REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE 20 août 1955-Skikda

Faculté des Science
Département d'informatique

N° d'ordre : . . . | . . . | . . . .
Série :. . . . . | . . . | . . . .

THESE DE DOCTORAT EN LMD

SPECIALITE : INFORMATIQUE

Présentée et soutenue publiquement par

Ouareth Selma

le : .....

Un Système Multi-Agents Auto-Adaptatifs Sensibles aux Contextes à Boucles de Contrôle Hiérarchisées

---

Redjimi	Mohammed	**Prof.**	Univ. de 20 Août 1955, Skikda	**Président**
Mazouzi	Smaine	**Prof.**	Univ. de 20 Août 1955, Skikda	**Rapporteur**
Boulehouache	Soufiane	**M.C.A**	Univ. de 20 Août 1955, Skikda	**Co-Rapporteur**
Brahimi	Said	**M.C.A**	Univ. de 08 Mai 1945, Guelma	**Examinateur**
Zeghida	Djamel	**M.C.A**	Univ. de 20 Août 1955, Skikda	**Examinateur**

Année Universitaire 2022/2023.

Remerciements

Je remercie tout d'abord Dieu d'avoir me donner la santé et la force pour réaliser ce travail.

En premier lieu, je tiens à remercier mon directeur de thèse *Pr. Mazouzi Smaine* pour sa patience et sa motivation tout au long de ces années. Mes remerciements vont également à mon co-encadreur *Dr. Boulehouache Soufiane* pour ses orientations et ses conseils, qu'il m'a guidé tout au long de la recherche et de la rédaction de cette thèse. Aussi, je remercie *Pr.Redjimi Mohamed* pour avoir accepté de présider cette soutenance, et je remercie *Dr.Zeghida Djamel* et *Dr. Brahimi Said* pour avoir accepté d'examiner cette thèse et d'avoir contribué à sa amélioration par leurs remarques.

Un grand remerciement à *mes parents* qui m'ont encouragé dès le début et m'ont soutenu toutes ces années pour atteindre mon objectif. Il n'y a pas assez de mots pour les remercier de leurs efforts. Je remercie également *mes frères* et *sœurs* pour leur soutien, en particulier mon frère *Mohamed Saleh*, je lui souhaite de devenir un innovateur dans le domaine de l'informatique à l'avenir.

L'aboutissement de ce travail de thèse doit aussi beaucoup à mon mari, *Hakim*, et je tiens à le remercier, dont les encouragements m'ont permis de terminer ce projet doctorale. Il a toujours été là pour me soutenir dans les bons comme dans les mauvais moments. En fait, il n'y a pas assez de mots pour lui donner sa vraie part de remerciements.

Enfin, je voudrais exprimer mes remerciements à un morceau de mon cœur, mon fils *Amir*, qui, malgré mes lacunes à son égard, m'a donné de la force et m'a rendu plus insistante pour terminer ma thèse. J'espère que tu seras fier de ta mère quand tu seras grand, mon amour.

À mes chers parents...

À mon cher mari ...

À mon petit fils Amir...

À mes sœurs et frères...

À toute la famille…

S. Ouareth… 

Résumé

La centralisation des décisions d'auto-adaptation hétérogènes des Systèmes Auto-* et le couplage fort inter-constituants ont engendré des Entités Autonomiques trop volumineuses et trop complexes à construire et à maintenir. Donc, ce travail concerne la re-ingénierie de la partie Contrôle Autonomique des Systèmes Auto-Adaptatifs (SAAs). Il s'agit de séparer les capacités d'auto-adaptations en plusieurs boucles de contrôle MAPE-K (Monitor, Analyze, Plan, Execute et Knowledge base) spécifiques pour chaque type d'auto-adaptation. Cependant, le modèle MAPE-K est trop basique et même les améliorations existantes de sa conception sont très générales. Ils manquent suffisamment de détails concernant la structure et la dynamique de ses constituants. De plus, ces conceptions n'utilisent ni un paradigme de conception ni des modèles de conception bien établis. Nous avons donc trouvé nécessaire de raffiner plus le modèle MAPE-K et le remodéliser utilisant un modèle à base de composant. Ce modèle facilitera le processus de la conception des SAAs.

Lorsque le système est complexe et à grande échelle, il est nécessaire d'utiliser plusieurs boucles de contrôle. Dans la littérature, différents modèles ont été proposés de composition de boucles de contrôle, basés sur les approches de contrôle suivantes : centralisé, décentralisé et hiérarchique. Dans ce contexte, nous avons proposé un nouveau modèle de contrôle partiellement centralisé basé sur le modèle MAPE-K à base de composant. Ce modèle consiste à centraliser les composants Plan et Knowledge, et à décentraliser les composants Monitor, Analyze et Execute. Nous nous sommes appuyés sur le modèle de composant Fractal pour modéliser les composants Plan et Knowledge en tant que composants partagés. De plus, le composant Execute doit se coordonner avec ses pairs. Ce modèle assure une décision correcte et une exécution dans un temps raisonnable. En raison de la nature évolutive des SAAs, il existe d'autres défis affectent le fonctionnement des SAAs, tels que : la distribution des actions d'adaptation peut imposer un surcoût de communication important, le composant Plan représente un point de défaillance unique et une surcharge de communication importante est effectué sur le composant Plan. Par conséquent, nous avons développé ce modèle en passant du contrôle partiellement centralisé au contrôle hiérarchique afin d'améliorer la fiabilité globale du système.

