Une étude de la décision pour l'adaptation autonomique des systèmes

Imen Abdennadher*, Mohamed Jmaiel**

*Université de Sfax, ReDCAD, B.P. 1173, 3038 Sfax, Tunisie
imen.abdennadher@redcad.org,
** Research Center for Computer Science, Multimedia and Digital Data Processing of Sfax,
B.P. 275, Sakiet Ezzit, 3021 Sfax, Tunisia
mohamed.jmaiel@enis.rnu.tn

Résumé. Le paradigme d'autonomic computing est apparu afin d'aborder la complexité des systèmes dynamiques hétérogènes s'exécutant dans des environnements ubiquitaires. Dans ce type d'environnements, le contexte change fréquemment, ce qui nécessite l'adaptation des systèmes à ces changements. Particulièrement, les systèmes autonomes adaptent dynamiquement leurs architectures basées sur la boucle autonomique MAPE-K. Des décisions pendant l'exécution des différentes phases de la boucle MAPE-K doivent être prises afin de sélectionner la configuration la plus adéquate du système. Dans notre travail, nous nous focalisons à la décision pour l'adaptation autonomique des systèmes. Nous effectuons un état de l'art de la décision qui résume des travaux traitant la décision dans différents domaines d'application. Nous présentons ainsi une discussion sur les approches de décision qui existent dans la littérature, afin de définir notre approche permettant de combler les lacunes des appoches existantes et d'assurer une adaptation adéquate des systèmes.

1 Introduction

Le concept de l'informatique ubiquitaire a été introduit par Mark Weiser en 1991 : "Les technologies les plus profondément enracinées sont les technologies invisibles. Elles s'intègrent dans la trame de la vie quotidienne jusqu'à ne plus pouvoir en être distinguées." (Weiser, 1991). L'informatique ubiquitaire désigne donc le fait que l'informatique est omniprésente, c'est-à-dire qu'elle est parfaitement intégrée dans les objets et les endroits de notre vie quotidienne, et ceci d'une manière tellement invisible qu'on ne s'aperçoit pas de sa présence.

Les environnement ubiquitaires se caractérisent par un contexte qui change fréquemment. Ceci pose de nouveaux défis et possibilités pour les systèmes distribués s'exécutant dans ce type d'environnements, comme c'est mentionné dans le travail de (Sancho et al., 2010). En fait, les systèmes ubiquitaires doivent s'auto-adapter aux changements de leur contexte afin d'assurer leurs bons fonctionnements et satisfaire les besoins et les exigences de leurs utilisateurs. Néanmoins, la complexité des environnements ubiquitaires distribués génère des difficultés à créer des applications adaptées aux changements de ces environnements. Plusieurs approches

ont été proposées afin d'aborder cette complexité. Particulièrement, nous nous intéressons à l'approche de l'adaptation autonomique des systèmes.

(IBM, 2006) propose une architecture pour les systèmes autonomes qui englobe des capacités d'auto-gestion comme l'auto-configuration, l'auto-optimisation et l'auto-protection. Cette architecture comporte un gestionnaire autonome (autonomic manager) qui intéragit avec des ressources gérées (managed resources) représentant des capteurs et des actionneurs. L'architecture proposée par IBM suit les étapes de la boucle autonomique MAPE-K (monitoring, analyse, planification, exécution). L'exécution des modules de cette boucle engendre des décisions qui doivent être prises afin d'assurer l'adaptation du système. Ces décisions peuvent être prises par le développeur au moment de conception ou de compilation, comme elle peuvent être effectuées par le système au moment de son exécution. Dans ce papier, nous nous intéressons à l'étude des différentes approches de décision qui existent dans la littérature. Ceci nous permettra de proposer une approche de décision pour l'adaptation des systèmes autonomes permettant de combler les lacunes des approches de décision existantes.

