme que les modules sont construits par des architectes différents et indépendants. Dans cette approche, un module est un maillon d'un ensemble plus vaste de modules.

Ainsi, un module fournit un service à un ou plusieurs autres modules ou à un utilisateur final. Pour permettre cela, il faut mettre en place un couplage lâche entre les modules permettant ainsi à des architectes indépendants de réutiliser des services pour ajouter une plus-value au service qu'ils proposent. Ainsi, un service est défini comme suit :

Définition 2 : Service logiciel

Un service est un mécanisme permettant l'accès à une ou plusieurs fonctionnalités, où l'accès est fournit à travers une interface prescrite et est utilisée en accord avec des contraintes et des politiques spécifiées par la description du service [OAS06].

L'approche orientée services répond aux propriétés suivantes.

Abstraction. L'approche orientée services vise à s'abstraire de l'implémentation d'un service. La manière dont un service est rendu par un module est donc cachée aux clients, *i.e.*, boite noire. Ainsi seule les préconditions et postconditions du module sont disponibles pour une réutilisation par un architecte tiers.

Contrat de service. Dans l'approche orientée services, un module est une boite noire et seules les préconditions et les postconditions sont disponibles pour l'architecte tiers. Toutefois, pour être certain que deux modules peuvent interagir, même s'ils sont conçus par deux architectes indépendants, il faut que le module qui consomme le service sache comment le consommer. Dans l'approche orientée services, le service fourni par un module est décrit au travers d'un contrat de services, *e.g.*, WSDL [6].

Interopérabilité. Dès lors que le contrat de service est disponible, un architecte tiers peut le récupérer afin de consommer le service. Pour cela il est nécessaire que le service soit

6. *Web Services Description Language*

interopérable, *i.e.*, qu'il puisse fonctionner avec d'autres modules indépendamment de leur implémentation. Pour cela, l'approche orientée services utilise des standards ouverts, *e.g.*, services web [CDK+02], *Universal Plug and Play* (UPnP) [UPn08].

Couplage faible. L'approche orientée services permet un couplage faible entre le fournisseur d'un service et le consommateur du service. En effet, l'usage de standards ouverts et de contrat de services permet à d'autres modules de consommer les services qui sont proposés de manière transparente, *i.e.*, sans que les architectes des deux modules ne se soient concertés lors de la phase de conception.

Sans états. Un service doit pouvoir être consommé par plusieurs clients en même temps. Pour cela, il est nécessaire que les clients n'impactent pas le fonctionnement du service. Ainsi, les services ne doivent pas être fourni au travers d'états antérieurs. C'est la propriété d'idempotence qui prime pour les modules fournissant des services, *i.e.*, que le service fournisse le même résultat qu'il soit appelé une ou plusieurs fois avec les mêmes paramètres.

Réutilisation et composition de services. La réutilisation des services permet à plusieurs clients de consommer le service issu d'un seul fournisseur. Grâce à un couplage faible et l'idempotence des services, plusieurs consommateurs peuvent requérir le service en parallèle sans s'impacter les uns les autres.

En facilitant la réutilisation de services conçus par des tiers, *i.e.*, grâce à l'interopérabilité et au couplage faible, un architecte peut concevoir un service ayant une plus-value par rapport aux services dont il dépend. Ainsi, l'approche orientée services permet de faire de la composition de services afin de fournir des services enrichis. Ce nouveau service peut à son tour être sujet à composition.

Prenons l'exemple d'un service de cartographie accessible sur Internet. Ce service fournit une carte basique avec les villes et les routes. Le service peut être consommé tel quel par un utilisateur. Cependant des entreprises non affiliées à celle fournissant le service de cartographie peuvent consommer ce service pour fournir un service enrichi de nouvelles fonctionnalités, *e.g.*, service de routage, affichage de centres d'intérêt. Ce service enrichi est également disponible à l'utilisateur.

Découverte du service. Si les architectes proposant des services ne se connaissent pas, il se pose alors le problème de savoir où le service se trouve, *i.e.*, la découverte de ce service par un consommateur du service. L'approche standard des Architectures Orientées Services (SOA [7]) [OAS06] consiste à utiliser un annuaire dans lequel les modules fournissant des services se déclarent lorsqu'ils apparaissent sur le réseau. Le consommateur du service n'a plus qu'à interroger cet annuaire afin de récupérer le contrat de service et l'adresse du service (cf. Figure 2.1).

Initialement, SOA a été conçu pour utiliser des équipements différents interagissant au

7. *Service Oriented Architecture*

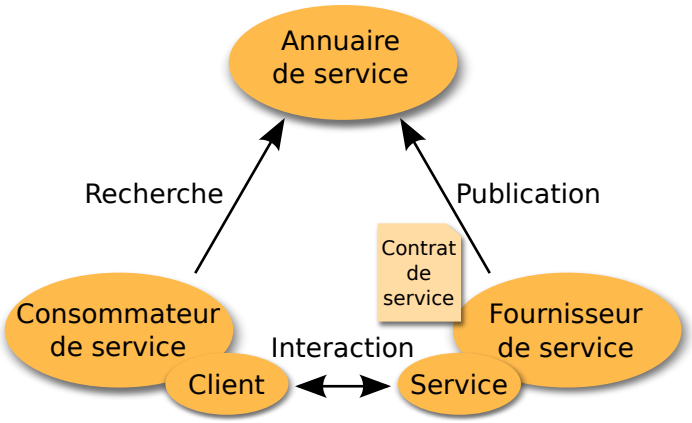

FIGURE 2.1 – **Architecture Orientée Service.** L'architecture orientée service se compose de trois entités : le fournisseur de service, l'annuaire de services et le consommateur de service. La première entité va publier son contrat de service dans l'annuaire. Puis le consommateur de service va interroger l'annuaire pour récupérer les informations relatives au fournisseur de service. Enfin, la liaison est établie entre le fournisseur et le consommateur

travers de services web, *i.e.*, WSDL, SOAP[8], UDDI[9]. Mais l'architecture fournisseur-consommateur-annuaire se retrouve aussi pour l'exécution d'applications au sein d'un même équipement. Le canevas logiciel OSGi[10] est basé sur ce principe [OSG12]. Il permet de déployer des modules qui publient leurs services dans un annuaire local. Un consommateur peut alors interroger cet annuaire et être mis en relation avec le fournisseur du service.

Dans les environnements informatique restreints, *i.e.*, se limitant à quelques équipements, il est envisageable de se passer d'un annuaire de services. Dans le standard UPnP, un équipement diffuse les services qu'il recherche. Si un équipement héberge un de ces services, il lui répond et lui envoie le contrat de service. À l'évidence, cette approche nécessite la diffusion de messages dans tout le réseau dès lors qu'un équipement requiert un service et le passage à l'échelle est donc impossible, notamment si cette architecture doit être mise en place sur Internet.

2.2.3 Composants orientés services

Nous avons décrit l'approche à composants (cf. Section 2.2.1) et l'approche à service (cf. Section 2.2.2)[11]. Chaque approche est porteuse de propriétés différentes. Cependant, l'ap-

8. *Simple Object Access Protocol*
9. *Universal Description Discovery and Integration*
10. *Open Services Gateway initiative*
11. Pour une comparaison plus poussée de ces deux concepts, voir [BL07].

proche à composants à quelques points communs avec l'approche à services, *i.e.*, réutilisation, contrat de service, composition.

L'approche des composants orientés services cherche à marier ces deux approches afin d'hériter de l'ensemble des propriétés évoquées [CH04]. Cette approche est décrite au travers de la spécification des Architectures de Composants à Services (SCA) [12] [OAS07] définie en 2007. Il existe plusieurs implémentations de ce standard : FraSCAti [SMF+09, SMR+12], Tuscany [LCF+11].

Dans notre cas, nous nous intéressons à la modification de l'architecture d'un service à l'exécution, permettant ainsi de déplacer les composants sur des équipements hétérogènes dans un environnement volatile. L'objectif non fonctionnel qui consiste à réduire la consommation d'énergie sans modifier les objectifs fonctionnels des services est ainsi atteint.

L'approche des composants orientés services propose la modification de l'architecture d'un service à l'exécution. Mais elle ne permet pas à elle seule de décider comment adapter automatiquement ces services à un environnement changeant. C'est pour cela que la section suivante discute de l'adaptation autonome des systèmes informatiques à un environnement amené à changer.

2.3 Adaptation autonome des systèmes informatiques

L'adaptation des systèmes à un environnement changeant est une approche qui est déjà largement utilisée dans d'autres domaines, *e.g.*, intelligence artificielle [Sar83], médical [SSS89]. Elle se base généralement sur une boucle de rétroaction. Un système rétroactif est composé d'un ensemble de composants qui agissent ensemble afin de maintenir les valeurs des attributs réels d'un système proche des valeurs cibles. L'idée principale est d'observer les valeurs de sortie pour ajuster les valeurs d'entrée afin de satisfaire les objectifs [MKS09].

Par exemple, une pompe à insuline doit injecter de l'insuline dans le corps d'un patient régulièrement. La dose admissible du patient à chaque injection dépend du taux de sucre présent dans son corps au moment de l'injection. La pompe doit s'adapter à ce taux lorsqu'elle injecte l'insuline. Il est donc nécessaire d'avoir une boucle de rétroaction afin que la dose injectée ne soit pas fatale pour le patient.

En ingénierie logicielle, l'adaptation autonome des systèmes informatiques cherche à construire des systèmes capable de s'auto-gérer. Cette approche vise à décharger l'utilisateur de la nécessité de s'occuper du système informatique que ce soit pour le déploiement, ou lorsque des pannes surviennent. L'auto-gestion d'un système informatique, appelée également auto-$\star$, est divisée en quatre parties [KC03] :

Auto-configuration. L'auto-configuration permet à un système informatique de s'installer, de configurer et d'incorporer de nouveaux composants de manière transparente.

12. *Service Component Architecture*

Auto-optimisation. L'auto-optimisation cherche à constamment améliorer les performances d'un système informatique.

Auto-protection. L'auto-protection s'occupe de protéger le système contre des attaques externes ou des pannes en séries.

Auto-guérison. L'auto-guérison permet de détecter des problèmes internes au système, de les analyser et enfin de les résoudre.

Ces systèmes peuvent ensuite être classés suivant leur degré d'autonomie, du plus simple au plus sophistiqué. Pour évaluer l'autonomie d'un système informatique, [HM08] propose une classification de ces systèmes au travers de quatre degrés.

Support. L'architecture possède un aspect ou un composant qui améliore ses performances. Par exemple, il s'agit d'un composant de surveillance autonome intégré au sein d'une application plus large.

Cœur. L'architecture dispose d'une solution d'auto-gestion de bout en bout pour une seule application. Par exemple, il s'agit d'une application qui s'auto-gère de bout en bout afin de toujours atteindre son objectif fonctionnel.

Contrôle. L'architecture dispose d'une solution d'auto-gestion de bout en bout pour plus d'une application. Par exemple, il s'agit d'une architecture qui gère plusieurs applications de bout en bout afin d'atteindre les objectifs fonctionnels de chaque application.

Autonome. L'architecture est générique pour considérer plusieurs applications et est capable de se surveiller et de s'adapter en conséquence. Par exemple, il s'agit d'une architecture qui est autonome et se considère elle-même dans la gestion.

De manière générale, l'adaptation des systèmes informatiques permet de modifier le fonctionnement d'un système informatique pour atteindre un objectif non fonctionnel, *e.g.*, sécurité [HLL10], fiabilité [DKM04], efficience énergétique [KHY08, GDPR12]. Le système informatique s'adapte ainsi à un environnement changeant sans remettre en cause son objectif fonctionnel.

L'adaptation autonome des systèmes informatiques se fait au travers d'une boucle de contrôle. Cette boucle permet d'observer l'environnement dans lequel le système informatique évolue, d'analyser les changements et d'effectuer des actions afin que le système change en fonction des observations.

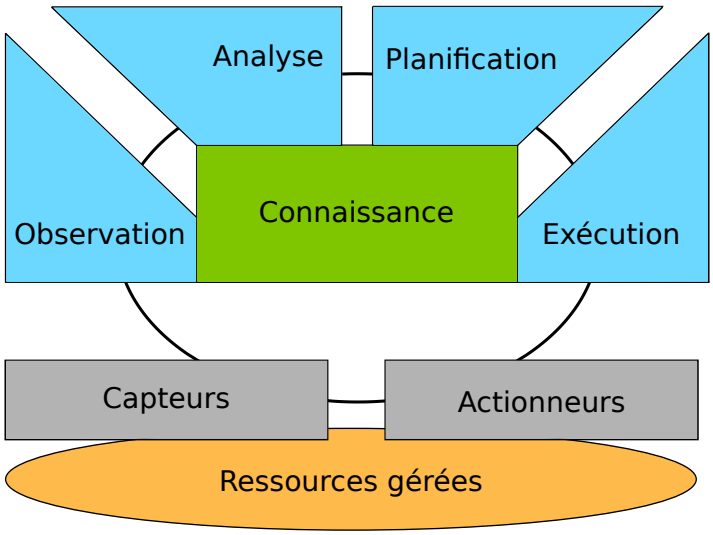

FIGURE 2.2 – **Boucle autonome MAPE-K.** La boucle autonome MAPE-K est un patron de conception permettant à un système informatique de s'adapter à un environnement changeant. Elle se divise en quatre entités (Observation, Analyse, Planification, Exécution) qui partagent une base de Connaissance commune. Cette boucle s'appuie sur des Capteurs et des Actionneurs présents dans l'environnement qu'elle gère.

2.3.1 Boucle autonome MAPE-K

En 2003, la compagnie IBM [13] a proposé un patron de conception d'architecture décrivant une boucle autonome : la boucle MAPE-K [14] [Com03]. Ce patron est depuis largement utilisé pour définir des systèmes informatiques autonomes.

La boucle autonome MAPE-K permet à un système de s'adapter à des événements extérieurs. En analysant ces événements, le système informatique est alors capable de planifier des actions et de les exécuter afin de s'adapter à l'environnement dans lequel il évolue.

La boucle MAPE-K se base sur un processus divisé en quatre sous-entités afin de déterminer les changements à effectuer pour s'adapter : Observation, Analyse, Planification, Exécution (cf. Figure 2.2). La boucle MAPE-K compte également une sous-entité de Connaissance qui est partagée entre l'ensemble des entités du processus de décision.

Enfin, cette boucle autonome nécessite des Capteurs et des Actionneurs afin de respectivement récupérer les événements impliquant une adaptation du système informatique et effectuer des actions afin de mettre en application la décision prise par la boucle de contrôle.

Observation. La sous-entité Observation récupère les événements issus des capteurs placés dans l'environnement. Les événements collectés au travers de capteurs sont des

13. *International Business Machines*
14. MAPE-K : *Monitor*, *Analyze*, *Plan*, *Execute* - *Knowledge*

événements significatifs pour le processus d'adaptation. Les événements significatifs sont donc à définir par l'architecte lors de la conception de la boucle autonome. Par exemple, l'événement significatif reçu fait état d'une attaque informatique.

Analyse. La sous-entité Analyse modélise l'environnement au travers des informations récupérées précédemment. En corrélant ces informations, le processus de décision peut en retirer des informations de plus haut niveau. Par exemple, cette sous-entité analyse le type d'attaque informatique.

Planification. La sous-entité Planification cherche les actions à effectuer afin d'atteindre l'objectif fixé. Cette sous-entité se base sur l'analyse précédente pour déterminer les actions à mener. Par exemple, l'analyse a déterminé le type d'attaque, la sous-entité Planification décide des actions à mener pour que cette attaque ne se reproduise pas.

Exécution. La sous-entité Exécution met en application les actions planifiées en utilisant les actionneurs placés dans l'environnement. Cette sous-entité contacte les actionneurs et leur transmet les actions à exécuter. Par exemple, une des actions à prendre est d'instancier un composant chiffrant les communications.

Connaissance. La sous-entité Connaissance est commune à l'ensemble des entités de la boucle autonome. Cette mémoire est persistante afin de conserver les informations issues de l'environnement dès lors que des événements significatifs surviennent.

La boucle autonome MAPE-K est générique afin de laisser le choix à l'architecte de l'objectif non fonctionnel à atteindre. Ainsi, au delà de la conception d'un système apte à s'adapter à un environnement changeant, il faut également introduire une politique décisionnelle décidant des actions à mener afin d'atteindre l'objectif non fonctionnel.

2.3.2 Politiques de décision

La boucle autonome MAPE-K est un patron de conception permettant de concevoir des systèmes informatiques capables de s'adapter à un environnement changeant. La boucle en elle même propose seulement la partie « auto » de l'auto-gestion. La gestion est donc à réaliser au travers d'une politique de décision.

Pour mettre en place des politiques de décision, il faut prendre l'hypothèse qu'un système informatique est décrit par un état à un instant donné. Un état est un ensemble de variables dont la valeur est connue à un instant donné. Par exemple, dans notre cas, il peut s'agir de l'état énergétique d'un équipement, *i.e.*, allumé.

L'état courant ne remplit pas nécessairement l'ensemble des besoins définis dans la spécification. Par exemple, la spécification décrit qu'un système informatique autonome doit se trouver dans un état dans lequel il minimise sa consommation d'énergie. Toutefois, ce système se trouve dans l'état dans lequel la consommation d'énergie est maximisée. Cet état dans lequel il se trouve doit donc être changé.

Pour passer dans un état qui satisfait les besoins de la spécification, il faut décider et exécuter des actions. Une action est un événement décidé par la politique de décision modifiant la valeur d'une variable. Par exemple, pour atteindre l'état qui minimise la consommation d'énergie, une action est de placer dans un état de basse consommation un équipement.

Les politiques de décision sont classées en trois niveaux de spécification comportementale : politique d'action, politique d'objectif ou politique de fonction d'utilité [KW04]. Chaque niveau défini un comportement plus sophistiqué que le précédent.

Politique d'action. La politique d'action est équivalente à une politique d'Événement-Condition-Action (ECA) [MD89]. L'ECA permet d'atteindre les objectifs fixés par l'architecte au travers de règles. Une règle décrit un événement, *i.e.,* un signal déclenchant l'invocation de la règle, une condition, *i.e.,* une évaluation à satisfaire, et enfin une action à mener qui modifie l'état courant du système informatique.

Dans ce genre de politique, une action est définie pour chaque événement que le système considère. Cette approche nécessite de définir les règles à la conception. Ainsi, l'architecte doit connaître l'ensemble des événements et des actions à mener lorsque les conditions sont satisfaites.

Politique d'objectif. La politique d'objectif définit les états possibles d'un système, mais ne permet pas de distinguer quel état est le meilleur. Ainsi la politique d'objectif cherche à trouver les actions permettant de passer de l'état courant à un des états prédéfinis.

Comme pour la politique d'action, cela nécessite de connaître l'ensemble des états que peut prendre un système informatique dès la conception. Toutefois, par rapport à la politique précédente où l'ensemble des actions et des états est prédéfini, la politique d'objectif a plus de liberté pour choisir les actions à mener pour changer d'état.

Politique de fonction d'utilité. Cette politique définit une fonction d'utilité qui décrit l'ensemble des états que peut prendre un système informatique. Cette politique est utilisée lorsque l'architecte ne connaît pas à l'avance quel état est le meilleur dans une situation donnée. L'état est choisi à l'exécution en fonction du contexte.

La mise en place d'une politique d'action ou d'objectif se heurte à la capacité de l'architecte de spécifier l'ensemble des états possibles d'un système. L'utilisation d'une fonction d'utilité permet de palier ce problème puisqu'au travers de cette fonction, l'ensemble des états sont définis. De fait, cette politique est adaptée aux environnements où l'architecte doit considérer l'ensemble des états sans les connaître à la conception.

2.4 Conclusion

L'ingénierie logicielle propose des approches rendant un système informatique capable de s'adapter à un environnement changeant. L'usage d'une architecture faite de composants

orientés services permet de bénéficier des avantages de l'approche à composants, *e.g.,* modularité, cycle de vie, réflectivité, et de l'approche orientée services, *e.g.,* contrat de services, couplage faible. L'architecture d'un service de bout en bout est alors plus facilement adaptable à l'exécution.

L'approche des composants orientés services permet donc de développer des architectures de service plus flexibles sans perturber l'objectif fonctionnel. Par la suite, une boucle de contrôle utilise cette flexibilité architecturale afin d'atteindre un ou plusieurs objectifs non fonctionnels. Enfin, pour définir l'objectif non fonctionnel à atteindre, il est nécessaire de spécifier une politique de décision.

Dans le chapitre suivant, nous présentons les différentes approches existantes afin de minimiser la consommation d'énergie d'un équipement ou d'un ensemble d'équipements. Ces approches considèrent différents niveaux de fonctionnement, *i.e.,* du matériel aux applications.

Chapitre 3

Gestion énergétique des systèmes informatiques

La planète est pleine d'énergie Mako. Ici, les gens l'utilisent tous les jours. C'est l'âme de la planète. Mais la Shinra continue de l'aspirer avec ses machines.
– Barret, *Final Fantasy VII*

Sommaire

3.1 Introduction 30
3.2 La gestion énergétique d'un équipement informatique 30
3.2.1 Gestion des états énergétiques 31
3.2.2 Adaptation énergétique des équipements 34
3.2.3 Optimisation énergétique des applications 37
3.3 La gestion énergétique des systèmes répartis 39
3.3.1 Contrôle des états énergétiques 40
3.3.2 Gestion énergétique du réseau d'équipements 42
3.3.3 Répartition des applications 44
3.3.4 Mandatement 45
3.4 Modification des usages grâce aux systèmes informatiques 47
3.5 Conclusion 47

3.1 Introduction

L'énergie prend de plus en plus une place centrale dans la conception des systèmes informatiques. La recherche et l'industrie ont fait d'énormes progrès pour minimiser la consommation de ces systèmes. Ces progrès touchent l'ensemble des aspects des systèmes informatiques : des couches matérielles aux couches applicatives, des équipements isolés aux systèmes répartis.

La mise en œuvre de ces solutions dans l'électronique grand public passe par la standardisation, permettant ainsi une meilleure intégration dans les environnements informatiques. Toutefois, le taux de pénétration de ces solutions est assez faible. Bien que les équipements isolés en bénéficient depuis quelques années, *e.g.*, ACPI [Hew10], les succès dans les systèmes répartis sont plus mitigés, *e.g.*, UPnP Low Power [UPn07].

Dans ce chapitre, nous dressons un panorama des solutions visant à faire des économies d'énergie. Nous commençons par décrire les solutions relatives aux équipements isolés avant de discuter des solutions relatives aux systèmes répartis. Enfin, parce qu'un système informatique est avant tout destiné à un utilisateur humain, nous présentons comment l'utilisation des systèmes informatiques est un vecteur de modification des usages informatiques des utilisateurs.

Structure du chapitre

Ce chapitre s'articule comme suit : la Section 3.2 décrit les approches visant à réduire la consommation d'un équipement informatique isolé. La Section 3.3 expose les approches existantes afin de réduire la consommation d'énergie d'un système réparti. La Section 3.4 présente succinctement les stratégies permettant des réductions d'énergie en modifiant l'usage qui est fait des équipements informatiques. La Section 3.5 conclut ce chapitre.

3.2 La gestion énergétique d'un équipement informatique

Pour mieux comprendre l'intérêt de rendre les équipements et les applications efficients énergétiquement, nous posons une analogie. Un cycliste souhaite se rendre de Grenoble à Lille. Pour cela, il a à sa disposition un vélo, *i.e.*, l'équipement, ainsi que son savoir-faire, *i.e.*, l'application. Pour rallier Lille, il doit à la fois utiliser le vélo grâce à ses jambes, *i.e.*, énergie mécanique, et utiliser son savoir-faire pour pédaler.

Pour minimiser l'énergie utilisée pour parcourir la distance Grenoble-Lille, des améliorations sont possibles sur le vélo pour que le cycliste dépense le moins d'énergie possible. Par exemple, le vélo peut être allégé ou son aérodynamisme amélioré. Mais il est également possible d'améliorer le savoir-faire du cycliste. Par exemple, le cycliste peut limiter ses mouvements lorsqu'il est en train de descendre ou bien avoir une plus faible prise au vent en améliorant son aérodynamisme, *i.e.*, position sur le vélo.

En informatique, la réduction de la consommation d'énergie passe par une amélioration de l'équipement et de l'application. Les travaux présentés dans cette section s'intéressent à la réduction de la consommation énergétique d'un équipement. Par exemple, une approche participant à contrôler l'énergie d'un équipement est le bouton d'allumage. Au même titre que le cycliste n'a pas besoin de fournir un effort lorsqu'il ne se sert pas du vélo, ce bouton permet de couper ou de rétablir l'alimentation électrique de l'équipement à la demande de l'utilisateur.

Par la suite, les approches cherchent à réduire la consommation d'énergie lorsque l'équipement est utilisé. Cette gestion de l'énergie est étudiée lors de la conception de ressources matérielles, *e.g.,* le processeur [TSR+98] ou la mémoire vive [LFZE00]. L'équipement électronique est amélioré en cherchant à ce que les actions propres à ses ressources matérielles, *e.g.,* traitement d'une instruction, écriture/lecture dans la mémoire, consomment le moins d'énergie possible.

Cette section s'intéresse également aux approches existantes au niveau des applications pour minimiser l'usage de l'équipement. L'application utilise les ressources matérielles de l'équipement pour fournir son service. Elle peut utiliser plus de ressources qu'elle a besoin pour fournir ce service. Cette surconsommation de ressource induit une consommation d'énergie accrue.

Dans le cas de notre cycliste, c'est équivalent à un savoir-faire de mauvaise qualité. Si le savoir-faire est mauvais alors le cycliste mobilise davantage d'énergie pour parvenir au même résultat, *i.e.,* rallier le point B. Pour lui éviter d'arriver exténué à sa destination, il a besoin d'un savoir-faire de qualité, lui permettant de mieux contrôler sa dépense énergétique.

En informatique, la problématique est similaire. Contrôler l'usage qui est fait des ressources matérielles par une application est nécessaire. Autrement, l'énergie dépensée est plus importante que l'énergie minimale pour rendre un service, amenant à une surconsommation. Par exemple, un moyen de minimiser l'énergie est de couper l'alimentation des ressources matérielles qui ne sont pas nécessaires à la fourniture du service.

3.2.1 Gestion des états énergétiques

Un équipement est un ensemble de ressources matérielles interconnectées, *e.g.,* carte mère, processeur, disque dur, écran. Chaque ressource a un rôle particulier. Par exemple, le disque dur stocke des informations, l'écran permet une interaction avec l'utilisateur. Toutefois, les applications qui consomment ces ressources ne les consomment pas toujours en totalité. Aussi, une première approche vise à concevoir des ressources matérielles qui passent dans des états de basse consommation lorsqu'elles ne sont pas utilisées.

Un équipement informatique est un ensemble de ressources matérielles de différents constructeurs. Aussi, un même type de ressource n'est pas contrôler de la même manière. Un standard est nécessaire afin que chaque constructeur intègre les mêmes états et les mêmes méthodes de contrôle des états.

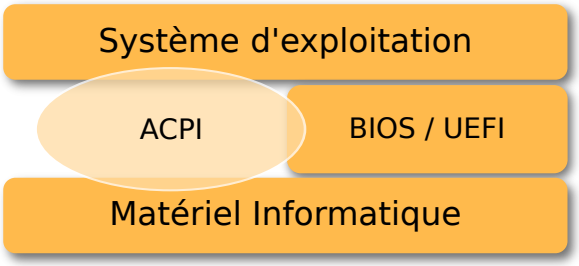

FIGURE 3.1 – **L'ACPI dans un équipement informatique.** L'ACPI est une interface de commande des ressources matérielles par les couches supérieures, *i.e.*, BIOS/UEFI et système d'exploitation.

Le contrôle des ressources matérielles permet de modifier la manière dont un équipement consomme. Pour cela, il existe un standard présent sur la plupart des équipements actuels et qui tend à intégrer les équipements futurs : l'ACPI [15].

L'ACPI est un standard publié dans sa première version en 1996 par un consortium d'industriels. La dernière version est la version 4.0 sortie en 2010 [Hew10]. Cette version de l'ACPI est disponible uniquement pour les processeurs sous architecture x86. La version 5.0 de l'ACPI étend le standard sur les architectures ARM qui sont devenues courantes dans les systèmes embarqués tels que les téléphones intelligents ou les tablettes.

L'ACPI permet au système d'exploitation ainsi qu'au BIOS [16] ou à l'UEFI [17], successeur du BIOS, de contrôler les ressources matérielles d'un équipement informatique. Cette interface est utilisée par différentes couches d'un équipement informatique (cf. Figure 3.1). Cependant, afin de les commander, les ressources matérielles doivent supporter ce standard. Aussi, les fabricants de ressources matérielles doivent participer à cet effort.

États énergétiques

L'ACPI définit un ensemble fini d'états énergétiques. Ces états sont définis en fonction des capacités de contrôle des ressources matérielles. Dans l'ACPI, il y a $4 + 1$ états globaux, 4 états de veille, n états de fonctionnement pour le processeur ou encore 4 états pour les ressources matérielles annexes, *e.g.*, disque dur, modem (cf. Figure 3.2).

Les $4 + 1$ états globaux détaillés dans l'ACPI décrivent les états possibles d'un équipement (nous ne détaillons pas les états propres à chaque ressource matérielle) :

G0 : actif. Le système est dans un état dans lequel il peut exécuter des applications et fournir des services.

15. *Advanced Configuration and Power Interface*
16. *Basic Input Output System*
17. *Unified Extensible Firmware Interface*

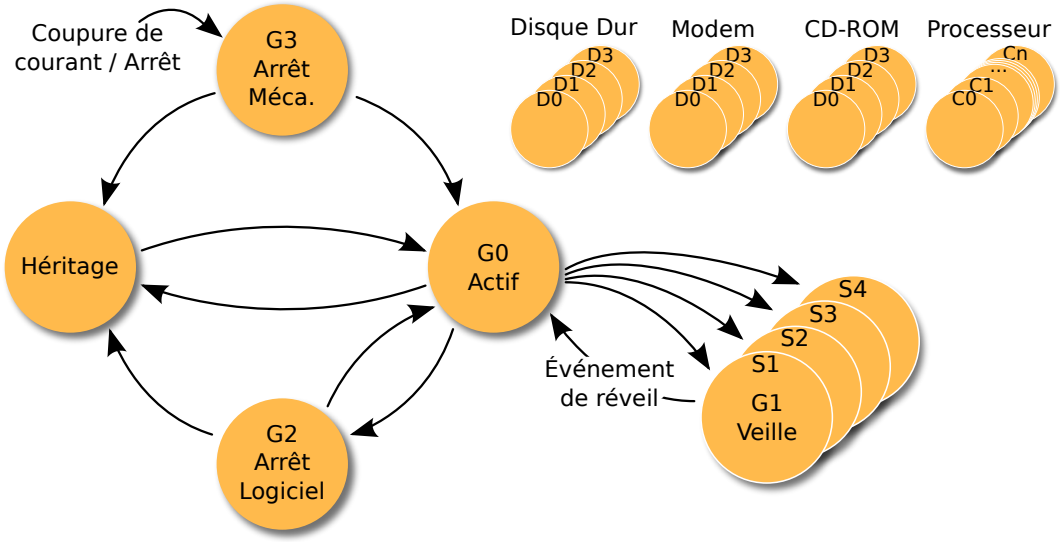

FIGURE 3.2 – **Relations entre les états énergétiques de l'ACPI.** Il y a $4+1$ états plus ou moins profond permettant des économies d'énergie plus ou moins importantes. Chaque ressource matérielle possède également divers états de consommation d'énergie permettant des combinaisons importantes. Cela participe à adapter la consommation d'énergie au besoin de l'application.

G1 : veille. Le système consomme un peu d'énergie. Une partie du contexte système est conservée. L'état G1 est composé à son tour de 4 états de basse consommation.

- **S1.** Cet état de veille permet un retour rapide à l'état G0. Aucun contexte matériel n'est perdu. L'alimentation des ressources matérielles reste en service.
- **S2.** Cet état est similaire à l'état S1 à la différence près que le contexte contenu dans le processeur et le cache est perdu. C'est au système d'exploitation de restaurer les différents contextes matériels.
- **S3.** Dans cet état, seule la mémoire reste alimentée, le reste des ressources matérielles ne l'est plus.
- **S4.** Cet état est le plus économe en énergie de l'état G1 mais la durée pour revenir dans l'état actif est aussi le plus long. Le contenu de la mémoire est sauvegardé sur le disque dur. L'ensemble des ressources matérielles ne sont plus alimentées.

G2 : arrêt logiciel. Le système consomme un minimum d'énergie. Les ressources matérielles sont arrêtées. Le contexte système n'est pas préservé.

G3 : arrêt mécanique. Le système est à l'arrêt total. Seule une action mécanique peut le redémarrer. Aucune énergie n'est consommée.

Héritage. Cet état permet de gérer les états énergétiques n'étant pas décrits dans le standard.

L'ACPI ne gère pas cet état spécifique à l'équipement. Un logiciel de plus haut niveau doit le prendre en charge.

Suivant la profondeur de l'état énergétique, l'équipement consomme moins d'énergie mais met plus de temps à redevenir actif, *i.e.*, état G0. Par exemple, un équipement dans l'état G1/S1 consomme plus d'énergie que dans l'état G1/S2. Cependant, le temps de transition de l'état actif à un état de basse consommation, ou inversement, consomme de l'énergie. Plus cette période de transition est importante, *i.e.*, plus la transition s'opère vers un état énergétique profond, plus la consommation est importante.

Ainsi, une réduction de ces phases de transition permet de réduire la consommation d'énergie de l'équipement. L'usage des états de basse consommation permettant la préservation du contenu de la mémoire volatile, *i.e.*, G1/S1 à G1/S3, assure un temps de transition faible entre l'état actif et un état de basse consommation [MGW09].

Enfin, à noter que dans les états autres que l'état G0, l'équipement ne peut pas communiquer avec un autre équipement. Seules les messages réveillant les équipements sont considérés par les cartes réseau.

Réveil d'un équipement localement

Le temporisateur est le seul moyen pour un équipement se réveiller de lui-même, *e.g.*, *Real Time Clock Alarm*. Un temporisateur peut être activé, dans le BIOS ou au niveau du système d'exploitation, afin que l'équipement se réveille automatiquement au bout d'un temps donné.

Un autre moyen est d'utiliser le *Wake-on-AC*. Ce mécanisme réveille l'équipement dès lors qu'il passe sous tension. Toutefois, il ne permet pas à un équipement de se réveiller de lui même. Il est nécessaire d'avoir une action extérieure, *i.e.*, un prise commandable ou un utilisateur, pour que l'équipement passe dans l'état actif.

Enfin, sur certains équipements, il existe un bouton de démarrage permettant à l'équipement de passer dans l'état actif. Cela nécessite également une action extérieure venant généralement de l'utilisateur.

3.2.2 Adaptation énergétique des équipements

L'ACPI décrit les états globaux que peut prendre un équipement. Elle décrit également des états pour les ressources matérielles d'un équipement lorsqu'il est dans l'état actif. Une ressource matérielle pouvant fonctionner dans différents états adapte sa consommation d'énergie pour fournir un service. Cette optimisation, nommée *Dynamic Power Management* [BBDM00], nécessite que la ressource soit conçue afin de fonctionner dans différents états d'activité et qu'elle puisse changer d'état à tout moment.

Toutefois, il existe une approche antagoniste qui n'est pas spécifiée dans l'ACPI. Cette approche vise à saturer la ressource matérielle pendant une courte période de temps avant

de la placer dans un état de basse consommation. Pour l'utilisateur final, aucune différence notable n'est constatée. Il s'agit seulement d'une autre approche de réduction de la consommation d'énergie des ressources matérielles.

Ces deux approches se retrouvent dans certaines des ressources matérielles des équipements informatiques modernes, *e.g.,* le processeur, la carte réseau. Pour d'autres ressources, l'approche se limite principalement à les placer dans un état de basse consommation dès lors qu'elles ne sont plus nécessaires. Enfin, pour le reste des ressources, il n'existe pas de mode de basse consommation, il n'est donc pas possible de les arrêter à moins de couper leur alimentation électrique. Nous présentons quelques exemples de ces technologies pour certaines ressources matérielles.

Le processeur

Le processeur d'un équipement est l'une des ressources matérielles qui consomme le plus d'énergie. Aussi, une majorité de travaux portent sur son optimisation énergétique. La distinction est généralement faite entre deux approches : le *Dynamic Voltage Frequency Scaling* et le *race-to-sleep*.

Dynamic Voltage Frequency Scaling (DVFS). Le DVFS est une technique qui réduit la consommation énergétique d'un processeur [ZBSF04]. Lorsque le besoin en ressource de calcul change, la fréquence et le voltage du processeur s'adaptent en conséquence. Cela minimise la consommation d'énergie en ne consommant que l'énergie nécessaire à la fourniture du service. Le DVFS est présent dans la plupart des architectures des processeurs actuels, *e.g.,* ARM, x86.

Race-to-sleep. Le *race-to-sleep* (ou *race-to-idle*) est à l'opposé du DVFS [PS07]. Son objectif est d'effectuer la tâche le plus rapidement possible avant de passer dans un état de basse consommation. Cela a pour effet d'augmenter l'efficience énergétique d'un équipement [BH07]. Plus la charge du processeur est importante, plus l'équipement est efficient énergétiquement (cf. Figure 3.3).

Cette technique est intéressante uniquement si la charge de travail est espacée dans le temps. En effet, lorsqu'un processeur est réveillé, un pic de consommation d'énergie non négligeable survient. Réveiller trop fréquemment un processeur conduit donc à une surconsommation par rapport au DVFS.

La carte réseau

Suivant le type d'équipement, la carte réseau est la ressource matérielle qui consomme le plus d'énergie. Par exemple, la consommation de cette ressource est importante dans les systèmes embarqués où la gestion de l'énergie est critique. Au contraire, sur les ordinateurs personnels, la carte réseau n'est pas très consommatrice d'énergie au regard des autres ressources, *e.g.,* processeur. Les deux approches décrites plus haut sont retrouvées, mais appelées ici *Adaptive Link Rate* et bourrasque de paquets.

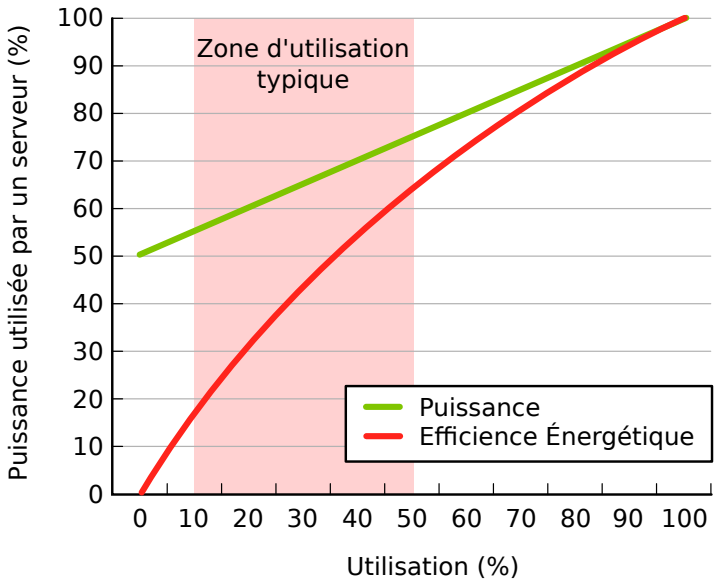

FIGURE 3.3 – **Efficience énergétique d'un processeur.** Cette courbe, tirée de [BH07], présente l'efficience énergétique d'un processeur de serveur en fonction de sa charge. Plus la charge du processeur est élevée, plus le processeur est efficient énergétiquement.

Adaptive Link Rate (ALR). Le standard 802.3az, ou *Adaptive Link Rate*, a été adopté par l'IEEE en 2010 [IEE10]. Il permet d'adapter la vitesse de transmission des données sur un réseau Ethernet en fonction des besoins et des données transférées. Cette approche rejoint l'adaptation appliquée au processeur au travers du DVFS, en l'imposant cette fois à la carte réseau Ethernet.

Bourrasque de paquets. Une approche de saturation existe également pour les cartes réseaux : la bourrasque de paquets [MCD+03]. Cette approche consiste à accumuler suffisamment de données avant de réveiller la carte réseau afin de tout envoyer en même temps. Par la suite, la carte réseau est replacée en état de basse consommation. Dans cette approche, la qualité de la ligne est un facteur important pour son succès. La bourrasque de paquets est inutilisable si un paquet met trop de temps a être émis. Au delà, les paquets sont jetés, ce qui abaisse la qualité de service et nécessite de les envoyer à nouveau.

L'écran

L'écran est un grand consommateur d'énergie dans un équipement informatique. Il représente environ 20 % de la consommation d'énergie pour les ordinateurs portables [MV05] et jusqu'à 68 % pour les téléphones intelligents, suivant les usages [CH10]. Il est impossible de l'arrêter dès lors que l'utilisateur est devant. Il n'est donc pas possible d'appliquer une approche du type *race-to-sleep* pour réduire sa consommation d'énergie.

Toutefois, il est possible d'adapter l'intensité de l'écran pour réduire la consommation d'énergie. Par exemple pour un écran LCD, en fonction des images projetées sur l'écran, il est possible de réduire l'intensité du rétroéclairage. Cela nécessite toutefois de compenser les données des pixels en ajustant leur luminance à la nouvelle luminance issue de la perte d'intensité du rétroéclairage [CK09].

3.2.3 Optimisation énergétique des applications

L'optimisation des ressources matérielles réduit la consommation énergétique de l'équipement. Cela équivaut à améliorer le vélo de notre cycliste sans toucher à son savoir-faire. Aussi, n'importe quel cycliste utilisant ce vélo, indépendamment de son savoir-faire, consomme moins d'énergie que s'il utilise un vélo non amélioré.

Toutefois, à vélo identique, deux cyclistes ne consomment pas la même quantité d'énergie. La musculature mise à part, un cycliste professionnel sait mieux gérer son effort qu'un débutant. En informatique, les choses peuvent être vues de la même manière. Des gains sont possibles dans la manière dont les ressources matérielles sont utilisées par l'application. L'application est donc un autre vecteur de réduction de la consommation d'énergie.

Pour minimiser la consommation des ressources matérielles par une application, il est possible de la minimiser à sa conception et de la minimiser lors de son exécution.

Conception des applications

Suivant le service considéré et l'environnement dans lequel il est fourni, une application est plus ou moins liée à un équipement. Par exemple, dans un système embarqué, où l'énergie tient un rôle important pour l'autonomie, les applications sont le plus souvent dédiées. Dans d'autres environnements, où l'énergie est moins critique, l'application n'est pas développée pour un équipement précis. C'est ce que nous appelons les applications génériques.

Les applications dédiées. L'approche consiste à chercher l'ensemble des instructions mettant en œuvre un service tout en consommant le moins de ressources matérielles sur un équipement ou un type d'équipement précis. Consommer moins de ressources matérielles permet de limiter l'énergie dépensée pour fournir ce service. Ce choix est généralement effectué lors de la conception de l'application où l'environnement de développement intégré propose les meilleures instructions qui sont fonction de l'équipement cible [KZ08].

Par exemple, pour faire communiquer deux équipements, l'architecte a le choix entre envoyer les messages compressés ou non (*e.g.*, méthodes `FileStream` et `GZipStream`). Dans le cas où les messages sont compressés, la consommation d'énergie de la carte réseau est moindre que lorsque les messages ne sont pas compressés puisque moins d'octets sont envoyés. Par contre, il est nécessaire d'utiliser d'avantages de ressources processeur pour compresser et décompresser le message. À l'inverse, s'il n'y a pas de compression, la carte

réseau consomme d'avantage puisqu'elle envoie plus d'octets et le processeur un peu moins qu'avec les solutions précédentes [BA06, KZ08].