Le contrôle hiérarchique est prouvé être un bon compromis pour gérer la complexité de l'auto-adaptation. Cependant, il est confronté à un certain nombre de défis, notamment la séparation des préoccupations et l'affectation des différentes boucles de contrôles à différentes préoccupations. Motivés par ce fait, nous arrivons au HCLs (Hierarchical Control Loops), une proposition architecturale qui se concentre sur trois aspects : la séparation des préoccupations dans la modélisation des systèmes, le contrôle hiérarchique et l'adaptation des différentes entités des boucles de contrôle MAPE-K. Pour séparer les préoccupations, nous nous appuyons sur le modèle de composant Fractal qui peut simplifier la séparation. Dans HCLs, chaque niveau de la hiérarchie correspond à un type d'adaptation spécifique. Trois types d'adaptation pouvant être appliqués par HCLs qui sont Adaptation Locale, Adaptation Régionale et adaptation Supérieure. De plus, HCLs offre la possibilité de modifier la structure et le comportement des boucles de contrôle au moment de l'exécution sur la base de stratégies d'exploration pour l'apprentissage par renforcement en ligne utilisant le modèle de caractéristiques pour adapter différentes entités des boucles de contrôle MAPE-K.

Enfin, une expérimentation a été menée avec cette approche dans la gestion d'une connexion Internet sans fil dans un aéroport.

Mots clés : Sensibilité au contexte, Systèmes Auto-Adaptatifs, Boucle de Contrôle Autonomique, MAPE-K, Contrôle hiérarchique, Composants logiciels, Séparation des préoccupations, Fractal, Apprentissage par renforcement, Modèle de caractéristiques.

Abstract

The centralization of the heterogeneous auto-adaptive decisions of Auto-* Systems and the strong inter-constituent coupling have generated Autonomic Entities that are too voluminous and too complex to build and maintain. So, this work concerns the re-engineering of the Autonomic Control part of Self-Adaptive Systems (SASs). This involves separating the self-adaptation capacities into multiple specific MAPE-K (Monitor, Analyze, Plan, Execute and Knowledge base) control loops for each type of self-adaptation. However, the MAPE-K model is too basic and even the existing improvements in its design are very general. They lack sufficient detail regarding the structure and dynamics of its constituents. Additionally, these designs do not use a well-established design paradigm or design patterns. Therefore, we found it necessary to further refine the MAPE-K model and re-model it using a component-based model. This model will facilitate the process of designing SASs.

When the system is complex and large-scale, it is necessary to use multiple control loops. In the literature, different models have been proposed for the composition of control loops, based on the following control approaches : centralized, decentralized and hierarchical. In this context, we have proposed a new partially centralized control model based on our extension of the MAPE-K model. This model consists of centralizing the Plan and Knowledge components, and decentralizing the Monitor, Analyze, and Execute components. We relied on the Fractal component model to model the Plan and Knowledge components as shared components. Additionally, the Execute component must coordinate with its peers. This model ensures a correct decision and execution in a reasonable time. Due to the evolutionary nature of SASs, there are other challenges affecting the functioning of SASs, such as the distribution of adaptation actions can impose a large communication overhead, the Plan component represents a single point of failure, and significant communication overhead is performed on the Plan component. Therefore, we developed this model by moving from partially centralized control to hierarchical control in order to improve the overall reliability of the system.

Hierarchical control is proven to be a good compromise to manage the complexity of self-adaptation. However, it faces a number of challenges, including the separation of concerns and the assignment of different control loops to different concerns. Motivated by this fact, we arrive at HCLs (Hierarchical Control Loops), an architectural proposal that focuses on three aspects : the separation of concerns in the modeling of systems, hierarchical control and the adaptation of the different entities of MAPE-K control loops. To separate the concerns, we rely on the Fractal component model which can simplify the separation. In HCLs, each level of the hierarchy corresponds to a specific adaptation type. Three types of adaptation can be applied by HCLs which are Local adaptation, Regional adaptation and Superior adaptation. Moreover, HCLs provides the possibility to modify the structure and behavior of control loops at runtime based on exploration strategies for online reinforcement learning using the feature model to adapt different entities of the MAPE-K control loops.

Finally, an experiment was conducted with this approach in managing an airport wireless Internet connection.

Keywords : Context Awareness, Self-Adaptive Systems, Autonomic Control Loop, MAPE-K, Hierarchical Control, Software Components, Separation of Concerns, Fractal, Reinforcement Learning, Feature Model.