La suite de ce papier est organisée comme suit. La section 2 présente des travaux de recherche traitant la décision appartenant à plusieurs domaines d'application. La section 3 présente les leçons apprises à partir de l'étude des travaux traitant la décision. La section 4 présente une discussion sur les approches de décision. Finalement, la section 5 présente la conclusion.

2 Travaux traitant la décision relatifs à divers domaines d'application

Dans cette section, nous présentons des travaux traitant la décision dans plusieurs domaines. Pour chaque domaine, nous citons des travaux représentatifs que nous exposons en deux phases : Dans la première phase, nous expliquons l'adaptation proposée par le travail puis nous nous focalisons sur la phase de décision pour l'adaptation.

2.1 Décision dans le domaine des systèmes auto-adaptatifs

Dans le domaine des systèmes auto-adaptatifs, nous présentons le travail de (Geihs et al., 2009) et celui de (Kephart et Das, 2007).

Le travail de (Geihs et al., 2009) se focalise sur l'adaptation des systèmes sensibles au contexte. La conception de ces systèmes est basée sur le concept de variabilité, c'est-à-dire sur la spécification des points de variation représentant les éléments dynamiquement variables de l'architecture de l'application, et qui permettent lors de l'exécution d'avoir les différentes configurations architecturales possibles. Les auteurs présentent une infrastructure basée-composants qui supporte l'adaptation au niveau application. La décision dans ce travail est basée sur des politiques qui utilisent des fonctions d'utilité. Ces fonctions permettent de sélectionner la variante de l'application qui maximise l'utilité de l'application tout en satisfaisant les contraintes de ressources.

Le travail de (Kephart et Das, 2007) assure une auto-adaptation des systèmes qui définit des objectifs de haut niveau. La décision dans ce travail est aussi basée sur les fonctions d'utilité qui sont considérées comme une méthode générale, pratique et basée sur des principes pour représenter les objectifs de haut niveau.

2.2 Décision dans le domaine des systèmes distribués basés service

Dans le domaine des systèmes distribués basés service, nous présentons le travail de (Gauvrit et al., 2010) et celui de (Bastide et al., 2008).

Le travail de (Gauvrit et al., 2010) suit le processus de la boucle autonomique MAPE-K afin d'adapter les applications basées services s'exécutant dans des environnements distribués et hétérogènes. Les auteurs proposent un framework d'adaptation qui permet l'évolution dynamique des architectures basées services en fournissant toutes les fonctionnalités du modèle MAPE-K (voir Figure 1). La décision dans le processus d'adaptation apparaît à la phase d'analyse en sélectionnant la configuration cible selon une politique d'adaptation comme l'efficacité énergétique. La décision apparaît aussi à la phase de planification en sélectionnant l'algorithme de planification le plus adéquat selon des contraintes (comme la durée d'exécution et la consommation des ressources). Finalement, la décision apparaît à la phase d'exécution en sélectionnant la meilleure action concrète selon des contraintes. Gauvrit et al. distinguent deux types de décision : décision à court-terme et décision à long-terme. Le premier type définit des décisions rapides et simples basées sur des règles "Evénement-Condition-Action". Le deuxième type (décision à long-terme) nécessite des stratégies complexes, permet l'optimisation de l'application à long-terme et utilise des algorithmes génériques basés sur des fonctions d'utilité.

Dans le travail de (Bastide et al., 2008), les auteurs confirment que la création d'applications capables de s'exécuter dans des environnements ubiquitaires nécessite une meilleure considération du contexte d'exécution afin d'assurer la continuité du service. Les auteurs proposent une approche visant la reconfiguration de la structure du composant permettant un déploiement felexible de ses services selon le contexte d'utilisation. Cette reconfiguration est basée sur des décisions qui englobent trois types de tâches de classification : La première tâche permet de classifier les services produits par le composant adapté selon leurs priorités de déploiement dans le dispositif de l'utilisateur. Cette stratégie de classification est basée sur des règles ($< condition > \Rightarrow < action >$) conçues par l'administrateur de l'application. La deuxième tâche assure la sélection des sites où les services peuvent être déployés en prenant en considération la disponibilité des ressources dans les différents nœuds de l'infrastructure distribuée. Cette sélection est basée sur des politiques d'adaptation définies par le concepteur du composant. La troisième tâche permet de classifier les services du composant selon les données reliées à la structure et le comportement du composant. Cette tâche peut être entièrement automatique, contrairement à la deuxième et la troisième tâche où un traitement spécifique à l'application est requis.