Aussi, pour savoir quelle approche utiliser dans ce cas, il est nécessaire de connaître l'environnement qui va fournir le service. Si le même service doit être fourni par un autre type équipement, le processus de conception pour porter ce service doit être recommencé. Ce portage requiert davantage de ressources financières et humaines.

Les applications génériques. Les applications génériques sont conçues pour s'exécuter sur un ensemble hétérogène d'équipements. Par exemple, une application générique d'encodage vidéo s'exécute aussi bien sur des téléphones intelligents que sur des ordinateurs personnels. Pour minimiser l'utilisation des ressources matérielles de cette application, il faut utiliser à l'exécution les instructions qui consomment le moins en fonction de l'équipement. Pour cela, rajouter des éléments non fonctionnels à la conception, *e.g.,* annotations [CZSL12] ou méta-données dans les fichiers [MCO+05], permet à l'application de modifier sa structure en fonction de l'équipement tout en fournissant le même service.

L'intérêt des applications génériques est qu'elles peuvent s'exécuter sur un ensemble hétérogène d'équipements, *e.g.,* ayant des systèmes d'exploitation différents. Aussi, le coût financier et humain sont limités pour porter le service sur d'autres équipements. L'optimisation de ces applications a lieu uniquement à l'exécution mais doit être pensée à la conception rendant incertain le gain énergétique obtenu. De plus, *a priori*, elles ne sont pas aussi optimisées énergétiquement que les applications dédiées.

Exécution des applications

Pour réduire la consommation d'énergie d'une application à l'exécution, il faut la faire consommer moins de ressources matérielles. Pour minimiser les ressources, il y a deux approches. La première consiste à dégrader le service fourni. La deuxième consiste à optimiser le fonctionnement de l'application afin de consommer moins de ressources tout en fournissant le même service.

Dans les deux cas, il est nécessaire de définir les critères caractérisant un service. Ces critères sont définis à la conception au travers de la spécification décrivant l'application.

La dégradation d'un service. La dégradation d'un service consiste à réduire la qualité de service d'une application afin d'effectuer des économies d'énergie [FS99, PDR08, XKYJ09]. Cela suppose que certains des critères qui caractérisent le service sont soit variables, soit optionnels. Par exemple, considérons que la qualité de service d'un film est seulement définie au travers de sa résolution. Ce film est disponible dans différentes résolutions, *i.e.,* la résolution est un critère variable. Plus la résolution est grande, plus le processeur est utilisé pour le décoder et l'afficher. Aussi, pour réduire l'usage du processeur, il est plus intéressant d'utiliser une résolution moins importante.

Dans ce cas, la qualité de service et la réduction de la consommation d'énergie sont des objectifs contradictoires. Si la qualité de service est mauvaise alors l'utilisateur ne voudra pas de ce service. Au contraire, si aucun gain énergétique n'est atteint alors il est inutile de mettre en œuvre cette approche. Il faut donc trouver un compromis entre la dégradation du service et le gain énergétique qui en résulte.

De manière générale, la dégradation de service vise à prendre des décisions en se basant sur les critères optionnels ou variables qui définissent la qualité de service d'un service. Par l'utilisation de cette approche, moins de ressources matérielles sont utilisées et donc moins d'énergie est consommée.

L'optimisation d'un service. L'optimisation d'un service consiste à consommer moins d'énergie tout en ayant une qualité de service équivalente [YNA+06]. Cela suppose que les critères qui décrivent un service ne sont pas variables ou optionnels, il faut tous les satisfaire lors de l'exécution. Par exemple, le film précédent est disponible dans une seule résolution. Aussi, pour économiser de l'énergie, il faut utiliser d'autres approches ne remettant pas en causes les critères définissant la qualité de service du service. Une solution est donc d'utiliser une application dédiée, spécialement conçue pour l'équipement sur lequel l'utilisateur visionne le film.

Dans cette approche, la qualité de service est toujours atteinte. La marge de manœuvre pour réduire la consommation des ressources est déportée sur la mise en œuvre du service. Cette mise en œuvre est définie lors de la conception.

Synthèse

Cette section a présenté les approches existantes permettant à un équipement isolé de réduire sa consommation d'énergie. Ces approches concernent chaque couche d'un équipement, *i.e.*, des ressources matérielles aux applications, et cherchent à satisfaire les besoins avec moins de ressources. Cela passe donc par l'adaptation des ressources en fonction des besoins.

Toutefois, dans les usages modernes, un équipement est rarement utilisé isolément. L'apparition des systèmes répartis considère plusieurs équipements pour fournir un service. Par exemple, pour regarder un film en vidéo à la demande, le service requiert au moins deux équipements, *i.e.*, le serveur hébergeant le film et l'équipement sur lequel le film est joué. Cet usage peut être adapté afin de consommer le moins d'énergie possible.

3.3 La gestion énergétique des systèmes répartis

La gestion de l'énergie d'un équipement isolé est un sujet étudié depuis longtemps. Toutefois, les usages actuels imposent l'utilisation de systèmes répartis. La gestion énergétique

des systèmes répartis est encore balbutiante dans l'informatique du quotidien, *i.e.,* bureaux, maisons, contrairement aux centre de traitement de données.

Pourtant, réduire la consommation d'énergie à l'échelle d'un ensemble d'équipement est prometteur. Il suffit de prendre l'analogie du cycliste pour constater qu'il est possible de réduire la consommation d'énergie à l'échelle d'un groupe d'individu. Par exemple, supposons que notre cycliste se mette à rouler en peloton. Rouler en peloton lui permet de moins subir la résistance de l'air et de profiter de l'aspiration du coureur devant lui. Aussi, rouler en peloton permet de réduire son effort pour parcourir la même distance, en un temps qui peut même être plus court.

Aussi, une coopération accrue entre différents équipements permet de proposer des approches pour réduire la consommation d'énergie à l'échelle du système réparti. Ces approches touchent l'ensemble des couches d'un système réparti, des cartes réseaux aux applications. Les approches impliquent soit qu'un équipement, ou sous ensemble d'équipements, fasse le plus gros de l'effort pour l'ensemble du groupe, soit qu'en fonction des spécificités de chaque équipement, les applications soient réparties au mieux.

3.3.1 Contrôle des états énergétiques

Un système réparti possède autant d'états énergétiques différents que le nombre de combinaisons d'état existants pour l'ensemble des équipements qui composent le système réparti. Par exemple, si tous les équipements possèdent seulement deux états énergétiques, alors pour n équipements, il existe 2^n états énergétiques différents. Le passage d'un état énergétique du système réparti à un autre état est fait au travers de la gestion de l'ensemble des équipements pris individuellement.

Pour passer dans un état de basse consommation, l'équipement suit généralement sa propre politique énergétique, *e.g.,* équipement inutilisé, ou bien laisse l'environnement lui ordonner de changer d'état, *e.g.,* demande de mise en veille à distance. Les approches pour placer en état de basse consommation un équipement à distance sont courantes mais dépendent du système d'exploitation présent sur l'équipement. Pour parvenir à ce résultat, il existe différentes solutions, commerciales ou libres.

Pour passer dans un état actif, il existe également plusieurs approches à distance et qui dépendent principalement des médiums de communication intégrés à l'équipement. À cause de cette diversité, le réveil d'un équipement est impossible si aucun autre équipement ne supporte la méthode de réveil de l'équipement en état de basse consommation. Aussi, il est nécessaire de mettre en place une couche de contrôle répartie afin de faciliter le réveil des équipements.

Réveil d'un équipement à distance

Afin de réveiller un équipement à distance, il existe différentes méthodes qui sont liées au médium de communication, *e.g.,* Ethernet, ZigBee. Il existe généralement une seule méthode par médium de communication. Nous en listons quelques unes ici :

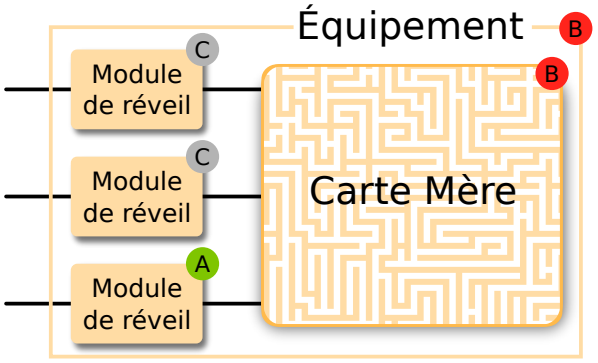

FIGURE 3.4 – **Réveil de la carte mère.** Pour réveiller la carte mère dans l'état de basse consommation (point B), il faut que les modules de réveil soient alimentés (point A) pour recevoir les ordres de réveil. D'autres module de réveils existent mais ne sont pas nécessairement activés (point C).

- Réveil sur un réseau Ethernet : *Wake-on-LAN* (WoL) [Lie02];
- Réveil sur le réseau WiFi : *Wake-on-Wireless LAN* (WoWLAN) [SBS02];
- Réveil sur un réseau ZigBee : HOPE [YFB+13];

L'ensemble de ces méthodes nécessite qu'une partie de l'équipement soit en état de basse consommation, *i.e.*, généralement la carte mère, et une autre partie, que nous nommons module de réveil, soit alimentée afin de recevoir les ordres de réveil provenant du reste du système réparti (cf Figure 3.4).

Le protocole de réveil distant est toujours le même :

- L'équipement distant envoie un message sur le médium de communication intégré à l'équipement qui se trouve dans l'état de basse consommation, *e.g.*, Ethernet, WiFi, ZigBee. Ce message est reçu par le module de réveil de l'équipement à réveiller;
- Le module de réveil envoie une interruption à la carte mère via une liaison série, *e.g.*, PCI, afin qu'elle passe dans un état actif.

Le message transmis sur le médium de communication reliant les deux équipements diffère d'une méthode de réveil à une autre. Par exemple, dans le cas du *Wake-on-LAN* [Lie02], il faut envoyer un « paquet magique » composé des octets `FF FF FF FF FF FF` suivis de 16 fois l'adresse MAC de l'équipement à réveiller. L'adresse MAC de l'équipement à réveiller doit être connue *a priori*.

Des extensions matérielles peuvent être ajoutées afin d'ajouter de nouvelles méthodes de réveil. Ces nouvelles méthodes de réveil se basent sur les ports série des équipements afin d'envoyer des interruptions à la carte mère. Il est toutefois nécessaire de modifier la configuration des ports séries pour activer ces nouvelles méthodes de réveil. Cela limite donc la diffusion de ces approches auprès du grand public.

Par exemple, l'option `Device Remote Wakeup` du protocole USB permet de laisser sous tension le contrôleur USB et de réveiller l'équipement si il en reçoit l'ordre extérieur [USB11]. En activant cette option, il est possible d'insérer un adaptateur USB-ZigBee. L'adaptateur joue alors un rôle de module de réveil lorsqu'un équipement distant, sur le même réseau ZigBee, lui ordonne de réveiller l'équipement auquel il est attaché [YFB+13].

Couche de contrôle

Au dessus des méthodes permettant de réveiller des équipements à distance, il est possible de mettre en place une couche de contrôle. Cette couche permet de collecter les informations de réveil des équipements (*e.g.*, méthodes de réveil, adresses de réveil), de conserver les données d'état des équipements ou encore de contrôler les états des autres équipements. L'objectif de cette couche de contrôle est de réveiller ou de faire passer dans un état de basse consommation les équipements en fonction des besoins des utilisateurs.

La couche de contrôle est utile lorsqu'un utilisateur cherche à consommer un service réparti sur un ensemble d'équipements. Par exemple, un utilisateur souhaite regarder un film sur sa télévision. Ce film est stocké sur un serveur de stockage en réseau. Toutefois, ce film nécessite un accès à un serveur de DRM sur Internet pour pouvoir autoriser le décodage sur la télévision. Jouer ce film requiert donc trois équipements de l'environnement domestique : la télévision, le serveur de stockage en réseau et un accès à Internet sur la passerelle résidentielle. Ces équipements sont dans un état de basse consommation.

Lorsque l'utilisateur démarre la télévision et sélectionne le menu permettant de consulter les vidéos stockées dans le réseau local, la couche de contrôle réveille le serveur de stockage en réseau. Lorsque le film est lancé, la couche de contrôle réveille la passerelle résidentielle pour accéder à Internet. Ainsi, les équipements sont démarrés en fonction du besoin.

Plusieurs couches de contrôle implémentent cette approche. *Universal Play and Play Low Power* (UPnP LP) [UPn07] est apparu en 2007. Ce protocole est basé sur UPnP [UPn08] qui est un standard présent sur beaucoup d'équipements de l'environnement domestique. UPnP LP propose une architecture permettant à un équipement de décrire son état, de le diffuser sur le réseau et de permettre à un autre équipement d'utiliser ces informations afin de le réveiller. L'ajout d'un mandataire de service basse consommation permet aux services d'être constamment visibles sur le réseau et donc consommables par les utilisateurs.

Une autre réalisation de cette approche est proposée par HOPE [YFB+13]. HOPE se base à la fois sur UPnP LP et un réseau ZigBee recouvrant. L'ajout du ZigBee permet le réveil des équipements intermédiaires qui ne sont pas visibles sur le réseau domestique, tels que les prises de Courant Porteur en Ligne.

3.3.2 Gestion énergétique du réseau d'équipements

Dans la Section 3.2, nous avons décrit comment les composants matériels d'un équipement s'adaptent à l'usage qui en est fait. Si la charge de calcul d'un équipement change, alors

le processeur s'adapte en conséquence adaptant ainsi la consommation d'énergie au besoin. Dans un environnement réparti, cette adaptation s'applique toujours aux équipements pris individuellement mais il faut également considérer une adaptation des relations entre les équipements.

Afin de réduire la consommation d'énergie de la communication entre deux équipements, ils doivent s'accordent sur différents points. Par exemple, ils choisissent le médium de communication ou le protocole le moins gourmand pour interagir. Ils peuvent également choisir le chemin de communication le moins énergivore. Sans cet accord, la communication risque d'échouer parce que l'un des équipements prend une décision qui impacte le fonctionnement de l'autre équipement.

Choix du moyen de communication

La première approche permettant de réduire l'énergie concerne le moyen de communication utilisé pour échanger des messages entre les équipements. Cela concerne donc l'ensemble des couches du modèle *Open Systems Interconnection* (OSI), de la couche Physique à la couche Application [JSAC01].

Dans un environnement hétérogène en équipements, chaque équipement possède différents médiums de communication (couche Physique), *e.g.,* WiFi, Bluetooth, Ethernet. Le standard IEEE P1905.1 [IEE13] récupère les données de communications relatives à ces médiums puis choisit quels médiums utiliser pour l'échange de messages entre plusieurs équipements. Parmi ces médiums de communication, l'un d'eux permet l'échange de messages pour un coût énergétique moindre que les autres. Ainsi, la consommation d'énergie est réduite en choisissant le médium de communication le plus adapté énergétiquement pour l'échange de messages entre deux équipements [CL12].

Pour les couches plus hautes, *i.e.,* Réseau ou Transport, des économies d'énergie sont également envisageables. Il est possible d'utiliser des protocoles plus économes en informations [YK03] ou de modifier les protocoles existants afin d'éviter l'envoi de messages considérés inutiles [PDR08], *e.g.,* messages d'acquittement. Enfin, il est possible de choisir le protocole de communication le plus adapté entre deux équipements [SHB08].

Optimisation de la topologie

Une autre approche consiste à choisir le chemin emprunté par un message pour communiquer avec un autre équipement. Dans un réseau maillé, il existe plusieurs chemins de communication entre deux équipements. Aussi, il existe un chemin qui nécessite moins d'énergie pour délivrer le message.

Ainsi, des protocoles de routage choisissent le chemin le moins consommateur en énergie afin de délivrer un message [SR02]. Le désavantage de cette approche est qu'elle implique d'utiliser constamment le même chemin, au risque de surcharger les équipements intermédiaires. Pour y remédier, il faut choisir le chemin le moins consommateur en énergie tout en tenant compte de la charge des équipements intermédiaires [PS08].

3.3.3 Répartition des applications

Gérer un ensemble d'équipements hébergeant des applications, réparties ou non, suppose qu'il y a plus d'équipements qu'il n'en faut pour exécuter ces applications. L'hypothèse est qu'il y a plus de ressources matérielles que la demande issue des applications. Ainsi, la marge de maœuvre est plus importante pour décider des équipements participant à l'exécution les applications.

Dans un tel cas, répartir « efficacement » les applications sur les équipements minimise la consommation d'énergie. Les objectifs amenant à répartir efficacement les applications sur les équipements diffèrent suivant les environnements considérés.

Nous présentons trois approches produisant des répartitions différentes en vue de faire des économies d'énergie. La première approche consiste à déporter des applications depuis des équipements sur batterie vers des équipements alimentés par le secteur. La deuxième approche consiste à minimiser le nombre d'équipements actifs. Enfin, la dernière approche consiste à placer les applications sur les équipements les plus adaptés pour les exécuter.

Dans l'ensemble des approches, il y a une décorrélation plus ou moins modélisée entre les équipements et les applications déployées dessus. Ainsi, il existe différents critères permettant de prendre les décisions de placement des applications. Ces critères définissent les contraintes permettant à une application d'être déployée ou non sur un équipement.

Déport des applications

De plus en plus d'environnements homogènes, mais foncièrement différents, sont connectés. Par exemple, un centre de traitement de données peut être connecté à un réseau mobile *ad hoc*. Il est donc envisageable de déporter certaines parties des applications dans des centres de traitement de données depuis des téléphones intelligents [MV03]. Cette approche augmente l'autonomie des équipements sur batterie tout en leur permettant de consommer le service.

Cependant, cette approche se borne souvent à considérer la consommation d'énergie d'une seule partie de l'environnement. Dans le cas des équipements sur batterie, l'objectif est d'augmenter leur autonomie. La consommation d'énergie est donc réduite sans préciser le coût du déport et de l'exécution de code dans l'autre partie de l'environnement considéré.

Par exemple, entre ces deux environnements, il existe un ensemble d'équipements réseaux à prendre en compte afin de calculer le coût global d'un déport d'une partie d'une application dans un environnement différent de l'environnement initial [KL10]. Ainsi, énergétiquement, il n'est pas toujours intéressant de déporter des services dans un autre environnement informatique.

Minimisation des équipements actifs.

La minimisation des équipements actifs, aussi appelée « consolidation », est principalement utilisée dans les environnements homogènes, *e.g.*, les centres de traitement de données [HLM+09]. La consolidation cherche à réduire le nombre d'équipements actifs en rassemblant l'ensemble des applications sur un minimum d'équipements. Au travers de différents algorithme décisionnel, *e.g.*, *round-robin*, *unbalanced*, le gain énergétique varie [LO10].

En plus de minimiser le nombre d'équipements, l'efficience énergétique d'un équipement est améliorée (cf. Figure 3.3). Cette approche est souvent combinée avec une mise en état de basse consommation des équipements considérés inactifs, *i.e.*, qui n'hébergent plus d'applications. Lorsque les ressources matérielles viennent à manquer ou que le nombre d'applications change, la répartition des applications est modifiée et si besoin, des équipements sont réveillés pour que la demande en ressources des applications soit satisfaite.

La consolidation considère qu'en réduisant le nombre d'équipements actifs, la consommation globale d'énergie diminue. Ce postulat est vrai dans les environnements homogènes où chaque équipement consomme pratiquement de la même manière et dans les mêmes proportions.

Adaptation aux équipements.

Dans les environnements plus hétérogènes, l'objectif n'est pas de réduire le nombre d'équipements mais de répartir les applications en tenant compte de l'hétérogénéité de l'environnement [SNS07]. Cette répartition considère différents critères, *e.g.*, autonomie, ressources matérielles disponibles. Elle vise simplement à répartir les applications suivant les caractéristiques des équipements qui composent l'environnement.

Par cette approche, chaque composant de l'application répartie voit son efficience énergétique augmenter car ils sont déployés sur les équipements qui sont les plus appropriés pour les exécuter. Par exemple, un composant décodant un film est déployé sur un équipement possédant un processeur de signal numérique, tandis qu'un composant faisant du stockage est déployé sur un serveur de stockage en réseau. Ainsi, par cette augmentation de l'efficience énergétique de chaque parties de l'application, l'efficience énergétique de l'application augmente également.

3.3.4 Mandatement

Le mandatement vise à déléguer une application ou un service à un équipement tiers afin de passer dans un état de basse consommation. La différence avec l'approche précédente est que l'équipement qui demande un mandatement reste le propriétaire de l'application ou du service.

Cette approche est particulièrement adaptée aux environnements hétérogènes en équipement où les applications ou les services sont liés à un équipement particulier. Cela permet soit de continuer l'exécution d'une application, soit de donner l'illusion que le service est disponible, malgré que le mandant soit en état de basse consommation.

Mandatement d'applications

Le mandatement d'application vise à déléguer l'exécution d'une application à un équipement tiers afin de passer en état de basse consommation. Par exemple, un équipement exécute une application de téléchargement. Si l'équipement est rendu inactif, le téléchargement peut être continué sur un équipement tiers, afin que le mandant passe en état de basse consommation. Dès que l'équipement redémarre, l'exécution de l'application continue sur le mandant si l'application n'est pas terminée, ou le fichier téléchargé est transféré si le téléchargement est terminé.

Le mandataire de l'application est un équipement à part entière qui se trouve quelque part dans le réseau [ZSZX11] ou bien est une extension d'un équipement qui dispose des ressources matérielles nécessaires à exécuter des applications simples [AHC+09]. Dans les deux cas, le mandataire est toujours dans un état actif même lorsqu'il n'exécute aucune application afin d'être toujours disponible pour héberger des applications.

Mandatement de services

Le mandatement de service vise à rendre visible sur le réseau les services fournis par des équipements dans un état de basse consommation. Cela permet de donner l'illusion que le service est disponible, malgré que l'équipement qui le propose soit dans un état de basse consommation. Si un équipement essaye de consommer le service, le mandataire ou l'équipement client réveille alors l'équipement qui fournit ce service.

Cette approche est proposée pour les environnements bureautiques, dans lesquels l'ensemble des équipements considérés peut s'avérer très grand [RGKP10, ASG10, SLH+12]. Mais cette approche est également disponible au travers du standard UPnP LP dans les environnements domestiques [UPn07] ou dans les réseaux mobiles *ad hoc* [SHB08]. Dans cette approche, le service reste lié à un équipement, *i.e.*, un équipement précis fournit le service.

Synthèse

Cette section a présenté les différentes approches qui minimisent la consommation d'énergie d'un ensemble d'équipements. Cette minimisation s'effectue sur différentes couches des systèmes répartis, des communications entre les équipements à la répartition des applications. Au niveau des communications, les approches cherchent à mieux s'adapter aux équipements ou aux chemins reliant les équipements.

Au niveau applicatif, la minimisation se fait soit au détriment d'un autre équipement, *e.g.*, consolidation, soit en répartissant/déléguant efficacement les applications sur les équipements. Suivant le type d'environnement considéré, certaines approches sont plus appropriées. Par exemple dans le cas d'une environnement homogène, la consolidation semble appropriée car n'importe quelle application peut s'exécuter sur n'importe quel équipement. Dans le cas des environnements hétérogènes, c'est la répartition ou le mandatement des applications qui sont les plus adaptés car ils tiennent compte du caractère singulier de chaque équipement.

3.4 Modification des usages grâce aux systèmes informatiques

Jusqu'à maintenant, nous avons discuté des systèmes informatiques et de la modification de leur fonctionnement afin de faire des économies d'énergie. Cela nécessite de spécifier concrètement le service en se basant sur les besoins des utilisateurs puis de mettre en place des mécanismes pour le délivrer pour un coût énergétique le plus faible possible, voir à le dégrader. Les sections précédentes reposent toutes sur ce principe : adapter la manière dont est fourni le service, sans changer le besoin qui a conduit à développer ce service.

Toutefois, il est important de mentionner, même succinctement, ce que peut offrir l'informatique afin de modifier les besoins en vue de faire des économies d'énergie. La modification des besoins des utilisateurs est un levier important pour faire des économies d'énergie. Cela ne signifie pas dégrader la qualité de service ou s'adapter à l'utilisateur mais bien réduire ou modifier les besoins des utilisateurs.

Pour cela, il faut une interaction entre le système informatique et l'utilisateur, notamment en lui fournissant des informations sur la consommation d'énergie. Ce phénomène de « rétroaction » [18] permet à l'utilisateur d'adapter son comportement et de réduire sa consommation d'énergie [Dar06, Fis08].

Par exemple, chez les utilisateurs, certaines activités sont faites plus régulièrement que d'autres, notamment à leur domicile [Bea09]. L'ajout d'un indicateur sur la consommation énergétique d'un utilisateur, consommation qui varie en fonction des usages, l'amène à modifier son comportement en conséquence [WN03, SCG+11].

Un autre exemple est de mettre en concurrence les utilisateurs afin de les inciter à réduire leur consommation d'énergie [NSC+08]. La mise en concurrence vise à positionner l'individu par rapport à une valeur de consommation d'énergie moyenne par utilisateur. Ainsi, l'utilisateur modifie ses besoins en conséquence et cherche à faire mieux que la moyenne.

D'un point de vue ingénierie de système, cela signifie qu'en plus d'adapter le système à l'utilisateur pour réduire la consommation d'énergie, l'utilisateur peut modifier ses besoins pour augmenter le gain énergétique. Pour que l'utilisateur modifie ses besoins, il faut mettre en place des indicateurs lui restituant ce que coûtent ses besoins en énergie.

3.5 Conclusion

Des approches de réduction de la consommation d'énergie existent aussi bien au niveau de l'équipement qu'au niveau du système réparti. Nous n'avons pas décrit toutes les approches existantes pour chaque parties des systèmes informatiques, *e.g.*, autres composants matériels, chaque couche du modèle OSI. Toutefois, ce panorama permet de classer ces approches les unes par rapport aux autres et de définir une « pîle énergétique ». Cette pîle énergétique est scindée en quatre parties (cf. Figure 3.5) :

18. En psychologie, il s'agit de « conditionnement opérant » .

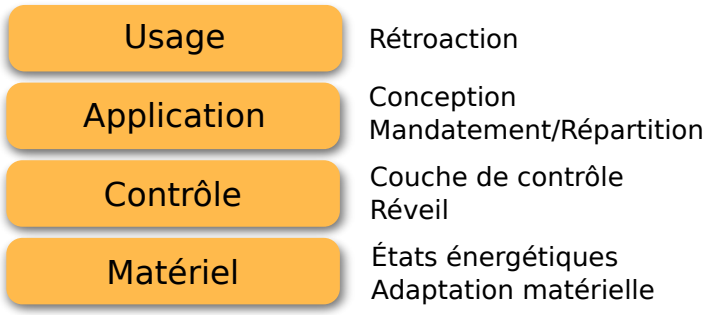

FIGURE 3.5 – **Pîle énergétique d'un système informatique.** Chaque couche détaille les approches qui participant à la réduction de la consommation d'énergie d'un système informatique.

Matériel. Cette couche propose des approches permettant de réduire la consommation d'énergie des ressources matérielles, *e.g.*, processeur, carte réseau.

Contrôle. Cette couche permet le contrôle énergétique des ressources matérielles par la couche Application. Il s'agit de l'interface entre les applications et le matériel.

Application. Cette couche s'occupe de l'exécution des applications en cherchant à minimiser l'usage des ressources matérielles.

Usage. Cette couche informe l'utilisateur de la consommation énergétique des usages qu'il fait des systèmes informatiques. En conséquence, il modifie son comportement.

Cet état de l'art permet de déduire une approche qui résume toutes ces approches : adaptation. L'ensemble de ces approches cherche à adapter la consommation d'énergie aux besoins des couches supérieures. L'adaptation se retrouve sur toutes les couches, des couches matérielles aux usages. Ces besoins, auxquels les systèmes informatiques s'adaptent, sont caractérisés à la conception et évoluent à l'exécution.

Dans le chapitre suivant, nous présentons les systèmes adaptatifs existants combinant certaines des approches décrites dans ce chapitre afin de réduire la consommation d'énergie d'un ensemble d'équipements. Par exemple, il est possible de combiner des approches aux niveaux du matériel, *e.g.*, DVFS, et au niveau des applications, *e.g.*, répartition des applications. Le chapitre suivant présente également notre positionnement par rapport à ces systèmes.

Chapitre 4

Systèmes adaptatifs et énergie

Nous sommes, en tant que source d'énergie, facilement renouvelable et entièrement recyclable, les morts sont liquéfiés et nourrissent par intraveineuse les vivants.
– Morpheus, *Matrix*

Sommaire

4.1 Introduction 50
4.2 Systèmes adaptatifs pour la réduction d'énergie 50
4.2.1 Entropy 51
4.2.2 HOPE 52
4.2.3 Mobile Service Overlay 53
4.2.4 Parasite 54
4.2.5 PARM 54
4.2.6 SANDMAN 55
4.2.7 Transhumance 56
4.2.8 UPnP Low Power 57
4.3 Conclusion 58

4.1 Introduction

Le chapitre précédent détaille les approches réduisant la consommation d'énergie d'un équipement, d'un ensemble d'équipements ou d'un besoin issu de l'utilisateur. Notre approche cherche à améliorer l'efficience énergétique d'un ensemble d'équipement. Aussi, nous présentons les travaux connexes existants réduisant la consommation d'énergie d'un ensemble d'équipement de manière autonome, *i.e.*, sans l'intervention d'un utilisateur. Ces travaux sont issus de la littérature scientifique ou de travaux de standardisation.

Les travaux connexes, décrits dans ce chapitre, s'inscrivent dans des environnements répartis différents. Ces travaux mettent en œuvre une ou plusieurs approches du Chapitre 3 afin de minimiser la consommation d'énergie de l'environnement qu'ils considèrent. Au travers de ces travaux, nous analysons les avantages et les inconvénients de chacun afin de positionner notre approche.

Structure du chapitre

Ce chapitre s'articule comme suit : la Section 4.2 présente différents systèmes dont l'objectif est de réduire la consommation d'énergie d'un ensemble d'équipements. La Section 4.3 présente une synthèse de ces différents systèmes et décrit notre positionnement par rapport à eux.

4.2 Systèmes adaptatifs pour la réduction d'énergie

Cette section décrit différents travaux proposant des systèmes autonomes cherchant à réduire la consommation d'un ensemble d'équipements. Chacun de ces systèmes est conçu pour un environnement précis en prenant en compte des contraintes différentes pour décider des adaptations à effectuer. Aussi, afin de positionner notre travail par rapport à ces systèmes, nous les présentons au travers des critères suivants :

Environnement. Ce critère précise dans quel environnement le système s'inscrit, *e.g.*, maison numérique, centre de traitement de données.

Approches utilisées. Ce critère détaille l'ensemble des approches, issues du chapitre précédent, qui sont utilisées afin de réduire la consommation d'énergie de l'ensemble des équipements, *e.g.*, mandatement de service, répartition d'applications.

Événements considérés. Ce critère décrit les événements modélisés et pris en compte en vue d'effectuer une adaptation du système. Ces événements définissent la volatilité des systèmes, *i.e.*, les événements considérés mais imprévisibles.

Contraintes décisionnelles. Ce critère présente les contraintes modélisées et prises en compte lors des décisions visant à s'adapter. Ces contraintes définissent les hétérogénéités considérées par les systèmes, *i.e.*, les différences caractérisant les équipements ou les applications de l'environnement d'étude.

Qualité de service. Ce critère spécifie si le système conserve la qualité de service ou la dégrade en vue de faire des économies d'énergie (cf. Section 3.2.3).

Degré d'autonomie. Ce critère décrit quel degré d'autonomie est supporté par le système, *i.e.*, cœur, support, contrôle ou autonome (cf. Section 2.3).

Politique de décision. Ce critère détaille la politique de décision utilisée et qui définit les actions à effectuer pour s'adapter à l'environnement, *i.e.*, politique d'action, politique d'objectif ou politique de fonction d'utilité (cf. Section 2.3).

4.2.1 Entropy

Entropy [HLM+09] se base sur le principe de la consolidation. Ainsi, Entropy vise à minimiser le nombre d'équipements actifs dans un centre de traitement de données. Cette minimisation est possible en migrant des machines virtuelles d'un serveur à un autre en fonction de la charge processeur de ces derniers et de la mémoire vive disponible.

Entropy décrit un modèle dans lequel le temps de migration d'une machine virtuelle est pris en compte dans le calcul du placement des machines virtuelles sur les serveurs. Entropy insiste sur la nécessité de limiter les migrations des machines virtuelles pour que la consolidation soit la plus efficace possible.

Environnement. Centre de traitement de données.

Approches utilisées. Minimisation des équipements actifs, passage dans un état de basse consommation des équipements inutilisés.

Événéments considérés. Apparition et disparition de machines virtuelles.

Contraintes décisionnelles. Ressources matérielles, temps de migration.

Qualité de service. Conservation.

Degré d'autonomie. Contrôle.

Politique de décision. Politique de fonction d'utilité.

Entropy se limite à un environnement homogène. Les équipements ont tous les mêmes types de ressources matérielles, *i.e.*, processeur, mémoire vive. Au contraire, la maison numérique nécessite de prendre en compte d'autres hétérogénéités dues aux différences plus importantes entre les équipements, *e.g.*, caméra, écran.

Enfin, le système décidant de la répartition des machines virtuelles ne se prend pas en compte pour réduire la consommation d'énergie de l'environnement. En effet, dû à la consommation d'énergie d'un tel environnement, la consommation de l'équipement en charge de la répartition des machines virtuelles est négligeable.

4.2.2 HOPE

HOPE [YFB+13] est un intergiciel destiné à la maison numérique. Cet intergiciel se base sur les technologies UPnP Low Power et ZigBee pour créer un réseau recouvrant sur l'ensemble des équipements informatique de l'environnement. Il propose notamment l'ajout d'un adaptateur USB-ZigBee permettant à l'équipement de passer en état de basse consommation sans que le nœud ZigBee disparaisse. Ainsi, l'équipement distant peut être réveillé au travers de ZigBee.

Le réveil est effectué lorsqu'un équipement cherche à consommer un service présent sur un autre équipement. HOPE capture le message et, au travers du réseau recouvrant, réveille l'équipement qui héberge le service pour qu'il soit consommé ainsi que les équipements intermédiaires. Par exemple, un téléphone intelligent envoi un message en UPnP sur le réseau. Ce message est capturé par HOPE qui réveille la télévision hébergeant le service ainsi que les prises de courant porteur en ligne grâce au réseau recouvrant ZigBee.

La deuxième approche utilisée pour réduire la consommation d'énergie est de choisir le chemin le plus économe en énergie pour fournir un service. Pour cela, HOPE considère qu'il existe plusieurs chemins permettant d'accéder au service. HOPE se charge alors de choisir le chemin en fonction de la bande passante disponible, de l'énergie consommée et de la fréquence radio.

Environnement. Maison numérique.

Approches utilisées. Passage dans un état de basse consommation des équipements inutilisés, couche de contrôle, optimisation de la topologie.

Événéments considérés. Apparition et disparition d'applications.

Contraintes décisionnelles. Bande passante, énergie consommée, fréquence radio.

Qualité de service. Conservation.

Degré d'autonomie. Contrôle.

Politique de décision. Politique d'action.

HOPE cherche à réduire la consommation d'énergie de la maison numérique en passant en état de basse consommation les équipements inutilisés. Pour cela, il utilise une couche de contrôle se basant à la fois sur ZigBee et UPnP Low Power. Cette couche de contrôle

réveille ou fait passer dans un état de basse consommation les équipements participant à la fourniture d'un service à l'utilisateur.

L'adaptation de HOPE se limite à placer dans un état de basse consommation les équipements qui ne participent pas à la fourniture d'un service. HOPE ne cherche pas à optimiser cette consommation à l'aide d'approches telles que la répartition de services. Aussi, il ne tire pas partie de l'apparition de nouvelles ressources matérielles ou de la mutualisation des applications sur un équipement pour réduire la consommation d'énergie.

4.2.3 Mobile Service Overlay

Mobile Service Overlay (MSO) [SNS07] est un intergiciel destiné aux réseaux mobiles *ad hoc*. Il déploie les composants d'une application en fonction de l'autonomie des nœuds ainsi que de leurs ressources matérielles disponibles. MSO prend en considération les changements dans l'environnement afin de fournir une répartition des composants qui minimise toujours la consommation d'énergie de l'ensemble des nœuds.

Par exemple, MSO observe constamment le niveau d'énergie restant sur chaque nœud ainsi que la charge de leur processeur. À tout moment, MSO peut modifier la répartition des applications en tenant compte de ces deux paramètres. Il peut notamment déplacer une application s'exécutant sur un nœud ayant un niveau d'énergie bas vers un nœud possédant plus d'énergie.

Environnement. Réseau mobile *ad hoc*.

Approches utilisées. Adaptation aux équipements.

Événéments considérés. Autonomie faible, ressources disponibles, disparition d'équipements.

Contraintes décisionnelles. Autonomie, charge du processeur.

Qualité de service. Conservation.

Degré d'autonomie. Contrôle.

Politique de décision. Politique de fonction d'utilité.

MSO propose un algorithme heuristique pour définir le placement des applications sur les nœuds. Cet algorithme est spécialement conçu pour cet environnement, rendant difficile son utilisation dans un autre environnement avec d'autres hétérogénéités. Par exemple, MSO se limite à un environnement homogène en type d'équipements, *i.e.*, tous les nœuds ont les mêmes ressources matérielles. Les auteurs prévoient de revoir l'algorithme pour l'adapter à des environnements plus hétérogènes.

4.2.4 Parasite

Parasite [ZSZX11] est un intergiciel qui capture des événements de téléchargement provenant d'équipements bureautiques, *e.g.,* ordinateur fixe. Ces événements sont ensuite envoyés à un équipement dédié au téléchargement de fichiers. Cela permet ainsi de libérer l'équipement bureautique du service de téléchargement. Notamment, si l'utilisateur laisse l'équipement inactif, ce dernier passe alors en état de basse consommation.

Parasite propose également un système de cache permettant de ne télécharger qu'une seule fois le fichier. Ainsi, si un autre utilisateur demande à télécharger le même fichier, il est redirigé par la copie présente localement dans le cache de l'entreprise. Cela accélère la fourniture du service et mobilise donc moins de ressources matérielles.

Environnement. Informatique de bureau.

Approches utilisées. Mandatement d'applications.

Événements considérés. Apparition d'une application (limité au téléchargement).

Contraintes décisionnelles. -

Qualité de service. Conservation.

Degré d'autonomie. Cœur.

Politique de décision. Politique d'action.

Cette approche nécessite un équipement dédié qui ne peut pas passer dans un état de basse consommation. Si aucun téléchargement n'est en cours et que les ordinateurs de bureau sont actifs, l'équipement dédié reste également dans l'état actif. Cela augmente d'autant plus la consommation globale d'énergie.

Cette approche se limite également à un seul service, *i.e.,* le service de téléchargement. Parasite ne propose pas d'architecture générique pour prendre en considération d'autres services de l'informatique de bureau.

4.2.5 PARM

PARM [MV03] est un intergiciel permettant de déporter certains de ses composants sur des mandataires, géographiquement proches des mandants, *i.e.,* dans le même sous réseau. Suivant les décisions prises par le « power broker », *i.e.,* l'entité en charge de la décision, les composants de l'intergiciel sont répartis soit sur le mandataire soit sur l'équipement basse consommation.

Dans le cas où le composant s'exécute sur l'équipement et doit être migré sur le mandataire, la migration est transparente pour l'application s'exécutant sur l'équipement basse

consommation. Un composant *stub* simule alors localement le composant migré et fait la liaison avec ce dernier s'exécutant sur le mandataire.

Dans la cas où la migration n'est pas possible, PARM dégrade le service fournit par l'intergiciel à l'application le requérant ou bien arrête simplement le service pour économiser de l'énergie.

Environnement. Réseau mobile *ad hoc*, centre de traitement de données.

Approches utilisées. Déport des applications, mandatement de services, dégradation de services.

Événements considérés. Apparition d'une application.

Contraintes décisionnelles. Autonomie, énergie consommée.

Qualité de service. Conservation, dégradation.

Degré d'autonomie. Contrôle.

Politique de décision. Politique de fonction d'utilité.

PARM met en œuvre diverses approches pour réduire la consommation d'énergie. Il propose notamment de déporter les applications sur des équipements plus à même de les exécuter. La portée de cette approche est limitée car elle ne considère pas une vue de l'ensemble des équipements mais un environnement par équipement mobile. Il est donc nécessaire de prendre la décision de déporter les applications pour chaque équipement mobile.

De plus, PARM ne prend pas en compte le « power broker » dans le calcul de la répartition des composants de l'intergiciel. Cette entité se trouve dans un centre de traitement de données. Aussi, pour avoir une solution totalement automatique, il est nécessaire de considérer tous les équipements intermédiaires, *e.g.*, routeurs, dans le calcul de la consommation totale de PARM. Cela limite donc le degré d'autonomie de la solution.

4.2.6 SANDMAN

SANDMAN [SB07] est une extension de l'intergiciel BASE [BSGR03] conçu pour les réseaux mobiles *ad hoc*. SANDMAN propose de passer dans un état de basse consommation les nœuds qui sont inutilisés afin de réduire la consommation d'énergie globale. Pour cela, il propose l'utilisation de *sessions* permettant de préciser si le nœud fournit actuellement un service à un autre nœud.

Dans le cas où un nœud passe dans un état de basse consommation, SANDMAN permet de le laisser visible sur le réseau. En effet, lorsqu'un nœud change d'état, il informe le leader de la grappe de nœuds dans laquelle il se trouve. Il lui fournit notamment la durée de son

état de basse consommation. Ainsi, lorsqu'un nœud tiers cherche à le joindre, le leader joue le rôle de mandataire afin de l'informer du temps restant avant le réveil du nœud cible.

Le deuxième mécanisme mit en avant pour réduire la consommation d'énergie est le choix du protocole de communication le plus adapté à un instant donné. Toutefois, les auteurs ne précisent pas sur quels critères ils prennent les décisions.

Environnement. Réseau mobile *ad hoc*.

Approches utilisées. Passage dans un état de basse consommation des équipements inutilisés, mandatement de service, choix du protocole de communication.

Événéments considérés. -

Contraintes décisionnelles. ?

Qualité de service. Conservation.

Degré d'autonomie. Support.

Politique de décision. -

SANDMAN ne permet pas de réveiller un nœud lorsqu'un client en a besoin. Lors du passage dans un état de basse consommation, les nœuds informent les leaders auxquels ils sont attachés, de la durée de cet état. Il existe aucune méthode permettant à un nœud d'en réveiller un autre à moins d'attendre qu'il passe dans un état actif de lui même.

SANDMAN propose de choisir le protocole de communication entre deux nœuds afin de réduire la consommation d'énergie. Ce choix est indépendant de l'application s'exécutant sur l'intergiciel. Toutefois, cette adaptation se limite à la communication entre les parties d'une application répartie. C'est pour cela que le degré d'autonomicité est faible puisqu'il ne concerne qu'un seul élément, à savoir les communications.

4.2.7 Transhumance

Transhumance [PDR08] est un intergiciel qui adapte son comportement, *i.e.*, sécurité, mode de transport, en fonction de l'autonomie restante des nœuds d'un environnement de type réseau mobiles *ad hoc*. Pour cela, l'intergiciel définit des seuils, *i.e.*, des niveaux d'énergie, au delà desquels des adaptations dégradant les services qu'il propose sont définies. Par exemple, au delà d'un certain seuil, les composants assurant la sécurité des communications sont supprimés. Cette suppression permet d'économiser de l'énergie car les ressources dédiées au chiffrement des communications sont libérées.

Environnement. Réseau mobile *ad hoc*.

Approches utilisées. Adaptation à l'exécution, dégradation de service.

Événements considérés. Niveau d'énergie des équipements.

Contraintes décisionnelles. Autonomie.

Qualité de service. Dégradation.

Degré d'autonomie. Autonome.

Politique de décision. Politique d'action.