2.3 Décision dans le domaine de lignes de produits logiciels dynamiques (DSPL)

Dans le domaine de lignes de produits logiciels dynamiques, nous présentons le travail de (Pascual et al., 2014) et celui de (Capilla et al., 2014).

Le travail de (Pascual et al., 2014) modélise la variabilité architecturale à la phase de conception, en spécifiant les éléments de l'architecture qui peuvent être adaptés dynamiquement. Après avoir modélisé la variabilité de l'architecture de l'application, les différentes variantes du système seront générées au moment d'exécution, puis le processus de décision choisit la variante la plus adaptable au contexte courant. Les auteurs ont distingué deux catégo-

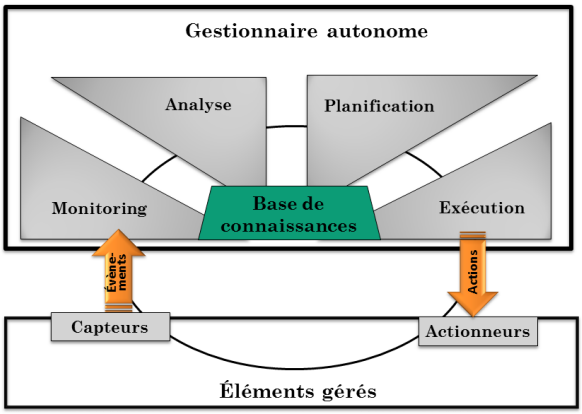

FIG. 1 – *La boucle autonomique MAPE-K de (IBM, 2006)*

ries de processus d'adaptation : les processus qui assurent l'analyse et la dérivation des plans de reconfiguration au moment de conception, suivant généralement des règles "Evénement-Condition-Action", et les processus orientés objectif qui n'énumèrent pas l'ensemble des paires "changement de contexte/configuration" à la phase de conception car c'est le système qui est responsable de calculer la configuration adéquate à un changement de contexte. Les auteurs mentionnent aussi les processus de décision basés sur des fonctions d'utilité, et qui sont considérés comme une extension des approches de décision qui se basent sur les objectifs.

Le travail de (Capilla et al., 2014) présente une vue d'ensemble des architectures et des techniques des lignes de produits logiciels dynamiques (DSPL). Les auteurs confirment qu'un DSPL doit être capable de manipuler les adaptations nécessaires et les tâches de la reconfiguration courante après le déploiement original. En effet, le DSPL doit manipuler les propriétés sensibles au contexte qui sont utilisées comme données d'entrée pour changer dynamiquement les valeurs des variantes du système et/ou pour sélectionner des nouvelles options du système selon les conditions de l'environnement. Les informations qui changent fréquemment dans le DSPL sont utilisées pour effectuer de nouvelles décisions ou pour sélectionner des options du système différentes à la volée. Les décisions effectuées au moment d'exécution sont souvent basées sur les informations de contexte, les niveaux de qualité demandés par le logiciel exécuté et les préférences des utilisateurs.

2.4 Décision dans le domaine de l'intelligence artificielle

Dans le domaine de l'intelligence artificielle, nous présentons le travail de (Kurdjokova et al., 2013) qui vise l'adaptation des affichages publics au contexte social. La prise de décision pour assurer cet objectif est basée sur des réseaux bayésiens qui comportent : un nœud de décision représentant toutes les actions que le système peut choisir pour réagir à un changement de contexte, et un nœud d'utilité qui calcule les valeurs d'utilité de toutes les actions possibles et leurs conséquences et retourne l'action ayant la valeur d'utilité la plus élevée.

Le travail de (Pascual et al., 2014) est parmi les travaux qui se basent sur un algorithme génétique afin d'adapter l'architecture du système. L'algorithme génétique appelé DAGAME permet de trouver approximativement la configuration architecturale optimale au moment d'exécution. Ceci est fait en optimisant une fonction d'utilité permettant de quantifier les points de variabilité de l'architecture selon des critères comme la précision et la consommation des ressources.

2.5 Décision dans le domaine de génie logiciel

Dans le domaine de génie logiciel, nous présentons le travail de (Zimmermann et al., 2009), celui de (Abrams et al., 2008) et celui de (Schuster et al., 2007).

Le travail de (Zimmermann et al., 2009) propose un framework conceptuel adapté aux exigences des architectes. Il définit un méta-modèle en UML pour la capture et la réutilisation de la décision architecturale. Cette décision suit trois étapes principales : l'identification de la décision, la prise de décision et la réalisation de la décision. La première étape consiste à identifier les décisions individuelles à partir d'un ensemble d'exigences et de décisions antérieures. Pendant la deuxième étape (l'étape de prise de décision), les architectes sélectionnent les alternatives selon des guides de décision qui peuvent être des exigences spécifiques au contexte ou des attributs de qualité logiciel. La dernière étape (l'étape de réalisation de la décision) consiste à partager et valider les résultats de la décision prise lors de la deuxième étape avec les différents acteurs et l'équipe de projet.

Le travail de (Abrams et al., 2008) s'intéresse à la collecte et à l'organisation des informations architecturales du système. Il utilise l'outil Architects' Workbench (AWB) qui supporte le processus de raisonnement et de modélisation architecturaux correspondant aux exigences des utilisateurs. Les décisions architecturales effectuées à travers les aspects de l'architecture, les alternatives considérées et la justification des choix effectués sont documentés par un "work product" de décision architecturale. Ce dernier est essentiel pour la compréhension d'une solution, la préservation de l'intégrité de cette solution même en subissant des opérations de maintenance ou d'évolution, et la réutilisation de ceratines parties de cette solution dans d'autres conceptions architecturales.

Le travail de (Schuster et al., 2007) propose un système de collaboration basé-web, nommé AD_{kwik}, qui fournit un support explicite pour le partage et la réutilisation des éléments de connaissance du domaine de capture de la décision architecturale. La conception de AD_{kwik} utilise des concepts de modélisation du domaine et une architecture en couches. AD_{kwik} comporte une API permettant de créer le modèle de décision initial à partir des modèles des exigences existants, les architectures de référence et d'autres atouts de communité. L'approche de gestion de la connaissance de AD_{kwik} englobe un modèle de domaine riche avec une gestion de sa relation avec la décision, un support de prise de décision collaborative et un résultat de projet partagé via Internet.

2.6 Décision dans le domaine Machine à Machine (M2M)

Dans le domaine Machine à Machine (M2M), nous présentons le travail de (BenAlaya et Monteil, 2012) et celui de (Moummadi et al., 2011).

Le travail de (BenAlaya et Monteil, 2012) présente une architecture autonomique générale basée sur la sensibilité au contexte. Les auteurs proposent un framework qui utilise une

TAB. 1 – *Exemple d'algorithme de calcul de la valeur d'utilité*

```
1 Calcul_utility(){
2  Soit utility=0
3  Soit utility_op_duration=0
4  Soit utility_op_memory=0
5  Si(Exec_duration(Op) ≤ 20) alors
     utility_op_duration ← 1
6  sinon
     utility_op_duration ← 1-(Exec_duration(Op)-20)/Exec_duration(Op)
7  fin si
8  Si(memory_used(Op) ≤ 10) alors
     utility_op_memory ← 1
9  sinon
     utility_op_memory ← 1-(memory_used(Op)-10)/memory_used(Op)
10 fin si
11 utility ← W_Dur * utility_op_duration + W_mem * utility_op_memory
12 retourner utility
13 }
```

représentation multi-modèles basée sur les ontologies et les graphes, décrivant les concepts et les relations du réseau M2M dans une base de données multi-niveaux. Les auteurs utilisent un modèle générique de décision basé sur le moteur de règles logiques Drools.