Transhumance vise à s'adapter en fonction de l'environnement sur lequel il s'exécute. Les adaptations sont faites indépendamment des applications de plus haut niveau qui s'exécutent sur les nœuds du réseau mobile *ad hoc* et visent à dégrader les services fournis aux applications. Ainsi, une application répartie utilisant cet intergiciel est amenée à voir sa qualité de service être à son tour dégradée, indépendamment de ses besoins.

Par exemple, l'application utilise les composants de chiffrement de l'intergiciel pour communiquer. Si un seuil d'énergie est atteint, Transhumance supprime alors les composants de chiffrement pour augmenter l'autonomie des nœuds. Cette adaptation ne tient pas compte des besoins de chiffrement des communications de l'application.

4.2.8 UPnP Low Power

UPnP Low Power [UPn07] permet à un équipement de représenter les services d'un autre équipement sur le réseau domestique. Ainsi, ces services sont toujours visibles sur le réseau afin qu'un client puisse y accéder. Lorsqu'un client souhaite consommer ce service, le mandaté fournit au client les informations de réveil du mandant afin qu'il le réveille.

Environnement. Maison numérique.

Approches utilisées. Couche de contrôle, mandatement de services.

Événéments considérés. Apparition et disparition d'applications.

Contraintes décisionnelles. -

Qualité de service. Conservation.

Degré d'autonomie. Contrôle.

Politique de décision Politique d'action.

La couche de contrôle décrite dans UPnP Low Power nécessite que l'équipement client ou le mandataire réveillent l'équipement fournissant le service. Si aucun des deux équipements ne possède la méthode de réveil, alors ils ne peuvent pas faire appel à un équipement tiers la possédant pour réaliser le réveil.

De plus, UPnP LP ne propose pas d'adaptation autre que celle de réveiller un équipement sur demande d'un utilisateur ou d'un service tiers. Aussi, UPnP LP ne fournit pas une modélisation de la maison numérique et de ses propriétés.

4.3 Conclusion

Les systèmes actuels, présentés dans ce chapitre, mettent en œuvre plusieurs des approches décrites dans le chapitre précédent. Le cumul de certaines de ces approches améliore les gains énergétiques, *e.g.*, minimisation des équipements actifs et passage dans un état de basse consommation des équipements inactifs. Tous ces systèmes sont mis en œuvre au travers d'intergiciels déployés sur l'ensemble des équipements de l'environnement considéré.

Toutefois, la majorité d'entre eux ne propose pas un degrés d'adaptation automatique, *i.e.*, ils ne se considèrent pas dans la réduction de la consommation d'énergie. Or, quel que soit l'environnement, le système ne doit pas consommer plus d'énergie qu'il ne permet d'en économiser. Il est donc nécessaire que le système se prenne en compte dans la réduction de la consommation d'énergie.

De plus, une partie des systèmes est limitée par la politique mise en place pour réduire la consommation d'énergie, *i.e.*, politique d'action. Ces politiques nécessitent de définir l'ensemble des états énergétiques que l'environnement est amené à prendre à la conception. Comme chaque ménage possède un ensemble d'équipements différents, cette définition à la conception est impossible. Il faut donc une politique de fonction d'utilité, décrivant tous les états énergétiques.

Ces systèmes s'inscrivent principalement dans des environnements homogènes, où chaque équipement est l'équivalent des autres équipements de l'environnement, *i.e.*, réseau mobile *ad hoc* ou centre de traitement de données. Cela limite donc la réutilisation de leur modèle dans des environnements plus hétérogènes où des contraintes supplémentaires doivent être considérées, *e.g.*, présence de l'utilisateur.

Enfin, ces systèmes gèrent souvent peu d'événements dans l'adaptation à l'environnement. Ces événements se limitent souvent soit aux applications, soit aux équipements. Or dans un environnement comme la maison numérique, les événements issus des équipements et des applications sont à considérer pour toujours atteindre l'efficience énergétique.

Le Tableau 4.1 et le Tableau 4.2 synthétisent notre positionnement par rapport à ces travaux. Par la suite, le Chapitre 5 décrit le modèle sur lequel nous nous basons pour déduire la fonction d'utilité. Cette politique de décision est ensuite incorporée dans le Chapitre 6 où nous décrivons notre système autonome prenant en compte les événements et modifiant l'architecture des services afin d'atteindre notre objectif d'efficience énergétique.

	Environnement	Approches utilisées	Événements considérés
Entropy	Centre de traitement de données	Minimisation des équipements actifs, passage dans un état de basse consommation des équipements inutilisés	Apparition et disparition de machines virtuelles
HOPE	Maison numérique	Passage dans un état de basse consommation des équipements inutilisés, couche de contrôle, optimisation de la topologie	Apparition et disparition d'applications
Mobile Service Overlay	Réseau mobile *ad hoc*	Adaptation aux équipements	Autonomie faible, ressources disponibles, disparition d'équipements
Parasite	Informatique de bureau	Mandatement d'applications	Apparition d'applications
PARM	Réseau mobile *ad hoc*, centre de traitement de données	Déport des applications, mandatement de services, dégradation de services	Apparition d'une application
SANDMAN	Réseau mobile *ad hoc*	Passage dans un état de basse consommation des équipements inutilisés, mandatement de service, choix du protocole de communication	-
Transhumance	Réseau mobile *ad hoc*	Adaptation à l'exécution, dégradation de service	Niveau d'énergie des équipements
UPnP Low Power	Maison numérique	Couche de contrôle, mandatement de services	Apparition et disparition d'applications
Notre approche	Maison numérique	Adaptation aux équipements, passage dans un état de basse consommation des équipements inutilisés	Apparition et disparition d'équipements et d'applications

TABLE 4.1 – **Positionnement de notre approche par rapport à l'existant.** Ce tableau présente les différents travaux évoqués dans l'état de l'art et permet de positionner notre approche par rapport à eux.

	Contraintes décisionnelles	Qualité de service	Degré d'autonomie	Politique de décision
Entropy	Ressources matérielles	Conservation	Contrôle	Politique de fonction d'utilité
HOPE	Bande passante, énergie consommée, fréquence radio	Conservation	Contrôle	Politique d'action
Mobile Service Overlay	Autonomie, charge du processeur	Conservation	Contrôle	Politique de fonction d'utilité
Parasite	-	Conservation	Cœur	Politique d'action
PARM	Autonomie, énergie consommée	Conservation, dégradation	Contrôle	Politique de fonction d'utilité
SANDMAN	?	Conservation	Support	-
Transhumance	Autonomie	Dégradation	Autonome	Politique d'action
UPnP Low Power	-	Conservation	Contrôle	Politique d'action
Notre approche	Ressources matérielles, usages, consommation d'énergie	Conservation	Autonome	Politique de fonction d'utilité

TABLE 4.2 – **Positionnement de notre approche par rapport à l'existant (suite).** Ce tableau présente les différents travaux évoqués dans l'état de l'art et permet de positionner notre approche par rapport à eux.

Deuxième partie

Contribution

Chapitre 5

Modélisation

La Pierre Philosophale : ceux qui la possèdent ne sont plus limités par les lois de l'échange équivalent en alchimie, peuvent gagner sans sacrifier ... créer sans échange équivalent.
– Edward Elric, *Fullmetal Alchemist*

Sommaire

5.1 Introduction 64
5.2 L'efficience énergétique des équipements informatiques 64
5.2.1 Travail utile et processus 65
5.2.2 La mutualisation des ressources 69
5.2.3 Maximiser l'efficience énergétique d'un ensemble d'équipements 69
5.3 La maison numérique 70
5.3.1 Hétérogénéité 71
5.3.2 Volatilité 71
5.3.3 Répartition 71
5.3.4 Ouverture aux tiers 72
5.3.5 Qualité de service 72
5.4 Modélisation de la maison numérique 73
5.4.1 Le plan de répartition 73
5.4.2 Les actions 73
5.4.3 Les contraintes de déploiement 75
5.4.4 La consommation électrique 78
5.5 Utilisation des composants 79
5.5.1 La mobilité des applications 80
5.5.2 Décomposition et regroupement 83
5.5.3 Les composants sans contraintes 84
5.6 Conclusion 84

5.1 Introduction

L'efficience énergétique est devenue un champ de recherche à part entière. Ce champ est important pour que nos systèmes informatiques aient un faible impact financier ou écologique. En informatique, ce champ de recherche est principalement mis en avant dans les centres de traitement de données et les réseaux de capteurs sans fils, où l'énergie est cruciale pour leur fonctionnement.

Dans la maison numérique, les initiatives sont peu nombreuses. Il est intéressant de s'interroger sur la pertinence de la mise en application pour la maison numérique dans la mesure où le gain n'est pas aussi important que dans les autres domaines. Toutefois, il faut garder en mémoire qu'un petit gain pour chaque ménage correspond à un gain plus important à l'échelle d'un pays.

L'efficience énergétique dans la maison numérique doit prendre en compte les propriétés particulières de cet environnement. Contrairement aux autres environnements, les équipements de la maison numérique sont très hétérogènes et très volatiles. Aussi, nous définissons avant tout les deux grands thèmes entremêlés dans ce chapitre : l'efficience énergétique et la maison numérique. Ces définitions sont la base du modèle proposé dans ce chapitre.

Le modèle proposé s'inspire des approches réduisant la consommation des environnements répartis à l'exécution des applications. Cette approche cherche à minimiser la consommation d'énergie tout en garantissant que les services sont correctement fournis aux ménages. Pour cela, nous utilisons des applications à base de composants qui sont plus facilement répartissable et facilitent la satisfaction des besoins en ressources matériels de chaque composant.

Structure du chapitre

Le chapitre s'articule comme suit : la Section 5.2 spécifie ce qu'est l'efficience énergétique et son lien avec l'informatique. La Section 5.3 décrit les propriétés de la maison numérique. La Section 5.4 définit le modèle qui permet d'appliquer l'efficience énergétique à la maison numérique. La Section 5.5 étend le modèle au travers de l'utilisation des composants logiciels. Enfin, la Section 5.6 conclut ce chapitre.

5.2 L'efficience énergétique des équipements informatiques

Il existe plusieurs définitions de l'efficience énergétique, dépendant principalement du domaine d'étude considéré (*e.g.*, thermodynamique, économie) [Pat96]. Toutefois, la Définition 3 est la plus générale et la plus communément admise.

Définition 3 : L'efficience énergétique

$$\text{Efficience énergétique} = \frac{\text{Travail utile du processus}}{\text{Apport d'énergie dans le processus}}$$

L'amélioration de l'efficience énergétique d'un processus consiste à maximiser le rapport du « travail utile fourni par un processus » par « l'apport d'énergie pour alimenter ce processus ». Cette problématique peut être ramenée à un problème d'optimisation dans lequel il faut consommer le moins d'énergie pour fournir un même travail utile. Cette définition peut donc être redéfinit comme suit :

Définition 4 : La transformation de l'énergie en travail utile

Un travail utile S est le résultat d'une transformation d'une quantité d'énergie ϵ par un processus P :

$$S = P(\epsilon)$$

Toutefois ces définitions sont assez abstraites et ne donnent pas de détails sur la nature du processus permettant de transformer l'énergie en travail utile. Ce « processus » étant lié au domaine d'application, nous détaillons ce que nous entendons par processus en informatique. De plus, ces définitions ne détaillent pas ce qu'est un « travail utile ». Une définition de ces deux termes est donc nécessaire afin de comprendre ce chapitre.

5.2.1 Travail utile et processus

Pour qu'un travail soit utile, il faut que le fruit de ce travail satisfasse un besoin. Sans besoin, pas de raison de produire ce travail. Le besoin est donc le préalable à un travail utile.

Pour concevoir le système satisfaisant ce besoin, il faut donc avant tout exprimer ce besoin. Sans cette expression, l'architecte, *i.e.*, la personne chargée de concevoir le système satisfaisant le besoin, ne peut pas commencer son travail. Aussi, le besoin doit être défini au travers d'une spécification qui décrit clairement ce qui est attendu d'un travail pour qu'il soit considéré utile, *i.e.*, ensemble de critère à satisfaire. Ainsi, un travail utile est défini comme suit :

Définition 5 : Le travail utile

Un travail utile est un travail qui satisfait à un ensemble de besoins clairement décrits.

Cette définition nécessite de satisfaire à un ensemble de critères pour considérer le travail comme utile. Par exemple, si le travail utile est de regarder une vidéo sans l'audio, les

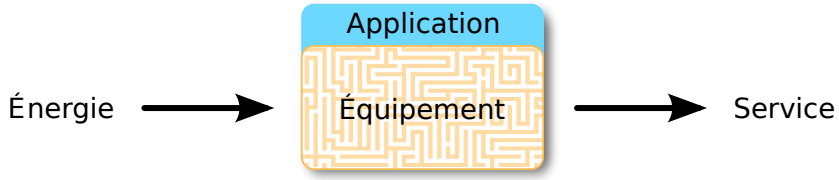

FIGURE 5.1 – **Processus de transformation de l'énergie en service.** Afin de transformer de l'énergie en service, il faut l'association d'un équipement et d'une application.

critères peuvent être que cette vidéo soit jouée sur un écran de taille $L \times l$ et que l'utilisateur soit devant cet écran. Si l'utilisateur n'est pas devant l'écran alors le travail n'est pas utile puisqu'il ne satisfait pas le besoin de l'utilisateur.

Pour fournir un travail utile, un équipement et une application sont nécessaires. En effet, un équipement informatique en sortie d'usine, *i.e.*, l'électronique brute, n'est pas capable de fournir un service à l'utilisateur (en informatique, le terme service correspond à celui de travail utile). Il faut concevoir et exécuter une application pour fournir un service sur cet équipement.

L'application dédie, le temps de l'exécution, l'équipement informatique à un service spécifique. Pour cela, elle commande les ressources matérielles qui composent l'équipement, *e.g.*, modifier l'état des portes logiques du processeur, modifier les valeurs de la mémoire vive. Ces modifications permettent à l'équipement de fournir le service souhaité.

Sans application, l'équipement ne permet pas de fournir un service spécifique, et *vis versa*. Un service est donc une transformation d'une énergie par l'association d'un équipement et d'une application (cf. Figure 5.1). Cela nous permet donc de poser la Définition 6.

Définition 6 : Le processus

Un processus P est l'association d'un équipement E et d'une application A. Un processus est par la suite noté comme suit : $P_{\langle E,A \rangle}$.

Un service est un ensemble de critères à satisfaire. Soit l'ensemble des critères sont satisfaits et ainsi le service est rendu, soit ils ne le sont pas et le service n'est pas rendu. Ainsi, un service ne peut pas être rendu partiellement [19].

Pour atteindre les critères définissant le service, le processus convertit une certaine quantité d'énergie. Il existe donc une quantité d'énergie minimum ϵ_0 pour un processus donné afin que le service soit rendu. *A priori*, dépenser plus d'énergie pour fournir le même service

19. Nous ne traitons pas le cas d'un service rendu partiellement car cela revient à dégrader un service. Se pose alors la question de savoir si un service dégradé correspond au même service puisque l'ensemble des critères ne sont pas atteints.

ne l'empêche pas d'être fourni. Aussi, un service est similaire à une fonction marche de tel sorte que :

Équation 1 : Seuil de fourniture d'un service

$$S = P_{\langle E,A \rangle}(\epsilon) = \begin{cases} 0 & \text{if } \epsilon < \epsilon_0 \\ 1 & \text{if } \epsilon \geq \epsilon_0 \end{cases}$$

La quantité d'énergie minimum ϵ_0 à fournir pour que le service soit rendu dépend du processus $P_{\langle E,A \rangle}$. Mais un service n'est pas nécessairement rendu par un unique $P_{\langle E,A \rangle}$. Un service peut être satisfait par un autre processus $P_{\langle E,A \rangle}$ où E et/ou A sont différents du premier processus. Il est donc nécessaire de prendre quelques hypothèses sur ces combinaisons afin de simplifier le problème.

La première hypothèse concerne l'unicité de l'application, *i.e.*, il existe une seule et unique application capable de rendre le service souhaité dans l'environnement. La seconde permet de décorréler l'application de l'équipement au travers d'une couche d'abstraction.

Unicité de l'application

Le service est défini au travers d'un ensemble de critères à satisfaire. Cependant, l'application fournissant le service n'est pas nécessairement unique. En effet, il n'existe pas toujours une seule et unique manière de modifier la structure interne de l'équipement, *i.e.*, une seule et unique implémentation, afin de fournir ce service. Il en résulte que sur un équipement E, un service S peut être rendu par une application A_a ou A_b, nécessitant des quantités d'énergie différentes : $S = P_a(\epsilon_a) = P_b(\epsilon_b)$ avec $P_a = P_{\langle E,A_a \rangle}$ et $P_b = P_{\langle E,A_b \rangle}$.

Toutefois, pour simplifier le problème, nous mettons de côté le choix de l'application pour un service donné sur un équipement donné et prenons l'hypothèse suivante sur l'unicité de l'application sur un équipement :

Hypothèse 1 : Unicité de l'application sur un équipement

À un service donné, sur un équipement donné, correspond une seule et unique application.

La couche d'abstraction

Une autre manière de fournir un même service est de changer l'équipement E qui fournit le service. En effet, dès lors que l'ensemble des critères définissant un service sont satisfaits, il importe peu de savoir quel équipement fournit le service. Nous supposons que ce nouvel équipement est structurellement différent du premier de par ses ressources matérielles mais possède des ressources matérielles de même nature, *e.g.*, processeur, mémoire vive.

Pour fournir un même service sur un équipement différent, il faut tenir compte de ses spécificités matérielles. Ainsi, l'équipement étant structurellement différent du premier, l'application diffère afin de prendre en compte cette différence structurelle entre les deux équipements. Cela nécessite donc de disposer d'applications spécifiques à chaque équipement pour un même service afin de gérer l'hétérogénéité des ressources matérielles. Il y a donc un service S pouvant être rendu par un processus P_1 issu de la combinaison d'un équipement E_1 et d'une application A_1 ou d'un processus P_2 issu de la combinaison E_2 et A_2 : $S = P_1(\epsilon_1) = P_2(\epsilon_2)$ où $P_1 = P_{\langle E_1, A_1 \rangle}$ et $P_2 = P_{\langle E_2, A_2 \rangle}$.

Cependant, des mécanismes permettent d'exécuter une même application sur des équipements hétérogènes. Il s'agit de la couche d'abstraction (*e.g.*, systèmes d'exploitation, machines virtuelles). La couche d'abstraction permet de cacher les différences structurelles des équipements afin de pouvoir exécuter une seule et même application sur chacun des équipements. Cela entraîne que : $S = P_1(\epsilon_1) = P_2(\epsilon_2)$ où $P_1 = \langle E_1, A \rangle$ et $P_2 = \langle E_2, A \rangle$.

Toutefois, cette affirmation n'est pas vraie dans tous les cas. Par exemple, une application a besoin d'une caméra pour fonctionner mais l'équipement sur lequel elle souhaite tourner ne possède pas cette ressource. Il y a donc un problème physique lié au fait que l'équipement ne possède pas les ressources matériels pour fournir le service [20].

La couche d'abstraction ne gomme donc pas toutes les hétérogénéités des équipements. La majorité des équipements possèdent des types de ressources matérielles communs, *e.g.*, processeur, mémoire vive, mémoire morte. Ce sont ces ressources que la couche d'abstraction cache à l'application. Toutefois, des ressources plus spécialisées comme des caméras, des microphones ou des écrans peuvent ne pas être présents dans un équipement. Ainsi une application peut s'exécuter sur différents équipements dès lors que l'ensemble des ressources requises par l'application est présent. L' Hypothèse 1 peut être ainsi étendue :

Hypothèse 2 : Unicité de l'application dans l'environnement

À un service donné, sur des équipements différents mais ayant des types de ressources communs, correspond une seule et unique application.

Le service est fourni par différents équipements au travers d'une seule et unique application. La couche d'abstraction modifie la structure interne de l'équipement afin de fournir le même service. Cependant, étant structurellement différents, les équipements ne sont pas modifiés de la même manière. La couche d'abstraction cache l'hétérogénéité des équipements à l'application, mais doit tout de même s'adapter aux différentes ressources des équipements. Comme ces ressources diffèrent d'un équipement à l'autre, l'apport d'énergie pour fournir le service diffère également. Cela permet de déduire que :

20. Nous ne considérons pas l'émulation de ressources si elles n'existent pas sur l'équipement.

Hypothèse 3 : Différence énergétique pour un même service

À service identique, l'apport d'énergie diffère d'un équipement à l'autre. Il existe donc un équipement, parmi un ensemble d'équipements, qui nécessite un apport d'énergie moindre afin de fournir le service souhaité.

5.2.2 La mutualisation des ressources

Les équipements informatiques sont polyvalents. Un équipement fournit un service qu'il ne fournissait pas auparavant grâce à l'utilisation d'une application. De plus, il peut exécuter plusieurs applications en parallèle. Un équipement peut donc fournir plusieurs services en parallèle.

Toutefois, un équipement ne peut pas fournir une infinité de services en parallèle. Il est limité par ses ressources matérielles. Par exemple, supposons qu'une application utilise la souris de l'équipement pour fournir un service et qu'une autre application souhaite également l'utiliser. La ressource étant occupée, cette nouvelle application ne peut pas fournir son service. Nous prenons donc l'hypothèse que :

Hypothèse 4 : Un équipement fournit plusieurs services

Un équipement est susceptible de fournir plusieurs services dès lors qu'il possède les ressources matérielles nécessaires et suffisantes afin que l'application y soit déployée.

L'autre effet de la mutualisation des ressources est l'augmentation de l'efficience énergétique de l'équipement [BH07]. Cela est dû au fait que certaines ressources peuvent être utilisée simultanément entre plusieurs applications (*e.g.*, processeur, écran). Une ressource matérielle nécessite un apport minimal d'énergie pour fonctionner. Par la suite, sa consommation est fonction de son utilisation. Ainsi, plus la ressource est utilisée, plus le coût énergétique du service est faible.

5.2.3 Maximiser l'efficience énergétique d'un ensemble d'équipements

La Définition 5 définit les objectifs à atteindre afin de considérer un travail comme utile. Nous avons vu qu'il existe plusieurs manières de fournir un service. Toutefois, l'Hypothèse 1 et l'Hypothèse 2 laissent de côté la recherche d'une solution au travers de la manière dont le service est fournit, *i.e.*, son application.

Nous nous basons donc sur l'Hypothèse 3 qui stipule qu'un même service est fourni par différents équipements et que l'un d'eux le fournit pour un coût énergétique moindre. Cette

hypothèse permet de proposer une approche basée sur l'identification de l'équipement le plus adapté énergétiquement pour fournir un service à un instant donné.

Nous nous appuyons également sur l'Hypothèse 4 qui stipule qu'un équipement peut fournir plusieurs services à la fois. En réunissant le maximum de services sur un équipement, l'efficience énergétique de chaque service est améliorée.

Nous définissons donc l'objectif à atteindre pour maximiser l'efficience énergétique : il s'agit de trouver le sous-ensemble de l'ensemble des équipements qui consomme le moins d'énergie pour fournir l'ensemble des services souhaités. Les équipements ne fournissant pas de service sont alors placés dans un état de basse consommation pour réduire autant que possible la consommation énergétique.

5.3 La maison numérique

La maison numérique est fait d'un ensemble d'équipements informatiques, *e.g.*, ordinateurs, set-top box, téléphones intelligents. La polyvalence des ces équipements leur permet de fournir un ensemble de services, *e.g.*, regarder un film, écouter de la musique, téléphoner.

Avec les usages modernes, cet ensemble de services est amené à s'exécuter en partie en dehors de la maison numérique. Aussi, des parties des services peuvent s'exécuter dans des centres de données, *e.g.*, stockage, traitement des données. Nous fixons donc les limites de notre étude au travers de l'hypothèse suivante :

Hypothèse 5 : Environnement autonome

La maison numérique est un environnement informatique limité en équipements et auto-suffisant en ressources.

Il est limité car à un instant donné, l'ensemble des équipements dont il est constitué est fini, *i.e.*, à aucun moment, nous ne considérons des équipements présents en dehors de la maison numérique. Aussi, les solutions visant à migrer les services dans les centres de traitement de données ne sont pas considérées. Il est auto-suffisant car l'ensemble des équipements suffit à fournir l'ensemble des services souhaités.

L'Hypothèse 5 restreint l'étude de l'efficience énergétique à l'ensemble des équipements informatiques et des services de la maison numérique. Toutefois, elle nous permet de considérer l'ensemble du cycle de vie d'un service, depuis son état sous forme d'énergie jusqu'à sa transformation par l'équipement.

Mais, il faut également prendre en compte les propriétés intrinsèques à la maison numérique. Ces propriétés sont décrites dans la suite de cette section : hétérogénéité, volatilité, répartition, ouverture aux tiers et qualité de service.

5.3.1 Hétérogénéité

L'hétérogénéité constitue une des propriétés fondamentales de la maison numérique et se retrouve au travers des équipements qui la composent. Elle se caractérise par des ressources matérielles différentes d'un équipement à l'autre et en différentes quantités. C'est pour cela que nous considérons qu'il existe une couche d'abstraction qui est présente sur chaque équipement (cf. Hypothèse 2).

L'hétérogénéité de la maison numérique se retrouve également dans les services qui sont délivrés à l'utilisateur. Cette hétérogénéité se caractérise au travers de besoins différents d'un utilisateur à l'autre ou d'une période de la journée à l'autre.

Enfin, nous avons vu au travers de la Section 5.2 qu'un service est initialement une transformation d'énergie. Nous avons également vu que suivant l'équipement qui fournit le service, l'apport en énergie est différent. Les deux hétérogénéités évoquées plus haut en impliquent donc une troisième, invisible pour l'utilisateur final : l'hétérogénéité de consommation d'énergie.

Cependant, cette troisième hétérogénéité est également un atout pour maximiser l'efficience énergétique. En effet, si l'hétérogénéité des consommations énergétiques correspond à un ensemble de solutions dans lequel un ensemble de services s'exécute sur un ensemble d'équipements, alors il existe une solution qui est plus efficiente énergétiquement que les autres.

5.3.2 Volatilité

Dans la maison numérique, une solution à un instant n'est pas valable *ad vitam æternam*. En effet, la maison numérique est également marquée par sa volatilité : les équipements arrivent et sortent du réseau domestique de manière imprévisible. Les ressources matérielles à disposition changent au cours du temps. Enfin, les besoins des utilisateurs varient au cours du temps. Les services sont donc fournis provisoirement et peuvent à tout moment s'arrêter ou démarrer. Ainsi, une solution efficiente énergétiquement ne l'est plus forcément dès lors que l'un des ensembles, d'équipements ou d'applications, change.

5.3.3 Répartition

La répartition des services est également une propriété forte de la maison numérique. Il n'est pas rare qu'un service nécessite plusieurs équipements afin de fournir le service final souhaité. L'un des exemples les plus typique est celui du partage de contenu entre un serveur de stockage en réseau et l'écran de télévision. Le service final rendu à l'utilisateur est celui du visionnage d'un film sur la télévision. Deux équipements sont impliqués : le serveur de stockage en réseau qui contient le film et qui le transmet à la télévision pour qu'elle le décode et l'affiche. Le service de visionnage de film considère donc deux équipements et s'exécute de manière répartie.

5.3.4 Ouverture aux tiers

L'hétérogénéité cache également une autre propriété qui est plus difficile à appréhender : l'ouverture aux tiers. La maison numérique est hétérogène en équipements et en services. Cependant, ces équipements ou ces services ne sont pas issus du même équipementier ou du même fournisseur de services. Comme il s'agit d'industriels, chacun cherche donc à imposer ses produits au risque d'éclater la maison numérique, *i.e.,* protocoles de communication divers et pas toujours correctement implémentés, applications non portables d'un équipement à un autre ou encore applications perturbant d'autres applications.

Toutefois, nous ne considérons pas cette propriété dans la suite de cette étude. Elle nécessite notamment d'ajouter des éléments d'isolation pour des questions de sécurité ou des concertations autour de normes avec d'autres industriels afin que cet ensemble fonctionne sans accrocs. Surtout, elle contraint le présent travail à être beaucoup moins intrusif ou d'abandonner les équipements tiers au profit d'un ensemble d'équipements contrôlé par un unique industriel.

5.3.5 Qualité de service

La qualité de service est la propriété de la maison numérique qui est la plus visible par l'utilisateur final. C'est celle qui détermine si l'utilisateur accepte de consommer un service. La qualité de service est définie comme suit par l'ITU [21].

Définition 7 : Qualité de service

Ensemble des caractéristiques d'un service qui lui permet de satisfaire aux besoins explicites et aux besoins implicites de l'utilisateur du service [ITU08].

Les besoins explicites correspondent aux critères énoncés dans la spécification du service, *e.g.,* résolution de l'image, débit minimal, bouton en haut à gauche. Ces critères doivent être satisfait lors de la consommation du service pour considérer que la qualité de service est atteinte. Mais il existe également des besoins implicites, non définis dans la spécification du service.

Par exemple, un service propose une interface avec l'utilisateur. Ce service requiert que l'utilisateur soit devant cet interface. En l'absence d'avoir spécifié ce qu'il faut faire dans ce cas là, le système tend à ne pas changer. D'un point de vue énergétique, ce type de besoin doit passer dans la catégorie des besoins explicites. En effet, l'interface peut être arrêtée si l'utilisateur n'est pas devant l'écran, réduisant la consommation d'énergie.

21. International Telecommunication Union

5.4 Modélisation de la maison numérique

La modélisation de la maison numérique doit prendre en compte ses propriétés d'hétérogénéité et de volatilité. La modélisation est proposée au travers de la description de l'état de l'environnement à instant donné t, *i.e.,* le plan de répartition, et d'un ensemble d'actions qui modifient l'état dans lequel se trouve l'environnement.

5.4.1 Le plan de répartition

La maison numérique est caractérisée par un ensemble d'équipements et un ensemble d'applications. Les applications utilisent les ressources des équipements pour fournir un service au client. À un instant donné, il est possible de dire sur quel équipement une application s'exécute [22]. Ce constat définit l'état de l'environnement à un instant donné.

Cet état est représenté au travers d'une matrice $D^t = (d^t_{ea})_{1 \leq e \leq |E^t|, 1 \leq a \leq |A^t|}$ où E^t est l'ensemble non ordonné des équipements disponibles dans la maison à l'instant t et où A^t correspond à l'ensemble non ordonné des applications à l'instant t. Nous nommons cette matrice : le plan de répartition.

Le plan de répartition est rempli de la manière suivante, où A^t_e représente l'ensemble des applications qui sont hébergées sur l'équipement e à l'instant t.

Équation 2 : Plan de répartition

$$d^t_{ea} = \begin{cases} 1 & \text{si } a \in A^t_e \\ 0 & \text{sinon} \end{cases}$$

En ne considérant que ce plan de répartition, le nombre de solutions, *i.e.,* nombre de plan de répartitions, possibles s'élève à $|E^t|^{|A^t|}$. Par exemple, dans le cas d'une matrice de 3 équipements et 5 applications, il y a $3^5 = 243$ solutions possibles. Ainsi, même si la dimension de l'ensemble A^t augmente peu (quelques éléments), le nombre de solutions possibles augmente exponentiellement. Nous verrons dans la Section 5.5 comment atténuer cet effet.

5.4.2 Les actions

Le plan de répartition représente la répartition des applications sur les équipements à un instant t donné. Cependant, ce plan est amené à être modifié de manière imprévisible en raison de la volatilité de l'environnement. La volatilité modifie l'état courant de l'environnement aux travers d'actions connues mais imprévisibles, *i.e.,* des événements significatifs. Suite à un événement significatif, l'environnement se trouve dans un nouvel état. Toutefois, ce nouvel état n'est pas nécessairement efficient énergétiquement.

22. Nous considérons dans un premier temps qu'une application n'est pas répartie

Pour y remédier, il faut définir des actions que le système peut utiliser afin de tendre vers un nouvel état efficient énergétiquement. Aussi, nous prenons l'hypothèse que les applications peuvent migrer d'un équipement à l'autre. Ces actions de migration sont modélisées au travers du plan de déploiement. Elles modifient le plan de répartition pour atteindre l'objectif d'efficience énergétique de la maison numérique.

Nous distinguons deux types d'actions modifiant le plan de répartition : les actions issues de la volatilité de l'environnement et les actions mise à la disposition du système afin de changer l'état de l'environnement

La volatilité

Les événements significatifs modifient les dimensions du plan de répartition de manière imprévisible, *i.e.*, apparition ou disparition d'un équipement ou d'une application. Ces événements modifient les ensembles E^t et A^t correspondants. Si un des ensembles change, la dimension du plan de répartition et son contenu changent en conséquence. Ainsi le plan de répartition à l'instant t n'a pas la même dimension qu'à l'instant $t+1$, après l'événement. Ces changements sont modélisés de la manière suivante :

Équation 3 : Événements significatifs

$$\begin{array}{ll} \text{L'équipement } e \text{ apparaît:} & E^{t+1} = \{e\} \cup E^t \\ \text{L'équipement } e \text{ disparaît:} & E^{t+1} = \{e\} \setminus E^t \\ \text{L'application } a \text{ apparaît:} & A^{t+1} = \{a\} \cup A^t \\ \text{L'application } a \text{ disparaît:} & A^{t+1} = \{a\} \setminus A^t \end{array}$$

La volatilité impose donc de passer d'un plan de répartition de dimension $(|E^t|, |A^t|)$ à un plan de répartition de dimension différente, qui rajoute ou enlève un élément à l'un des ensembles. Nous prenons l'hypothèse que deux événements survenant à l'exact même instant est impossible, aussi les dimensions des ensembles sont modifiées uniquement d'une unité à la fois.

Le plan de déploiement

Définir des actions modifiant le plan de répartition courant participe à atteindre l'efficience énergétique. Modifier la répartition des applications sur les équipements se fait entre deux plans de répartition de même dimension. Seule le placement des applications sur les équipements change. Pour cela, le plan de déploiement est défini. Le plan de déploiement fait passer d'un plan de répartition d'une dimension $|E^t| \times |A^t|$ à un autre plan de répartition de même dimension. Un plan de déploiement est représenté comme suit :

Équation 4 : Plan de déploiement

$$\Delta_{ea} = d_{ea}^{t+1} - d_{ea}^{t} = \begin{cases} 1 & \text{si } a \text{ doit être déployé sur } e \\ -1 & \text{si } a \text{ doit être retiré de } e \\ 0 & \text{si } a \text{ ne bouge pas ou n'est pas présente sur } e \end{cases}$$

A noter qu'en ne considérant que ces actions, une application peut migrer sur l'ensemble des équipements. Cette hypothèse est relativement loin de la réalité puisque toutes les applications ne peuvent pas se déplacer sur tous les équipements de la maison (cf. Hypothèse 2). Pour limiter les déplacements des applications et garantir que les services fournis sont toujours les mêmes, il faut caractériser les applications et notamment leurs besoins fonctionnels.

5.4.3 Les contraintes de déploiement

Le déplacement des applications d'un équipement à un autre doit prendre en considération l'hétérogénéité de l'environnement. Une application ne peut pas se déplacer n'importe où sous prétexte de réduire la consommation d'énergie. Pour atteindre l'efficience énergétique, il faut être capable de fournir le même service avec moins d'énergie.

Pour garantir que le service fournit est bien celui qui est attendu par l'utilisateur, il faut caractériser, à sa conception, les besoins de l'application qui le délivre. Et après migration, pour s'assurer que l'application fournit le même service sur le nouvel équipement, ses besoins doivent être satisfait. La satisfaction de ces besoins permet de s'assurer qu'une fois déplacée, l'application fournit toujours le même service.

Cette caractérisation des besoins est faite au travers de « contraintes de déploiement ». Ces contraintes sont définies lors de la conception de l'application. Elles garantissent que le service fourni est le même, quel que soit l'équipement qui exécute l'application, dès lors qu'elles sont satisfaites. Une contrainte est constituée d'une variable, *e.g.*, `Location`, et d'une valeur, `kitchen` et s'écrit « variable = valeur », *e.g.*, `Location = kitchen`.

Nous avons vu dans la Section 5.2.1 que nous considérons des besoins clairement énoncés. Les besoins permettent donc de définir nos contraintes de déploiement. Il n'existe pas de liste exhaustive contenant l'ensemble des contraintes de déploiement qu'une application pourrait nécessiter. Toutefois, nous en énumérons certaines nous paraissant fondamentales.

Ressources et Quantité de ressources matérielles

Les ressources matérielles correspondent aux outils utilisés par une application pour fournir un service. Beaucoup d'applications se satisfont d'un simple processeur et d'une quantité raisonnable de mémoire vive. Ces ressources se retrouvent sur l'ensemble des équipements de la maison numérique. Toutefois, il existe des applications qui requièrent des

ressources matérielles plus spécifiques telles que des microphones ou des enceintes. Ces ressources ne se trouvent pas sur tous les équipements de la maison domestique, réduisant l'ensemble des équipements susceptibles d'accueillir les applications les nécessitant.

Il faut donc que l'application décrive ses besoins afin de déterminer l'équipement qui l'hébergera. Cette description s'effectue au travers de conditions booléennes telles que `WebcamPresence = true`. L'équipement de son côté doit également fournir sa description et spécifier si il possède ce qui est requis par l'application : `WebcamPresence = true`. Si l'équipement ne décrit pas une ressource, c'est qu'il ne la possède pas. Cela revient à avoir la valeur `false` par défaut.

En plus des ressources matérielles, les applications peuvent également en demander une certaine quantité. Pour vérifier que l'application a suffisamment de ressources pour fournir son service une fois déployée, il faut mettre en relation les besoins de l'application et ce que l'équipement possède. Par exemple, si une application requiert `RAM = 20 Mo` alors il faut que l'équipement qui l'héberge ait `RAMPresence = true` et qu'il lui en reste suffisamment : `RAM` $\geq$ `20 Mo`.

De manière générale, les ressources et les quantités de ressources requises par une application sont décrites au travers d'une table ressource/quantité (cf. Équation 5 où r correspond à un type de ressource, Q à la quantité associée et n au nombre total de ressources). Afin de s'assurer que les besoins de l'application seront satisfaits, les quantités données sont des maximums même si l'application ne les utilise pas dans leur globalité.

Équation 5 : Ressources et quantités requises pour une application

$$R_a = \begin{pmatrix} r^1 = Q_{r^1} \\ r^2 = Q_{r^2} \\ \dots \\ r^n = Q_{r^n} \end{pmatrix}_a$$

De la même manière, les ressources et les quantités de ressources disponibles sur un équipement sont décrites au travers de l'Équation 6. Si la ressource n'est pas mentionnée, cela revient à avoir `"Resource"Presence = false`.

Équation 6 : Ressources et quantités disponibles sur un équipement

$$R_e = \begin{pmatrix} r^1 = Q_{r^1} \\ r^2 = Q_{r^2} \\ \dots \\ r^n = Q_{r^n} \end{pmatrix}_e$$

Pour savoir si une application a peut être déployée sur un équipement e, il suffit alors de faire la soustraction : $R_e - R_a$. Si une des valeurs est négative alors l'application ne peut pas être déployée sur l'équipement. Et si plusieurs applications doivent être déployées sur

un même équipement, il faut alors additionner l'ensemble des ressources de toutes ces applications et procéder à la soustraction : $R_e - \sum_{a \in A^t} R_a$.

La définition des ressources ne tient pas compte de leur évolution tant du côté de l'application que de l'équipement. Ainsi, quel que soit le comportement de l'application, elle a toujours besoin des mêmes ressources et toujours dans les mêmes quantités. Pour l'équipement, nous supposons que ses ressources maximales ne varient pas au cours du temps, *e.g.*, défaillance d'un composant matériel.

La complexité de cette contrainte est similaire au problème du sac à dos (ou *bin packing*), où en considérant les ressources requises par une application, l'algorithme doit maximiser l'utilisation des ressources disponibles sur chaque équipement. Cette contrainte est connue comme étant un problème NP-difficile [CJGJ96]. Voir l'Annexe A pour plus de précisions sur le *bin packing* et les méthodes de résolution.

Présence de l'utilisateur

Dans certains cas, la migration n'est pas envisageable du fait que l'utilisateur interagit directement avec l'application. Déplacer l'application sur un équipement moins énergivore, à travail utile équivalent, est alors impossible : l'utilisateur fait parti des contraintes de déploiement.

Par exemple, l'utilisateur utilise un navigateur web sur un ordinateur. Pour réduire l'énergie, le navigateur peut être déplacé sur un téléphone intelligent. Dans le cas présent, l'utilisateur est ennuyé et ne souhaite pas que le service soit rendu par un autre équipement. Le navigateur peut s'exécuter sur un autre équipement, le travail fournit est le même mais n'est pas considéré comme utile car il ne profite à personne.

Dans d'autres cas, cette contrainte apporte des économies d'énergie puisqu'elle permet de mieux cerner le besoin de l'utilisateur à un instant donné. Par exemple, un utilisateur regarde le journal télévisé dans son salon. À un moment donné, il sort de la pièce et va préparer le repas dans la cuisine qui est attenante au salon. Depuis la cuisine, il peut entendre le journal télévisé mais ne peut pas le voir. Le service de visionnage est-il alors considéré comme utile ou non ?

En décomposant ce service en deux, un service audio et un service vidéo, alors le service audio est un travail utile puisqu'il est consommé par l'utilisateur dans la cuisine. Toutefois, le service vidéo n'est pas considéré comme utile car il ne satisfait à aucun besoin de l'utilisateur qui ne peut pas le consommer depuis la cuisine. Dans ce cas là, le service audio doit continuer tandis que le service vidéo doit être arrêté car l'utilisateur n'est pas devant l'écran.

Cette contrainte doit être exprimée dans la spécification d'un service. Elle est modélisée au travers d'une condition booléenne dans la description de l'application : `UserPresence = true`. Elle doit par conséquent être confrontée avec la même description donnée par l'équipement, à savoir `UserPresence = true` ou `UserPresence = false`.

La complexité de cette contrainte est en $o(1)$ parce qu'elle ne nécessite pas particulièrement de ressources de calcul pour être satisfaite. Soit l'utilisateur utilise l'équipement, soit il ne l'utilise pas.

5.4.4 La consommation électrique

La puissance correspond à une quantité d'énergie utilisée par unité de temps. En considérant le système informatique dans un état donné à un instant donné, il est possible de dire quel équipement consomme quelle puissance. Ainsi, l'énergie ne permet pas de décrire le plan de répartition puisqu'il s'agit d'une vue de notre système à un instant donné. La puissance est donc utilisée pour déterminer la consommation du plan de répartition.

L'énergie se calcule sur une durée. Aussi pour réduire la consommation d'énergie, il faut également prendre en compte le paramètre temporel dans le choix d'un plan de répartition plus efficient énergétiquement. Le gain d'énergie est calculable dès lors que la période entre deux événements significatifs est connue. Toutefois, comme l'environnement étudié est très volatile, il est difficile de prédire quand un événement significatif va survenir. Il est donc difficile de calculer quel est le gain énergétique d'un plan de répartition puisque nous ne savons pas quand un nouvel événement va nécessiter de modifier ce plan de répartition. Il faut donc se baser sur la puissance pour déterminer un nouveau plan de répartition.

L'hétérogénéité des équipements entraîne également différentes manières de gérer l'énergie de l'équipement [Jos12, MC09]. Afin de modéliser la consommation électrique de la maison numérique, nous considérons uniquement deux états Φ_e : un état *actif* on, dans lequel l'équipement fournit des services, et un état de *basse consommation* lp, dans lequel il ne peut pas fournir de services et ne peut pas communiquer avec d'autres équipements. À tout moment, un équipement est soit à l'état actif, soit à l'état de basse consommation.

Dans notre modèle, nous considérons seulement la puissance consommée par des équipements, P_e. Cette consommation peut prendre deux valeurs qui dépendent de l'état de l'équipement : (a) P_e^{lp}, une consommation constante reflétant la puissance utilisée par l'équipement dans l'état de basse consommation, et (b) $P_e^{on}(R_e^t)$, une consommation dépendant de la charge de l'équipement dans l'état actif à un instant t [ERKR06].