Le travail de (Moummadi et al., 2011) présente un modèle générique pour les services M2M et pour le support des décisions. Les auteurs proposent une approche de décision hybride basée sur les concepts de systèmes multi-agents et de la programmation des contraintes. Cette approche est fondée sur des agents de raisonnement multiples faisant des décisions locales indépendantes qui servent à la résolution d'un problème global commun de satisfaction de contraintes.

3 Leçons apprises à partir de l'étude des travaux traitant la décision

Dans cette section, nous présentons les leçons apprises à partir de l'étude des travaux existants traitant la décision. La première leçon consiste à classifier les approches de décision existantes et la deuxième leçon consiste à fixer le positionnement de la décision dans la boucle autonomique MAPE-K.

3.1 Classification des approches de décision

Après avoir étudié les travaux existants qui traitent la décision, nous pouvons distinguer principalement quatre types d'approches de décision.

– *Les approches situation-action* : nécessitent une spécification exacte des réactions à faire dans un état courant du système. Les règles de décision sont spécifiées par le concepteur du système au moment de sa conception.

- *Les approches orientées objectif* : donnent la responsabilité au système de calculer les actions permettant de passer de l'état courant à un état désiré.
- *Les approches orientées utilité* : attribuent une valeur réelle de désirabilité pour chaque état possible du système appelé variante du système. Elles calculent les valeurs d'utilité pour chaque variante et sélectionnent la variante ayant la valeur d'utilité la plus élevée.
- *Les approches ad hoc* : sont définies par le concepteur du processus de décision.

Nous prenons un exemple simple pour expliquer le principe de la décision prise par chaque approche. L'exemple considère une opération Op effectuée par le système afin d'assurer l'une de ses fonctionnalités. Exec_duration(Op) correspond à la durée d'exécution de l'opération Op. Une exigence du système impose que la durée d'exécution de l'opération Op doit être toujours inférieure ou égale à 20 ms quelles que soient les conditions. Nous expliquons ainsi comment chacune des approches de décision traite cette exigence.

Pour *les approches situation-action*, le concepteur doit spécifier des règles précises. L'exemple définit deux règles :

Règle 1 : If (Exec_duration(Op) >20 ms) Then increase CPU by 3%
Règle 2 : If (Exec_duration(Op) >20 ms) Then increase Memory by 5%

La première règle (**Règle 1**) augmente la fraction de CPU réservée pour l'exécution de Op par 3% si la période d'exécution de Op dépasse 20ms et la deuxième règle (**Règle 2**) augmente la quantité de mémoire réservée à l'opération Op par 5% si la période d'exécution de Op dépasse 20ms. Lors de l'exécution du système, si la condition Exec_duration(Op) dépasse 20ms se présente, le système choisit l'une des deux actions agissant sur la fraction de CPU ou la quantité de mémoire.

Pour *les approches orientées objectif*, le concepteur définit seulement l'objectif qui exige (dans cet exemple) que la durée d'exécution de l'opération Op ne doit pas dépasser 20ms (Exec_duration(Op) $\leq$ 20 ms). Lors de l'exécution du système, ce dernier aura la responsabilité de calculer les actions permettant d'assurer cette exigence.