Toutefois, à cause des hétérogénéité de ressources matérielles des équipements, chaque ressource a son propre modèle de consommation d'énergie. Ces modèles sont complexes afin de représenter fidèlement la consommation de chaque ressource matérielle [MAC+11]. Intégrer l'ensemble des modèles de consommation des ressources de la maison numérique requiert de tous les connaître. Or les équipementiers ne fournissent pas ces informations.

Aussi, la modélisation de la consommation d'un équipement est abstraite des modèles de consommation des ressources qui le compose. Cette modélisation est faite par l'Équation 7. Cette équation distingue deux consommations : la première est constante lorsque l'équipement est dans un état de basse consommation et la seconde est fonction de la charge des ressources matérielles d'un équipement lorsqu'il est dans l'état actif.

Équation 7 : Modélisation de la consommation d'un équipement

$$P_e = \begin{cases} P_e^{lp} & \text{si } \Phi_e = lp \\ P_e^{on}(R_e^t) & \text{si } \Phi_e = on \end{cases}$$

Comme les quantités de ressources matérielles sont des maximums, ce modèle donne toujours une valeur haute de la puissance consommée. Et comme les quantités ne varient pas au cours du temps, la puissance consommée entre deux événements significatifs est constante. Aussi, en réduisant la puissance consommée, l'énergie dépensée est réduite.

Pour connaître la consommation énergétique de l'ensemble de l'environnement P_s^t, les consommations de chacun des équipements qui le compose à un instant t sont additionnées (cf. Équation 8). Cette équation définit l'ensemble des consommations d'énergie potentiels du plan de répartition : c'est une fonction d'utilité. La fonction d'utilité prend en compte l'activité des équipements δ^t : δ^t est égal à 1 si l'équipement héberge des applications, 0 sinon.

Équation 8 : Fonction d'utilité

$$P_s^t = \sum_{e \in E^t} (\delta_e^t \times P_e^{on}(R_e^t) + (1 - \delta_e^t) \times P_e^{lp})$$

$$\text{avec} \qquad \delta_e^t = 1 - \prod_{a \in A^t} (1 - d_{ea}^t)$$

5.5 Utilisation des composants

Jusqu'à présent, nous avons considéré que les applications sont monolithiques, qu'elles forment un tout non décomposable. Ainsi, un service est une transformation de l'énergie électrique opérée par un processus, *i.e.*, l'association d'un équipement et d'une application. Ce service satisfait le besoin d'un utilisateur.

Cependant, plus le déploiement d'une application est contraignant, moins l'application est mobile puisque peu d'équipements peuvent satisfaire à l'ensemble des contraintes (cf. Définition 8). Un déploiement est considéré comme de plus en plus contraignant dès lors que l'ensemble des équipements sur lequel l'application peut être déployée se réduit. Cela passe donc soit par l'ajout de nouvelles contraintes, soit la requalification de certaines contraintes avec des critères plus stricts, réduisant les équipements candidats à son hébergement.

Définition 8 : La mobilité

La mobilité d'une application est sa capacité à être déployée, à un instant donné, sur différents équipements, en considérant ses contraintes de déploiement.

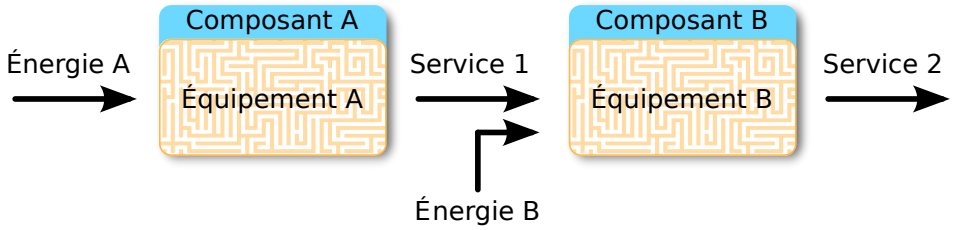

FIGURE 5.2 – **Transformation de l'énergie en service final via plusieurs processus $\langle E, C \rangle$.** Le service final correspond ici au service 2 puisqu'il s'agit du dernier service sur la chaîne de services. L'application correspond à l'ensemble des composants participant à la fourniture du service final, à savoir les composants A et B.

Pour augmenter la mobilité d'une application, nous prenons l'hypothèse que les applications sont réparties. Aussi, le service issu d'un processus peut satisfaire le besoin d'un autre processus. Sans ce service, ce dernier processus n'est alors pas capable de fournir, à son tour, son service. Comme l'application est alors décomposable et définie au travers d'un ensemble de composants C, elle peut être répartie sur différents équipements E afin de fournir son service final (cf. Figure 5.2). Dès lors, nous ne parlons plus de couples équipement/application, mais de couples équipement/composant. Un service final est donc défini au travers de la Définition 9.

Définition 9 : Service final

Un service final est fourni par un ensemble de processus $\langle E, C \rangle$. La taille de cet ensemble est de 1 ou supérieur.

5.5.1 La mobilité des applications

Si la mobilité d'une application décroit avec ses contraintes, supprimer ou assouplir des contraintes lui permet d'être déployée sur un ensemble plus vaste d'équipements. Toutefois, afin de s'assurer que le service est satisfait, il ne faut pas supprimer ou assouplir des contraintes de l'application.

Pour avoir un ensemble de solutions de déploiement plus large, nous pouvons considérer que les applications sont réparties et constituées de composants logiciels. Les composants logiciels permettent de répartir les fonctionnalités de l'application à des éléments dédiés et plus mobiles dans l'environnement. Les composants sont plus mobiles car les contraintes de l'application sont distribuées à ses composants et chaque composant se retrouve donc avec moins de contraintes de déploiement que l'application dans sa globalité.

L'ensemble des solutions de déploiement pour une application répartie S_{da} est plus grand que l'ensemble des solutions de déploiement d'une application monolithique S_{ma}.

En effet, une solution dans S_{ma} est forcément une solution qui satisfait les contraintes de tous ses composants. Elle satisfait également la contrainte implicite d'avoir tous les composants sur un seul et unique équipement. S_{ma} correspond donc à l'intersection de l'ensemble des solutions de déploiements pour chacun des composants de l'application : $S_{ma} = \bigcap_{i=0}^{n} S_{c_i}$.

Les applications réparties supprime la contrainte, implicite, de déploiement de l'ensemble des composants sur un unique équipement. Les composants sont plus mobiles, leur permettant de se déployer indépendamment les uns des autres. Cela permet d'augmenter le nombre de solutions car cet ensemble S_{da}, plus large, correspond à l'union de l'ensemble des solutions de déploiement pour chacun des composants de l'application : $S_{da} = \bigcup_{i=0}^{n} S_{c_i}$.

Finalement, l'ensemble des solutions possibles S_{da} est plus grand que S_{ma} : $|S_{ma}| < |S_{da}|$. Ainsi, toute solution dans S_{ma} existe aussi dans S_{da}. En effet, la solution dans laquelle tous les composants de l'application sont déployés sur un unique équipement revient à dire que l'application, dans sa forme monolithique, est déployée sur un unique équipement. Cela entraîne donc $S_{ma} \subset S_{da}$.

L'effet de bord de cette approche est qu'en augmentant le nombre de solutions possible, le temps nécessaire à la recherche de la solution optimale, *i.e.*, le temps d'optimisation, s'accroît. Par exemple, si il y a 5 applications et que chacune d'elle est constituée de 3 composants, il y a donc $5 \times 3 = 15$ composants à la place de 3 applications dans le plan de répartition. Si la répartition s'effectue sur 3 équipements, le nombre de répartitions possibles passe de $3^5 = 243$ à $3^{15} = 14348907$. Cette augmentation drastique du nombre de solutions nécessite de passer plus de temps dans la recherche du plan de répartition le plus efficient énergétiquement. Et invariablement, chercher pendant plus longtemps une solution nécessite d'avantage d'énergie.

Pour réduire le temps d'optimisation, il faut réduire le nombre d'éléments considérés. Cette réduction ne peut pas se faire en réduisant le nombre d'équipements, au risque de passer à côté de la solution optimale. Il faut donc réduire l'ensemble des composants considérés. Ce regroupement se fait au travers d'une unité de réflexion : la « grappe de composants » et est utilisé uniquement au cours de la phase d'optimisation. Ce regroupement n'est pas considéré lors de la mise en application du plan de répartition.

Cependant, tous les regroupements ne sont pas bons à prendre. Il ne faut pas grouper les composants n'importe comment, au risque de perdre en mobilité et donc de réduire l'ensemble des solutions. Par exemple, si l'ensemble des composants d'une application sont regroupés au sein d'une même grappe, cela revient à considérer l'application comme monolithique. Nous l'avons vu plus tôt, S_{ma} limite le choix des solutions, contrairement à S_{da}.

Il faut donc grouper les composants sans que la grappe de composants ne limite leur déplacement à travers l'environnement. En d'autres termes, il faut grouper les composants de tel sorte que le nouvel ensemble de plans de répartition possibles S_{ga} soit égal à S_{da}. Par exemple, si les composants ayant les mêmes contraintes sont regroupés, la grappe possède également la même contrainte. Pour autant, la mobilité des composants n'est pas réduite

puisque chaque composant pris individuellement requiert cet contrainte pour finalement être déployé au même endroit.

La création d'une grappe rajoute tout de même une contrainte aux composants. Un composant dans une grappe est lié aux autres composants de cette grappe. Ils doivent donc être déployés sur le même équipement. Ainsi $S_{ga} \subset S_{da}$. Toutefois, S_{ga} demeure plus grand que S_{ma} puisque le nombre de contraintes est plus faible sur une grappe que sur une application.

Aussi, sur l'ensemble des composants présents dans l'environnement, toutes applications confondues, il existe des composants qui peuvent être déployés sur les mêmes équipements. Par exemple, prenons des composants ayant la contrainte `UserPresence = true`, qui nécessitent d'être déployés sur l'équipement actuellement utilisé par l'utilisateur. Si plusieurs composants possèdent cette contrainte, le système n'aura d'autres choix que de les déployer sur le même équipement. Pour lui éviter de parvenir à la même conclusion pour plusieurs composants, ils sont groupés sous une grappe qui possède cette contrainte.

Toutefois, grouper des composants qui ont les mêmes contraintes n'est pas possible dans tous les cas. Par exemple, la contrainte commune entre deux composants est `RAM = 20 Mo`. Séparément, chaque composant doit être placé sur un équipement où 20 Mo de `RAM` est disponible. Si les deux composants sont déployés sur un même équipement, alors il faut 40 Mo de `RAM`. Donc en les regroupant, la contrainte de la grappe est de `RAM = 40 Mo`. Comme la valeur est plus élevée, il est plus difficile de placer la grappe que les deux composants pris séparément. Ces deux exemples permettent de définir la grappe de composants :

Définition 10 : La grappe de composants

La grappe est un ensemble de un ou plusieurs composants ayant les mêmes contraintes de déploiement et qui conserve la mobilité des composants qui le constituent. La grappe est considérée uniquement lors de la recherche d'une solution.

Ces deux exemples permettent de conclure que les politiques de groupement ne sont pas les mêmes suivant les contraintes. Cela amène donc à considérer deux catégories de contraintes : les contraintes « à valeurs quantitatives » et les contraintes « à valeurs énumérées ». Dans la description de ces contraintes, nous considérons toujours des ensembles ordonnés par inclusion.

Les contraintes à valeurs quantitatives. Les contraintes à valeurs quantitatives sont caractérisées par des quantités. Par exemple, un composant C_1 peut avoir une contrainte sur la quantité de mémoire vive nécessaire à son fonctionnement, *e.g.*, `RAM = 20 Mo`. Si ce composant est groupé avec un autre composant C_2 ayant le même type de contrainte avec une valeur quelconque, *e.g.*, `RAM = 15 Mo`, la contrainte de la grappe en résultant est la somme des deux contraintes, *i.e.*, `RAM = 35 Mo`. Il est plus difficile de placer une grappe ayant une contrainte de `RAM = 35 Mo` que de placer deux grappes ayant des contraintes de `RAM =`

`20 Mo` et `RAM = 15 Mo`. La contrainte sur la grappe est plus contraignante et donc l'ensemble des solutions pour cette grappe S_{C_0}, *i.e.*, l'ensemble des équipements sur lesquels la grappe peut être déployée, est : $S_{C_0} \subseteq S_{C_1} \subseteq S_{C_2}$.

Les contraintes à valeurs énumérées. Les contraintes à valeurs énumérées correspondent à des contraintes qui ne dépendent pas d'une quantité. Par exemple, un composant C_1 peut nécessiter la présence de l'utilisateur : `UserPresence = true` ou d'être déployé dans un lieu précis : `Location = kitchen`. Si ce composant est groupé avec un autre composant C_2 ayant la même contrainte, la grappe en résultant possède alors la même contrainte. L'ensemble des solutions de S_{Co} est donc : $S_{Co} = S_{C_1} = S_{C_2}$.

5.5.2 Décomposition et regroupement

Fort des ces deux types de contraintes, nous proposons maintenant une méthodologie afin de décomposer une application puis de regrouper les composants en fonction de leurs contraintes. La première phase est effectuée par l'architecte lors de la conception de son application. La deuxième phase est réalisée par le système amené à migrer les composants à l'exécution.

La décomposition

La phase de décomposition est effectuée par l'architecte grâce à sa connaissance de l'application. Cette phase augmente l'ensemble des plans de répartition possibles en décomposant les applications en composants. Chaque composant se voit alors attribuer son propre ensemble de contraintes. Pour augmenter la mobilité des composants, il faut donc que cet ensemble de contraintes, propre à chaque composant, soit le plus petit possible.

La décomposition suppose que l'application est définie au travers d'un ensemble de contraintes et d'un ensemble de composants. Premièrement, l'architecte isole et analyse les besoins de chaque composant, *e.g.*, présence de l'utilisateur, ressources matérielles. Ensuite, les contraintes de l'application sont distribuées aux composants, en prenant en compte leurs besoins. Une contrainte peut être distribuée à plusieurs composants et plusieurs contraintes peuvent être appliquées sur un même composant.

La répartition des contraintes diffère pour chacun des types de contraintes évoqués plus haut. La répartition d'une contrainte à valeur énumérée à un ensemble de composants consiste à dupliquer la contrainte puis à l'assigner à chaque composant. Par exemple, si plusieurs composants nécessitent la contrainte à valeur énumérée `UserPresence = true` alors cette contrainte est dupliquée et assignée à l'ensemble des composants la requérant.

La répartition d'une contrainte à valeur quantitative doit être divisée parmi les composants qui en ont besoin. Une fois divisée, différentes quantités de cette contrainte peuvent être réparties aux composants la nécessitant, mais la somme de chacune de ces valeurs ne doit pas excéder la valeur de départ. Par exemple, une application nécessite `RAM = 20 Mo`.

Pour distribuer cette contrainte à deux composants nécessitant également de la mémoire vive, la quantité 20 Mo doit être divisée entre les deux composants. Cette division n'est pas forcément équitable de tel sorte que l'un peut se retrouver avec 15 Mo et l'autre avec 5 Mo. La somme des deux nouvelles contraintes doit être égale à 20 Mo.

Le regroupement

Une fois que la phase de décomposition est achevée, chaque application est vue comme un ensemble de composants, chacun possédant son propre ensemble de contraintes. Maintenant, en remplaçant les applications par leurs composants dans le plan de répartition, il y a plus de solutions possibles à considérer.

Pour réduire l'ensemble des solutions considérées lors de la recherche de la solution optimale, nous utilisons les grappes (cf. Définition 10). Les grappes ne se limitent pas à chaque application mais peuvent être composées à partir de l'ensemble des composants des applications présentes dans l'environnement à un instant donné. La taille de la grappe est donc amenée à changer lors de l'apparition ou de la disparition de composants. De plus, la grappe est considérée uniquement lors de la phase d'optimisation. C'est un groupement de composants abstrait qui disparaît lors de la mise en application de la solution où chaque composant est déployé individuellement.

5.5.3 Les composants sans contraintes

Tous les composants ne nécessitent pas d'être déployés sous contraintes. Certains composants sont considérés comme « sans contraintes ». Ces composants sont toujours déployés sur des équipements qui fournissent un minimum de mémoire vive et de processeur. Ils nécessitent tellement peu de ressources que nous considérons qu'ils peuvent s'exécuter sur n'importe quel équipement sans spécifier de contraintes.

La phase d'optimisation ne les considère pas comme des composants ayant des contraintes. Parce que l'objectif est de minimiser les équipements actifs, et ainsi la consommation d'énergie, les composants sans contraintes sont déployés sur des équipements qui hébergent déjà des composants ayant des contraintes.

Mais s'il y a trop de composants sans contraintes sur un équipement, leur charge sur les ressources est non négligeable. Il faut donc les répartir sur l'ensemble des grappes considérées lors de la recherche d'une solution. En les répartissant, le regroupement s'assure que leur charge reste négligeable par rapport aux composants avec contraintes.

5.6 Conclusion

Ce chapitre définit un service comme étant une transformation d'une quantité d'énergie par un processus. En informatique, il existe plusieurs processus, *i.e.,* couples équipement/application, amenant à fournir un même service. Maximiser l'efficience énergétique

consiste donc à trouver le processus fournissant le service pour le plus faible coût énergétique.

Aussi, nous modélisons les différents processus courant, *i.e.*, le plan de répartition, ainsi que les actions amenant à considérer d'autres processus, *i.e.*, le plan de déploiement et les événements significatifs issus de la volatilité de la maison numérique. La définition d'une fonction d'utilité décrit la consommation énergétique de l'ensemble des états possibles, *i.e.*, un état est définit par un ensemble de processus.

Pour choisir l'état le plus efficient énergétiquement, nous prenons l'hypothèse qu'une application n'est pas liée à un équipement. Cette décorrélation entre l'application et l'équipement permet de décider des actions à mener pour atteindre cet état. Ces actions amènent à déplacer des applications d'un équipement à un autre.

Cependant, une application n'est pas totalement déliée d'un équipement. Le placement d'une application doit considérer ses propres spécificités et l'hétérogénéité de la maison numérique. L'utilisation de contraintes de déploiement caractérisent ces hétérogénéités et limite les déplacements d'une application.

Pour les rendre plus mobile, nous proposons l'utilisation d'applications réparties à base de composants logiciels. Ces applications accroissent le nombre d'équipements pouvant héberger une application, ou des parties de l'application. En augmentant le nombre de déploiement, l'efficience énergétique est également améliorée puisqu'il existe plus de solutions de déploiement. Parmi l'ensemble de ces solutions, l'une d'elle est plus efficiente énergétiquement que les autres.

L'utilisation des applications réparties à base de composants logiciels accroît le nombre de solutions et maximise davantage l'efficience énergétique. Le revers de la médaille est que cela demande plus de temps pour trouver la solution optimale. Aussi le regroupement des composants ayant les mêmes contraintes sous une grappe de composants diminue le temps d'optimisation du plan de répartition.

Le Chapitre 6 s'intéresse à la mise en œuvre de cette modélisation au travers d'une architecture spécifiquement conçue pour l'environnement domestique. Cette architecture se veut autonome et peu gourmande en énergie.

Chapitre 6

Architecture

La matière exotique est une substance transdimensionnelle découverte comme un produit dérivé de la recherche du Boson de Higgs au CERN. En théorie, la matière exotique est à la fois matière et énergie et existe simultanément dans plusieurs dimensions.

– P.A. Chapeau's, *Niantic Project*

Sommaire

6.1 Introduction . . . 88
6.2 Vue d'ensemble de l'architecture . . . 89
6.2.1 Système d'adaptation du placement des composants . . . 90
6.2.2 Système de contrôle des états énergétiques des équipements . . . 91
6.2.3 Interaction entre les systèmes . . . 92
6.2.4 Gestion de la dette énergétique . . . 93
6.3 Adaptation du placement des composants . . . 94
6.3.1 Description du coordinateur . . . 94
6.3.2 Description du gestionnaire . . . 98
6.4 Contrôle des états énergétiques des équipements . . . 99
6.4.1 Description du collecteur . . . 100
6.4.2 Description du contrôleur . . . 100
6.5 Gestion des pannes . . . 101
6.5.1 Les pannes et leur détection . . . 101
6.5.2 Mécanismes de rétablissement . . . 103
6.6 Conclusion . . . 105

6.1 Introduction

Le modèle du chapitre précédent définit la manière dont la consommation d'énergie dans la maison numérique est réduite. Pour cela, nous prenons l'hypothèse qu'un service n'est pas nécessairement fournit par un seul équipement (cf. Hypothèse 2). La migration des applications d'un équipement à un autre minimise la consommation d'énergie à un instant donné. Les équipements inactifs sont ensuite placés en état de basse consommation.

Toutefois, la solution proposée a un coût énergétique qui doit être prit en compte lors de la recherche d'un nouveau plan de répartition et de sa mise en application. Nous nommons ce coût la « dette énergétique ». Cette dette est calculée par rapport au même environnement sans la mise en place de la solution. Pour minimiser cette dette, il faut que la mise en œuvre de la solution soit peu énergivore afin d'éviter que le système consomme plus d'énergie qu'il n'en fait gagner. Cette minimisation de la consommation d'énergie passe avant tout par la conception d'une solution automatique, *i.e.*, qui se considère elle même dans la maximisation de l'efficience énergétique (cf. Section 2.3).

Pour concevoir notre solution, nous séparons les objectifs fonctionnels et les attribuons à deux systèmes indépendants : (a) un système répartissant les composants sur les équipements et (b) un système plaçant dans un état de basse consommation les équipements inactifs, *i.e.*, qui ne fournissent pas de services.

Ces systèmes proposent de manière autonome un plan de répartition maximisant l'efficience énergétique de la maison numérique et le mettant en œuvre. Pour cela, ils considèrent la volatilité de cet environnement en réagissant aux événements significatifs, *i.e.*, apparition ou disparition d'applications ou d'équipements. Puis ils définissent le plan de répartition optimisé, *i.e.*, le plan de répartition qui maximise l'efficience énergétique, qui conserve la qualité de service avant de l'appliquer. Aussi notre solution considère les objectifs suivants, par ordre croissant de priorité :

① Conserver la qualité de service;

② Réduire globalement la consommation d'énergie, *i.e.*, l'ensemble des équipements de la maison numérique;

③ Réduire localement la consommation d'énergie, *i.e.*, les équipements isolés.

Pour réduire la dette énergétique de notre solution, les systèmes sont conçus pour se considérer eux mêmes dans la recherche du plan de répartition optimisé. Pour cela ils sont conçus comme des applications qui peuvent migrer d'un équipement à un autre. De plus, nos systèmes sont réactifs, *i.e.*, ils consomment de l'énergie seulement lorsqu'un événement significatif survient.

Ainsi notre solution est construite autour de trois critères :

- Considérer la volatilité de la maison numérique;

- Toujours proposer le plan de répartition le plus efficient énergétiquement qui conserve la qualité de service;
- Avoir un faible impact sur la consommation d'énergie au travers de systèmes efficient énergétiquement.

Structure du chapitre

Le chapitre s'organise comme suit : le Section 6.2 donne une vue d'ensemble des systèmes. La Section 6.3 décrit l'architecture du système permettant de modifier le placement des composants sur les équipements. La Section 6.4 détaille l'architecture du système permettant de contrôler l'état énergétique des équipements. La Section 6.5 détaille les défaillances qui peuvent survenir et les mécanismes qui permettent de rétablir le système. Enfin, la Section 6.6 conclut ce chapitre.

6.2 Vue d'ensemble de l'architecture

Les équipements actuels possèdent des mécanismes de réduction de la consommation d'énergie qui leur sont propres. Deux équipements ne possèdent pas nécessairement les mêmes mécanismes. Toutefois, ces mécanismes cherchent à adapter la consommation d'énergie au besoin. Au niveau matériel, ils cherchent à contrôler l'énergie consommée par les ressources matérielles. Au niveau des services, ces mécanismes cherchent à contrôler les ressources matérielles consommées. Il y a donc deux niveaux d'adaptation de l'énergie : un au niveau des équipements et un au niveau des services.

Ces deux niveaux sont complémentaires puisqu'ils ne considèrent pas les mêmes éléments, *i.e.*, services ou ressources matérielles. Ainsi, l'approche contrôlant l'énergie des équipements peut être séparée de l'approche contrôlant le placement des composants logiciels sur les équipements. Ces approches s'exécutent séparément l'une de l'autre. Cependant, une interaction entre ces approches permet une meilleure synergie afin de maximiser l'efficience énergétique.

Pour les appliquer à la maison numérique, il faut prendre en compte la volatilité de l'environnement. Il n'y a aucune certitude qu'un équipement soit toujours disponible. Par contre, comme nous avons pris l'hypothèse qu'un service n'est pas lié à un équipement, nous considérons que sa disponibilité est potentiellement plus élevée que celle d'un équipement. Il est donc possible de séparer ce qui relève des équipements de ce qui relève des services. Ainsi, la conservation d'informations sur l'environnement ou la prise de décision sont placées au niveau des services. Par contre, la mise en œuvre nécessite la coopération des équipements, que ce soit pour contrôler les états énergétiques des équipements ou bien pour déployer des composants.

Ainsi, l'architecture proposée se divise en deux systèmes complémentaires, chacun étant à la fois sur les niveaux service et équipement, comme l'illustre la Figure 6.1. Le premier

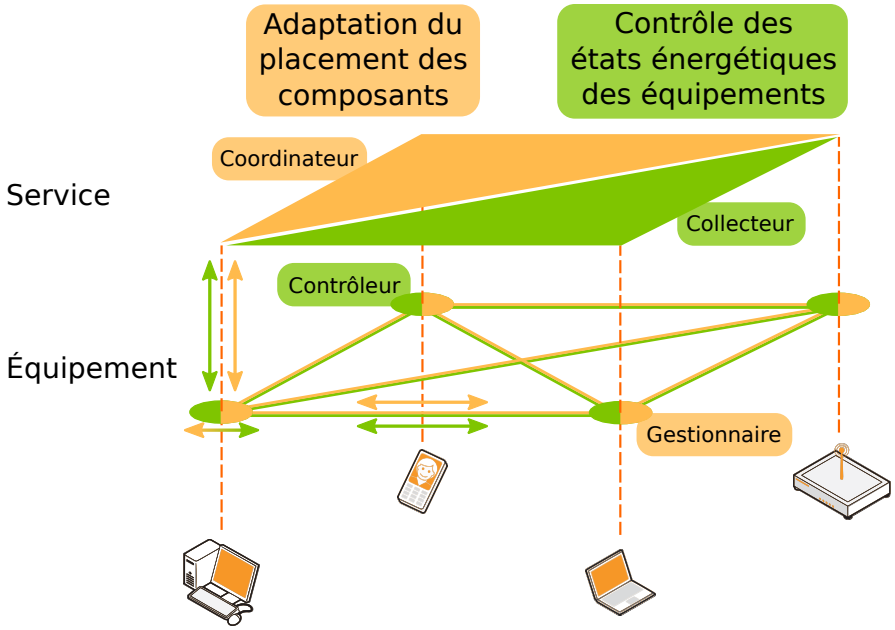

FIGURE 6.1 – **Architecture du système de réduction de la consommation d'énergie.** Le système d'adaptation du placement des composants, en orange, est composé des entités coordinateur et gestionnaire tandis que le système de contrôle des équipements, en vert, est composé des entités collecteur et contrôleur. Ces entités sont liées aux services ou aux équipements. Des interactions existent au sein de chacun des systèmes mais également entre les systèmes.

système cherche à maximiser l'efficience énergétique des services, tout en satisfaisant aux besoins de l'utilisateur. Ce premier système est composé de deux entités : le « coordinateur » qui s'occupe du placement des composants des applications réparties et le « gestionnaire » qui met en application les décisions de l'entité de coordination. Le gestionnaire nécessite des accès locaux, *i.e.*, démarrage ou arrêt des composants, et distants, *i.e.*, migration des applications sur un autre équipement.

Le deuxième système cherche à contrôler l'état des équipements. Il contrôle le réveil à distance des équipements mais doit également appliquer les politiques de gestion de l'énergie propre à chaque équipement. Ce système est composé d'entités dédiées : le « collecteur » qui recueille les informations liées aux méthodes de réveil à distance des équipements et le « contrôleur » qui s'occupe de la mise en état de basse consommation des équipements et ou de leur réveil.

6.2.1 Système d'adaptation du placement des composants

Le système qui s'occupe du placement des composants sur les équipements est lui même composé de deux entités : le coordinateur et le gestionnaire. Le rôle dévolu à ces entités

diffère mais est complémentaire afin de minimiser le nombre d'équipements actifs.

Entité Coordinateur

Nous parlons de « coordinateur » car il s'agit de rendre cohérent les ordres qui sont donnés à l'échelle de l'environnement. Pour prendre une analogie courante, cette entité correspond à un chef d'orchestre. Un chef d'orchestre cherche à rendre cohérent et harmonieux un ensemble d'instruments, notamment en donnant le tempo. En aucun cas, le chef d'orchestre ne va jouer d'un instrument autre que de sa baguette [23].

Le coordinateur effectue une optimisation de l'énergie en considérant la répartition des composants sur l'ensemble des équipements. Pour cela, cette entité a besoin d'informations issues aussi bien des équipements présents dans l'environnement que des applications qui s'y trouvent. Ainsi, le coordinateur nécessite avant tout que chaque équipement, pris individuellement, participe à alimenter une vision globale de l'environnement.

Une fois que l'entité possède suffisamment de données de l'environnement, elle peut envisager de changer la répartition des composants sur les équipements en utilisant le modèle décrit dans le Chapitre 5. Ce changement doit être autonome, *i.e.*, l'utilisateur ne doit pas avoir à s'en occuper, et doit s'adapter aux usages de l'utilisateur. Dans le cas contraire, l'utilisateur aurait du mal à accepter un tel système chez lui.

Entité Gestionnaire

Le « gestionnaire » participe également au placement des composants sur les équipements. Au même titre qu'un chef d'orchestre sourd et/ou manchot a du mal à diriger son orchestre, le coordinateur doit écouter et commander les équipements se trouvant dans l'environnement.

Comme la maison numérique est volatile, il est nécessaire de mettre à jour les informations le concernant. Cela passe avant tout par une écoute de ces événements. De plus, des actionneurs sont nécessaires pour mettre en applications les actions issues du coordinateur. Ces actionneurs sont disponibles sur les équipements.

Le gestionnaire participe activement à une bonne coordination des équipements entre eux. Il détecte notamment les événements sur les équipements afin de les remonter au niveau de la coordination. Il participe également à la mise en œuvre du plan de répartition en mettant en pratique les décisions du coordinateur.

6.2.2 Système de contrôle des états énergétiques des équipements

Le système d'adaptation du placement des composants ne considère pas les problématiques de gestion de l'énergie des équipements. Il s'attelle seulement à placer les composants afin de maximiser l'efficience énergétique. Cependant, cela ne suffit pas afin de réduire la

23. Au lecteur de juger si une baguette de chef d'orchestre est un instrument de musique ou non.

consommation d'énergie de l'environnement. Il faut également contrôler les états énergétiques des équipements afin de les faire passer en état de basse consommation ou encore de les réveiller si besoin. Pour les contrôler il faut collecter les informations des équipements, *e.g.,* méthode de réveil. Ces informations sont stockées dans l'entité de collecte.

Entité Collecteur

Afin de contrôler les équipements, une entité au niveau service doit conserver les informations propres au réveil de chaque équipement : le « collecteur ». Comme le système évolue dans un environnement volatile, ces informations doivent être accessible a tout moment. Pour cela, ces informations ne doivent pas être liées à un équipement en particulier. Ainsi, le collecteur doit pouvoir migrer d'un équipement à un autre si besoin. En lui permettant de se déplacer, cela fiabilise ces données face à la volatilité de l'environnement.

Entité Contrôleur

Le « contrôleur » doit tout d'abord prendre en compte les politiques de gestion de l'énergie propre à chaque équipement. Nous supposons toutefois que l'équipement est paramétré de façon à ce qu'il passe en état de basse consommation dès lors qu'il se sait inactif, *i.e.,* qu'il ne fournit plus aucun service.

Cette entité a également un rôle de médiation entre les équipements. Comme nous considérons un environnement réparti, il faut que chaque équipement participe à la gestion énergétique de l'environnement. Une couche de communication est donc nécessaire sur l'ensemble des équipements afin de collecter et de contrôler l'état énergétique des autres équipements. Le contrôle à distance se limite toutefois au réveil des équipements distants. En aucun cas, un équipement ne peut mettre en état de basse consommation un équipement distant puisqu'il s'agit d'une ingérence dans la politique de gestion de l'énergie de ce dernier.

6.2.3 Interaction entre les systèmes

Notre architecture est composée de deux systèmes (cf. Figure 6.1). Il y a d'abord un système d'adaptation du placement des composants sur les équipements. Il y a ensuite un système de contrôle de l'état énergétique des équipements. Chacun de ces systèmes est composé de deux types d'entités qui sont liés au niveau des services, *i.e.,* collecteur et coordinateur, ou des équipements, *i.e.,* contrôleur et gestionnaire. Le système d'adaptation du placement des composants cherche à maximiser l'efficience énergétique des services tandis que le système de contrôle de l'état énergétique fait passer dans un état de basse consommation les équipements inactifs. A priori, les deux systèmes peuvent fonctionner séparément.

Toutefois, pour une meilleure réduction de la consommation d'énergie, ces deux systèmes nécessitent de travailler ensemble. L'interaction entre ces deux systèmes se résume ainsi : 1) le système d'adaptation des services commande le réveil d'un équipement afin d'y

placer des composants. Il a donc besoin du niveau de contrôle des états énergétiques des équipements pour y parvenir. Ensuite, 2) le système de contrôle des états énergétiques des équipements lui retourne s'il a réussi à réveiller l'équipement en question ou non. Dans le cas d'un échec, l'équipement peut être considéré comme ayant disparu de l'environnement domestique, équivalant à envoyer un événement significatif de disparition d'un équipement.

Une interaction est donc nécessaire afin que les deux systèmes appliquent l'approche de réduction de la consommation d'énergie présentée dans le Chapitre 5. Cette interaction se concrétise uniquement si les deux systèmes utilisent les mêmes identifiants pour les équipements. Nous supposons que l'identifiant unique utilisé par l'adaptation des services pour différencier les équipements est le même que celui utilisé pour le contrôle des états de énergétiques des équipements.

6.2.4 Gestion de la dette énergétique

La « dette énergétique » d'un système correspond à la différence de la consommation d'énergie de l'environnement qui l'exécute par rapport à ce même environnement ne l'exécutant pas. Par exemple, lorsqu'un utilisateur démarre un navigateur web, ce navigateur nécessite des ressources matérielles qui consomment davantage d'énergie par rapport à s'il n'avait pas été lancé. La dette énergétique de cette application correspond à la différence de consommation de l'équipement lorsqu'il exécute cette application et sa consommation lorsqu'il n'exécute pas l'application.

Notre approche cherche à faire des gains énergétique. Toutefois, notre approche nécessite une phase dans laquelle il faut trouver le plan de répartition optimisé et le mettre en application, *i.e.,* migrer les composants. Cette phase induit une dette énergétique car pendant cette période l'approche utilise des ressources matérielles et donc a un impact négatif sur la consommation d'énergie. De plus, plus la dette énergétique est importante, plus sa compensation, *i.e.,* un « gain énergétique positif », est difficile à atteindre. Aussi, pour minimiser la dette énergétique, il faut que les systèmes soient conçus de manière à ne pas consommer trop d'énergie.

Dans notre cas, les systèmes sont actifs uniquement lorsque l'environnement change, *i.e.,* lorsqu'un événement significatif survient. Nos systèmes cherchent alors à modifier l'emplacement des composants sur les équipements. Ainsi, le système de placement des composants exécute cherche un nouveau plan de répartition uniquement lorsqu'un événement significatif survient. De même, le système de contrôle des états énergétiques est actif uniquement lorsque le système d'adaptation des services a besoin de lui. Le reste du temps, les systèmes ne font qu'observer l'environnement sans rien changer. Cette « réactivité » des systèmes assure que la dette énergétique augmente ponctuellement et qu'un nouveau plan de répartition n'est pas constamment cherché, sollicitant davantage d'énergie.

Le deuxième mécanisme limitant la dette énergétique requiert que les systèmes se considèrent eux-mêmes dans le placement de leur composants. Cela évite l'utilisation d'un équi-

pement dédié pour héberger ces systèmes puisque les systèmes utilisent alors les équipements déjà dans un état actif. Il est donc nécessaire d'avoir des systèmes ayant des degrés d'autonomie « automatiques » (cf. Section 2.3). Pour cela, il faut concevoir ces systèmes afin qu'ils puissent migrer d'un équipement à un autre, en fonction des équipements actifs dans l'environnement, comme les applications qu'ils cherchent à migrer.

Nous décrivons maintenant en détail et séparément le système d'adaptation du placement des composants et le système de contrôle des états énergétiques des équipements.

6.3 Adaptation du placement des composants

Afin d'adapter le placement les composants à l'environnement, nous nous sommes rapprochés des architectures autonomes existantes. Ces architectures permettent une adaptation à l'environnement, notamment une adaptation à un environnement volatile. Pour cela, nous nous basons sur la boucle autonome MAPE-K. La boucle MAPE-K permet une gestion autonome des ressources. Dans notre cas, elle adapte le placement des composants en fonction de l'environnement afin de maximiser l'efficience énergétique des services . Au travers de Capteurs et d'Actionneurs, la boucle autonome est capable de s'adapter à l'environnement dans lequel elle est déployée.

Ces deux entités de coordination et de gestion reprennent les sous-entités décrites dans la boucle MAPE-K et représentées dans la Figure 6.2. Cette concordance entre les entités et la boucle autonome est répartie comme suit :

- le coordinateur, qui a une vue d'ensemble de l'environnement, *i.e.*, équipements, applications. Il est en charge de trouver le plan de répartition des composants le plus efficient énergétiquement. Cette entité implémente les sous-entités Observation, Analyse, Optimisation et Planification, Exécution ainsi que Connaissance de la boucle MAPE-K. Cette entité n'est pas liée à un équipement particulier, elle peut donc être (re)déployée n'importe où dans l'environnement domestique.
- le gestionnaire, qui a une connaissance parfaite de l'équipement sur lequel il s'exécute et, dans une moindre mesure, de l'environnement. Il est chargé d'envoyer des informations relatives à l'équipement ou à l'environnement au coordinateur, *i.e.*, apparition ou disparition du réseau domestique et démarrage et arrêt d'une application. Il est également en charge d'exécuter les ordres du coordinateur. Cette entité réalise les sous-entités Capteur et Actionneur.

6.3.1 Description du coordinateur

Le coordinateur doit être constamment à l'écoute d'un événement significatif survenant dans l'environnement. Toutefois, ce n'est pas possible si tous les équipements sont dans un

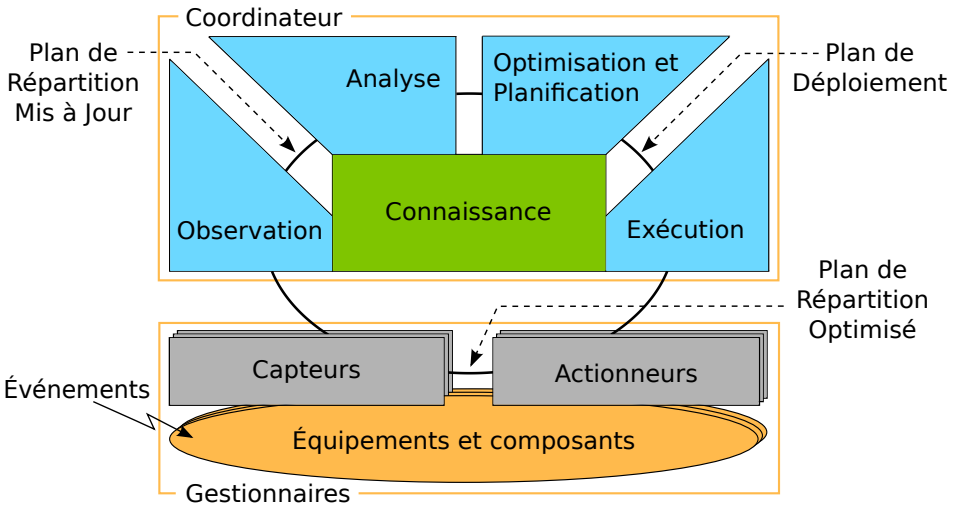

FIGURE 6.2 – **Répartition des fonctionnalités de la boucle MAPE-K entre le coordinateur et les gestionnaires.** Le coordinateur réalise les sous entités Observation, Analyse, Optimisation, Planification, Exécute et Connaissance. Le gestionnaire réalise les sous entités Capteur et Actionneur.

état de basse consommation au même instant, *i.e.*, ils sont imperméables aux communications. Aussi, au moins l'un d'entre eux doit être allumé pour traiter les événements, *e.g.*, départ ou arrivée d'un équipement ou d'une application.

Ainsi, le coordinateur est au cœur d'une architecture centralisée. L'utilisation d'une architecture décentralisée nécessite que plusieurs équipements soient dans l'état actif au même moment afin de définir le plan de répartition optimisé. Laisser plusieurs équipements actifs plutôt qu'un seul consomme davantage d'énergie. Toutefois, utiliser un unique équipement pour les gouverner tous pose le problème de savoir lequel, et de la panne de cet équipement.

Une partie du problème est résolu en séparant effectivement les services des équipements. Le coordinateur est donc naturellement une application répartie faite de composants. Dans son processus de décision, il se prend donc en compte pour savoir sur quels équipements ses composants doivent être déployés. De plus, le coordinateur est conçu à base de composants dont leur état peut être conservé afin de ne pas perdre les informations contextuelles déjà collectées. Le coordinateur peut migrer d'un équipement à un autre afin de s'exécuter sur l'équipement le plus approprié à un instant donné.

Par contre, ce choix apporte d'autres problèmes, notamment l'indisponibilité du coordinateur durant sa migration. Un événement significatif peut survenir pendant cette phase. Comme le processus de migration est déjà engagé, il faut seulement conserver les événements qui surviennent pendant cette période de temps. Une fois la phase de migration achevée, les événements peuvent être traités.

Comme nous l'avons évoqué, le coordinateur est l'entité en charge de produire le plan de

répartition le plus efficient énergétiquement. Pour cela, il est construit à l'aide d'une boucle MAPE-K dans laquelle il y a les sous-entités suivantes :

- Observation des événements liés aux équipements et aux applications;
- Analyse de l'impact d'un événement sur l'efficience énergétique du plan de répartition;
- Optimisation du plan de répartition;
- Planification du plan de déploiement;
- Exécution du plan de déploiement;

Ces sous-entités ont toutes un accès à la sous-entité Connaissance en charge de conserver les données issues de l'environnement, *i.e.,* équipements, applications, composants. Dans la suite de la section, nous détaillons davantage chacune des entités.

La sous-entité Observation. Pour éviter d'optimiser sans cesse le plan de répartition, et par conséquent consommer de l'énergie, nous supposons que le plan de répartition est toujours le plus efficient énergétiquement jusqu'à ce qu'un nouvel événement significatif survienne. De plus, pour éviter de créer de la dette énergétique sans pouvoir la rembourser, il faut limiter le nombre d'événements déclenchant une optimisation.

Aussi, nous limitons les événements significatifs aux seules apparition ou disparition d'un équipement et au démarrage ou à l'arrêt d'une application. Un équipement passant en état de basse consommation ou en sortant n'est pas considéré comme un événement significatif car l'équipement fait toujours parti de l'ensemble des équipements considérés. Il peut ainsi être réveillé à tout moment pour exécuter des composants. De même, une mise à jour de la description d'un équipement ou d'une application ne déclenche pas une optimisation.

Un événement significatif met à jour le plan de répartition en modifiant, soit l'ensemble des équipements, soit l'ensemble des applications (cf. Équation 3). Le plan de répartition en résultant est appelé « plan de répartition mis à jour » (cf. Figure 6.2). Ce plan ne maximise pas nécessairement l'efficience énergétique contrairement au plan avant que l'événement ne survienne, considéré comme optimisé. La sous-entité Observation déclenche le processus d'optimisation dès lors que l'un des ensembles change.

Le Tableau 6.1 résume quels événements sont considérés comme significatif par la sous-entité Observation. Les événements significatifs déclenchent toujours une optimisation, même si *in fine* le plan de répartition ne change pas. Les événements non significatifs ne déclenchent pas d'optimisation mais permettent au coordinateur de rester à jour de l'environnement.

La sous-entité Analyse. La sous-entité Analyse calcule la consommation du plan de répartition mis à jour, issue de la sous-entité Observation. Le calcul considère la consommation des équipements dans leur état actif et dans leur état de basse consommation envoyées

Événement	Optimisation ?
Apparition d'un équipement	✓
Disparition d'un équipement	✓
Mise à jour de la description d'un équipement	✗
Démarrage d'une application	✓
Arrêt d'une application	✓
Mise à jour de la description d'une application	✗
Impossibilité d'appliquer le plan de déploiement (*e.g.*, ressources insuffisantes)	✓

TABLE 6.1 – **Les événements déclenchant une optimisation du plan de répartition.** Les événements marqués d'une coche verte sont considérés significatifs par le coordinateur et déclenchent une optimisation. Les événements marqués d'une croix rouge ne sont pas significatifs mais sont nécessaires pour des optimisations futures.

lors de l'apparition de l'équipement (cf. Équation 8). Les données de consommation énergétique sont basées sur des formules mathématiques prenant en compte l'usage des ressources d'un équipement. La consommation d'un équipement est estimée en fonction des composants qu'il héberge. Certaines données propres aux équipements sont envoyées lorsque les équipements apparaissent dans l'environnement domestique, *e.g.*, consommation maximum dans l'état actif.

La consommation du plan de répartition mis à jour est ensuite envoyée à la sous-entité Optimisation. Cette valeur devient une valeur de référence pour déterminer si une modification du plan de répartition mis à jour est nécessaire.

La sous-entité Optimisation. En considérant le plan de répartition décrit dans la Section 5.4.1, la phase d'optimisation cherche à atteindre la fonction objectif, *i.e.*, minimiser la consommation d'énergie de la maison numérique, tout en considérant les contraintes de déploiement des composants. Cette entité implémente le modèle décrit dans le Chapitre 5.

La recherche d'une solution est effectuée par un solveur de contrainte qui cherche une solution optimale parmi l'ensemble des solutions existantes. Le temps de calcul est lié aux nombres de contraintes prises en compte ainsi qu'à la dimension du plan de répartition mis à jour. La solution optimale doit notamment conserver la qualité de service. La qualité de service est définie à la conception des services au travers des contraintes de déploiement des composants et satisfaite à l'exécution par le solveur de contraintes.

La sous-entité Optimisation est de loin la sous-entité requérant le plus de temps et de ressources matérielles pour fonctionner, et donc d'énergie. Cela est dû à l'utilisation du solveur de contraintes. Pour cela, le solveur nécessite l'utilisation intensive d'un processeur et

de la mémoire vive pendant toute la durée de la recherche d'une solution. La dette énergétique de notre système provient essentiellement de cette sous-entité.

La sous-entité Planification. Une fois que l'optimisation est terminée, le solveur de contraintes trouve une solution ou non. Dans le cas où aucune solution n'est trouvée, rien ne change. Dans le cas contraire, le nouveau plan de répartition issu de la phase d'optimisation est comparé au plan de répartition mis à jour, *i.e.,* celui issu de la phase d'observation.

Cette comparaison permet de déduire le plan de déploiement, décrit dans l'Équation 4 (cf. p.75). Ce plan de déploiement détaille, pour chaque gestionnaire, les actions qu'il a à exécuter pour que le plan de répartition optimisé soit mis en application.

La sous-entité Exécution. Une fois que le plan de déploiement a été calculé, la sous-entité Exécution envoie les actions à prendre aux gestionnaires concernés. Ces actions sont effectuées par les actionneurs des gestionnaires et sont détaillées dans la Section 6.3.2.

La sous-entité Connaissance. Enfin, les sous-entités décrites plus haut requièrent toutes un accès aux données de l'environnement afin d'exécuter leurs tâches respectives. Pour cela, la sous-entité Connaissance regroupe toutes les données issues des autres sous-entités. Ces dernières peuvent lire et écrire dans cette base de données.

6.3.2 Description du gestionnaire

Le coordinateur s'occupe de déterminer sur quels équipements les composants doivent être déployés. Afin de s'adapter à l'environnement volatile de la maison numérique et d'appliquer les actions décrites par le coordinateur, chaque équipement doit embarqué un gestionnaire. Le gestionnaire est en charge d'exécuter les sous-entités Capteur et Actionneur :

- Détection des événements issus des équipements sur lesquels ils sont déployés (cf. Tableau 6.1). Ces données sont ensuite envoyées au coordinateur;
- Exécution des actions ordonnées par le coordinateur afin de mettre en application le plan de répartition optimisé.

Dans notre architecture, chaque équipement est représenté par un gestionnaire. La boucle MAPE-K gére donc plusieurs capteurs et plusieurs actionneurs qu'il faut pouvoir distinguer les uns des autres. Ainsi, chaque gestionnaire est représenté au travers de l'identifiant de l'équipement sur lequel il s'exécute. Nous supposons que le nom de l'équipement est unique dans l'environnement.

La sous-entité Capteur. Pour calculer le plan de répartition le plus efficient énergétiquement, le coordinateur requiert des données de l'environnement domestique. Nous considérons que les gestionnaires sont les entités les plus pertinentes pour envoyer ces données car ils sont au plus proche des équipements et des applications. Chaque gestionnaire connaît l'équipement sur lequel il s'exécute ainsi que les composants actuellement déployés sur ce dernier. Ces données sont envoyées lorsqu'un événement significatif survient sur l'équipement. Par conséquent, le gestionnaire joue le rôle de capteur, sensible aux événements issus de l'équipement et des composants qu'il héberge.

Mais un gestionnaire remonte également la disparition des équipements qui l'entourent. Notamment, un gestionnaire cherchant à joindre un autre gestionnaire sait s'il a une réponse ou non. Il peut donc dire si un équipement a disparu de manière inattendue. Cette information est ensuite remontée au coordinateur pour qu'il mette à jour sa base de connaissance.

La sous-entité Actionneur. Le gestionnaire est aussi un actionneur. Il met en application les ordres du coordinateur au travers de trois actions prédéfinies : `start`, `stop` et `migrate`. Ces actions sont des compositions des activités de déploiement décrites dans [Hal99] : installation, activation, désactivation et désinstallation.

Ces actions ne sont pas exécutées sans être vérifiées au préalable. Le coordinateur est l'entité ayant une vision globale de l'environnement, cependant il ne connaît pas en détail chaque équipement. Notamment, le coordinateur ne sait pas si les ressources d'un gestionnaire ont été altérées depuis la dernière optimisation. Ainsi lorsqu'une action est demandée à un gestionnaire, ce dernier vérifie qu'il peut accomplir l'action. Par exemple, le gestionnaire doit vérifier que les composants ont effectivement les ressources matérielles promises par le coordinateur afin d'être déployés sur l'équipement. Si ce n'est pas le cas, le gestionnaire informe le coordinateur afin qu'il modifie le plan de répartition en conséquence.

Enfin, le gestionnaire a également un rôle d'actionneur à distance. Lorsqu'il cherche à faire migrer un composant sur un autre équipement, il transmet les données relatives à ce composant à l'équipement distant. Comme le coordinateur peut migrer, les gestionnaires ont besoin d'une autonomie suffisante afin de redéployer le coordinateur sur les équipements adéquates. C'est pour cela qu'un gestionnaire peut ordonner certaines actions à un autre gestionnaire. Par exemple, dans le cadre d'une migration de composant, un gestionnaire ordonne à un autre gestionnaire de démarrer ce composant présent sur ce premier.

6.4 Contrôle des états énergétiques des équipements

Le système de contrôle des états énergétiques des équipements est réparti sur l'ensemble des équipements. Son fonctionnement est séparé du système d'adaptation du placement des composants. Ce système est divisé en deux parties. La première est dédiée à la collecte des informations de réveil des équipements. La deuxième partie est dédiée au contrôle des états énergétiques des équipements, notamment au réveil à distance ou à la mise en état de basse consommation des équipements.

6.4.1 Description du collecteur

La collecte des informations de réveil est le premier obstacle dans un environnement volatile. Un nouvel équipement arrivant dans l'environnement ne connaît aucun autre équipement auquel transmettre ses informations de réveil. Il faut donc passer par une diffusion afin de s'assurer qu'un autre équipement puisse apprendre comment le réveiller.

Cependant, la diffusion de ces informations se heurte à l'état énergétique des autres équipements. En effet, un équipement ne peut communiquer avec un autre équipement que si les deux sont dans un état actif. Si le message est diffusé et que les autres équipements présents sont dans un état de basse consommation, alors ils ne recevront pas ce message. Ainsi ils seront incapables de réveiller le premier équipement s'il passe à son tour dans un état de basse consommation.

Grâce au système d'adaptation du placement des composants, un équipement doit être constamment allumé afin d'intercepter les événements survenant dans l'environnement. Aussi, il est possible d'utiliser ce même équipement afin qu'il stocke la méthode de réveil de l'équipement venant de diffuser son message. Un équipement est donc actif, à un instant donné, afin de collecter les informations de réveil.

Par contre, comme l'environnement est volatile, à un autre instant, rien ne garantit que l'équipement qui est allumé le reste. Il est donc important que ces informations puissent migrer d'un équipement à un autre. Cela permet ainsi au collecteur d'être toujours présent dans l'environnement. Aussi la mise en place d'une entité liée à un service, et non pas à un équipement, est une nécessité si nous voulons que les informations de réveil des équipements soient toujours disponibles pour l'entité de contrôle les utilisant.

6.4.2 Description du contrôleur

Le contrôleur est séparé en deux parties. La première, le contrôle local, est relative à l'équipement, *i.e.,* applique la politique de gestion de l'énergie propre à l'équipement en question. La deuxième, le contrôle à distance, est relative au réveil à distance des autres équipements, en se basant sur les informations stockées dans le collecteur.

Le contrôle local

Il y a d'abord un contrôle local de l'état énergétique d'un équipement. Ce contrôle s'occupe de mettre en pratique les politiques de gestion de l'énergie qui sont propres à l'équipement. Il s'agit de savoir si une des règles de mise en veille est respectée, *e.g.,* utilisateur absent au bout de x secondes, charge du processeur très faible. Ou si, plus finement, l'équipement peut placer dans un état de basse consommation certaines parties de ses ressources matérielles car elles sont inutilisées, *e.g.,* disque dur, carte réseau.

Le contrôleur est confondu avec un système d'exploitation enrichi d'un module plus poussé pour la gestion individuelle de l'énergie. Cependant, nous prenons l'hypothèse que

l'équipement intègre une règle. Cette règle spécifie que l'équipement doit passer en état de basse consommation dès lors qu'aucun composant ne s'exécute sur l'équipement, à l'exception des composants propres aux systèmes de notre solution.

Le contrôle à distance

Au delà du contrôle propre à chaque équipement, une fois dans l'état de basse consommation, nous supposons que seul un autre équipement, ou l'utilisateur, est capable de le sortir de cet état. Aussi il est nécessaire de pouvoir contrôler un équipement à distance pour le réveiller.

Dans un environnement hétérogène, chaque équipement est différent. Aussi, il n'est pas impossible qu'il existe plusieurs méthodes de réveil différentes et qu'elles ne soient pas implémentées sur tous les équipements. Ainsi, il y a des équipements qui ne peuvent pas réveiller certains équipements. Par exemple, l'équipement A implémente la méthode R_1 pour réveiller et être réveillé et l'équipement B implémente la méthode de réveil R_2. Il est donc impossible pour A de réveiller B et inversement. Toutefois, par chance, l'équipement C implémente les méthodes R_1 et R_2. Donc si A veut réveiller B, il doit passer par C.

Dans le cadre de ce travail, nous prenons l'hypothèse qu'il existe une seule méthode de réveil, *i.e.,* le *Wake-on-LAN*, disponible sur tous les équipements. Dans la réalité, il existe des méthodes de réveil différentes. De plus, il faut prendre en compte quel équipement peut réveiller quel autre équipement, complexifiant la problématique. Cette problématique nécessite de minimiser le nombre d'équipements à réveiller pour placer les composants.

6.5 Gestion des pannes

Jusqu'à présent, nous avons considéré que les systèmes fonctionnaient sans problème. Les événements significatifs, *i.e.,* l'apparition ou la disparition d'un équipement ou d'une application, déclenchent une optimisation. Nous avons pris l'hypothèse que pour chaque événement, le coordinateur est informé et qu'il modifie la répartition des composants sur les équipements.

Cependant, il peut arriver que le coordinateur ne soit pas mis au courant suite à un dysfonctionnement. Pour les événements d'apparition, cela ne pose pas beaucoup de problèmes, le message pouvant être retransmis plus tard par les gestionnaires concernés. Pour les événements de disparition par contre, c'est plus problématique.

6.5.1 Les pannes et leur détection

Afin de continuer à maximiser l'efficience énergétique, le système doit être capable de surmonter ces pannes. Comme il y a une grande diversité de pannes, nous discutons seulement de celles qui nous paraissent les plus graves.

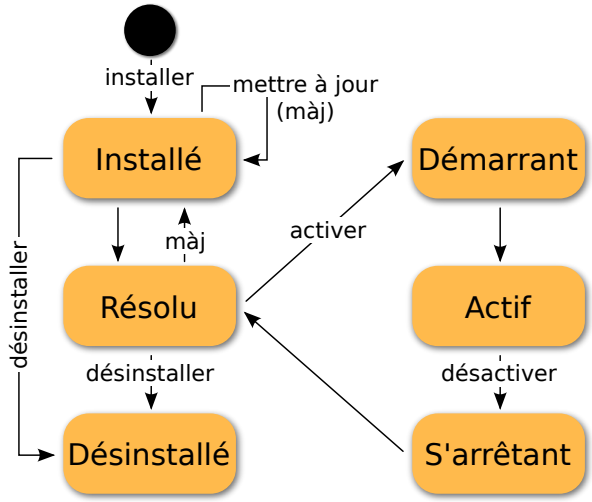

FIGURE 6.3 – **Cycle de vie des composants et activités de déploiement.** Un composant peut passer par plusieurs états au travers des actions décrites dans la figure. Un composant rend son service uniquement dans l'état `Actif`.

- disparition inattendue d'un composant d'une application répartie;
- disparition inattendue d'un équipement;
- disparition du coordinateur.

Disparition inattendue d'un composant

La disparition d'un composant d'une application répartie est l'une des premières défaillances à considérer. Elle peut survenir suite à une défaillance de l'équipement (*e.g.*, sortie inattendue du réseau, panne de l'équipement) ou bien à cause de sa propre défaillance (*e.g.*, panne du composant). Lorsqu'un composant disparaît sans prévenir, cela affecte les autres composants qui en dépendent. Si un composant client n'a plus de fournisseur, il n'est plus à même de fournir son propre service.

Pour détecter la disparition d'un composant, le gestionnaire hébergeant le composant défaillant détecte son changement d'état. Lorsqu'un composant passe dans un état dans lequel il n'est plus à même de fournir un service car une de ses dépendances n'est plus satisfaite, il passe dans l'état Résolu (cf. Figure 6.3). Le gestionnaire sait alors qu'un problème est survenu et peut remonter l'information au coordinateur qui propose un nouveau plan de répartition.

Toutefois, cette détection se limite à savoir qu'une application ne fonctionne plus, pas à savoir quel composant est le fautif. C'est pour cela que le coordinateur propose un nouveau plan de répartition qui redéploie l'ensemble des composants de l'application.

Disparition inattendue d'un équipement

La disparition inattendue d'un équipement est également considérée comme une défaillance. Lorsqu'un équipement disparaît sans s'annoncer, il emporte avec lui l'ensemble des composants qu'il exécute. S'il n'en exécute aucun, il n'y a pas de problèmes. Lors de la prochaine phase d'optimisation, un équipement en moins est disponible pour héberger des composants. Toutefois, s'il exécute des composants, alors certaines applications réparties peuvent tomber en panne à leur tour lorsqu'elles dépendent des composants disparus. La disparition d'un équipement correspond alors à une disparition de composants.

La disparition d'un équipement n'est détectable que si il ne communique plus. Ainsi, il faut que les équipements communiquent régulièrement lorsqu'ils sont actifs afin de s'assurer qu'ils sont toujours dans l'environnement. Cela est fait au travers d'un message régulier, diffusé dans l'environnement domestique, attestant de leur présence (*e.g.*, message SSDP alive dans UPnP). Dans notre cas, comme un gestionnaire doit remonter régulièrement son état, ce sont ces messages qui permettent au système de détecter la disparition d'un équipement.

Pour un équipement en état de basse consommation, l'émission de messages n'est pas possible. La détection s'effectue alors lorsqu'un gestionnaire essaye de le réveille et de communiquer avec lui. Cela revient à dire que tant qu'un équipement n'est pas nécessaire, le système ne sait pas si cet équipement est toujours dans le réseau domestique, prêt à exécuter des composants. C'est un problème lorsque ces équipements doivent héberger des composants puisque ne pouvant les déployer, le coordinateur doit modifier en conséquence le plan de répartition afin de concorder à la nouvelle réalité.

Disparition du coordinateur

Enfin, la disparition du coordinateur met en péril la capacité du système à s'adapter et à continuer à maximiser l'efficience énergétique. La disparition du coordinateur peut survenir suite à la perte d'un de ses composants ou d'un équipement hébergeant un des ses composants. Comme le coordinateur peut être réparti sur un ensemble d'équipements, il n'est pas à l'abri de perdre un de ses composants. S'il perd un de ses composants, il ne peut plus exercer son rôle de coordinateur. Et ainsi le coordinateur disparaît.

La disparition du coordinateur est détectée seulement par les gestionnaires. En effet, ce sont les seules entités à interagir avec lui. Cette interaction est possible uniquement lorsqu'un événement survient. Ainsi la détection de la disparition du coordinateur a lieu lorsque le gestionnaire n'a aucune réponse aux messages qu'il envoie au coordinateur, *e.g.*, message d'acquittement.

6.5.2 Mécanismes de rétablissement

Une fois les pannes détectées, il faut rétablir le système afin qu'il reprenne son travail. Il s'agit ici de guérison et pas de prévention. Pour cela, nous proposons un ensemble de mécanismes qui permettent de résoudre les dysfonctionnements.

Répartition des fichiers d'état

Les composants logiciels sont de deux types, avec et sans état. Les composants avec état peuvent être relancés à partir d'un état dès lors qu'une trace de cet état est conservé. Chaque gestionnaire doit donc régulièrement introspecter les composants avec état qu'il héberge, récupérer leur état et les sauvegarder dans un fichier.

La sauvegarde régulière de l'état des composants avec état participe à la résilience des applications. Sauvegarder régulièrement les états des composants permet de redémarrer plus rapidement une application suite à une panne. Mais pour renforcer la résilience des applications, cet état ne doit pas être stocké sur un équipement qui peut à son tour être soumis à une panne. L'état des composants doit être dupliqué et stocké sur différents équipements dans l'état actif.

Rétablissement des applications réparties

Dans le cas d'un composant sans état, sa remise en fonctionnement se fait au travers de son démarrage. Comme il est sans état il est immédiatement disponible et peut ainsi être reconnecté à ses clients de la même manière qu'après une migration.

Dans le cas d'un composant avec état, il existe deux manières de le redémarrer. La première est de récupérer le dernier état existant du composant qui s'est exécuté sur un équipement. Si il s'agit de sa propre défaillance, cet état se trouve sur l'équipement qui l'héberge. Cet équipement conserve un état du composant suite à une migration ou à une sauvegarde périodique. S'il s'agit d'une défaillance de l'équipement, il faut déployer le composant sur un autre équipement et chercher s'il existe un état antérieur se trouvant sur un équipement encore présent dans le réseau domestique.

La deuxième option, mais pas forcément la plus intéressante, est de suivre la même procédure que dans le cas d'un composant sans état. Dans ce cas extrême, tout le travail accompli doit être refait depuis le début. Cette dernière option est exécutée seulement si l'autre a échoué.

Rétablissement du coordinateur

Les gestionnaires sont conçus pour fonctionner sans coordinateur, dans une certaine mesure. Cela signifie qu'ils sont capables de prendre des décisions basiques sans l'aide du coordinateur. Ainsi lorsque le coordinateur n'est pas accessible, un gestionnaire est apte à déployer des composants de lui même sur l'équipement qui l'héberge. Ainsi, les composants du coordinateur sont redéployés lorsqu'il n'est plus disponible participant ainsi à sa remise en marche.

Mais comme chaque gestionnaire est autonome, chacun peut relancer un coordinateur sur son propre équipement. Il y a alors plusieurs coordinateurs dans l'environnement. Ainsi avant qu'un gestionnaire relance un coordinateur, les gestionnaires doivent choisir lequel d'entre eux l'hébergera. Le rétablissement du coordinateur s'établit comme suit (également décrit au travers d'un diagramme de séquence dans la Figure B.1 en Annexe B) :

① Chaque fois qu'un gestionnaire se réveille ou apparaît dans le réseau domestique, il envoie une notification au coordinateur. Le gestionnaire envoie également régulièrement son état au coordinateur.

② Si un gestionnaire ne reçoit pas de réponse de la part du coordinateur, il le considère alors comme disparu.

③ En conséquence, il diffuse un message afin de déclencher une nouvelle élection parmi l'ensemble des gestionnaires actifs.

④ Si une élection est déjà en cours, il arrête son processus d'élection et se joint à l'élection existante. Sinon, chaque gestionnaire actif envoie l'horodatage de l'état des composants du coordinateur qu'ils possèdent à l'initiateur de l'élection.

⑤ Le gestionnaire qui possède les horodatages les plus récents est élu pour déployer le nouveau coordinateur (à condition qu'il ait les ressources nécessaires également).

⑥ Le gestionnaire élu démarre alors tous les composants du coordinateur sur lui même en restaurant leur état.

Cette élection regroupe tous les composants sur un unique équipement. Ultérieurement, les composants du coordinateur peuvent être amenés à être répartis sur l'ensemble des équipements actifs de l'environnement.

6.6 Conclusion

L'architecture proposée rend la solution autonome et efficiente énergétiquement. Sa réactivité lui permet de s'endetter énergétiquement uniquement lorsqu'un événement significatif survient. De même, la solution est automatique, *i.e.*, elle se considère dans la recherche du plan de répartition optimisé et est donc capable de migrer d'un équipement à un autre, limitant son impact énergétique.

Dans ce chapitre, nous avons présenté deux systèmes indépendant mais fonctionnant de concert pour atteindre l'objectif d'efficience énergétique. Un premier système s'adapte aux événements significatifs survenant dans la maison numérique afin de trouver le plan de répartition le plus efficient énergétiquement. Un deuxième système gère l'état énergétique des équipements.

Le système d'adaptation du placement des composants se compose de deux entités : le coordinateur et le gestionnaire qui gèrent chacun un niveau d'optimisation différent. Le coordinateur s'occupe de répartir les composants sur l'ensemble des équipements de la maison tandis que le gestionnaire, un par équipement, s'occupe de gérer les composants sur les équipements et de remonter les événements significatifs.

Le système de contrôle de l'état énergétique des équipements se charge de collecter et contrôler les états énergétique des équipements en les passant en état de basse consommation dès lors qu'ils sont inactifs ou en réveillant à distance des équipements. Le collecteur

est indépendant des équipements, fiabilisant l'accès à ses informations. Le contrôleur gère l'état énergétique de l'équipement sur lequel il est déployé et modifie l'état énergétique des équipements distants.

Avec ces deux systèmes, nous considérons à la fois le niveau des applications, *i.e.*, placement des composants, et celui des équipements, *i.e.*, gestion des états énergétiques. De plus, ces systèmes permettent une gestion globale de l'énergie ainsi qu'une gestion locale, la première étant prioritaire face à la seconde.

Enfin, l'architecture peut être sujette à différentes pannes. Seules quelques unes sont considérées, notamment lorsqu'un composant disparaît de manière inattendue. Nous avons proposé divers mécanismes permettant de palier ces défaillances.

Troisième partie

Validation

Chapitre 7

Mise en œuvre d'*HomeNap*

Propre, durable. Pas de pollution. Aucune infrastructure requise. Seulement l'énergie; la même énergie qui alimente nos vaisseaux. L'énergie bleue est la réponse aux maux de votre monde, et pour le peuple de Timbal, c'était la réponse à leur crise.

– Anna, *V*

Sommaire

7.1 Introduction 110
7.2 Extension du modèle 111
7.2.1 Consommation d'un équipement 111
7.2.2 Mise sous forme d'un problème de satisfaction de contraintes . 112
7.2.3 Discussion 114
7.3 Mécanismes de migration 116
7.3.1 Actions sur les composants et leurs états 116
7.3.2 Sauvegarde et restauration des états 117
7.3.3 Reconnexion des services 118
7.3.4 Processus de migration 119
7.3.5 Dépôt de composants et de leur état 119
7.3.6 Discussion 119
7.4 Implémentation d'*HomeNap* 121
7.4.1 Canevas logiciels utilisés 121
7.4.2 Description des équipements et des applications 123
7.4.3 Intégration des technologies dans l'approche 126
7.4.4 Discussion 130
7.5 Conclusion 131

7.1 Introduction

La partie validation est constituée de deux chapitres. Le premier présente la mise en œuvre de l'approche au travers de notre intergiciel *HomeNap* tandis que le deuxième cherche à l'évaluer. En plus de mettre en œuvre le modèle décrit précédemment, notre intergiciel a pour objectif de réaliser et valider les propriétés suivantes :

Prise en compte de l'hétérogénéité. L'hétérogénéité est une contrainte forte de l'environnement domestique numérique. Il s'agit des différences qui caractérisent au moins deux équipements, *e.g.*, système d'exploitation, ressources matérielles. Les différentes formes de l'hétérogénéité nécessitent d'être prises en compte à la conception pour produire une solution adaptée à la maison numérique.

Prise en compte de la volatilité. La volatilité est également une contrainte forte de la maison numérique. La volatilité est l'apparition ou la disparition d'un équipement ou d'une application. Elle impose une solution s'adaptant à des changements imprévisibles.

Conservation de la qualité de service. La qualité de service est la propriété la plus visible pour l'utilisateur. Il est nécessaire de la prendre en compte pour l'acceptation d'une solution. Si cette propriété venait à être fortement dégradée alors l'utilisateur ne voudrait pas d'une solution permettant des réductions de la consommation d'énergie.

Amélioration de l'efficience énergétique. L'efficience énergétique est l'objectif à atteindre dans cette étude. Cette amélioration doit tenir compte des propriétés évoquées plus haut afin de proposer des gains énergétiques.

Pour commencer, nous présentons la mise en œuvre du modèle et de l'architecture au travers de notre intergiciel *HomeNap*. Pour parvenir à une solution mettant en œuvre l'approche, nous étendons le modèle de la maison numérique, complétant celui du Chapitre 5. Cette extension du modèle définit la manière dont consomme un équipement ainsi que la fonction objectif et les contraintes qui sont utilisées dans la sous-entité Optimisation de la boucle autonome.

Ensuite, nous détaillons les mécanismes utilisés pour déplacer les composants d'un équipement à un autre. Migrer des composants doit tenir compte de l'état des équipements ainsi que de l'état dans lequel se trouve le composant. La migration doit suivre un processus strict assurant que le service est correctement rétabli une fois que les composants le fournissant ont été déplacés.

Enfin, ce chapitre présente les choix technologiques et techniques pour la mise en œuvre d'*HomeNap*. Nous présentons les différents canevas logiciels sur lesquels se base notre solution afin de s'intégrer facilement dans la maison numérique. L'utilisation de standards développés pour cet environnement, *e.g.*, UPnP, nous assure une meilleure intégration.

Structure du chapitre

Le chapitre s'articule comme suit : la Section 7.2 étend le modèle utilisé au sein de la boucle autonome pour trouver le plan de répartition optimisé. La Section 7.3 développe les mécanismes permettant la migration des composants d'un équipement à un autre. La Section 7.4 décrit le système *HomeNap*, système qui intègre le modèle et l'architecture décrient dans la contribution. Enfin, la Section 7.5 conclut ce chapitre.

7.2 Extension du modèle

Dans le Chapitre 5, nous évoquons le modèle de consommation des équipements. Le modèle nécessite de prendre certaines hypothèses supplémentaires afin d'être mis en œuvre. Par exemple, le modèle de la contribution ne décrit pas la manière dont consomme un équipement, et surtout ses ressource matérielles, car il n'existe aucun modèle générique applicable à l'ensemble des équipements. Toutefois, dans le cadre de la validation, un modèle est nécessaire, même simplifié.

Cette section définit également la fonction objectif cherchant à minimiser la consommation d'énergie de l'ensemble des équipements. Finalement, cette section définit les contraintes utilisées dans la validation et permettant de poser notre problème d'optimisation comme un problème de satisfaction de contraintes.

7.2.1 Consommation d'un équipement

Pour définir la consommation énergétique d'un équipement, nous utilisons une formule qui se base sur la charge du processeur d'un équipement. Le processeur est une ressource matérielle commune à chaque équipement. De plus, celle-ci est la plus consommatrice d'énergie dans un équipement. Cette ressource est ainsi la meilleure candidate pour notre modèle simplifié de consommation d'un équipement.

La puissance consommée par l'équipement est fonction de la charge du processeur, *i.e.*, du nombres d'instructions qu'il a à traiter à un instant donné [ERKR06]. Pour la validation, nous choisissons un modèle linéaire basé sur la charge du processeur pour définir sa puissance (cf. Équation 9).

Équation 9 : Consommation d'un équipement

$$P_e^{on}(R_e^{cpu}utilisée) = (P_{e_{max}} - P_{e_{min}}) \times \frac{R_e^{cpu}utilisée}{R_e^{cpu}totale} + P_{e_{min}}$$

$P_{e_{max}}$ correspond à la puissance maximale d'un équipement lorsque l'ensemble de ses ressources sont utilisées. $P_{e_{min}}$ correspond à la puissance minimale lorsqu'aucun composant

logiciel ne s'exécute sur l'équipement mais que ce dernier est encore dans un état actif. Ces deux constantes sont mesurées *a priori* de l'évaluation par l'utilisation d'un wattmètre.

L'Équation 9 permet de calculer la puissance consommée par un équipement à un instant donné. Cette équation décrit donc l'hétérogénéité des consommations d'énergie des équipements.

7.2.2 Mise sous forme d'un problème de satisfaction de contraintes

Afin de résoudre le problème de l'optimisation du plan de répartition, nous choisissons de le mettre sous la forme d'un problème de satisfaction de contraintes. Pour cela, nous définissons la fonction objectif, *i.e.*, la fonction que l'algorithme d'optimisation cherche à résoudre, ainsi que les contraintes qui pèsent sur la fonction objectif.

Fonction objectif

Trouver le plan de répartition optimisé consiste à trouver le plan de répartition qui minimise la puissance utilisée par l'ensemble des équipements parmi l'ensemble des plans de répartition satisfaisant l'ensemble des contraintes. L'Équation 10 est la fonction objectif qui minimise la puissance utilisée par l'ensemble des équipements informatiques considérés. Cette fonction est définie à partir de la fonction d'utilité (cf. Équation 8) et cherche toujours un optimum global.

Équation 10 : Fonction objectif

$$P_{d^t} = min(P_s^t) = min(\sum_{e \in E^t} (\delta_e^t \times P_e^{on}(R_e^{cpu} utilise) + (1 - \delta_e^t) \times P_e^{lp}))$$

Pour rappel, P_{d^t} correspond à la consommation de la solution optimale du plan de répartition à l'instant t. P_s^t correspond à la puissance totale utilisée par l'ensemble des équipements à l'instant t. $P_e^{on}(R_e^{cpu} utilise)$ représente la puissance de l'équipement e, qui est fonction de la charge du processeur, $R_e^{cpu} utilise$, à l'instant t. P_e^{lp} est la puissance de l'équipement lorsqu'il est dans un état de basse consommation. Enfin, δ_e^t prend la valeur 1 si l'équipement e est héberge au moins un composant, 0 sinon.

Contrainte sur la dette énergétique d'*HomeNap*

Modéliser la dette énergétique d'*HomeNap* se heurte à la connaissance *a priori* de la durée passée dans chacune des phases de la boucle MAPE-K. Par exemple, nous ne connaissons pas *a priori* le temps nécessaire au calcul du plan de répartition optimisé. Il est donc difficile d'estimer la dette énergétique de la sous-entité Optimisation. C'est pour cela que le chapitre suivant cherche à l'évaluer. Toutefois, il est nécessaire de modéliser, même grossièrement, la dette énergétique d'*HomeNap* afin d'éviter que ce dernier déclenche un processus de migration de composants pour un gain énergétique négatif, *i.e.*, qui ne compense pas la dette.

Aussi après la sous-entité Optimisation, nous supposons que seule la sous-entité Exécution engendre une dette énergétique conséquente. En effet, pendant que le composant migre, il ne fournit pas son service et fait donc consommer de l'énergie aux équipements qui participent à sa migration.

La migration d'un composant considère deux équipements : l'équipement e depuis lequel le composant migre et l'équipement e' sur lequel le composant migre. Ce processus de migration est divisé en trois étapes, de différentes durées : (a) t_s pour arrêter le composant sur l'équipement e, (b) t_t pour transférer le composant de l'équipement e à l'équipement e', et (c) t_r pour redémarrer le composant sur l'équipement e'.

L'Équation 11 définit l'énergie consommée $\mathcal{E}_m^t$ pour migrer un composant à un instant donné t. Nous prenons l'hypothèse que les deux équipements dédient toutes leurs ressources matérielles au processus de migration. Cela permet ainsi de modéliser grossièrement la dette énergétique de la migration car en réalité un équipement exécute plusieurs processus en parallèle.

Équation 11 : Dette énergétique de la migration

$$\mathcal{E}_m^t = P_e^{on} \times (t_s + t_t) + P_{e'}^{on} \times (t_t + t_r)$$

La contrainte utilisée dans *HomeNap* implique que la dette $\mathcal{E}_m^t$ doit être compensée par le gain énergétique issu du plan de répartition optimisé. Pour cela, nous adoptons la contrainte suivante :

Équation 12 : Contrainte sur la dette énergétique de la migration

$$P_{d^{t+1}} \times \Delta t + \mathcal{E}_m^t < P_{d^t} \times \Delta t$$

P_{d^t} est la puissance utilisée par le plan de répartition mis à jour et $P_{d^{t+1}}$ correspond à la puissance utilisée par le plan de répartition optimisé. $\mathcal{E}_m^t$ correspond à la dette énergétique de la migration. Enfin, Δt est le durée entre deux événements. À noter que si Δt tend vers 0, la migration est inutile parce que la dette énergétique tend à être plus élevée que le gain énergétique. Toutefois, si Δt tend vers ∞ alors la décision de migrer est toujours prise parce que la dette est négligeable par rapport au gain énergétique engendré.

L'utilisation de cette contrainte nécessite de connaître la durée entre deux événements. Cependant, dans *HomeNap*, il n'existe aucun mécanisme permettant de prédire quand va survenir le prochain événement. C'est pour cela que dans l'évaluation, la durée entre deux événements est connue à l'avance grâce à l'utilisation de scénarios minutés.

Contraintes sur les ressources et les quantités de ressource

Nous faisons l'hypothèse que la soustraction des quantités de ressources disponibles sur un équipement R_e par celles requises par un composant R_c détermine si le composant est

déployable sur l'équipement. Ainsi *HomeNap* sait quelles quantités de ressources il reste à l'équipement une fois que le composant est déployé (cf. Section 5.4.3).

Pour garantir qu'un service n'est pas dégradé en migrant sur un autre équipement, aucune valeur de $R_e(C_e^t \cup c)$ ne doit être négative. Si c'est le cas, le composant ne peut pas être déployé sur l'équipement. Cela garantit la qualité de service. Quand un composant s'arrête, les ressources utilisées sont libérées.

Cette contrainte décrit l'hétérogénéité des ressources matérielles disponibles dans la maison numérique.

Contrainte sur la migration des composants

La dernière contrainte utilisée dans la validation correspond à la possibilité pour un composant de migrer. Cette contrainte définit si, à tout instant, un composant peut être déplacé d'un équipement à un autre (après avoir été déployé une première fois). Il s'agit d'une contrainte booléenne définie pendant la phase de conception.

Cette contrainte spécifie quels composants sont liés à un équipement. C'est le cas notamment des interfaces graphiques qui sont souvent liées à un équipement particulier, *i.e.,* celui sur lequel se trouve l'utilisateur.

Cette contrainte est définie comme suit : si le composant c ne peut pas migrer, *i.e.,* `Migratable = false`, et c appartient à l'ensemble des composants s'exécutant sur l'équipement e, *i.e.*, $c \in C_e^t$, alors sa valeur dans le plan de répartition optimisé doit rester à 1, *i.e.,* $d_{ec}^t = 1$. Cette contrainte impose de laisser l'équipement e dans l'état actif pour que le service proposé par c soit fourni.

Cette contrainte décrit l'hétérogénéité des usages dans la maison numérique.

7.2.3 Discussion

Cette section décrit une extension du modèle de la contribution afin de mettre en œuvre l'approche. Les hypothèses prises, *i.e.,* la consommation d'un équipement, la description des ressources matérielles, définissent les hétérogénéités que nous considérons. Cette extension du modèle définit notamment les hétérogénéités de consommation d'énergie, de ressources matérielles liées aux équipements ou aux applications et d'usage par un utilisateur.

Ces hypothèses sont mises sous forme d'un problème de satisfaction de contraintes s'appliquant sur la fonction objectif. Cette extensibilité est nécessaire car toutes les hétérogénéités de la maison numérique ne sont pas encore considérées au travers du modèle. Par exemple, les améliorations du modèle portent sur une modélisation plus fine de la consommation des équipements, sur une modélisation des communications ou de la dette énergétique d'*HomeNap*.

Vers des modèles intégrés aux ressources matérielles

Les hypothèses posées pour la validation supposent une consommation énergétique linéaire des équipements, seulement basée sur la charge du processeur. Cette loi est ensuite appliquée pour tous les équipements, quel que soit leur type, *i.e.,* téléphone intelligent, passerelle résidentielle, ordinateur.

Cependant, suivant le type d'équipement, la part de consommation du processeur dans la consommation totale n'est pas la même. Par exemple, dans un téléphone intelligent, il faut prendre en compte la consommation de l'écran ou des communications [CH10]. Dans un ordinateur portable, la mémoire morte est également source de consommation [MV05]. Chaque équipement nécessite donc un modèle de consommation dédié.

De plus, un équipement ne consomme pas de la même manière suivant comment il est utilisé. Par exemple, lorsqu'il communique beaucoup, sa carte réseau consomme d'avantage que lorsque l'équipement est silencieux. De même, lorsque son écran est éteint, il ne consomme pas de la même manière que lorsque qu'il est allumé. Aussi, comme un équipement est composé de différentes ressources matérielles pouvant se trouver dans différents états alors le calcul de la consommation d'un équipement s'en trouve complexifié.

Toutefois, trouver un modèle propre à chaque équipement est du ressort des équipementiers ou des fondeurs. En introduisant les modèles de consommation de leurs équipements directement dans ce dernier, *e.g.,* dans les pilotes matériels, les couches supérieures peuvent alors analyser l'impact des usages sur la consommation. Ainsi, cela permet de proposer des solutions plus optimisées car plus proches de la réalité.

Extension du modèle

Cette section étend le modèle proposé dans la contribution en vue de la validation. Toutefois, cette extension ne prend pas en compte tous les aspects de la maison numérique. Il faut également considérer le coût des communications entre les composant répartis sur des équipements différents. Dans notre étude, nous estimons que ce coût est faible au regard de la consommation des équipements de la maison numérique. Par exemple, lorsqu'un routeur est saturé de messages afin qu'il les réémette un maximum de fois, sa consommation d'énergie augmente d'environ 4 W [Rem08]. Comparé à un ordinateur portable consommant entre 20 W et 40 W, cela demeure faible.

Toutefois, prendre en compte le coût énergétique des liens entre les équipements améliore le modèle de la maison numérique au travers d'une nouvelle contrainte. Cette contrainte impose aux composants fortement communiquant d'être déployés sur le même équipement ou sur des équipements ayant un faible coût de communication. Il s'agit là d'un travail à court terme à effectuer afin d'améliorer le modèle [KL10].

Enfin, cette extension ne modélise pas finement la dette énergétique de notre solution. Dans cette validation, nous modélisons uniquement la dette énergétique de la migration d'un composant, *i.e.,* la sous-entité Exécution. Une modélisation plus complète doit tenir

compte de la dette énergétique de notre système dans sa globalité lors de la recherche du plan de répartition optimisé et de son application, *e.g.*, temps de réveil des équipements, temps de l'optimisation.

7.3 Mécanismes de migration

Nous avons pris l'hypothèse que les composants d'une application répartie peuvent se déplacer d'un équipement à un autre. Le service est alors fourni pour un coût énergétique moindre. En effet, en regroupant les composants sur un même équipement, l'efficience énergétique de l'équipement est augmentée [BH07] et permet à d'autres équipements de passer en état de basse consommation.

Cependant, déplacer des composants est compliqué à mettre en œuvre. En effet, un composant en plein traitement ne peut pas migrer à tout instant, au risque de recommencer cette tâche depuis le début. Déplacer un tel composant nécessite de prendre en compte son état juste avant la migration afin que l'état soit restauré et qu'il puisse reprendre son exécution sur le nouvel équipement.

De plus, migrer un composant est un problème pour les composants qui en dépendent, *i.e.*, ses clients. Lorsqu'un composant migre, il faut que ses clients puissent de nouveau le contacter afin de continuer à fonctionner. Ainsi, avant de rétablir la liaison, le client nécessite de savoir où chercher son fournisseur. Il faut donc un mécanisme rétablissant les connexions quel que soit l'équipement sur lequel le fournisseur se trouve.

7.3.1 Actions sur les composants et leurs états

Le gestionnaire est l'entité en charge d'effectuer des actions modifiant l'état des composants. Pour les états, nous avons choisi le cycle de vie proposé par OSGi [OSG12]. Un composant peut donc prendre l'état : installé, résolu, démarrant, actif, s'arrêtant, désinstallé (cf. Figure 6.3). Les actions modifiant les états des composants sont des compositions des activités de déploiement décrites dans [Hal99] : installation, activation, désactivation et désinstallation. Ainsi, le gestionnaire met en application les ordres du coordinateur au travers de trois actions prédéfinies : `start`, `stop` et `migrate` détaillé ci-après.

- L'action `start` installe un composant depuis l'adresse URL indiquée dans le message d'action et l'active. Lorsque le composant a démarré sur un équipement, son *bundle* est rendu publiquement accessible sur l'équipement. L'URL du *bundle* est mise à jour dans sa description pour que les prochains équipements la récupère depuis le dernier équipement qui l'a déployé.
- L'action `stop` désactive puis désinstalle un composant. Nous supposons que chaque composant possède un identifiant unique dans l'environnement domestique afin d'éviter d'arrêter le mauvais composant.

- L'action `migrate` est une composition des actions `stop` et `start`. L'action `migrate` arrête le composant sur l'équipement qui l'exécute avant d'envoyer une action `start` à l'équipement amené à hébergé ce composant. Si l'équipement est dans un état de basse consommation, le gestionnaire se charge de le réveiller avant d'envoyer l'action `start`.

7.3.2 Sauvegarde et restauration des états

La sauvegarde de l'état d'un composant est essentielle pour mettre en application le plan de répartition optimisé. Cela permet de ne pas perdre le travail accompli et de le continuer sur un autre équipement. Toutefois, cela ne concerne que les composants avec état.