Pour *les approches orientées utilité*, l'exemple définit l'algorithme présenté dans le tableau 1. La variable "utility" est calculée pour chaque configuration possible du système, afin de sélectionner par la suite la configuration ayant la valeur d'utilité la plus élevée. Deux paramètres sont considérés : Exec_duration(Op) qui correspond à la durée d'exécution de l'opération Op et memory_used(Op) qui correspond à la quantité de mémoire utilisée par l'opération Op . Dans cet exemple, l'utilisateur exige que la durée d'exécution de Op soit inférieure ou égale à 20ms et que la quantité de mémoire utilisée par Op soit inférieure ou égale à 10Mo. La variable utility_op_duration est égale à 1 si Exec_duration(Op) $\leq$ 20 (Table 1- instruction 5) sinon elle prend comme valeur le décalage entre la valeur de la durée d'exécution de Op réelle et la valeur désirée (Table 1- instruction 6). De même, la variable utility_op_memory est égale à 1 si memory_used(Op) $\leq$ 10 (Table 1- instruction 8) sinon elle prend comme valeur le décalage entre la valeur de la mémoire utilisée réelle et la valeur désirée (Table 1- instruction 9). La valeur d'utilité est égale à la somme pondérée de utility_op_duration et utility_op_memory (Table 1- instruction 11). Les poids W_{Dur} et W_{mem} correspondent respectivement aux paramètres durée d'exécution et mémoire, et sont fixés par le concepteur.

Après avoir étudié les travaux existants traitant la décision, nous associons les domaines de ces travaux aux approches de décision : *Les approches situation-action* sont fréquemment utilisées par les travaux appartenant aux domaines des systèmes auto-adaptatifs, des systèmes distribués basés service et des lignes de produits logiciels dynamiques. *Les approches orien-*

tées objectif sont fréquemment utilisées par les travaux appartenant aux domaines des systèmes auto-adaptatifs, des lignes de produits logiciels dynamiques et le domaine M2M. *Les approches orientées utilité* sont fréquemment utilisées par les travaux appartenant aux domaines des systèmes auto-adaptatifs, des systèmes distribués basés service, des lignes de produits logiciels dynamiques et de l'intelligence artificielle. *Les approches ad hoc* sont fréquemment utilisées par les travaux appartenant au domaine de génie logiciel et le domaine M2M.

3.2 Positionnement de la décision dans la boucle autonomique MAPE-K

Dans la figure 2, nous précisons le positionnement de la décision dans la boucle autonomique MAPE-K. La décision commence dès la phase d'analyse. En effet, aprés avoir analysé les changements du système, juste après la phase de diagnostic, l'analyseur démarre un processus de décision permettant de préciser s'il est nécessaire d'effectuer une adaptation pour ces changements ou non. En cas de besoin d'une adaptation, la requête de changement est envoyée au planificateur et la décision se présente ainsi à la phase de planification en sélectionnant l'algorithme de planification le plus adéquat selon des contraintes dégagées à partir des exigences des utilisateurs. Finalement, la décision au niveau de la phase d'exécution permet de choisir les actions concrètes de reconfiguration les plus adéquates aux contraintes ressources.

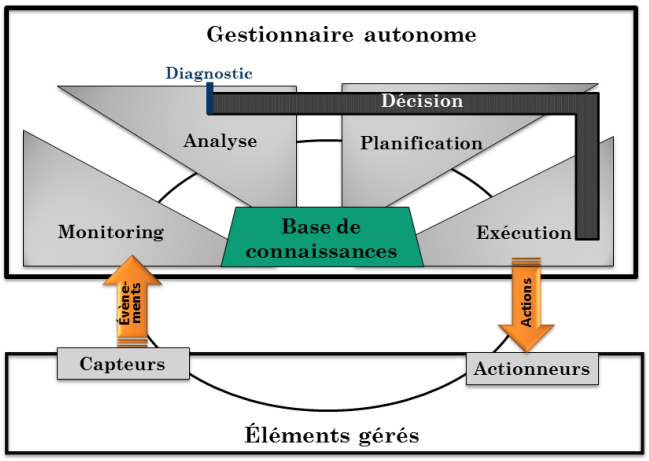

FIG. 2 – *Le positionnement de la décision dans la boucle autonomique MAPE-K*

4 Discussion

Dans cette section, nous présentons une vue critique des différentes approches de décision.