Pour sauvegarder l'état d'un tel composant, il y a deux manières de faire, liées à deux types de migration : la migration forte et la migration faible [BHRS98]. La migration forte consiste à sauvegarder l'environnement d'exécution du composant : son code objet, les données de son instance (*e.g.*, variables locales) et les données d'exécution (*e.g.*, pile d'exécution, pointeur d'instruction). La migration faible consiste seulement à migrer le code objet ainsi que les données de l'instance.

Dans notre cas, nous nous limitons à une migration faible. Aussi, un composant est alors décrit au travers de ses variables. Pour récupérer facilement ces variables, nous considérons que les variables correspondent aux propriétés fonctionnelles du composant. Ainsi lorsqu'une des variables à conserver change, la valeur de la propriété correspondante change également. Au moment de sauvegarder l'état du composant, le composant est introspecté et la valeur de ses propriétés est récupérée. De même, lorsque l'état d'un composant est restauré, le système intercesse le composant afin de lui réaffecter les valeurs des propriétés sauvegardées.

Cette sauvegarde a lieu à n'importe quel moment. Il suffit d'introspecter le composant et de sauvegarder ses propriétés. Toutefois, cette sauvegarde doit également se faire lorsque le composant s'arrête (action `stop`). C'est lors de cet arrêt que l'état du composant est le plus à jour. C'est l'état à l'arrêt du composant qui sert par la suite pour la phase de migration du composant.

La restauration de l'état du composant s'effectue lorsque le composant redémarre (action `start`). Aussi, le composant doit être développé en lui attribuant une méthode de démarrage l'amenant dans une phase de restauration d'état. Cette méthode doit être développée par l'architecte car il s'agit de la seule personne à savoir comment interpréter les états et ainsi pouvoir rétablir le bon fonctionnement d'un composant.

Enfin, la sauvegarde des états est nécessaire uniquement pour certains composants. Les composants sans état nécessitent seulement d'être démarrés afin de fournir le service souhaité.

7.3.3 Reconnexion des services

Une fois que la sauvegarde et la restauration de l'état d'un composant sur un équipement sont effectives, l'application ne redémarre pas immédiatement. En effet, si un composant bouge alors qu'il est en train de fournir un service à un autre composant, ce dernier doit pouvoir le joindre de nouveau après la migration afin de continuer à fournir son propre service à ses clients. Tant que l'application n'est pas capable de fournir le service souhaité par l'utilisateur, elle est dans un état instable.

Comme la topologie de l'application a changé, le composant client a besoin de savoir où se trouve le composant fournisseur afin de rétablir la liaison entre eux. À partir de là, quatre solutions sont possibles :

- utiliser un serveur centralisé, dans lequel les composants migrés publient leur nouvelle adresse et auquel les composants client s'abonnent (*e.g.*, RSS [RSS02]);
- utiliser un système de publication décentralisé (*e.g.*, Pubsubhubbub [pub13]) dans lequel les composants comptent sur d'autres équipements pour leur fournir l'information;
- diffuser le plan de répartition optimisé à chaque équipement afin qu'ils fassent les reconnexions eux-mêmes;
- diffuser les nouvelles adresses des composants migrés dans la maison numérique dès qu'ils démarrent (*e.g.*, SSDP [UPn08]).

La première et la deuxième solution impliquent que les équipements qui hébergent ces informations ne s'éteignent pas ou bien que les informations migrent sur d'autres équipements afin d'être toujours accessibles. Toutefois, migrer ces informations nécessite de connaître leur localisation afin de les récupérer. Cela revient au problème initial. Ainsi, ces informations ne peuvent pas être fournies par un système de publication.

La troisième solution permet à chaque gestionnaire de savoir où les composants sont déployés. Toutefois, ils ne savent pas quand ils seront effectivement actifs. Si par exemple, un composant ne démarre pas correctement, le composant client n'en sera jamais averti et essayera sans cesse de se reconnecter au composant qui n'a jamais démarré.

Il ne reste donc que la quatrième solution qui consiste à diffuser ces informations dans l'environnement domestique. Ainsi lorsqu'un composant démarre sur un équipement, sa localisation est diffusée dans l'environnement afin que les composants client s'y connectent.

Toutefois, cette solution nécessite que chaque composant fournisseur intègre un mécanisme pour diffuser sa nouvelle adresse. Ou encore, cela nécessite que chaque composant client possède un mécanisme pour se reconnecter. Pour éviter d'intégrer de tels mécanismes, chaque gestionnaire diffuse ces nouvelles adresses. Une fois que le gestionnaire sait où le composant fournisseur se trouve, il rétablit le lien de manière transparente entre les deux composants. Ainsi, l'architecte de services tiers n'a pas à s'occuper de ce problème.

7.3.4 Processus de migration

Pour éviter de commencer la migration d'un ensemble de composants sans être certain que l'environnement n'a pas évolué depuis les dernières mises à jour, le processus de migration sépare les différentes étapes. Ainsi, la migration d'un composant est évitée si le système n'a pas la certitude que le composant peut être redémarré à la suite de sa migration.

L'enchaînement suivant des actions permet de s'assurer que la migration des composants est un succès (voir la Figure B.2 en annexe pour le diagramme d'activité du processus de migration) :

① S'assurer que les équipements sur lesquels vont être déployés les composants sont effectivement présents dans l'environnement domestique. Une détection de leur présence ou leur réveil est requis pour s'assurer qu'ils sont joignables.

② Vérifier que l'équipement distant peut effectivement recevoir les composants, *i.e.*, toutes les contraintes sont satisfaites. Dans le cas contraire, les informations sur lesquelles le coordinateur s'est basé pour produire le plan de répartition optimisé sont obsolètes. Celui-ci doit alors prendre en compte ces nouvelles informations au travers d'une nouvelle optimisation.

③ Démarrer le processus de migration : arrêter les composants qui doivent l'être et sauvegarder leur état si nécessaire; transférer les composants et leur état s'il existe; démarrer les composants sur le nouvel équipement, en restaurant leur état si nécessaire.

7.3.5 Dépôt de composants et de leur état

Les composants sont amenés à migrer d'un équipement à un autre. Si un composant est amené à être régulièrement migré alors son *bundle* se trouve sur l'ensemble des équipements l'ayant déjà déployé auparavant. La redondance du composant est ainsi augmentée, participant à éviter certaines pannes. Notons que la question de la version du composant n'est pas considérée. Nous prenons l'hypothèse que le composant n'a qu'une seule version en cours.

Enfin, le gestionnaire est également un dépôt des états des composants s'exécutant dans la maison numérique. Chaque gestionnaire conserve notamment les états des composants du coordinateur afin de prévenir à une éventuelle panne de cette entité et ainsi la rétablir sans perdre trop d'informations.

7.3.6 Discussion

Cette section aborde la migration des composants dans l'environnement domestique numérique. Nous présentons les mécanismes mis en place afin que la fourniture du service puisse reprendre de manière transparente pour l'utilisateur, *i.e.*, reconnexion des composants, sauvegarde et restauration des états. Cela garantit la continuité de service et donc participe à l'acceptation de la solution auprès de l'utilisateur.

De plus, la récupération des états des composants avec état lors de leur migration économise de l'énergie en évitant de refaire un travail que les composants ont déjà fourni. Cela participe donc à la réduction de la consommation d'énergie.

Toutefois, les composants avec état doivent être conçus afin de récupérer facilement leur état. L'architecte doit identifier les variables à migrer et proposer des méthodes permettant au composant de restaurer son état.

Enfin, ces migrations interviennent suite au calcul d'un nouveau plan de répartition optimisé, l'optimisation intervenant à la suite d'un événement significatif, *i.e.*, apparition ou disparition d'un équipement ou d'une application. Cependant, il existe un événement qui peut poser problème lorsque la période de recherche du plan de répartition optimisé et de sa mise en application est longue. Cet événement correspond au départ d'un équipement qui se fait souvent de manière brutale et rapide.

Patron de conception

Afin de migrer un composant avec état, pour que celui-ci reprenne son exécution là où il s'est arrêté, il nécessite un développement particulier. Notamment, il faut identifier les variables nécessaires à la reprise de l'exécution sur un autre équipement afin de les rendre lisibles par le système de migration.

De même, il est nécessaire de réattribuer au composant les valeurs de ses variables après migration. Or seul l'architecte du composant est à même de spécifier, suivant les valeurs des variables, comment il faut procéder et quel est l'impact sur son fonctionnement.

À l'avenir et afin de faciliter le développement d'un tel composant, il est nécessaire de définir des patrons de conception. Ces patrons de conceptions doivent proposer des méthodes génériques afin d'identifier et de restaurer les valeurs des variables après migration.

Départ d'un équipement et migration

Dans cette étude, nous considérons qu'un équipement qui quitte la maison numérique envoie un message au coordinateur. Le coordinateur prend en compte cet événement puis propose un nouveau plan de répartition. Des actions sont menées avec l'équipement quittant l'environnement afin qu'il migre les composants qu'il exécute.

Toutefois, si un équipement quitte le réseau domestique, il n'a pas forcément le temps d'attendre que le processus d'optimisation soit achevé et ainsi il emporte l'ensemble des instances des composants qu'il héberge. Aussi, contrairement à l'approche existante dans notre solution, il est nécessaire à l'avenir de prévoir un cas particulier pour cet événement.

Une solution est de sauvegarder l'état des composants de l'équipement qui quitte le réseau puis de diffuser ces états à l'ensemble des gestionnaires actifs à ce moment là. Ainsi l'optimisation se fait en parallèle d'une partie de la migration. Une fois l'optimisation achevée, les gestionnaires restants applique le plan de déploiement en récupérant les composants d'autres équipements et en restaurant leur état si nécessaire.

7.4 Implémentation d'*HomeNap*

HomeNap est l'implémentation de l'approche décrite dans la contribution. Cette solution intègre aussi bien le modèle étendu que l'architecture. Pour parvenir à cette implémentation, différents outils librement accessibles sont utilisés. Ces outils sont issus soit de standards existants, *e.g.,* Java ou UPnP, soit d'autres domaines de recherche, *e.g.,* solveur de contraintes.

7.4.1 Canevas logiciels utilisés

L'implémentation d'*HomeNap*[24] se base sur différents canevas logiciels que nous détaillons. Nous discutons également de leur pertinence par rapport aux technologies utilisées dans la maison numérique.

Java

Java est une technologie apparue dans les années 1990 qui demeure populaire dans les équipements informatiques domestiques du quotidien, *e.g.,* téléphones intelligents, ordinateurs. Cette technologie est disponible pour différentes architectures de processeurs, *e.g.,* x86, ARM, et pour différents systèmes d'exploitation, *e.g.,* Linux, Windows. Nous prenons donc l'hypothèse qu'une machine virtuelle Java (JVM)[25] est présente sur tous les équipements de la maison numérique.

Or, bien que Java soit disponible pour un large ensemble d'équipements différents, il n'y est pas toujours installé par défaut. Et lorsqu'il est installé sur l'équipement, l'architecte tiers n'a pas toujours le droit d'installer son application sur l'équipement. Nous prenons donc l'hypothèse que les équipements sont suffisamment ouverts afin de pouvoir installer des applications tierces. Ainsi, grâce à Java, un composant peut s'exécuter sur tous les équipements de la maison numérique.

Aussi, pour faciliter l'exécution des applications dans un environnement hétérogène, *HomeNap*, ainsi que les services qui sont utilisés pour l'évaluation, sont développés en Java.

OSGi et Felix

OSGi [OSG12] est un standard apparu vers la fin des années 1990. La version actuelle est la version 5 (juin 2012). OSGi gère, entre autres, le cycle de vie des applications ou encore propose un registre de services. De par son ancienneté, OSGi est une technologie mature dont il existe plusieurs implémentations.

La plupart des canevas logiciels qui implémentent ce standard sont développés en Java. Le canevas logiciel utilisé pour l'évaluation est Felix[26]. Toutefois, comme Felix se base sur

24. https://github.com/Orange-OpenSource/HomeNap (2013)
25. Java Virtual Machine
26. http://felix.apache.org/ (2013)

un standard, il est possible de l'intervertir avec un autre canevas logiciel implémentant le standard OSGi.

Dans le cas d'*HomeNap*, l'usage d'OSGi permet de facilement créer des unités de déploiement, *i.e.*, *bundles*, et de gérer leur cycle de vie sur les différents équipements. Un composant est contenu dans un *bundle*, *i.e.*, un groupement de classes Java et d'autres ressources, qui fournissent des services à un client ou à un autre *bundle*.

Nous prenons également l'hypothèse que Felix est installé sur l'ensemble des équipements de la maison numérique.

Les composants iPOJO

La mise en œuvre de l'architecture et des services de l'évaluation se fait grâce au canevas logiciel iPOJO [EHL07]. iPOJO est utilisé au dessus du canevas logiciel Felix. L'usage de ce canevas logiciel facilite avant tout le développement en séparant clairement les services fournis par chaque composant de l'architecture ou des services de l'évaluation.

Dans le cadre de l'architecture, ce canevas logiciel permet de clairement séparer les différentes entités de la boucle autonome MAPE-K au travers de composants distincts. Chaque composant est ensuite placé dans un *bundle* à part afin de faciliter leur déploiement sur d'autres équipements.

L'usage des composants permet également d'utiliser la réflexivité afin de sauvegarder et de restaurer l'état d'un composant à l'exécution. Ces propriétés propres aux composants permettent d'implémenter un système de migration faible de leur états (cf. Section 7.3).

UPnP et le réseau domestique

Le protocole UPnP [UPn08] est spécifiquement conçu pour la maison numérique. Il a été largement adopté par les équipementiers qui peuplent ces environnements, *e.g.*, téléphone intelligent, boîtier décodeur, passerelle résidentielle. Il est donc naturel de l'utiliser pour une meilleure intégration d'*HomeNap* dans cet environnement.

Le protocole UPnP prend en compte la volatilité de la maison numérique. Ce protocole facilite la découverte et l'interaction entre les équipements. Lorsqu'un équipement apparaît ou disparaît, UPnP diffuse un message. Nous l'avons adapté pour qu'il fasse de même lorsqu'une application souhaite démarrer ou s'arrêter afin d'en informer le coordinateur.

Outre la possibilité de prendre en charge la volatilité de l'environnement, UPnP est le protocole permettant la communication entre les entités de notre architecture, à savoir le coordinateur et les gestionnaires présents sur les équipements. Comme le coordinateur est amené à migrer d'un équipement à un autre, l'adresse à laquelle les gestionnaires doivent envoyer les messages est amenée à changer également. Par l'utilisation d'UPnP et lorsque le coordinateur a migré, ce dernier diffuse sa présence dans l'environnement afin qu'*HomeNap* continue à fonctionner correctement.

Solveur de contraintes Choco

Le problème que nous considérons est similaire à celui du problème du *bin packing* où des heuristiques permettent de trouver des solutions rapidement. Toutefois, ces algorithmes se limitent à 2 ou 3 dimensions (cf. Annexe B). Dans nos cas, comme le nombre de ressources matérielles à prendre en compte est supérieur à 3, il faut trouver algorithmes plus génériques.

Afin de trouver le plan de répartition optimisé, le modèle est décrit sous la forme d'un problème de satisfaction de contraintes. Un problème de satisfaction de contraintes est une suite finie de variables ayant des domaines respectifs, ainsi qu'un ensemble fini de contraintes, chacune sur une sous-suite de ces variables [Apt03].

Pour résoudre ce problème, nous utilisons le solveur de contraintes Choco [JRL+08]. Choco est une librairie Java qui a été intégrée dans un *bundle* OSGi pour les besoins d'*HomeNap*. Choco permet d'implémenter le modèle étendu et de trouver la solution optimale quand elle existe.

Dans la boucle MAPE-K, Choco fait parti de la sous-entité Optimisation, *i.e.*, celle qui cherche le plan de répartition optimisé. Cette sous-entité cherche toujours la solution la plus optimale. Cette recherche peut prendre du temps et dépend en partie du nombre d'éléments considérés, *i.e.*, nombre d'équipements et nombre de grappes de composants, ainsi que des contraintes définies. Enfin, le temps de recherche dépend également de l'équipement qui traite le problème.

Mise en veille et réveil

Pour l'évaluation, nous prenons l'hypothèse que tous les équipements supportent le Wake-on-LAN. Ainsi chaque équipement est capable de réveiller les autres équipements au travers de cette méthode. Cela limite donc l'évaluation à l'utilisation d'équipements accessibles uniquement au travers d'un réseau Ethernet.

Pour les états de basse consommation, nous nous appuyons sur le standard ACPI pour les architectures x86 [Hew10]. Ce standard définit différent états de basse consommation. Toutefois, dans l'évaluation nous n'en considérons qu'un seul par équipement, *i.e.*, état G1/S1 de l'ACPI.

Pour les architectures ARM, l'ACPI n'est pas encore en place. Pour ces équipements, nous ne considérons donc pas de moyens de veille ni de réveil. Ainsi, un équipement ARM ne possède qu'un seul état : actif.

7.4.2 Description des équipements et des applications

Les contraintes des composants sont décrites dans un fichier séparé de leur code source. De même, les ressources matérielles disponibles sur les équipements sont décrites dans un fichier dédié. Ces fichiers sont envoyés au coordinateur, respectivement, lorsque l'application

```
{
    "name": "application-name",
    "components": ["componentA","componentB","componentC"]
}
```

Code 7.1 – Fichier de description d'une application

cherche à démarrer et lorsque l'équipement se connecte pour la première fois dans le réseau domestique. Ces fichiers sont au format JSON [27] permettant ainsi de les lire plus facilement puisqu'il existe des librairies Java permettant de les analyser rapidement, *e.g.*, GSON [28].

Description d'une application

Comme nous faisons l'hypothèse qu'une application est un ensemble de composants, il faut créer autant de descriptions qu'il existe de composants dans l'application. Aussi, pour décrire une application, il existe un fichier contenant l'identifiant de l'application ainsi que l'identifiant de chacun des composants qui la compose. Ce fichier d'architecture est présenté dans le Code 7.1.

Pour décrire un composant, il est nécessaire de fournir plus d'informations (cf. Code 7.2). Tout d'abord, il faut faire le lien avec l'identifiant du composant décrit dans la description de l'application (champ `name`). Il faut également l'adresse URL à laquelle se trouve le *bundle* afin qu'un équipement tiers puisse le déployer (champ `url`). Ensuite, il faut spécifier l'état du composant (champ `state`) : avec état ou sans état. Enfin, il faut décrire les contraintes appliquée à un composant (champ `constraints`).

Description d'un équipement

Pour relier les ressources requises par un composant et les ressources disponibles sur un équipement, il faut également décrire ces dernières. Dans le cadre de cette évaluation, les ressources d'un équipement sont décrites de manière statique. Ainsi, si des ressources sont ajoutées ou enlevées, il faut éditer le fichier décrivant ces ressources (cf. Code 7.3).

En plus de décrire les ressources de l'équipement, d'autres informations sont nécessaires pour le bon fonctionnement d'*HomeNap*. Par souci de simplicité, les informations utiles sont agrégées au sein du fichier de description de l'équipement. Il s'agit de l'adresse MAC de l'équipement (champ `mac`) et l'adresse IP (champ `ip`). Il y a également les diverses consommations énergétiques de l'équipement lorsqu'il est dans l'état actif mais qu'il n'exécute aucun composant (champ `consumptionOnMin`), lorsque l'équipement est saturé de travail et qu'il consomme le plus d'énergie (champ `consumptionOnMax`) et enfin, lorsqu'il est dans l'état de basse consommation (champ `consumptionOff`).

27. JavaScript Object Notation
28. https://code.google.com/p/google-gson/ (2013)

```
{
    "name": "componentA",
    "url": "http://url.du.bundle",
    "state": "STATELESS",
    "constraints": [
        {"resources": [
            {"name": "CPU", "value": 750},
            {"name": "RAM", "value": 42}
        ]},
        {"migratable": true}
    ]
}
```

Code 7.2 – Fichier de description d'un composant

```
{
    "mac": "03:0E:0F:5C:41:23",
    "ip": "192.x.x.x",
    "consumptionOnMin": 80,
    "consumptionOnMax": 124,
    "consumptionOff": 5,
    "resources": [
        {"name": "CPU", "value": 5984},
        {"name": "RAM", "value": 2059984},
        {"name": "SCR", "value": 1}
    ]
}
```

Code 7.3 – Fichier de description d'un équipement

Le fichier de description d'un équipement est ensuite placé sur l'équipement participant à l'évaluation. Il est lu par le gestionnaire de l'équipement à son démarrage.

7.4.3 Intégration des technologies dans l'approche

Les technologies présentées précédemment mettent en œuvre l'approche au travers du coordinateur et des gestionnaires. Chaque composant qui compose nos entités est fait à base de iPOJO. Ces composants sont ensuite intégrés dans un *bundle* OSGi afin d'être déployés séparément.

Mise en œuvre du coordinateur

La boucle autonome MAPE-K est mise en œuvre au travers des composants suivants (cf. Figure 7.1) :

Observation. L'observation est effectuée par le composant `GCDevice`. Ce composant est en charge de la récupération des événements des gestionnaires au travers du protocole UPnP. Il fournit le plan de répartition mis à jour.

Analyse. L'analyse est mise en œuvre par le composant `Analyser`. Il envoie à la sous-entité Planification la consommation courante du plan de répartition mis à jour.

Planification. La planification est composée des composants `Optimizer` et `Migrater`. Le premier intègre le solveur de contraintes Choco et cherche le plan de répartition optimisé. Le deuxième récupère le plan de répartition optimisé et calcul la différence entre ce plan et le plan de répartition mis à jour. Le résultat est le plan de déploiement.

Exécute. L'exécution est effectuée par le composant `Executer`. Il envoie les actions à mener par les gestionnaires pour atteindre le plan de répartition optimisé.

Connaissance. La connaissance est mise en œuvre au travers du composant `GlobalDatabase` disponible pour l'ensemble des composants de la boucle autonome.

Mise en œuvre du gestionnaire

Les entités Capteurs et Actionneurs sont mises en œuvre dans le gestionnaire (cf. Figure 7.2). Il existe plusieurs déclencheur d'une action de la part du gestionnaire:

- Le composant `Command` par lequel l'utilisateur démarre ou arrête les applications. Au travers de cette interface utilisateur, il donne le fichier de description d'une application (cf. Code 7.1). Ce fichier est ensuite extrait de sa forme JSON par le composant `ArchitectureReader`.

FIGURE 7.1 – **Architecture du coordinateur.** Le coordinateur reprend le patron de conception de la boucle autonome MAPE-K. Pour cela, il écoute les événements issus des gestionnaires grâce au composant `GCDevice`. Puis, le coordinateur analyse, optimise, planifie et exécute les actions pour atteindre la fonction objectif. Tous les composants sont développés en iPOJO et séparément inclus dans un *bundle* OSGi. Le composant `Optimize` intègre le solveur de contraintes Choco. Les composants `GCDevice` et `LMControlPoint` permettent la communication en UPnP avec les gestionnaires.

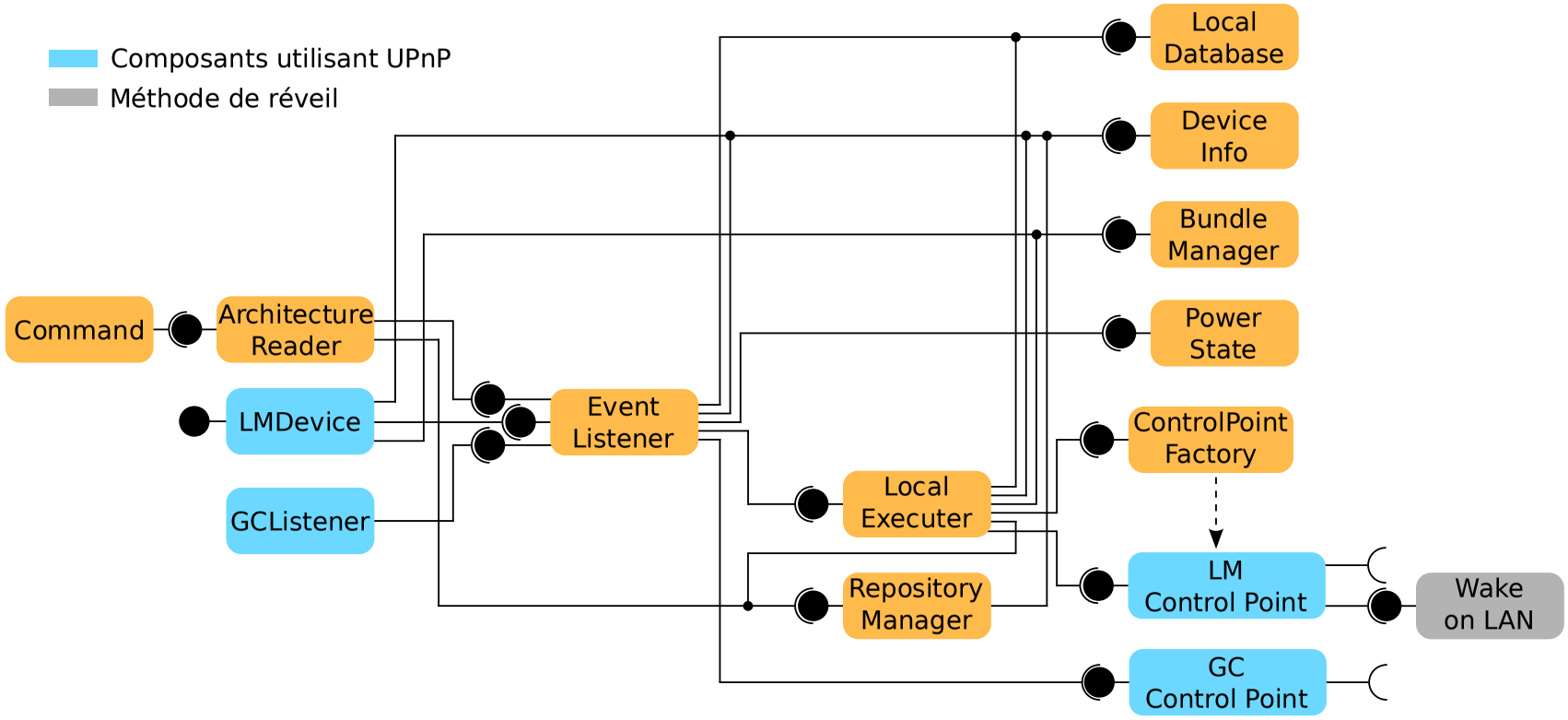

FIGURE 7.2 – **Architecture du gestionnaire.** Le gestionnaire écoute les actions issues d'un utilisateur local, *i.e.*, `Command`, du coordinateur ou d'un autre gestionnaire, *i.e.*, `LMDevice`. Ensuite, il exécute ces actions en prenant en compte les composants qui sont à sa disposition pour mettre en place le plan de répartition. Tous les composants sont développés en iPOJO et séparément inclus dans un *bundle* OSGi. Les composants `LMDevice`, `LMControlPoint` et `GCControlPoint` permettent la communication en UPnP avec les gestionnaires ou le coordinateur.

- Le composant `LMDevice` par lequel le coordinateur donne les actions à mener par le gestionnaire ou par lequel un gestionnaire transfert un composant à un autre gestionnaire.
- Le composant `GCListener` qui permet au gestionnaire de détecter la présence du coordinateur dans le réseau.
- Le composant `PowerState` remonte un événement lorsque le temporisateur d'inactivité de l'équipement atteint zéro. Cela signifie alors que l'équipement peut passer en veille.

Ces événements locaux ou distants sont ensuite transmis au composant `EventListener` qui les analyse et contacte le composant `LocalExecuter` pour les actions visant à démarrer ou à arrêter les composants, *i.e.,* composant `BundleManager`, à copier les *bundles* des composants dans un dossier public accessible depuis les autres gestionnaires, *i.e.,* composant `RepositoryManager`, à envoyer des actions à d'autres gestionnaires, *i.e.,* composant `LMControlpoint`, ou des événements au coordinateur, *i.e.,* composant `GCControlPoint`.

Le gestionnaire conserve également une base de données des composants démarrés au travers du composant `LocalDatabase` et a accès à la description de l'équipement au travers du composant `DeviceInfo`.

Enfin, le gestionnaire a accès au système de contrôle énergétique des équipements au travers du composant `LMControlPoint`. Dans l'architecture du gestionnaire, cela est représenté par le composant `Wake-on-LAN`.

Interaction entre les entités

La liaison entre le coordinateur et les gestionnaires est effectuée au travers du protocole UPnP. Pour cela, il est nécessaire de définir un client UPnP, *i.e.,* un *control point*, afin d'effectuer des actions sur le serveur, *i.e.,* le *device*. Dans *HomeNap*, il existe un seul et unique coordinateur. Aussi, son *device* est unique dans le réseau. Cette unicité est représentée par le déploiement d'un seul est unique composant chargé de représenté le coordinateur : le composant `GCDevice`. De même, chaque gestionnaire déploie un seul et unique *control point* : le composant `GCControlPoint`.

Dans le cas où le coordinateur est le client et souhaite envoyer les actions aux gestionnaires, ou bien que les gestionnaires doivent communiquer entre eux, il est nécessaire de déployer autant de clients qu'il y a de gestionnaires. Pour cela, chaque gestionnaire déploie un seul et unique *device* le représentant dans le réseau. Par contre, pour joindre d'autres gestionnaires, il existe une fabrique, *i.e.,* `ControlPointFactory`, qui instancie un *control point* par gestionnaire au moment où une autre entité cherche à le joindre : le composant `LMControlPoint`. Le composant `LMControlPoint` est le seul à avoir accès au système de contrôle des équipements et au composant chargé de réveiller les équipements distants : le composant `WakeOnLan`.

7.4.4 Discussion

Cette section présente la mise en œuvre de l'approche décrite dans le Chapitre 5 et le Chapitre 6. Pour cela, *HomeNap* se base sur différents canevas logiciels. L'utilisation de canevas logiciels matures permet d'envisager une adoption plus rapide de l'approche dans la maison numérique. Ils considèrent notamment certaines propriétés de cet environnement. Par exemple, Java permet de s'abstraire des équipements, cachant ainsi une partie de l'hétérogénéité de la maison numérique. UPnP permet de prendre en compte la volatilité de cet environnement. Enfin, les fichiers de description des composants et des équipements décrivent les hétérogénéités en termes de ressources matérielles et de consommation des équipements.

Cependant, certains canevas paraissent moins en phase avec ceux qui se trouvent dans une maison numérique. Par exemple, un solveur de contraintes est plus souvent utilisé au sein d'un centre de traitement de données que d'une maison numérique. Cela est notamment dû aux grandes quantités de processeur qu'il mobilise afin de résoudre un calcul le plus rapidement possible.

Enfin, l'usage d'une méthode de réveil unique est peu représentatif de l'hétérogénéité des méthodes de réveil présentes dans cet environnement. À l'heure des communications sans fil, il est nécessaire d'intégrer d'autres méthodes de réveils, telle que le *Wake-on-Wireless LAN*, afin de considérer davantage d'équipements.

Transfert vers l'industrie

La plupart des canevas logiciels présentés dans cette section se retrouvent déjà ou sont en passe de se retrouver dans les équipements de la maison numérique. Par exemple, Java se retrouve dans le système d'exploitation des téléphones intelligents fonctionnant sous Android. De même, les fournisseurs d'accès Internet travaillent à intégrer OSGi à leurs équipements. Aussi le choix de ces technologies est en accord avec l'existant bien qu'elles ne soient pas actuellement déployées sur tous les équipements de cet environnement.

Cette approche se heurte toutefois au passage dans la vie courante. Comme nous l'avons expliqué auparavant, il est nécessaire d'installer un gestionnaire sur les équipements. Or les équipementiers ne proposent pas toujours des équipements ouverts à des architectes tiers, notamment pour y installer des applications. Il faut donc que les équipementiers intègrent des gestionnaires pour que leurs équipements soient considérés lors de l'optimisation.

Du côté des fournisseurs de services, cette approche nécessite de repenser la manière dont sont développés les services. Notamment, les composants avec états nécessitent qu'*HomeNap* puisse sauvegarder et restaurer leur état. Pour cela, il faut que les architectes définissent les variables qu'il faut conserver et qu'ils fournissent les moyens de restaurer les états à partir des variables. L'utilisation de patrons de conception dédiés à cet aspect est à envisager afin de répandre plus facilement cette nouvelle manière de développer.

Dans les deux cas, pour les équipementiers et les architectes, le passage par des organismes de standardisation permettrait d'accélérer l'adoption de cette approche. Dans le cas contraire, il est peu probable qu'elle apparaisse dans la maison numérique comme un moyen de réduire la consommation d'énergie.

7.5 Conclusion

Ce chapitre décrit *HomeNap*, son implémentation ainsi que les mécanismes de migration qu'il intègre. L'implémentation d'*HomeNap* se base sur différents canevas logiciels. Ces canevas logiciels considèrent diverses problématiques, *e.g.,* abstraction de la couche matérielle pour Java, gestion du cycle de vie pour OSGi. Ces canevas considèrent également certaines propriétés qui nous intéressent, *e.g.,* l'hétérogénéité grâce à Java, la volatilité grâce à UPnP.

Toutefois, aucun de ces canevas ne considère ou n'est conçu pour minimiser sa consommation d'énergie. Aussi, bien qu'*HomeNap* soit conçu pour avoir un faible impact énergétique sur l'environnement, les outils sur lesquels il se base n'ont pas été développés dans cette optique. Le prochain chapitre cherche donc à estimer les performances énergétiques de l'approche proposée au travers de l'implémentation *HomeNap*.

Ce chapitre détaille également l'extension du modèle. L'extension du modèle définit la loi de consommation énergétique des équipements, la fonction objectif ainsi que les contraintes sur lesquelles les évaluations du chapitre suivant se basent. Ce modèle est ensuite inséré dans un solveur de contraintes qui recherche toujours la solution la plus efficiente énergétiquement. Cependant, suivant la taille du plan de répartition et les contraintes considérées, cette recherche de la solution optimale prend un certain temps. Aussi, le prochain chapitre cherche à mesurer les performances de l'algorithme d'optimisation du plan de répartition.

Chapitre 8

Évaluation

Le vif est la puissance la plus strictement individuelle de chacun. Il tient du néphèsh, ce vent vital qui circule en nous, qui nous fait ce que nous sommes. Rien ne peut s'y mêler. Il est pur, insécable et automoteur. Il peut seulement se disperser si sa vitesse vient à décliner, il peut s'ajouter à un autre vif, mais pas fusionner...

– Orosihi, *La Horde du Contrevent*

Sommaire

8.1 Introduction 134
8.2 Évaluation de l'optimisation 135
8.2.1 Performance de l'optimisation 135
8.2.2 Dette énergétique de l'optimisation 137
8.2.3 Discussion 139
8.3 Évaluation de l'approche 141
8.3.1 Évaluation par simulation 141
8.3.2 Évaluation pratique 144
8.3.3 Discussion 148
8.4 Conclusion 149

8.1 Introduction

Ce chapitre évalue l'approche au travers de l'implémentation *HomeNap*. Cette approche cherchant à réduire la consommation d'énergie, il ne faut pas qu'*HomeNap* consomme davantage d'énergie qu'il n'en fait gagner. La dette énergétique engendrée par *HomeNap* doit donc être remboursée et produire des gains énergétiques positifs.

Aussi, dans un premier temps, nous mesurons le temps de calcul et l'énergie dépensée par l'algorithme d'optimisation du plan de répartition. Ces deux critères influent sur l'efficience énergétique de la solution : plus longue est la recherche du plan de répartition optimisé, et donc l'énergie dépensée pour trouver la solution optimale, plus la dette est importante.

Le temps de calcul est évalué en modifiant les paramètres de l'algorithme d'optimisation, *i.e.*, les dimensions du plan de répartition ainsi que les contraintes. L'énergie dépensée est évaluée en effectuant le même calcul sur différents équipements, *e.g.*, ordinateur portable, ordinateur fixe, proches ou assimilables à des équipements se retrouvant dans une maison numérique.

Par la suite, nous évaluons l'approche au travers de scénarios concrets issus de la vie courante des ménages. Une évaluation par simulation puis une évaluation pratique, plus restreinte techniquement, sont mises en pratique. Ces évaluations démontrent que des gains énergétiques sont réalisés malgré la dette énergétique de l'approche, et notamment par son algorithme d'optimisation.

Ces évaluations valident les propriétés de conservation de la qualité de service par l'algorithme d'optimisation, de la prise en compte de la volatilité par l'architecture ainsi que de l'amélioration de l'efficience énergétique de l'approche.

Le Tableau 8.1 décrit les équipements utilisés pour l'ensemble des évaluations. Nous considérons que ces équipements sont représentatifs ou équivalents à des équipements d'une maison numérique. Par exemple, la BeagleBoard utilise un processeur ARM. Cet équipement est donc assimilé à un équipement basse consommation, *e.g.*, téléphone intelligent. De même, l'ordinateur fixe est assimilable à une console de jeu vidéo.

Structure du chapitre

Le chapitre s'articule comme suit : la Section 8.2 évalue l'algorithme d'optimisation du plan de répartition suite à un événement significatif. Cette évaluation concerne le temps de calcul de la solution optimale ainsi que l'énergie dépensée pour y parvenir. La Section 8.3 évalue *HomeNap* au travers de deux scénarios permettant une évaluation par simulation et une évaluation pratique. Enfin, la Section 8.4 conclut ce chapitre.

Équipements	État actif (W)	État de basse consommation (W)
Ordinateur fixe sans écran (Dell Precision T3400)	80 – 122	4,1
Ordinateur Portable 1 (Dell Latitude E6400)	25 – 48	1,6
Ordinateur Portable 2 (Dell Latitude D430)	23 – 33	2
BeagleBoard avec hub auto-alimenté	8 – 10	-

TABLE 8.1 – **Équipements utilisés pour les évaluations.** Ce tableau présente les équipements utilisés lors des différentes évaluations de ce chapitre. Il détaille leur consommation dans l'état actif (plage de consommation) et dans l'état de basse consommation (état G1/S1 de l'ACPI).

8.2 Évaluation de l'optimisation

La validation de l'optimisation considère le modèle décrit dans le Chapitre 5 et son extension décrite dans le Chapitre 7. Dans cette section, nous cherchons à valider ce modèle au travers des performances de l'algorithme en faisant varier le nombre d'équipements ou de grappes de composants mais également en l'exécutant sur différents équipements issus de la maison numérique.

8.2.1 Performance de l'optimisation

La première évaluation de l'algorithme porte sur le temps de calcul nécessaire pour trouver un plan de répartition optimisé. Cette évaluation est effectuée sur un unique équipement mais avec différents paramètres d'entrée pour l'algorithme, *i.e.*, nombre d'équipements, nombre de grappes de composants.

Contexte d'évaluation

Le nombre de solutions possibles, *i.e.*, plans de répartition réduisant la consommation d'énergie, varie en fonction du nombre d'équipements et du nombre de grappes considérées. Plus il y a de solutions possibles, plus le temps de recherche de la solution optimale augmente. Ainsi, les performances de l'algorithme d'optimisation du plan de répartition varient en fonction des dimensions du plan de répartition mis à jour.

Cependant, l'optimisation dépend aussi des contraintes utilisées par le solveur de contraintes. Par exemple, les contraintes sur les ressources matérielles (cf. Section 7.2) sont représentées par un problème de *bin packing*, *i.e.*, problème NP-difficile. L'ajout d'une dimension au plan de répartition, *i.e.*, un nouvel équipement ou une grappe, augmente exponentiellement le temps de calcul de l'optimisation.

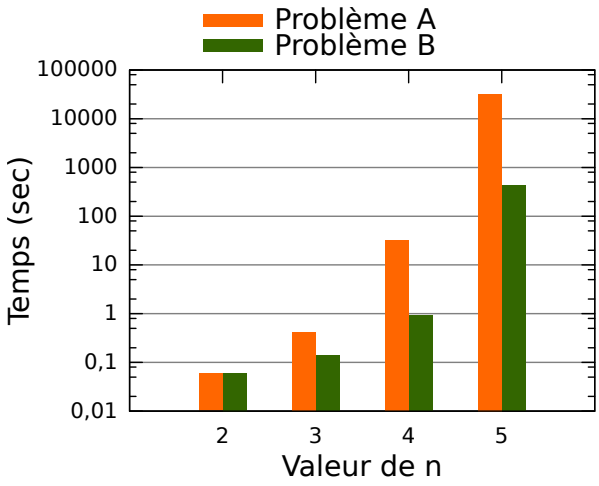

FIGURE 8.1 – **Temps de calcul de l'algorithme d'optimisation.** Le temps de calcul du plan de répartition optimisé dépend du nombre d'équipements et de grappes considérés, ainsi que des contraintes utilisées. Dans cette évaluation, le nombre de combinaisons suit une loi en n^{3n}.

Pour évaluer l'algorithme d'optimisation, nous procédons au calcul du plan de répartition optimisé suivant différentes dimensions sur un même équipement, *i.e.*, l'ordinateur fixe. Pour cela, nous faisons varier le nombre d'éléments considérés, *i.e.*, équipements et grappes, ainsi que les contraintes. Le nombre d'équipements augmente de 1 à chaque nouvelle mesure et le nombre de grappes augmente de 3. Ainsi, le nombre maximum de combinaisons à considérer à chaque mesure, lorsque les contraintes ne sont pas prises en compte, est de n^{3n}, avec $n \geq 2$ puisqu'il faut au moins deux équipements pour que l'algorithme fonctionne. Par la suite, nous observons le temps nécessaire à la résolution du problème d'optimisation.

Dans un premier temps, l'algorithme doit résoudre un problème d'optimisation sous-contraint, problème A, *i.e.*, le nombre de solutions possibles tend vers le nombre de combinaisons existantes lorsqu'il n'y a aucune contrainte. Pour cela, les équipements ont plus de ressources matérielles qu'il n'en faut pour héberger l'ensemble des composants.

Dans un deuxième temps, l'algorithme doit résoudre un problème presque sur-contraint, problème B, *i.e.*, le nombre de solutions possibles tend vers 0 en faisant en sorte que les équipements ne possèdent pas assez de ressources pour héberger tous les composants.

Résultats

La Figure 8.1 présente les résultats de l'évaluation de l'algorithme avec des dimensions et des contraintes différentes. Dans les deux cas, les résultats montrent que le temps de calcul croît exponentiellement avec le nombre d'éléments considérés (échelle de temps logarithmique). Toutefois, la durée de recherche de la solution optimale est beaucoup plus faible dans le cas où les contraintes sont plus fortes, *i.e.*, il existe moins de solutions possibles.

Dans les deux cas, lorsque n dépasse 4, *i.e.*, $\sim 1,6 \times 10^7$ combinaisons hors contraintes, la recherche de la solution optimale excède la minute. Or, plus cette durée augmente, plus la probabilité qu'un nouvel événement survienne nécessitant un nouveau calcul du plan de répartition augmente. Bien que l'utilisation des grappes de composants amène à réduire le nombre d'éléments considérés afin de trouver un résultat, excéder 12 grappes différentes ayant peu de contraintes pour 4 équipements différents rend le calcul inadapté à l'environnement.

Toutefois, l'ajout de contraintes plus fortes permet à l'algorithme de trouver plus rapidement une solution puisque qu'il existe moins de solutions possibles. Aussi, suivant les contraintes des composants, le temps de calcul de l'algorithme d'optimisation varie.

Cette évaluation montre que le système cherche à satisfaire les contraintes qui définissent la qualité de service d'une application tout en réduisant sa consommation d'énergie. Pour cela, il considère l'hétérogénéité, *i.e.*, ressources matérielles, usages, consommation d'énergie, au travers du modèle définit précédemment.