Les approches situation-action exigent une description explicite de chaque situation. Elles utilisent des règles compréhensibles et simples à utiliser et génèrent des actions rapides. De plus, celui qui spécifie les règles connaît exactement l'état que le système va atteindre en

exécutant l'action recommandée. Néanmoins, *les approches situation-action* ont des inconvénients majeurs comme les conflits qui peuvent apparaître entre les actions, le nombre de règles qui peut être très élevé et l'incapacité de capturer certaines dépendances entre l'adaptation et le contexte, puisqu'il existe toujours des situations qu'on ne peut pas prévoir.

Les approches orientées objectif offrent une flexibilité plus importante que celle offerte par *les approches situation-action*. En effet, au lieu de compter sur un humain pour encoder explicitement le comportement rationnel (comme dans *les approches situation-action*), le système génère lui-même le comportement rationnel. De plus, en suivant *les approches orientées objectif*, le développeur n'est pas sensé connaître les détails des fonctions du système de bas niveau, en laissant le système déterminer les actions requises pour atteindre les objectifs de haut niveau spécifiés. Cependant, *les approches orientées objectif* peuvent être face à des conflits entre les objectifs. Le travail de (Geihs et al., 2009) donne un exemple de conflits en mentionnant qu'un objectif visant l'optimisation de l'utilisation du CPU peut être atteint en faisant la suspension des applications de basses priorités, mais cette suspension peut entrer en conflit avec un autre objectif qui vise une haute disponibilité des services. Un autre inconvénient *des approches orientées objectif* est l'absence de mécanismes pour comparer entre les actions d'adaptation lorsque plusieurs actions peuvent être appliquées pour réaliser un objectif.

Les fonctions orientées utilité assurent une décision précise, résolvent le problème de conflits entre les objectifs en affectant une valeur réelle de désirabilité à chaque configuration possible et en sélectionnant la configuration ayant la valeur d'utilité la plus élevée. Par ailleurs, *les approches orientées utilité* abordent la complexité du raisonnement d'adaptation puisque leur sélection est basée sur des valeurs calculées. Néanmoins, *les approches orientées utilité* engendrent une difficulté à spécifier un ensemble multidimentionnel de préférences correspondant aux paramètres du contexte et leurs poids, qui demeurent difficiles à fixer.

D'après l'étude des approches de décision qui existent dans la littérature, nous remarquons que toutes ces approches engendrent des difficultés et des problèmes qui peuvent empêcher la prise des bonnes décisions ou même engendrer des défauts de fonctionnement du système. Pour cette raison, nous visons définir une approche hybride permettant de combler les lacunes des approches existantes et assurant une décision plus efficace pour l'adaptation des architectures des systèmes autonomes.

5 Conclusion

Dans ce papier, nous nous intéressons à l'étude de la décision dans le processus d'adaptation des systèmes. Nous avons dégagé plusieurs travaux appartenant à plusieurs domaines et traitant la décision pour l'adaptation. Ensuite, nous avons présenté les leçons apprises à partir de l'étude des travaux traitant la décision, et qui consistent à classifier les différentes approches de décision et à fixer le positionnement de la décision dans la boucle autonomique MAPE-K. Finalement, nous avons présenté une discussion sur les approches de décision qui existent dans la littérature. Dans notre futur travail, nous visons définir une approche de décision pour la gestion autonomique de l'adaptation des systèmes, qui permettra de combler les lacunes des approches de décision existantes.