8.2.2 Dette énergétique de l'optimisation

La deuxième évaluation effectuée sur l'algorithme d'optimisation a trait à l'équipement qui effectue ce calcul. Comme le coordinateur est amené à migrer entre les équipements, et que les équipements de la maison numérique sont hétérogènes alors le temps de calcul et la dette énergétique diffèrent de l'un à l'autre. Pour cela, nous exécutons l'algorithme pour différentes valeurs de n sur différents équipements. Ces équipements sont décrits dans le Tableau 8.1.

Contexte d'évaluation

Afin de mesurer la consommation d'énergie de l'environnement d'évaluation, nous avons recours à un *PowerSpy*[29]. Le *PowerSpy* est une prise gigogne, *i.e.*, un appareil qui se branche sur une prise de courant et qui laisse une prise de courant apparente, permettant de mesurer la puissance de l'équipement branché dessus.

Le *PowerSpy* mesure la consommation à une fréquence d'une mesure par seconde. Ces données sont ensuite envoyées en temps réel en Bluetooth à un équipement en dehors de l'environnement d'évaluation pour être stockées et traitées.

Résultats

La Figure 8.2 présente les résultats de l'évaluation portant sur la dette énergétique de l'optimisation sur différents équipements. La dette énergétique est calculée comme suit : $(P_{e_{moy}} - P_{e_{min}}) \times \Delta_{opt}$ où $P_{e_{moy}}$ correspond à la consommation moyenne relevée pendant

29. http://www.alciom.com/fr/produits/powerspy2.html (2013)

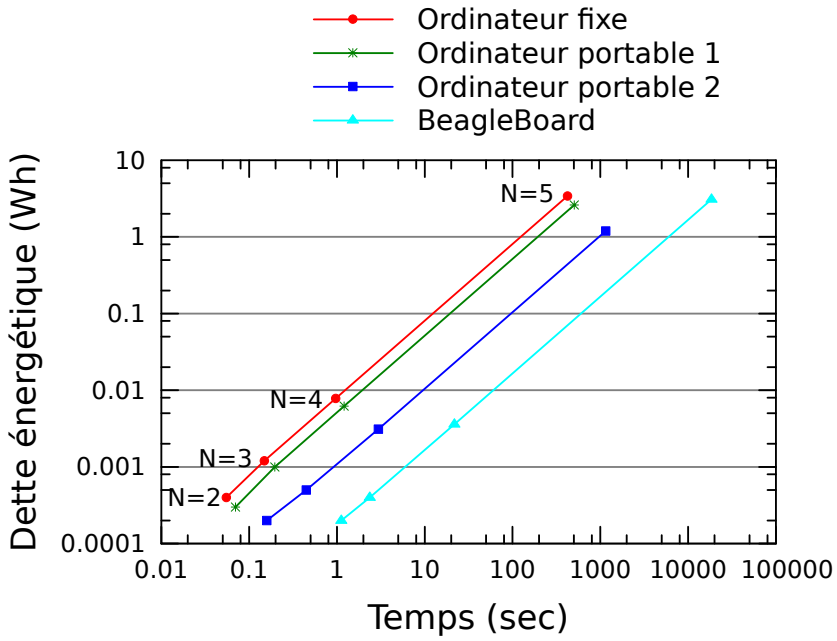

FIGURE 8.2 – **Dette énergétique de l'algorithme sur différents équipements.** La dette énergétique de l'algorithme diffère suivant l'équipement cherchant le plan de répartition optimisé. De même le temps de calcul diffère d'un équipement à un autre.

l'exécution de l'algorithme, $P_{e_{min}}$ est la consommation de l'équipement lorsqu'il est actif mais n'exécute aucun composant et enfin, Δ_{opt} est la durée de l'exécution de l'algorithme.

Les résultats montrent que l'ordinateur fixe est le plus rapide pour trouver le plan de répartition optimisé mais il n'est pas celui qui dépense le moins d'énergie pour y parvenir. Suivant la valeur de N, l'équipement le plus économe en énergie pour trouver le plan de répartition optimisé est soit la BeagleBoard, lorsque $n \leq 3$, soit l'ordinateur portable 2, lorsque $n \geq 4$.

Cette évaluation montre que la dette énergétique varie en fonction de l'équipement exécutant l'algorithme d'optimisation, pour les mêmes valeurs de n. Aussi, d'un point de vue énergétique, le coordinateur doit pouvoir migrer d'un équipement à un autre afin de rester le plus efficient énergétiquement.

Toutefois, si n est trop important, le temps de calcul augmente en conséquence. Aussi, il est nécessaire de trouver un compromis entre la dette énergétique du calcul du plan de répartition et le temps de calcul. En effet, pendant que l'algorithme cherche une solution, aucun gain énergétique ne s'opère. Bien que la dette énergétique soit limitée, sa compensation n'arrive alors que plus tardivement.

Aussi, le coordinateur doit pouvoir migrer mais pas toujours sur l'équipement le plus efficient énergétiquement. Il doit se placer sur l'équipement où le compromis entre la dette énergétique et le temps de calcul est le meilleur.

8.2.3 Discussion

Ces évaluations mettent en lumière le temps nécessaire à la recherche du plan de répartition optimisé et la dette énergétique engendrée. Le premier facteur impliquant une consommation d'énergie accrue est le temps passé à rechercher le plan de distribution optimisé. Ce temps de calcul augmente de manière exponentielle avec les dimensions du plan de répartition. Ainsi, plus il y a de dimensions au plan de répartition, plus le temps de recherche de la solution optimale augmente. Une solution pour résoudre ce problème est donc de diminuer le nombre de composants considérés grâce à l'utilisation de grappes de composants.

Le deuxième facteur influant sur le temps de calcul sont les contraintes. Plus les contraintes sont fortes, plus le temps de recherche de la solution optimale diminue. Dans ce cas, seules deux types de contraintes sont considérées : une contrainte sur les ressources matérielles et une contrainte sur la possibilité de déplacer un composant à l'exécution, notamment pour simuler la présence de l'utilisateur.

L'équipement fournissant le service doit également être pris en compte pour réduire la consommation d'énergie. Ces résultats montrent que bien que la dette énergétique soit minimale sur l'équipement qui consomme le moins, le temps de calcul s'accroît. Pour des plans de répartitions de plus grandes dimensions, les résultats montrent que ce n'est ni le plus performant, *i.e.,* l'ordinateur fixe, ni l'équipement qui consomme le moins, *i.e.,* la BeagleBoard, qui sont les plus efficients énergétiquement.

Pour résumer, afin de trouver le plan de répartition optimal dans un minimum de temps, il faut ajouter davantage de contraintes, *i.e.,* une modélisation plus fine de l'environnement, ou réduire le nombre d'éléments considérés, *i.e.,* utilisation des grappes de composants. Cependant, le trouver dans un minimum de temps ne garantit pas que ce service est effectué de manière efficiente énergétiquement. Il faut donc trouver un compromis entre le temps de calcul et l'énergie dépensée pour trouver le résultat.

Améliorer la modélisation de la maison numérique

L'environnement hétérogène de la maison numérique ne se borne pas aux deux seules contraintes utilisées dans cette évaluation. Par exemple, il y a des contraintes sur la localisation de l'équipement ou encore sur la nécessité pour deux composants différents d'être déployés sur le même équipement. Plus le modèle de la maison numérique est détaillé, plus les contraintes peuvent être ajoutées dans l'algorithme afin de réduire le temps de recherche d'une solution, *e.g.,* contraintes sur les communications.

Par contre, plus il y a de contraintes, plus le nombre de solutions possibles tend à diminuer. Ainsi, même si l'algorithme passe moins de temps dans la recherche d'une solution, il trouve alors moins souvent de solutions lorsqu'un événement survient. Aussi le choix des contraintes à utiliser doit être réfléchi plus avant afin de modéliser au mieux la maison numérique sans que la solution optimale soit écartée par des contraintes trop fortes.

Une stratégie est alors d'adapter les contraintes en fonction de l'environnement. Par exemple, si aucun utilisateur n'est présent dans la maison, la contrainte sur la présence de l'utilisateur n'est plus nécessaire et peut être retirée de l'algorithme temporairement, *i.e.*, jusqu'à ce que l'utilisateur revienne.

Ainsi, le modèle et l'architecture doivent être extensibles afin de facilement ajouter ou retirer des contraintes. Pour cela, l'architecture doit intégrer un mécanisme facilitant l'insertion ou le retrait de contraintes dans l'algorithme. L'utilisation d'un solveur de contraintes permet l'ajout ou la suppression de contraintes sans avoir à repenser la fonction objectif.

Solution optimale ou solution non optimale

L'algorithme d'optimisation n'est pas limité dans le temps afin de toujours trouver le plan de répartition optimisé. Le plan de répartition optimisé correspond à un optimum global, *i.e.*, la meilleure solution parmi l'ensemble des solutions possibles. Toutefois, trouver l'optimum global se fait au détriment du temps de calcul qui nécessite de l'énergie.

Nous pourrions chercher un optimum local, *i.e.*, une solution réduisant la consommation d'énergie mais pas forcément la plus optimale, afin de réduire le temps de calcul et ainsi l'énergie nécessaire à cette recherche. Cependant, cette solution non optimale ne rembourse pas nécessaire le temps passé à la chercher et à la mettre en application.

Aussi, deux choix sont possibles : soit le système cherche l'optimum global qui nécessite beaucoup de temps de calcul et donc d'énergie mais qui permet un gain énergétique maximal; soit le système cherche un optimum local dans un temps plus court et avec moins d'énergie mais qui permet un gain énergétique plus faible. De plus, dans les deux cas, il est possible qu'il n'existe aucune solution. Il se peut que la solution actuelle soit conservée puisqu'elle ne peut pas être optimisée. Le système effectue alors une recherche qui ne débouche sur aucun résultat, augmentant la dette énergétique.

Ce problème se rapproche d'un problème d'économie dans lequel le retour sur investissement doit être le plus élevé. Pour choisir quelle approche adopter, il faut savoir avant tout combien de temps s'écoule entre deux événements et donc deux optimisations. Or les événements sont supposés arriver de manière imprévisible (cf. Section 5.3). Cette dernière hypothèse ne reflète pas exactement la réalité puisque les utilisateurs ont des habitudes lorsqu'ils sont chez eux, limitant ainsi l'imprévisibilité. Ainsi, les événements se produisent à un instant donné avec une certaine probabilité.

À partir des probabilités de survenance d'un événement, *HomeNap* pourrait définir le temps à accorder à l'algorithme pour trouver une solution. L'utilisation d'algorithmes « à tout moment » [Zil96] permet alors de raffiner la solution, se rapprochant de la solution optimale, au fur et à mesure que le temps s'écoule.

Ainsi lorsque *HomeNap* sait qu'aucun événement ne va survenir pendant les prochaines heures, *e.g.*, cas du départ au travail/école des utilisateurs, il passe plus de temps afin de se rapprocher de l'optimum global puisqu'il sera remboursé plus facilement. Mais lorsque

les utilisateurs sont particulièrement actifs chez eux, *e.g.*, le soir ou le week-end, alors il cherche une solution moins optimale mais dont la dette énergétique est plus rapidement remboursée.

8.3 Évaluation de l'approche

L'évaluation de l'approche est scindée en deux parties. Nous nous intéressons d'abord à l'évaluation de l'approche au travers d'une simulation puis au travers d'une évaluation pratique. Nous faisons cette distinction car l'évaluation pratique étant relativement complexe à mettre en œuvre, elle n'est pas totalement représentative d'une maison numérique. En effet, il aurait fallu des équipements très différents sur lesquels installer *HomeNap*. Or les équipements du commerce ne permettent pas de le faire facilement. De plus, les composants qui sont utilisés dans l'évaluation pratique demandent à être développés d'une manière particulière. Cela limite le nombre de services que nous sommes aptes à développer, par manque de temps ou de ressources.

8.3.1 Évaluation par simulation

L'évaluation théorique propose de jouer un scénario comparable à un scénario de la vie courante d'une famille de 4 personnes. Ce scénario classique reprend des équipements et des services de la vie de tous les jours. Le scénario est construit en s'inspirant des résultats du comportement des utilisateurs décrits dans [Bea09] ainsi que des cas d'utilisations exprimés dans le projet « EconHome ».

Contexte d'évaluation

Pour parvenir à une simulation proche de la réalité, les valeurs de consommation énergétique des équipements sont celles d'équipements existants. De même, les services qui sont utilisés correspondent à des services qui se trouvent dans une maison numérique.

Pour jouer ce scénario, nous avons mis en place un simulateur prenant en compte un fichier décrivant le scénario. Il simule des événements significatifs qui sont envoyés au coordinateur. Le coordinateur est amené à chercher un plan de répartition optimisé.

Équipements utilisés. Afin de reconstituer un environnement peu différent d'une maison numérique, les équipements utilisés pour la validation sont des équipements se trouvant dans le commerce ou qui s'y apparentent. Le Tableau 8.2 décrit ces équipements ainsi que leur consommation d'énergie lorsqu'ils sont dans un état de basse consommation et lorsqu'ils sont dans l'état actif. Dans ce dernier état, leur plage de consommation est donnée. Elle est proportionnelle à leur charge processeur (cf. Section 7.2).

Équipements	État actif (W)	État de basse consommation (W)
DECT	2 – 3	1
Ordinateur fixe sans écran	80 – 128	5
Téléphone intelligent	1 – 4	1
Boîtier décodeur	20 – 30	3
Télévision	100 – 110	1

TABLE 8.2 – **Équipements utilisés pour la validation théorique.** Ce tableau décrit les équipements simulés utilisés afin d'évaluer l'approche.

Scénario d'évaluation. Afin de valider l'approche, *HomeNap* est évalué au travers d'un scénario impliquant une famille de quatre personnes (deux adultes et deux enfants). Le scénario, décrit en Figure 8.3, prend place un soir d'hiver et la veille d'un week-end [30].

Avant les événements. Pendant que les membres de la famille sont absents, la maison numérique est active. Un service de `Surveillance` est actif 24/7. Ce service détecte la présence d'intrus et alerte aussitôt les utilisateurs si nécessaire.

Événement 1. À 17 h, les premiers rentrés sont les enfants. Comme tous bons enfants qui se respectent lorsque leurs parents ne sont pas là, ils vont directement allumer l'ordinateur et la télévision. Ils lancent respectivement un service de `Navigation Web` et un service `Video` qu'ils vont consommer jusqu'à ce que le dîner soit prêt à 19 h 15.

Événement 2. À 18 h, l'un des parents rentre et pose son téléphone sur le chargeur sans fil de la famille. Ce téléphone s'annonce dans l'environnement domestique. Il annonce également qu'il héberge le service `Téléphonie`. Cet adulte commence à cuisiner.

Événement 3. À 18 h 15, l'enfant utilisant l'ordinateur lance le service `Téléchargement`. Le fichier qui est téléchargé est relativement volumineux (certainement un film en haute qualité ou un jeu vidéo, acheté légalement).

Événement 4. À 19 h 15, le second parent rentre du travail. À ce moment là, la famille se réunit autour de la table. Le service `Téléchargement` se poursuit mais les services `Vidéo` et `Navigation Web` sont arrêtés manuellement par les enfants.

Événement 5. À 20 h, une fois que la famille a terminé de manger, elle s'installe devant la télévision et lance le service `Vidéo` afin de regarder les actualités sur leur chaîne favorite. Bien que l'hiver débute à peine, le blocage des cyclistes reliant Grenoble à Lille par des chutes de neige importantes fait les gros titres du journal télévisé. À la suite de ce programme, la famille entière commence le visionnage d'un film.

30. L'auteur sait très bien que ce scénario n'est pas représentatif de l'ensemble des familles. Aussi, il convient de considérer ce scénario comme un exemple et non pas une généralité.

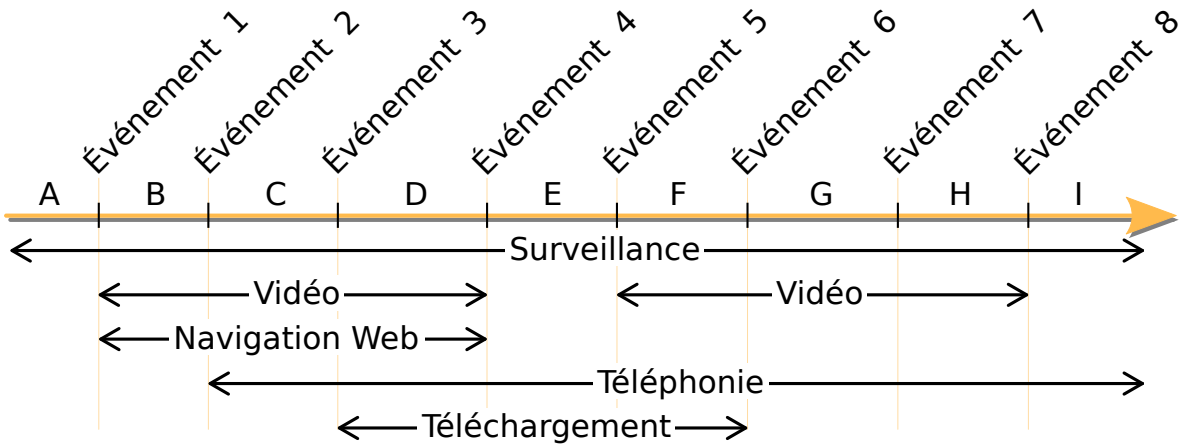

FIGURE 8.3 – **Scénario d'évaluation.** Ce scénario de 8 h décrit l'usage d'une maison numérique par une famille de quatre personnes (deux adultes et deux enfants).

Événement 6. À 22 h 25, le service `Téléchargement` se termine automatiquement.

Événement 7. À 22 h 45, alors que le film vient de se terminer, l'enfant qui avait lancé le téléchargement rallume fugacement l'ordinateur afin de vérifier le fichier. Il éteint ensuite cet équipement.

Événement 8. Enfin, à 23 h, l'ensemble de la famille se couche. Le service `Vidéo` est arrêté. À ce moment là, il ne reste plus que deux services actifs dans la maison numérique : la `Surveillance` et la `Téléphonie`.

Résultats

La Figure 8.4 présente les résultats de la simulation. Les courbes sont séparées en zone afin de discuter plus facilement des résultats. La courbe continue orange représente la consommation énergétique du scénario sans *HomeNap* tandis que la courbe en pointillés verts représente la consommation énergétique avec *HomeNap*.

Les zones dans lesquelles l'optimisation du plan de répartition est un succès sont les zones B, E et G. Dans les autres zones, l'optimisation a également lieu mais il n'existe pas un plan de répartition plus optimisé.

Ces résultats montrent que les deux courbes suivent les mêmes tendances puisqu'elles sont soumises aux même événements. Toutefois, des écarts se creusent dans les zones où l'optimisation a réussi, *i.e.*, les zones B, E et G. Ces écarts ont tendance à se répercuter dans les zones suivantes. C'est particulièrement visible pour la zone B puisque les résultats montrent un même écart entre les deux courbes dans les zones C et D. Ainsi, une optimisation réussie du plan de répartition a tendance à se répercuter au delà d'un unique événement menant à un gain qui est conservé jusqu'à ce qu'une autre optimisation réussie ait lieu.

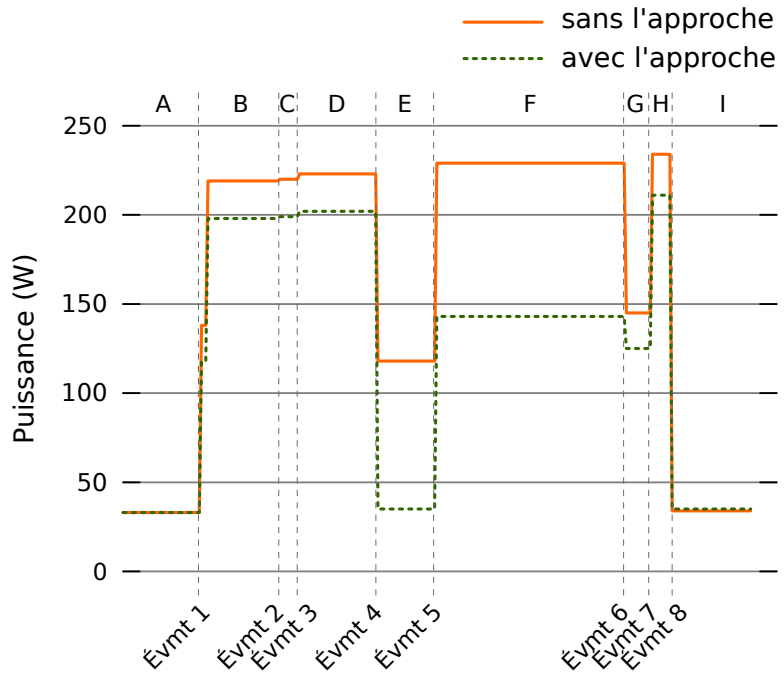

FIGURE 8.4 – **Résultats de la simulation.** Ces résultats montrent que des gains sont possibles en ne considérant que la répartition des applications sur les équipements de la maison numérique.

Toutefois, il s'agit d'une simulation. Elle ne prend pas en compte le temps de transfert des composants d'un équipement à un autre ainsi que les temps de passage en état de basse consommation et de réveil des équipements. Elle ne prend pas non plus en compte la dette énergétique de l'optimisation. Cette évaluation est une première étape permettant d'affirmer que l'approche utilisée est porteuse de résultats.

8.3.2 Évaluation pratique

L'évaluation pratique se heurte à l'ouverture aux tiers des équipements. Les équipements du commerce, et représentatifs de ceux se trouvant chez les ménages, sont souvent fermés aux développeurs tiers. Il n'est donc pas évident de mettre en place une évaluation utilisant ces équipements et ainsi rejouer le scénario de la Section 8.3.1 avec les équipements qui y sont décrits.

Pour contourner ce problème, nous utilisons des équipements ouverts, permettant d'installer facilement les composants qui nous sont utiles. Ce choix nous limite dans la représentativité des équipements qui sont utilisés pour l'évaluation. Le Tableau 8.1 décrit ces équipements. Nous considérons que ces équipements sont similaires ou assimilables à ceux qui se trouvent dans une maison numérique.

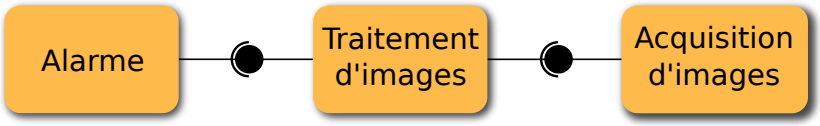

FIGURE 8.5 – **Service de surveillance.** Le service de surveillance est constitué de trois composants. Le composant `Acquisition d'Image` récupère les images issues d'une caméra. Le composant `Traitement d'Images` traite les images à la recherche d'un intrus. Enfin, le composant `Alarme` alerte l'utilisateur si un intrus est détecté.

Composants	Besoins
`Acquisition d'Image`	Caméra.
`Traitement d'Images`	500 MIPS de ressource processeur.
`Alarme`	Présence de l'utilisateur.
`Interface Utilisateur`	Présence de l'utilisateur.

TABLE 8.3 – **Composants utilisés pour l'évaluation pratique.** L'évaluation pratique considère ces quatre composants ainsi que ceux du gestionnaire, non décrits ici.

Contexte d'évaluation

Équipements et services utilisés. L'environnement de test décrit dans la Section 8.2 est réutilisé dans cette évaluation. Comme cette évaluation considère plusieurs équipements en même temps, nous branchons une multiprise sur le *PowerSpy*. C'est ensuite sur cette multiprise que sont connectés les équipements qui participent à cette expérience. Dans cette évaluation, nous considérons seulement trois équipements et neuf grappes composées d'un composant chacun.

Le service utilisé pour l'évaluation pratique est un service de surveillance. Ce service considère trois composants : (a) un composant `Acquisition d'Image`, (b) un composant `Traitement d'Images` et (c) un composant `Alarme` qui nécessite la présence de l'utilisateur. Un composant `Interface Utilisateur` est également utilisé. Enfin, les autres composants de l'expérience sont ceux du coordinateur qui est également considéré comme un service pouvant se déplacer d'un équipement à l'autre. Ces composants et leurs besoins sont décrits dans le Tableau 8.3. L'architecture de ce service est décrite dans la Figure 8.5.

Scénario d'évaluation. Afin de valider pratiquement l'approche, nous utilisons également un scénario. À cause des problèmes évoqués précédemment, il est beaucoup plus simple que le scénario exposé dans l'évaluation précédente.

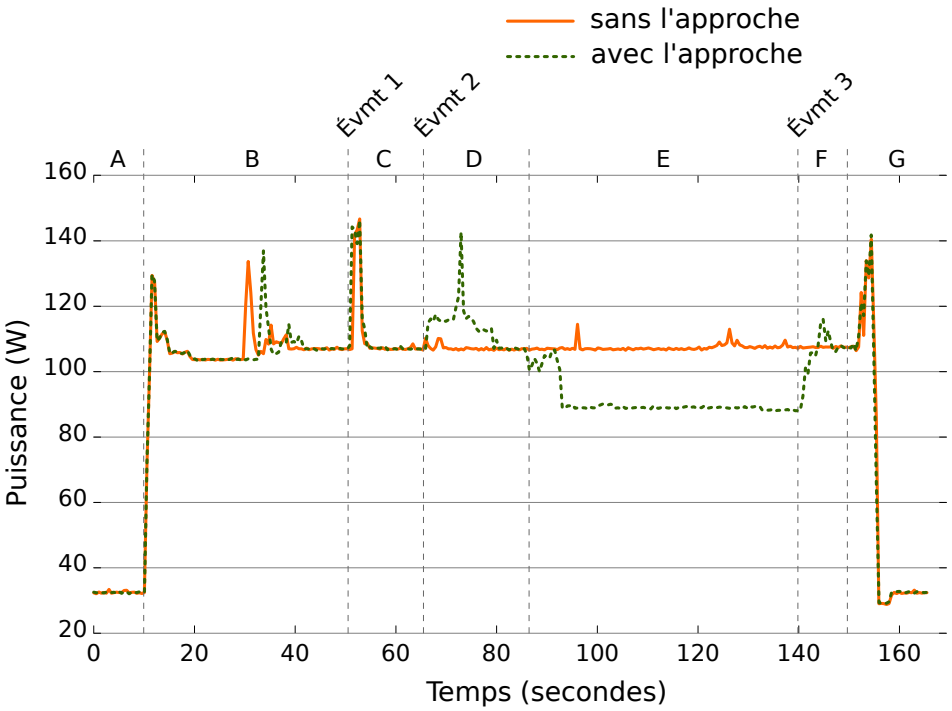

FIGURE 8.6 – **Résultats pratiques.** La dette énergétique (zone D) est complètement compensée par le retour sur investissement positif (zone E).

Avant les événements. Au démarrage, l'environnement numérique n'est constitué que d'un équipement x86 et d'une caméra. La BeagleBoard correspond à la caméra et exécute le composant `Acquisition d'Image`. L'ordinateur portable 2 exécute les composants `Traitement d'Images` et `Alarme`. Ce dernier équipement exécute également les composants du coordinateur.

Événement 1. Un utilisateur démarre l'ordinateur fixe.

Événement 2. L'utilisateur démarre le composant `Interface Utilisateur` sur l'ordinateur fixe.

Événement 3. L'utilisateur arrête le composant `Interface Utilisateur` et laisse l'ordinateur inactif.

Résultats

La Figure 8.6 présente les résultats de cette évaluation. La courbe continue orange présente la consommation d'énergie sans *HomeNap* tandis que la courbe en pointillés verts présente la consommation d'énergie du même scénario avec *HomeNap*. Pour comprendre ce qu'il s'est passé, nous reprenons le scénario en y ajoutant les actions menées par le coordinateur pour minimiser l'énergie.

Zone A. Il s'agit de l'état initial du système. Dans cette zone, seuls l'ordinateur portable et la BeagleBoard sont visibles par le coordinateur. Le coordinateur se trouve sur l'ordinateur portable. L'ordinateur fixe est en dehors du réseau et dans un état de basse consommation.

Zone B. L'utilisateur démarre l'ordinateur fixe. Son démarrage prend environ 40 secondes. Il n'est toujours pas visible sur le réseau par le coordinateur.

Zone C. Pour terminer son démarrage, l'ordinateur fixe lance son gestionnaire (le pic de la zone). Dès que le gestionnaire est lancé, il diffuse sur le réseau la présence de cet équipement (cf. événement 1 sur la figure). Dès lors que l'ordinateur fixe apparaît, le coordinateur démarre une optimisation. Cependant, l'apparition de l'ordinateur fixe ne change pas le plan de répartition car l'ordinateur est considéré comme inactif, *i.e.*, aucun composant n'est déployé. Ainsi, si l'utilisateur vient à le laisser dans cet état, l'ordinateur fixe passera en état de basse consommation.

Zone D. L'utilisateur démarre le composant `Interface Utilisateur` (cf. événement 2 sur la figure). Dès que ce composant démarre, une optimisation est lancée (zone D). Comme ce composant doit obligatoirement être déployé où se trouve l'utilisateur, *i.e.*, sur l'ordinateur fixe, le coordinateur ordonne aux composants présents sur l'ordinateur portable de migrer sur l'ordinateur fixe. Le composant présent sur la BeagleBoard ne peut pas se déplacer à cause de ses contraintes de déploiement le liant à la caméra.

Zone E. Suite à cette migration, l'ordinateur portable passe en état de basse consommation puisqu'il n'exécute plus aucun composant. L'ordinateur fixe et la BeagleBoard sont les deux seuls équipements actifs.

Zone F. L'utilisateur arrête le composant `Interface Utilisateur` (cf. événement 3 sur la figure), ce qui conduit à une nouvelle optimisation. Cette optimisation décide de réveiller l'ordinateur portable puis d'y migrer les composants présents sur l'ordinateur fixe.

Zone G. Comme l'ordinateur fixe est inactif, il passe en état de basse consommation.

Le Tableau 8.4 synthétise le scénario et les phases d'optimisation. Le gain énergétique est réalisé dans la zone E car l'ordinateur portable est dans un état de basse consommation. Malgré ce gain, les résultats montrent des pics de consommation qui apparaissent pendant la recherche d'un plan de répartition optimisé et de sa mise en application. Ces adaptations se produisent dans les zones D ($\sim$ 6 secondes) et F ($>$ 1 seconde).

La zone D augmente la consommation d'énergie de l'environnement par rapport à la consommation sans *HomeNap*. Cependant, la dette énergétique est compensée par le gain énergétique positif réalisé dans la zone E, amenant à une réduction de la consommation d'énergie sur l'ensemble du scénario. De plus, si l'utilisateur utilise plus longuement le composant `Interface Utilisateur` alors le gain énergétique augmente en proportion.

Zone	Description	Équipements actifs
A	État d'initial. Attente d'un nouvel événement	Ordinateur portable, BeagleBoard
B	Phase d'initialisation de l'ordinateur fixe	Tous
Événement 1 : Apparition de l'ordinateur fixe sur le réseau		
C	Échec de l'optimisation : pas de meilleur plan de répartition	Tous
Événement 2 : Démarrage du composant `Interface Utilisateur`		
D	Succès de l'optimisation & application du plan de répartition	Tous
E	Attente d'un nouvel événement	Ordinateur fixe, BeagleBoard
Événement 3 : Arrêt du composant `Interface Utilisateur`		
F	Succès de l'optimisation & application du plan de répartition	Tous
G	Attente d'un nouvel événement	Ordinateur portable, BeagleBoard

TABLE 8.4 – **Synthèse de l'évaluation pratique avec l'approche.**

Ces résultats montrent que la réactivité de l'architecture des systèmes ainsi que leur capacité à migrer d'un équipement à un autre minimise leur impact sur la consommation d'énergie de la maison numérique. Cela valide donc l'efficience énergétique de notre architecture. De même, cela valide la prise en compte de la volatilité par notre solution afin de produire une plan de répartition qui soit toujours le plus efficient énergétiquement.

8.3.3 Discussion

Ces évaluations mettent en perspective que les composants peuvent s'exécuter sur différents équipements tout en fournissant le même service. Grâce à ces migrations, l'efficience énergétique de la maison numérique est augmenté. Cela a pour effet de rendre des équipements inutilisés qui sont ensuite placés dans un état de basse consommation.

L'efficience énergétique est améliorée puisque pour un même service, moins d'énergie est consommée (cf. Définition 3). Ainsi, en plus de réduire la consommation d'énergie de la maison numérique, l'efficience énergétique d'un service est améliorée. Enfin, l'architecture limite sa consommation d'énergie en optimisant seulement sur les événements significatifs survenant dans l'environnement. De plus, grâce à la migration des composants d'*HomeNap*, ce dernier s'exécute toujours sur les équipements déjà utilisés par d'autres composants, augmentant d'autant plus l'efficience énergétique.

Les scénarios présentés ne concernent que de courtes périodes de temps, *i.e.*, de quelques minutes à quelques heures. Cependant, rien n'empêche *HomeNap* de s'exécuter toute l'année, 24 heures sur 24. Ainsi, bien que chaque optimisation ne génère pas forcément des gains

énergétiques importants, une utilisation prolongée permet d'augmenter son efficacité. Ainsi *HomeNap* joue le rôle d'un « collecteur d'énergie », se nourrissant de petits gains énergétiques engendrés par les habitudes des ménages.

Apprentissage des habitudes

Dans ces évaluations, les scénarios décrivent quelques heures dans la vie d'un ménage. Or, tous les ménages n'ont pas les mêmes habitudes, *i.e.*, ils ne possèdent pas les mêmes équipements et ne consomment pas les mêmes services. Aussi, les gains énergétiques présentés dans ces évaluations sont propres à ces scénarios. Dans la réalité, les gains varient d'un ménage à un autre, d'une maison numérique à une autre.

Il est donc nécessaire de prendre en considération les habitudes des ménages afin d'améliorer l'approche et ainsi d'en tirer des gains énergétiques plus importants. Aussi, proposer un modèle d'apprentissage complétant le modèle proposé dans ce document permettrait d'obtenir des plans de répartition plus optimisés et propres à chaque ménage.

Par exemple, l'apprentissage des habitudes des ménages évite que des optimisations aient lieu. Une optimisation est inutile, suite à un événement, car il existe une probabilité forte qu'un nouvel événement survienne peu de temps après, ne permettant pas de compenser la dette énergétique de l'optimisation si elle devait se produire. Il n'est donc pas nécessaire de créer une dette énergétique si le système prédit que le gain énergétique n'est pas positif.

Migration temporelle

À partir des habitudes des ménages, il est également envisageable de retarder l'exécution de certains services pour limiter la consommation d'énergie. Par exemple, si un équipement doit être réveillé spécialement pour héberger un composant mais que ce composant est facultatif alors son exécution peut être retardée pour éviter de réveiller exprès cet équipement.

La migration temporelle peut être décidée au travers du classement des services suivant leur importance pour le bon fonctionnement de la maison numérique, *e.g.*, service de surveillance, ou suivant leur importance pour l'utilisateur, *e.g.*, service de visionnage de film.

8.4 Conclusion

Ce chapitre décrit les évaluations du modèle défini dans le Chapitre 5 et de l'architecture proposée dans le Chapitre 6. La première partie de ce chapitre évalue l'algorithme optimisant le plan de répartition. Les résultats montrent que suivant les contraintes choisies, le temps de calcul d'un nouveau plan de répartition augmente plus ou moins vite. L'utilisation des grappes de composants ainsi que des contraintes correctement dimensionnées à la maison numérique limite ce temps de calcul.

La deuxième section de ce chapitre évalue l'implémentation *HomeNap* au travers de deux scénarios. Elle montre que le système déployé dans une maison numérique génère des gains énergétiques positifs. Cependant, ces gains énergétiques sont liés aux habitudes des ménages. Aussi, cela nous amène à proposer des pistes de réflexion pour améliorer l'approche.

Ces évaluations montrent également que l'usage des composants permet de répartir plus efficacement les services sur les équipements. La qualité de service est conservée, lorsqu'il s'agit d'un service fourni par un ensemble de composants, tout en consommant moins d'énergie. Cela a pour conséquence d'améliorer l'efficience énergétique du service et de réduire la consommation d'énergie de l'ensemble des équipements.

Toutefois, de nouvelles évaluations sont nécessaires afin de mesurer l'impact de ce système sur la qualité de service. Par exemple, la qualité de service est altérée lorsqu'un composant migre d'un équipement à un autre causant l'indisponibilité du service. Réduire ce temps d'indisponibilité participe à améliorer la qualité de service et l'acceptation de la solution.

Pour conclure la partie validation, le Chapitre 7 et le Chapitre 8 valident les propriétés décrites au début de cette partie :

Prise en compte de l'hétérogénéité. Cette propriété est prise en compte dans le modèle au travers de l'utilisation de contraintes décrivant l'environnement, *e.g.,* ressources matérielles, consommation d'énergie, équipements utilisés par l'utilisateur. Le modèle proposé se rapproche au plus près de la réalité de la maison numérique. L'algorithme d'optimisation propose une solution qui tient continuellement compte de ces contraintes dans la recherche du plan de répartition optimisé.

Prise en compte de la volatilité. Cette propriété est prise en compte au travers de la modélisation des événements significatifs, *i.e.,* apparition et disparition d'un équipement ou d'une application. Dans l'architecture, c'est au travers des informations remontées par les sous-entités Capteur des gestionnaires que le coordinateur est capable de modifier le plan de répartition en conséquence. La prise en compte dans le modèle et dans l'architecture de cette propriété permet au système de s'adapter à un événement significatif.

Conservation de la qualité de service. La description des besoins des services spécifie la qualité de service attendue d'un service. La satisfaction de ces besoins par l'algorithme d'optimisation veille à ce que la qualité de service soit toujours conservée face à l'efficience énergétique. Si les besoins d'un service ne sont pas satisfaits alors la solution proposée par l'algorithme n'est pas viable. Dans ce cas, il est préférable de conserver le plan de répartition courant plutôt que d'essayer de faire des économies d'énergie.

Amélioration de l'efficience énergétique. Malgré que la qualité de service soit la priorité d'*HomeNap*, il lui reste assez de marge de manœuvre pour trouver un plan de répartition consommant moins d'énergie. La recherche d'un plan de répartition optimisé

génère des gains énergétiques dans la maison numérique. De plus, l'efficience énergétique d'*HomeNap* est validée au travers de sa réactivité et de sa capacité à migrer d'un équipement à un autre.

Quatrième partie

Conclusion et Perspectives

Chapitre 9

Conclusion et Perspectives

L'énergie ne peut être créée ou détruite, elle peut seulement être transformée d'une forme à une autre.
– Albert Einstein

Sommaire

9.1 Synthèse 156
9.2 Contributions 157
9.2.1 Hétérogénéité 157
9.2.2 Volatilité 157
9.2.3 Qualité de service 157
9.2.4 Efficience énergétique 158
9.2.5 Publications 158
9.3 Perspectives 158
9.3.1 Perspectives à court terme 159
9.3.2 Perspectives à long terme 161

9.1 Synthèse

L'énergie est une ressource omniprésente et capitale pour la société moderne. Elle est source de progrès mais son utilisation est également source de dommages. Concevoir des services efficient énergétiquement participe au contrôle de cette ressource. À terme cela permet de réduire notre dépendance à cette ressource et de lutter contre ses effets secondaires, *e.g.,* pollution, conflits. Pour être efficace, cette réduction doit être entreprise par tous, les industriels comme les ménages, les fournisseurs de services comme ceux qui les consomment.

Toutefois, limiter notre consommation d'énergie ne passe pas seulement par la conception de services efficient énergétiquement. Cela passe également par une gestion « intelligente » de l'ensemble des services s'exécutant dans un environnement donné, quelque soit leur nombre et leur objectif fonctionnel. Aussi, les services doivent s'adapter à l'exécution à nos usages afin de limiter l'utilisation des ressources matérielles et donc l'énergie quelles consomment.

C'est pour cela que notre approche tend à s'adapter automatiquement à l'environnement de la maison numérique afin de réduire la consommation d'énergie. Cette réduction se fait en considérant les propriétés intrinsèques de cet environnement, *i.e.,* hétérogénéité, volatilité, qualité de service. Pour cela, nous avons proposé un modèle tenant compte de l'hétérogénéité des équipements, de la volatilité de l'environnement ou encore de la qualité de service requise par l'utilisateur.

Ce modèle est intégré dans une architecture logicielle s'adaptant à la volatilité de l'environnement. Cette adaptation permet de toujours proposer un plan de répartition efficient énergétiquement. Les équipements inutilisés, *i.e.*, qui n'exécutent aucun composant, passent dans un état de basse consommation, entraînant une réduction globale de la consommation d'énergie.

La validation montre qu'*HomeNap* réduit la consommation d'énergie de la maison numérique malgré sa dette énergétique. Toutefois, il est difficile d'estimer le gain réelle de notre approche pour des ménages. En effet, ces derniers ont des ensembles d'équipements ou des usages différents d'un ménage à un autre. Les gains énergétiques potentiels dépendent donc fortement des caractéristiques de chaque maison numérique.

De plus, les gains énergétiques qui en découlent peuvent paraître faibles au regard des autres postes de consommation de la maison, *i.e.*, chauffage, eau chaude sanitaire. Toutefois, notre « collecteur d'énergie », fonctionnant toute l'année, permet de petits gains énergétiques. Un petit gain énergétique tout au long de l'année et dans chacun des 26 millions de foyers permet de faire des économies substantielles à l'échelle d'un pays comme la France.

L'approche ne se limite pas à la seule maison numérique. Elle peut également s'appliquer au monde de l'entreprise qui utilise aussi des équipements hétérogènes pour différentes tâches, *e.g.,* administration, mercatique. Des économies d'énergie y sont aussi possibles. Toutefois le modèle doit être adapté pour rendre compte de la réalité de cet environnement.

Enfin, il est nécessaire de considérer l'impact que ces économies d'énergie peuvent avoir sur les utilisateurs. En effet, il a été démontré que des personnes réalisant des économies d'énergie, et donc des économies financières, ont tendances à réinvestir ces gains dans de nouveaux équipements. Par cette action, les utilisateurs consomment alors toujours autant d'énergie puisque leur parc d'équipement s'agrandit. Il s'agit de l'« effet rebond » [AGGD00].

9.2 Contributions

Dans cette section, nous synthétisons nos contributions présentées dans le Chapitre 5 et le Chapitre 6 et validées dans le Chapitre 7 et le Chapitre 8 au travers de l'implémentation de l'intergiciel *HomeNap*.

9.2.1 Hétérogénéité

Le modèle décrit plusieurs hétérogénéités : ressources matérielles, usages et consommation d'énergie. Ces hétérogénéités rendent compte des différences, parfois importantes, entre les équipements de la maison numérique. Elles permettent ainsi de proposer un plan de répartition des composants qui correspondent aux spécificités des équipements comme aux comportements des utilisateurs.

9.2.2 Volatilité

La volatilité est définie au travers des quatre événements significatifs pris en compte par le modèle : apparition d'un équipement, disparition d'un équipement, apparition d'une application, disparition d'un application. Ces événements modifient la taille du plan de répartition, diminuant potentiellement l'efficience énergétique de la maison numérique.

Les événements significatifs sont considérés à l'exécution grâce à une boucle autonome MAPE-K. Un composant, *i.e.*, la sous-entité Observation, est en charge d'écouter et de récupérer ces événements. Ce composant déclenche alors une optimisation du plan de répartition.

9.2.3 Qualité de service

La qualité de service est définie dans la spécification du service. Cette spécification conduit ensuite à définir des contraintes sur le placement des composants, *i.e.*, ressources matérielles, présence de l'utilisateur. La mise sous forme de contraintes permet à l'algorithme d'optimisation de les prendre en compte lorsqu'il recherche le plan de répartition le plus efficient énergétiquement.

9.2.4 Efficience énergétique

L'efficience énergétique est prise en compte dans le modèle au travers de la fonction objectif issue de la fonction d'utilité. La fonction d'utilité décrit l'ensemble des états de notre environnement réparti et la fonction objectif trouve l'état dans lequel le minimum d'énergie est consommé. La mise en œuvre de cet état revient à déplacer des composant logiciels d'un équipement à un autre et à mettre dans un état de basse consommation les équipements n'exécutant aucun composant. Cela permet d'améliorer l'efficience énergétique de la maison numérique.