6 Remerciement

Les auteurs remercient l'Université de Toulouse qui a permis cette opportunité de recherche dans IDEX "chaire d'attractivite" délivré à Pr. Gene COOPERMAN.

Références

Abrams, S., B. Bloom, P. Keyser, D. Kimelman, E. Nelson, W. Neuberger, T. Roth, I. Simmonds, S. Tang, et J. M. Vlissides (2008). Architectural thinking and modeling with the architects' workbench. *IBM Systems Journal 45*(3), 481–500.

Bastide, G., A. Seriai, et M. Oussalah (2008). A self-adaptation of software component structures in ubiquitous environments. In *Proceedings of the 5th International Conference on Pervasive Services*, pp. 173–176.

BenAlaya, M. et T. Monteil (2012). Frameself : A generic context-aware autonomic framework for self-management of distributed systems. In *Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on*, pp. 60–65.

Capilla, R., J. Bosch, P. Trinidad, A. Ruiz-Cortés, et M. Hinchey (2014). An overview of dynamic software product line architectures and techniques : Observations from research and industry. *Journal of Systems and Software 91*, 3–23.

Gauvrit, G., E. Daubert, et F. Andre (2010). Safdis : A framework to bring self-adaptability to service-based distributed applications. In *Proceedings of the 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, SEAA '10, pp. 211–218.

Geihs, K., P. Barone, F. Eliassen, J. Floch, R. Fricke, E. Gjorven, S. Hallsteinsen, G. Horn, M. U. Khan, A. Mamelli, G. A. Papadopoulos, N. Paspallis, R. Reichle, et E. Stav (2009). A comprehensive solution for application-level adaptation. *Software-Practice and Experience 39*(4), 385–422.

IBM (2006). An architectural blueprint for autonomic computing.

Kephart, J. et R. Das (2007). Achieving self-management via utility functions. *Internet Computing, IEEE 11*(1), 40–48.

Kurdjokova, E., M. Wissner, S. Hammer, et E. Andre (2013). Trust-based decision-making for the adaptation of public displays in changing social contexts. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, pp. 317–324.

Moummadi, K., R. Abidar, et H. Medromi (2011). Generic model based on constraint programming and multi-agent system for m2m services and agricultural decision support. In *Multimedia Computing and Systems (ICMCS), 2011 International Conference on*, pp. 1–6.

Pascual, G. G., M. Pinto, et L. Fuentes (2014). Self-adaptation of mobile systems driven by the common variability language. *Future Generation Computer Systems*.

Sancho, G., I. B. Rodriguez, T. Villemur, et S. Tazi (2010). What about collaboration in ubiquitous environments ? In *New Technologies of Distributed Systems (NOTERE), 2010 10th Annual International Conference on*, pp. 143–150.

Schuster, N., O. Zimmermann, et C. Pautasso (2007). Adkwik : Web 2.0 collaboration system for architectural decision engineering. In *Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 2007)*, pp. 255–260.

Weiser, M. (1991). The computer for the 21st century. *Scientific American 265*, 66–75.

Zimmermann, O., J. Koehler, F. Leymann, R. Polley, et N. Schuster (2009). Managing architectural decision models with dependency relations, integrity constraints, and production rules. *Journal of Systems and Software 82*(8), 1249–1267.

Summary

Dynamic systems running in ubiquitous environments are characterized by a context that changes frequently. The adaptation of this kind of systems to the context changes is a necessary and complex task. The autonomic computing paradigm arises to tackle this complexity. In fact, autonomic systems dynamically adapt their architectures, based on a MAPE-K autonomic loop, to the context changes. Decisions during the execution of the MAPE-K loop phases must be taken in order to select the most suitable configuration of the system. In our work, we focus on decision for autonomic system adaptation. We present a state of the art which summarizes research activities dealing with decision in many fields. Thus, we present a discussion about the existing decision approaches, in order to define our approach aiming at bridging the gap of the existing decision approaches and ensuring a suitable adaptation of systems.