HomeNap est lui même conçu pour être efficient énergétiquement. Pour cela, le système peut migrer d'un équipement à un autre : il est considéré comme n'importe quelle application dans la maison numérique. De plus, le système limite sa consommation d'énergie en étant réactif, *i.e.*, il est actif uniquement lorsqu'un événement significatif survient.

9.2.5 Publications

Les travaux présentés dans ce manuscrit, *i.e.*, le modèle ainsi que l'architecture, ont donné lieu aux publications suivantes :

- Rémi Druilhe, Anne Matthieu, Laurence Duchien et Romain Rouvoy. *La réduction de la consommation d'énergie dans les environnements domestiques répartis*. Conférence Française en Systèmes d'Exploitation, 2011 [DMDR11]
- Rémi Druilhe, Matthieu Anne, Jacques Pulou, Laurence Duchien et Lionel Seinturier. *Energy-driven Consolidation in Digital Home*. Symposium on Applied Computing, Software Engineering Aspects of Green Computing Track, 2013 [DAP+13b]
- Rémi Druilhe, Matthieu Anne, Jacques Pulou, Laurence Duchien et Lionel Seinturier. *Components mobility for energy efficiency of digital home*. Conférence Component-Based Software Engineering, 2013 [DAP+13a]

9.3 Perspectives

Ces travaux s'appuient sur plusieurs hypothèses décrites en détail dans le Chapitre 5. Ces hypothèses délimitent le champ de recherche ainsi que la contribution apportée. Elles simplifient également le problème posé. Toutefois, certaines hypothèses peuvent être levées au regard des travaux existants dans d'autres domaines ou des direction dans lesquelles l'industrie technologique s'oriente. Nous en donnons quelques exemples.

L'Hypothèse 1 stipule qu'un service est fourni par une seule et unique application sur un équipement. Cette hypothèse peut être levée car pour un même équipement il existe généralement plusieurs applications fournissant le même service. Par exemple, un service de navigation sur Internet est fourni par différentes applications, *e.g.*, Firefox, Chrome, qui peuvent

être déployées sur le même équipement. Dès lors, pour atteindre l'efficience énergétique, il est nécessaire de trouver l'application qui consomme le moins en fonction de l'équipement ciblé.

L'Hypothèse 5 limite le problème à la seule maison numérique et nécessite ainsi de prendre en compte tout le cycle de vie du service dans la recherche du placement des composants du service, *i.e.*, de sa création à sa consommation. Cependant, cette hypothèse peut également être levée. Les directions prisent tendent à déporter une partie de plus en plus conséquente d'un service dans des centres de traitement de données où la maintenance et l'évolution de l'application est plus aisée. Aussi, les travaux futurs doivent considérer ces nouveaux environnements dans la recherche de l'efficience énergétique globale.

La levée de certaines des hypothèses amène à l'enrichissement de la solution visant à réduire davantage la consommation d'énergie. Cette levée amène également à définir les orientations que nous envisageons concernant ces travaux. Aussi, nous présentons les perspectives, à court terme et à long terme, prolongeant ces travaux.

9.3.1 Perspectives à court terme

Enrichir le modèle

Notre modèle décrit l'environnement de la maison numérique ainsi que ses propriétés intrinsèques comme l'hétérogénéité ou la volatilité. Cependant, notre modèle peut encore être enrichi. Par exemple, le modèle ne tient pas compte du coût énergétique de la communication entre deux composants. Si deux composants communiquent beaucoup entre eux sans être localisés sur le même équipement, alors le coût énergétique de leurs échanges sur le réseau est plus important que lorsqu'ils sont localisés sur le même équipement.

Pour éviter cette dette énergétique des communications, le modèle doit la prendre en compte. Il faut alors définir une contraintes spécifiant que deux composants très communiquant doivent être placés sur le même équipement.

Nouvelles grappes de composants

La grappes de composants évoqués dans ces travaux cherchent à faire des groupements de composants en fonction de leur contraintes. Nous différencions les contraintes à valeurs quantitatives et les contraintes à valeurs énumérées. Cependant, d'autres groupements, sur d'autres aspects, sont envisageables.

Par exemple, il est possible de grouper les composants en fonction de leur fréquence d'utilisation. Certains composants sont plus souvent utilisés que d'autres. Suivant la fréquence d'utilisation, ces composants sont alors placés sur des équipements différents. Les équipements hébergeant les composants les moins utilisés peuvent alors passer dans un état de basse consommation. Lorsque ces composants sont appelés, les équipements repassent dans un état actif. Ainsi, le mandatement de service peut être considéré dans la recherche d'une solution.

Améliorer l'algorithme d'optimisation

La validation met en évidence les limites de l'algorithme d'optimisation, notamment celle du temps. Le temps de calcul augmente exponentiellement à cause de la taille du plan de répartition. Toutefois, il n'est peut être pas nécessaire de prendre en compte l'ensemble des ressources matérielles pour l'ensemble des composants. Par exemple, si nous prenons l'hypothèse que la mémoire vive est présente en grande quantité, alors pour placer un composant en requérant peu, il n'est peut être pas nécessaire de prendre en compte cette contrainte pour ce composant.

Il est également possible de changer le type d'algorithme utilisé pour trouver une solution. Une autre approche est d'utiliser des algorithmes à tout moment. Ces algorithmes permettent de trouver des solutions en considérant le facteur temps. Plus le temps passé pour chercher une solution est important, plus la solution est proche de l'optimum, si elle existe. Aussi, ce type d'algorithme trouve des solutions dans des périodes de temps qui varient.

Architecture extensible

Enrichir le modèle de la maison numérique implique d'ajouter de nouvelles contraintes dans l'algorithme d'optimisation. Cela permet ainsi de proposer des plans de répartition qui tiennent mieux compte de l'hétérogénéité de cet environnement. Toutefois, avec l'ajout de nouvelles contraintes, le nombre de solutions possibles diminue.

Ainsi, pouvoir ajouter ou retirer des contraintes en fonction du contexte, *e.g.*, période de la journée, nombres d'utilisateurs, permet d'adapter le modèle en conséquence sans pénaliser les gains énergétiques potentiels. Il faut alors ajouter un mécanisme permettant l'ajout ou le retranchement de contraintes avant l'exécution de l'algorithme d'optimisation.

Meilleure gestion de la volatilité

La volatilité de la maison numérique n'est pas la même à chaque heure de la journée. Il existe des périodes où les utilisateurs sont très actifs chez eux et sont donc amenés à lancer ou arrêter plusieurs services ou encore démarrer des équipements dans un laps de temps court, *e.g.*, le soir ou le week-end. Au contraire, il existe des périodes où les utilisateurs sont peu présents chez eux, *e.g.*, en journée et où le nombre d'événements significatifs est faible.

L'adaptation du système en fonction de ces périodes amène à avoir une stratégie à court terme, *i.e.*, lorsqu'il y a beaucoup d'événements significatifs, et à long terme, *i.e.*, lorsque la durée entre deux événements est grande. La stratégie à court terme tend à proposer des plans de répartition rapidement déployables avec des gains énergétiques minimes. Au contraire, la stratégie à long terme tend à toujours proposer le gain énergétique maximum mais demande plus de temps pour être trouvée.

Considérer davantage de pannes

Seules trois pannes sont évoquées dans ce document : disparition du coordinateur, défaillance d'un composant, défaillance d'un équipement. Ces trois pannes ne couvrent pas l'ensemble des pannes susceptibles de survenir dans la maison numérique. Par exemple, un équipement peut disparaître avant la migration des composants qu'il exécute sur un nouvel équipement. Cette panne amène une dégradation de la qualité de service.

Aussi, il est nécessaire de considérer davantage de cas de pannes et de déterminer si les résultats d'*HomeNap* en sont altérés. Il est notamment nécessaire de déterminer si la dette énergétique augmente avec la mise en place de mécanismes permettant l'auto-guérison de notre système.

9.3.2 Perspectives à long terme

Apprentissage des habitudes

La validation a montré que le gain énergétique dépend des habitudes des utilisateurs. Aussi, en apprenant de leurs habitudes, nous pensons qu'il est possible d'adapter plus finement le comportement de notre approche afin d'améliorer les gains énergétiques existants. Par exemple, cela évite de lancer des optimisations car la probabilité qu'un nouvel événement survienne dans peu de temps rend obsolète l'optimisation en cours. Aussi, autant ne pas commencer cette optimisation pour éviter d'accumuler de la dette énergétique si elle ne peut pas être remboursée ultérieurement.

Migration temporelle

Dans notre approche, nous avons seulement considéré l'instant présent sans nous préoccuper des services qui peuvent être décalés dans le temps. Par exemple, il est possible qu'un service soit retardé afin d'éviter de réveiller un équipement pour exécuter cet unique service. Il s'agit de le migration temporelle puisque la fourniture du service est retardée.

Toutefois, cela ne doit pas aller à l'encontre de la qualité de service. Aussi, il est nécessaire de spécifier si ce service peut être décalé dans le temps afin de satisfaire à la réduction de la consommation d'énergie. Cela passe donc par un nouveau type de contrainte à ajouter dans le modèle qui considère l'aspect temporel de l'exécution d'un service.

Transfert vers l'industrie

L'approche que nous proposons implique plusieurs acteurs, *i.e.,* les équipementiers, les utilisateurs et les fournisseurs de services. Pour voir arriver cette solution sur le marché, il est donc nécessaire d'avoir une coordination de l'ensemble de ces acteurs. Pour cela, il faut enrichir les systèmes d'exploitation actuels pour leur intégrer les fonctionnalités du gestionnaire. Aussi, le passage par un organisme de standardisation nous parait le plus simple pour voir un jour cette solution dans nos maisons.

Le deuxième point permettant un meilleur transfert vers l'industrie est la nécessité de produire des services qui sont conçus pour fonctionner avec cette approche. Dans notre cas, nous avons supposé qu'il était possible de sauvegarder les états de certains composants afin de les déplacer puis de les exécuter sur un équipement en restaurant leur état. Or, les développements actuels ne se concentrent pas sur cet aspect. Il est donc nécessaire de fournir des patrons de conception afin que les développeurs puissent facilement concevoir ce type de composants et les services qui en découlent.

Estimer le coût énergétique minimal d'un service

Nous avons montré que certains des services peuvent être fournis pour un coût énergétique moindre. Mais nous n'avons pas défini l'énergie minimale pour rendre un service. Aussi, il peut être intéressant de calculer pour chaque service, en se basant uniquement sur les critères qui le définissent, le minimum d'énergie qu'il peut consommer.

Au même titre qu'il existe des concours où l'objectif est de parcourir le plus de kilomètres avec le minimum d'essence, il est possible d'imaginer un concours qui cherche à fournir des services avec un minimum d'énergie. Les kilomètres à parcourir sont alors les critères à satisfaire pour considérer qu'il s'agit du service souhaité. Et pour éviter la conception d'équipements dédiés à un service, il faut alors que différents services puissent s'exécuter sur le même équipement.

Les consommations énergétiques qui en résultent peuvent ensuite être définies comme des limites atteignables pour arrêter l'algorithme d'optimisation. Couplé à un algorithme à tout moment, cela évite de consacrer du temps supplémentaire pour trouver une solution à peine plus efficiente énergétiquement. Par exemple, cela permet d'arrêter l'algorithme lorsqu'il est proche de l'optimum mais nécessite encore beaucoup de temps pour y parvenir.

Bibliographie

[AGGD00] Lorna A Greening, David L Greene, and Carmen Difiglio. Energy efficiency and consumption—the rebound effect—a survey. *Energy policy*, 28(6):389–401, 2000. *cité page 157*

[AHC+09] Yuvraj Agarwal, Steve Hodges, Ranveer Chandra, James Scott, Paramvir Bahl, and Rajesh Gupta. Somniloquy: augmenting network interfaces to reduce pc energy usage. In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 365–380, 2009. *cité page 46*

[Apt03] K.R. Apt. *Principles of constraint programming*. Cambridge Univ Pr, 2003. *cité page 123*

[ASG10] Yuvraj Agarwal, Stefan Savage, and Rajesh Gupta. Sleepserver: A software-only approach for reducing the energy consumption of pcs within enterprise environments. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, pages 22–22. USENIX Association, 2010. *cité page 46*

[BA06] Kenneth C Barr and Krste Asanović. Energy-aware lossless data compression. *ACM Transactions on Computer Systems*, 24(3):250–291, 2006. *cité page 38*

[BBDM00] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. A survey of design techniques for system-level dynamic power management. *IEEE Transactions on Very Large Scale Integration Systems*, 8(3):299–316, 2000. *cité page 34*

[BCL+06] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The fractal component model and its support in java. *Software: Practice and Experience*, 36(11-12):1257–1284, 2006. *cité page 18*

[Bea09] Thomas Beauvisage. Computer usage in daily life. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 575–584. ACM, 2009. *2 citations pages 47 et 141*

[BH07] Luiz André Barroso and Urs Holzle. The case for energy-proportional computing. *Computer*, 40(12):33–37, 2007. *4 citations pages 35, 36, 69, et 116*

[BHRS98] J. Baumann, F. Hohl, K. Rothermel, and M. Straßer. Mole–concepts of a mobile agent system. *World Wide Web*, 1(3):123–137, 1998. *cité page 117*

[BL07] Hongyu Pei Breivold and Magnus Larsson. Component-based and service-oriented software engineering: Key concepts and principles. In *33rd EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 13–20. IEEE, 2007. *cité page 21*

[BSGR03] Christian Becker, Gregor Schiele, Holger Gubbels, and Kurt Rothermel. Base-a micro-broker-based middleware for pervasive computing. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, pages 443–451. IEEE, 2003. *cité page 55*

[BW97] Martin Buchi and Wolfgang Weck. A plea for grey-box components. *Turku Centre for Computer Science*, 1997. *cité page 19*

[CDK+02] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing*, 6(2):86–93, 2002. *cité page 20*

[CFH+98] Antonio Carzaniga, Alfonso Fuggetta, Richard S Hall, Dennis Heimbigner, Andre Van Der Hoek, and Alexander L Wolf. A characterization framework for software deployment technologies. Technical report, DTIC Document, 1998. *cité page 18*

[CGD12] Chiffres Clés de l'Énergie, édition 2012. Technical report, Commissariat Général du Développement Durable, Décembre 2012. *cité page 4*

[CH04] Humberto Cervantes and Richard S Hall. Autonomous adaptation to dynamic availability using a service-oriented component model. In *Proceedings of the 26th International Conference on Software Engineering*, pages 614–623. IEEE Computer Society, 2004. *cité page 22*

[CH10] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, pages 21–21, 2010. *2 citations pages 36 et 115*

[CJGJ96] E.G. Coffman Jr, M.R. Garey, and D.S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996. *3 citations pages 77, 175, et 176*

[CK09] Hyunsuk Cho and Oh-Kyong Kwon. A backlight dimming algorithm for low power and high image quality lcd applications. *IEEE Transactions on Consumer Electronics*, 55(2):839–844, 2009. *cité page 37*

[CL12] Aidan Cully and Oleg Logvinov. Optimizing power consumption in networked devices. In *16th IEEE International Symposium on Power Line Communications and Its Applications*, pages 248–255. IEEE, 2012. *cité page 43*

[Com03] A. Computing. An architectural blueprint for autonomic computing. *IBM White Paper*, 2003. *cité page 24*

[CZSL12] Michael Cohen, Haitao Steve Zhu, Emgin Ezgi Senem, and Yu David Liu. Energy types. In *Proceedings of the ACM international conference on Object Oriented Programming Systems Languages and Applications*, pages 831–850. ACM, 2012. *cité page 38*

[DAP+13a] Rémi Druilhe, Matthieu Anne, Jacques Pulou, Laurence Duchien, and Lionel Seinturier. Components mobility for energy efficiency of digital home. In *Component-Based Software Engineering*, 2013. *cité page 158*

[DAP+13b] Rémi Druilhe, Matthieu Anne, Jacques Pulou, Laurence Duchien, and Lionel Seinturier. Energy-driven Consolidation in Digital Home. In *Symposium on Applied Computing, Software Engineering Aspects of Green Computing Track*, 2013. *cité page 158*

[Dar06] Sarah Darby. The effectiveness of feedback on energy consumption. *A Review for DEFRA of the Literature on Metering, Billing and direct Displays*, 486, 2006. *cité page 47*

[dev06] Face au changement climatique, agissons ensemble. Technical report, Ministère de l'écologie, du développement durable et de l'énergie, Novembre 2006. *cité page 4*

[DKM04] Alan Dearle, Graham NC Kirby, and Andrew J McCarthy. A framework for constraint-based development and autonomic management of distributed applications. In *Proceedings of the International Conference on Autonomic Computing*, pages 300–301. IEEE, 2004. *cité page 23*

[DMDR11] Rémi Druilhe, Anne Matthieu, Laurence Duchien, and Romain Rouvoy. La réduction de la consommation d'énergie dans les environnements domestiques répartis. In *Conférence Française sur les Systèmes d'Exploitation*, 2011. *cité page 158*

[Dyc90] Harald Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44(2):145–159, 1990. *cité page 175*

[EHL07] Clément Escoffier, Richard S Hall, and Philippe Lalanda. ipojo: An extensible service-oriented component framework. In *IEEE International Conference on Services Computing*, pages 474–481. IEEE, 2007. *cité page 122*

[ERKR06] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan. Full-system power analysis and modeling for server environments. In *Proceedings of Workshop on Modeling, Benchmarking, and Simulation*, pages 70–77. Citeseer, 2006. *2 citations pages 78 et 111*

[Fis08] Corinna Fischer. Feedback on household electricity consumption: a tool for saving energy? *Energy efficiency*, 1(1):79–104, 2008. *cité page 47*

[FS99] Jason Flinn and Mahadev Satyanarayanan. Energy-aware adaptation for mobile applications. *ACM SIGOPS Operating Systems Review*, 33(5):48–63, 1999. *cité page 38*

[Fur12] Samuele Furfari. *Politique et Géopolitique de l'Énergie: Une Analyse des Tensions Internationales au XXI^{ème} siècle*, volume 1. Editions Technip, 2012. *cité page 3*

[GDPR12] Soguy Mak Karé Gueye, Noel De Palma, and Eric Rutten. Coordinating energy-aware administration loops using discrete control. In *The Eighth International Conference on Autonomic and Autonomous Systems*, pages 99–106, 2012. *cité page 23*

[Hal99] Richard Scott Hall. *Agent-based software configuration and deployment*. PhD thesis, University of Colorado, 1999. *2 citations pages 99 et 116*

[Hew10] Hewlett-Packard Corporation, Intel Corporation, Microsoft Corporation, Phoenix Technologies Ltd, Toshiba Corporation. *Advanced Configuration and Power Interface Specification*, Avril 2010. *3 citations pages 30, 32, et 123*

[HLL10] Ruan He, Marc Lacoste, and Jean Leneutre. A policy management framework for self-protection of pervasive systems. In *Sixth International Conference on Autonomic and Autonomous Systems*, pages 104–109. IEEE, 2010. *cité page 23*

[HLM+09] Fabien Hermenier, Xavier Lorca, Jean-Marc Menaud, Gilles Muller, and Julia Lawall. Entropy: a consolidation manager for clusters. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, pages 41–50. ACM, 2009. *2 citations pages 45 et 51*

[HM08] Markus C Huebscher and Julie A McCann. A survey of autonomic computing-degrees, models, and applications. *ACM Computing Surveys*, 40(3):7, 2008. *cité page 23*

[iae12] Key World Energy Statistics. Technical report, Agence Internationale de l'Énergie, 2012. *cité page 3*

[IEE10] IEEE. *Energy Efficient Ethernet*, Septembre 2010. *cité page 36*

[IEE13] IEEE Standards Association. *Convergent Digital Home Network for Heterogeneous Technologies*, Avril 2013. *cité page 43*

[ITU08] E.800 : Definitions of terms related to quality of service. Technical report, International Telecommunication Union, 2008. *cité page 72*

[Jos12] M. Josefiok. Towards an energy abstraction layer. *Energy-Efficient Applications*, pages 25–30, 2012. *cité page 78*

[JRL+08] Narendra Jussien, Guillaume Rochart, Xavier Lorca, et al. Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on*

Open-Source Software for Integer and Contraint Programming, pages 1–10, 2008. *cité page 123*

[JSAC01] Christine E Jones, Krishna M Sivalingam, Prathima Agrawal, and Jyh Cheng Chen. A survey of energy efficient network protocols for wireless networks. *wireless networks*, 7(4):343–358, 2001. *cité page 43*

[KC03] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003. *cité page 22*

[KHY08] Bithika Khargharia, Salim Hariri, and Mazin S Yousif. Autonomic power and performance management for computing systems. *Cluster computing*, 11(2):167–181, 2008. *cité page 23*

[KL10] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56, 2010. *2 citations pages 44 et 115*

[Kum92] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI magazine*, 13(1):32, 1992. *cité page 177*

[KW04] Jeffrey O Kephart and William E Walsh. An artificial intelligence perspective on autonomic computing policies. In *Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 3–12. IEEE, 2004. *cité page 26*

[KZ08] Aman Kansal and Feng Zhao. Fine-grained energy profiling for power-aware application design. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):26–31, 2008. *2 citations pages 37 et 38*

[Lav11] Elisabeth Laville. Pour une consommation durable. Technical report, Centre d'Analyse Stratégique, Janvier 2011. *2 citations pages 7 et 8*

[LCF+11] Simon Laws, Mark Combellack, Raymond Feng, Haleh Mahbod, and Simon Nash. *Tuscany SCA in action*. Manning, 2011. *cité page 22*

[LFZE00] Alvin R Lebeck, Xiaobo Fan, Heng Zeng, and Carla Ellis. Power aware page allocation. *ACM SIGPLAN Notices*, 35(11):105–116, 2000. *cité page 31*

[Lie02] P. Lieberman. Wake-on-lan technology. 2002. *cité page 41*

[LLVH+13] Bart Lannoo, Sofie Lambert, Ward Van Heddeghem, Mario Pickavet, Fernando Kuipers, George Koutitas, Harris Niavis, Anna Satsiou, Michael Till, Andreas Fischer Beck, et al. Deliverable d8.1. overview of ict energy consumption. Février 2013. *cité page 5*

[LMM02] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002. *cité page 176*

[LO10] Laurent Lefèvre and Anne-Cécile Orgerie. Designing and evaluating an energy efficient cloud. *The Journal of Supercomputing*, 51(3):352–373, 2010. *cité page 45*

[MAC+11] J.C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A.C. Snoeren, and R.K. Gupta. Evaluating the effectiveness of model-based power characterization. In *USENIX Annual Technical Conference*. USENIX Association, 2011. *cité page 78*

[MBNR68] M Douglas McIlroy, JM Buxton, Peter Naur, and Brian Randell. Mass-produced software components. *Software Engineering Concepts and Techniques (1968*, pages 88–98, 1968. *cité page 17*

[MC09] F. Maker and Y. Chan. A survey on android vs. linux. *University of California*, pages 1–10, 2009. *cité page 78*

[MCD+03] Shivajit Mohapatra, Radu Cornea, Nikil Dutt, Alex Nicolau, and Nalini Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 582–591. ACM, 2003. *cité page 36*

[MCO+05] Shivajit Mohapatra, Radu Cornea, Hyunok Oh, Kyoungwoo Lee, Minyoung Kim, Nikil Dutt, Rajesh Gupta, Alex Nicolau, Sandeep Shukla, and Nalini Venkatasubramanian. A cross-layer approach for power-performance optimization in distributed mobile systems. In *Proceedings of the 19th IEEE International on Parallel and Distributed Processing Symposium*, pages 8–pp. IEEE, 2005. *cité page 38*

[MD89] Dennis McCarthy and Umeshwar Dayal. The architecture of an active database management system. *ACM Sigmod Record*, 18(2):215–224, 1989. *cité page 26*

[MGW09] David Meisner, Brian T Gold, and Thomas F Wenisch. Powernap: eliminating server idle power. In *ACM Sigplan Notices*, volume 44, pages 205–216. ACM, 2009. *cité page 34*

[MKS09] Hausi A Müller, Holger M Kienle, and Ulrike Stege. Autonomic computing now you see it, now you don't. In *Software Engineering*, pages 32–54. Springer, 2009. *cité page 22*

[MPV00] Silvano Martello, David Pisinger, and Daniele Vigo. The three-dimensional bin packing problem. *Operations Research*, 48(2):256–267, 2000. *cité page 176*

[MV03] Shivajit Mohapatra and Nalini Venkatasubramanian. Parm: Power aware reconfigurable middleware. In *Proceedings. 23rd International Conference on Distributed Computing Systems*, pages 312–319. IEEE, 2003. *2 citations pages 44 et 54*

[MV05] Aqeel Mahesri and Vibhore Vardhan. Power consumption breakdown on a modern laptop. In *Power-aware computer systems*, pages 165–180. Springer, 2005. *2 citations pages 36 et 115*

[NSC+08] Jessica M Nolan, P Wesley Schultz, Robert B Cialdini, Noah J Goldstein, and Vladas Griskevicius. Normative social influence is underdetected. *Personality and social psychology bulletin*, 34(7):913–923, 2008. *cité page 47*

[OAS06] OASIS. *Reference Model for Service Oriented Architecture 1.0*, 2006. *2 citations pages 19 et 20*

[OAS07] OASIS. *Service Component Architecture*, 2007. *cité page 22*

[OSG12] OSGi Alliance. *OSGi Service Platform, Core Specification, Version 5*, Juin 2012. *3 citations pages 21, 116, et 121*

[Pap77] Christos H Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical Computer Science*, 4(3):237–244, 1977. *cité page 175*

[Pap03] Mike P Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pages 3–12. IEEE, 2003. *cité page 19*

[Pat96] M.G. Patterson. What is energy efficiency?: Concepts, indicators and methodological issues. *Energy policy*, 24(5):377–390, 1996. *cité page 64*

[PDR08] Guilhem Paroux, Isabelle Demeure, and Laurent Reynaud. A power-aware middleware for mobile ad-hoc networks. In *Proceedings of the 8th International Conference on New Technologies in Distributed Systems*, page 22. ACM, 2008. *3 citations pages 38, 43, et 56*

[PS07] Venkatesh Pallipadi and Suresh B Siddha. Processor power management features and process scheduler: Do we need to tie them together. *LinuxConf Europe*, pages 1–8, 2007. *cité page 35*

[PS08] Eunjeong Park and Heonshik Shin. Reconfigurable service composition and categorization for power-aware mobile computing. *Transactions on Parallel and Distributed Systems*, 19(11):1553–1564, 2008. *cité page 43*

[pub13] *PubSubHubbub Core 0.4 – Working Draft*, Juin 2013. *cité page 118*

[PVD+08] Mario Pickavet, Willem Vereecken, Sofie Demeyer, Pieter Audenaert, Brecht Vermeulen, Chris Develder, Didier Colle, Bart Dhoedt, and Piet Demeester. Worldwide energy needs for ict: The rise of power-aware networking. In *2nd International Symposium on Advanced Networks and Telecommunication Systems*, pages 1–3. IEEE, 2008. *cité page 5*

[Rem08] Projet REMODECE : Mesure de la consommation des usages domestiques de l'audiovisuel et de l'informatique. Technical report, Électricité de France, Agence de l'environnement et de la maîtrise de l'énergie, Juillet 2008. *2 citations pages 8 et 115*

[RGKP10] Joshua Reich, Michel Goraczko, Aman Kansal, and Jitendra Padhye. Sleepless in seattle no longer. In *USENIX ATC*, volume 10, 2010. *cité page 46*

[RSS02] RSS Advisory Board. *Really Simple Syndication*, 2002. *cité page 118*

[Sal10] Julien Salanave. Impact Environnemental de la Filière TIC en France. Technical report, IDATE, Janvier 2010. *cité page 5*

[Sar83] George Saridis. Intelligent robotic control. *Transactions on Automatic Control*, 28(5):547–557, 1983. *cité page 22*

[SB07] Gregor Schiele and Christian Becker. Sandman: An energy-efficient middleware for pervasive computing. *Report 2007*, 2007. *cité page 55*

[SBS02] Eugene Shih, Paramvir Bahl, and Michael J Sinclair. Wake on wireless: an event driven energy saving strategy for battery operated devices. In *Proceedings of the 8th Annual InternationalConference on Mobile Computing and Networking*, pages 160–171. ACM, 2002. *cité page 41*

[SCG+11] Anna Spagnolli, Nicola Corradi, Luciano Gamberini, Eve Hoggan, Giulio Jacucci, Cecilia Katzeff, Loove Broms, and Li Jönsson. Eco-feedback on the go: Motivating energy awareness. *Computer*, 44(5):38–45, 2011. *cité page 47*

[SGM02] Clemens Szyperski, Dominik Gruntz, and Stephan Murer. *Component software: beyond object-oriented programming*. Addison-Wesley, 2002. *cité page 17*

[SHB08] Gregor Schiele, Marcus Handte, and Chistian Becker. Experiences in designing an energy-aware middleware for pervasive computing. In *Pervasive Computing and Communications*, pages 504–508. IEEE, 2008. *2 citations pages 43 et 46*

[SLH+12] Siddhartha Sen, Jacob R Lorch, Richard Hughes, Carlos Garcia Jurado Suarez, Brian Zill, Weverton Cordeiro, and Jitendra Padhye. Don't lose sleep over availability: The greenup decentralized wakeup service. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation*, pages 1–16, 2012. *cité page 46*

[SMF+09] Lionel Seinturier, Philippe Merle, Damien Fournier, Nicolas Dolet, Valerio Schiavoni, and J-B Stefani. Reconfigurable sca applications with the frascati platform. In *IEEE International Conference on Services Computing*, pages 268–275. IEEE, 2009. *cité page 22*

[SMR+12] Lionel Seinturier, Philippe Merle, Romain Rouvoy, Daniel Romero, Valerio Schiavoni, and Jean-Bernard Stefani. A component-based middleware platform for reconfigurable service-oriented architectures. *Software: Practice and Experience*, 42(5):559–583, 2012. *cité page 22*

[SNS07] Balasubramanian Seshasayee, Ripal Nathuji, and Karsten Schwan. Energy-aware mobile service overlays: Cooperative dynamic power management in distributed mobile systems. In *Fourth International Conference on Autonomic Computing*, pages 6–6. IEEE, 2007. *2 citations pages 45 et 53*

[Som10] Ian Sommerville. *Software Engineering 9th Edition*. Addison Wesley, 2010. *cité page 17*

[SR02] Rahul C Shah and Jan M Rabaey. Energy aware routing for low energy ad hoc sensor networks. In *Wireless Communications and Networking Conference*, volume 1, pages 350–355. IEEE, 2002. *cité page 43*

[SSS89] H Schwilden, H Stoeckel, and J Schüttler. Closed-loop feedback control of propofol anaesthesia by quantitative eeg analysis in humans. *British Journal of Anaesthesia*, 62(3):290–296, 1989. *cité page 22*

[TSR+98] Vivek Tiwari, Deo Singh, Suresh Rajgopal, Gaurav Mehta, Rakesh Patel, and Franklin Baez. Reducing power in high-performance microprocessors. In *Proceedings of the 35th annual Design Automation Conference*, pages 732–737. ACM, 1998. *cité page 31*

[UPn07] UPnP Forum. *UPnP Low Power Architecture*, Août 2007. *4 citations pages 30, 42, 46, et 57*

[UPn08] UPnP Forum. *UPnP Device Architecture 1.1*, Octobre 2008. *4 citations pages 20, 42, 118, et 122*

[USB11] USB Implementers Forum, Inc. *Universal Serial Bus 3.0 Specification*, Juin 2011. *cité page 42*

[WN03] G Wood and M Newborough. Dynamic energy-consumption indicators for domestic appliances: environment, behaviour and design. *Energy and Buildings*, 35(8):821–841, 2003. *cité page 47*

[XKYJ09] Yu Xiao, Ramya Sri Kalyanaraman, and Antti Ylä-Jääski. Middleware for energy-awareness in mobile devices. In *Proceedings of the Fourth international ICST Conference on Communication System Software and middlewaRE*, page 13. ACM, 2009. *cité page 38*

[YFB+13] Han Yan, Fabrice Fontaine, Olivier Bouchet, Jean-Paul Vuichard, Jean-Philippe Javaudin, Maryline Lebouc, Marie-Helene Hamon, Cedric Gueguen, and Bernard Cousin. Hope: Home power efficiency system for a green network. 2013. *3 citations pages 41, 42, et 52*

[YK03] Stephen S Yau and Fariaz Karim. An energy-efficient object discovery protocol for context-sensitive middleware for ubiquitous computing. *Transactions on Parallel and Distributed Systems*, 14(11):1074–1085, 2003. *cité page 43*

[YNA+06] Wanghong Yuan, Klara Nahrstedt, Sarita V Adve, Douglas L Jones, and Robin H Kravets. Grace-1: Cross-layer adaptation for multimedia quality and battery energy. *Transactions on Mobile Computing*, 5(7):799–815, 2006. *cité page 39*

[ZBSF04] Bo Zhai, David Blaauw, Dennis Sylvester, and Krisztian Flautner. Theoretical and practical limits of dynamic voltage scaling. In *Proceedings of the 41st annual Design Automation Conference*, pages 868–873. ACM, 2004. *cité page 35*

[Zil96] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73, 1996. *cité page 140*

[ZSZX11] Wumi Zhong, Gaotao Shi, Zenghua Zhao, and Feng Xia. Parasite: A system for energy saving with performance improvement in networked desktops. In *IEEE/ACM International Conference on Green Computing and Communications*, pages 79–84. IEEE, 2011. *2 citations pages 46 et 54*

Annexes

Annexe A

Problème d'optimisation

Les problèmes d'optimisation sont largement traités dans la littérature scientifique. L'optimisation cherche à trouver le meilleur élément d'un ensemble d'éléments. Pour cela, l'optimisation doit se baser sur un critère quantitatif. Ensuite, l'optimisation consiste à trouver l'élément qui correspond le mieux au critère quantitatif spécifié en les comparant.

Par exemple, un problème d'optimisation est celui du voyageur de commerce. Dans ce problème, un voyageur cherche à rallier l'ensemble des villes en utilisant le chemin le plus court [Pap77]. Dans cet exemple, la longueur du chemin est le critère quantitatif et celui qui est le plus court est l'élément qui est considéré comme le plus optimisé, *i.e.*, l'optimum.

Le problème d'optimisation qui nous intéresse est le problème du *bin packing* [CJGJ96]. Il fait partie des problèmes dit « de découpe et de conditionnement » qui regroupe entre autres le problème du sac à dos ou le problème d'ordonnancement sur plusieurs processeurs [Dyc90].

A.1 Le problème de *bin packing*

Le problème du *bin packing* cherche à placer le maximum d'objets dans un contenant. Ces objets sont de tailles différentes et le contenant est limité en taille, ne permettant pas d'y stocker l'ensemble des objets.

Ce problème peut être étendu en considérant plusieurs objets et plusieurs contenants ayant des capacités de stockage similaires ou différentes. Cette version s'applique facilement au domaine de l'informatique dans lequel un utilisateur cherche à placer le plus de fichiers sur le minimum de disques dur.

Enfin, le problème du *bin packing* peut être étendu à plusieurs dimensions. Par exemple pour deux dimensions, les objets ont une longueur et une largeur. Les contenants sont limités

FIGURE A.1 – **Problème de *bin packing* à 2 dimensions.** Le problème du *bin packing* à 2 dimensions consiste à essayer de faire rentrer un maximum d'objets de taille finie dans un ou plusieurs contenants de taille finie, *i.e.*, ici le sac à dos. La taille des objets et la contenance maximum du contenant sont représentés par les carrés. Cet exemple considère seulement la hauteur et la longueur et la largeur des objets.

en longueur et en largeur. Il s'agit donc d'un problème à deux dimensions, *i.e.*, la longueur et la largeur doivent être considérées lors de la recherche d'un optimum (cf. Figure A.1).

Ce problème nous intéresse car nous cherchons à placer un maximum de services sur un équipement. Il s'agit donc d'un problème d'optimisation qui peut ramener à un problème de *bin packing*.

Le problème de *bin packing* est un problème NP-difficile. La recherche d'une solution n'est pas calculable en un temps polynomial. Il n'existe pas de méthode générale permettant de trouver une solution optimale à moins de comparer chacune des solutions possibles une à une. Plus le nombre de solutions possibles est grand, plus le temps de calcul s'en voit rallongé afin de les comparer unes à unes.

A.2 Résolution du problème de *bin packing*

Afin de résoudre des problèmes NP-difficiles, il existe deux types d'algorithmes qu'il convient de différencier : les algorithmes d'approximation et les algorithmes exacts [LMM02]. Les algorithmes d'approximation accélèrent la recherche d'une solution. Toutefois, cette solution n'est pas nécessairement la solution optimale. Au contraire, les algorithmes exacts trouvent toujours la solution la plus optimale mais le temps de calcul s'en voit considérablement rallongé.

Le problème de *bin packing* ayant été largement étudié, il existe des méthodes de résolution heuristiques, *e.g.*, *first-fit decreasing*, *best-fit decreasing* [CJGJ96]. Ces méthodes de résolution se bornent à deux [LMM02], voir trois dimensions [MPV00].

La recherche d'une solution d'un problème de *bin packing*, *i.e.,* maximum d'objet stocké dans le contenant, tout en considérant des contraintes, *e.g.,* poids, volume, peut également être résolu au travers de méthodes exactes. La recherche d'une solution optimale en considérant les contraintes qui s'appliquent sur les variables se rapproche des problèmes de satisfaction de contraintes (CSP) [31] [Kum92].

Les CSP peuvent être décrit comme suit : il y a un ensemble de variables, définies au travers de domaines finis et discrets, ainsi qu'un ensemble de contraintes. Chaque contrainte est définie sur certains sous-ensembles de l'ensemble initial de variables et limite les combinaisons de valeurs que les variables peuvent prendre dans ce sous-ensemble. L'objectif est de trouver une valeur aux variables de telle sorte que cette valeur satisfasse à l'ensemble des contraintes [Kum92].

Les CSP permettent l'ajout de contraintes sur les variables plus facilement que les méthodes heuristiques, permettant ainsi de faire évoluer le modèle plus facilement. Ainsi, cela ne limite pas le problème à deux ou trois dimensions. Toutefois, cela demeure une méthode de résolution exacte, trouvant une solution optimale pour un temps de calcul qui peut s'avérer plus long que pour les méthodes approximatives.

Les solveurs de contraintes permettent, suivant l'approche utilisée, de résoudre les CSP. Dans le cadre de ce manuscrit, nous ne détaillons pas ces approches. Notre utilisation d'un solveur de contrainte se limite à modéliser notre problème sous la forme d'un CSP. Pour de plus amples informations sur les méthodes de résolution, nous invitons le lecteur à lire [Kum92].

31. *Constraint Satisfaction Problem*

Annexe B

Diagrammes d'activité

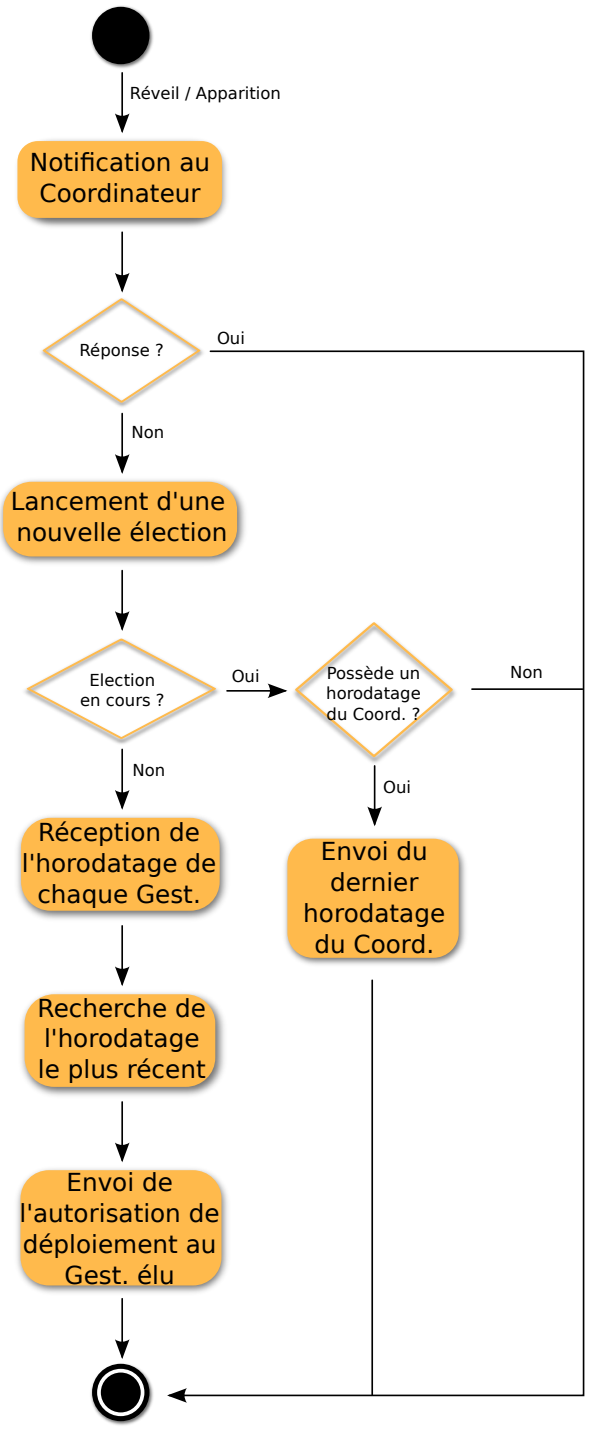

FIGURE B.1 – **Mécanisme d'élection d'un nouveau coordinateur.** Chaque gestionnaire est autonome et cherche avant tout à relancer le coordinateur au travers du scénario présenté. L'élection se base sur l'état connu le plus récent du coordinateur pour décider quel équipement va redémarrer le coordinateur.

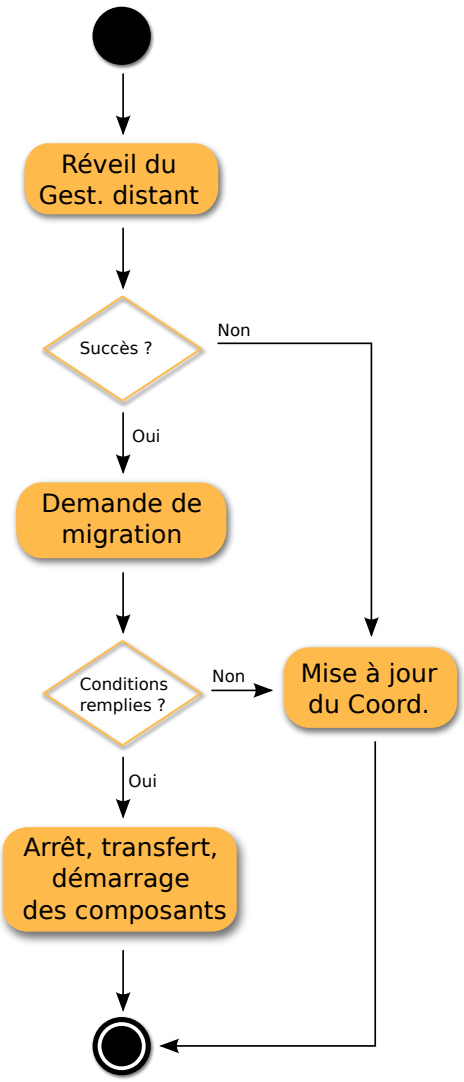

FIGURE B.2 – **Processus de migration.** Lorsqu'une migration de composants est ordonnée, le gestionnaire doit réveiller le gestionnaire distant, s'assurer que les composants peuvent migrer puis commencer la migration. Dans les autres cas, le coordinateur doit en être informé afin de modifier le plan de répartition en conséquence.