الملخص

أدت مركزية قرارات التكيف التلقائي غير المتجانسة للأنظمة التلقائية *Auto- والاقتران القوي بين المكونات إلى إنشاء كيانات مستقلة ضخمة جداً ومعقدة جداً بحيث يصعب بناؤها وصيانتها. لذلك، هذا العمل يتعلق بإعادة هندسة جزء التحكم الذاتي في أنظمة التكيف الذاتي. يتضمن ذلك فصل قدرات التكيف الذاتي إلى عدة حلقات تحكم MAPE-K محددة لكل نوع من أنواع التكيف الذاتي. ومع ذلك ، فإن نموذج MAPE-K قاعدي للغاية وحتى التحسينات الحالية في تصميمه عامة جدا. حيث إنهم يفتقرون إلى التفاصيل الكافية فيما يتعلق بهيكل وديناميكيات مكوناته. بالإضافة إلى ذلك ، لا تستخدم هذه التصميمات نموذج تصميم راسخا. لذلك، وجدنا أنه من الضروري تحسين نموذج MAPE-K وإعادة تصميمه باستخدام نموذج مكون. سيسهل هذا النموذج عملية تصميم أنظمة التكيف الذاتي.

عندما يكون النظام معقداً وواسع النطاق ، فمن الضروري استخدام عدة حلقات تحكم. في الأدبيات ، تم اقتراح نماذج مختلفة لتكوين حلقة التحكم ، بناءً على مناهج التحكم التالية: المركزية، اللامركزية و الهرمية. في هذا السياق ، اقترحنا نموذج تحكم مركزيا جزئيا جديدا يعتمد على نموذج MAPE-K القائم على المكونات. يتصف هذا النموذج بمركزية مكونات الخطة والمعرفة و لامركزية مكونات المراقبة والتحليل والتنفيذ. لقد اعتمدنا على نموذج المكون Fractal لنمذجة مكونات الخطة والمعرفة كمكونات مشتركة. بالإضافة إلى ذلك ، يتنسق مكون التنفيذ مع أقرانه. يضمن هذا النموذج اتخاذ القرار الصحيح والتنفيذ في وقت المناسب و المحدد. بسبب الطبيعة التطورية لنظام التكيف الذاتي ، هناك تحديات أخرى تؤثر على عمل أنظمة التكيف الذاتي مثل: توزيع إجراءات التكيف يمكن أن يفرض عبئاً هاماً على الاتصال، يمثل مكون الخطة نقطة فشل واحدة،...الخ . لذلك ، قمنا بتطوير هذا النموذج من خلال الانتقال من التحكم المركزي جزئيا إلى التحكم الهرمي من أجل تحسين الموثوقية الكلية للنظام.

ثبت أن التحكم الهرمي هو حل وسط جيد لإدارة تعقيد التكيف الذاتي. ومع ذلك ، فإنه يواجه عدداً من التحديات ، بما في ذلك فصل الاهتمامات وتخصيص حلقات تحكم مختلفة لاهتمامات مختلفة. بدافع من هذه الحقيقة ، وصلنا إلى HCLs (حلقات التحكم الهرمية) ، وهو اقتراح هندسي يركز على ثلاثة جوانب: فصل الاهتمامات في نمذجة الأنظمة ، والتحكم الهرمي وتكييف الكيانات المختلفة لحلقات التحكم MAPE-K. لفصل الاهتمامات ، نعتمد على نموذج المكون الكسري الذي يمكنه تبسيط الفصل. في HCLs ، يتوافق كل مستوى من التسلسل الهرمي مع نوع تكيف محدد. ثلاثة أنواع من التكيف يمكن تطبيقها بواسطة HCLs وهي التكيف المحلي والتكيف الإقليمي والتكيف المتفوق. علاوة على ذلك ، توفر HCLs إمكانية تعديل هيكل وسلوك حلقات التحكم في وقت التشغيل بناءً على استراتيجيات الاستكشاف لتعلم المعزز عبر الإنترنت باستخدام نموذج الميزة لتكييف كيانات مختلفة من حلقات التحكم MAPE-K.

أخيراً ، تم إجراء تجربة باستخدام هذا النهج في إدارة اتصال الإنترنت اللاسلكي في المطار.

الكلمات الرئيسية: الوعي بالسياق ، أنظمة التكيف الذاتي ، حلقة التحكم الذاتي ، MAPE-K، التحكم الهرمي ، مكونات البرامج ، فصل الاهتمامات ، Fractal ، التعلم المعزز ، نموذج الميزة.

Table des matières

Liste des figures

Liste des tables

Liste des symboles

DSL Domain Specific Language

$HCLs$ Hierarchical Control Loops

$MAPE-K$ Monitor, Analyze, Plan, Execute - Knowledge base

MSL MAPE Specification Language

SAA Systèmes Auto-Adaptatifs

SMA Systèmes Multi-Agents

Introduction

Ce chapitre donne un aperçu de la thèse. Tout d'abord, nous introduisons le cadre de cette étude qui passe par une description du contexte de travail afin de mettre en évidence la problématique. Ensuite, nous décrivons les objectifs de la solution souhaitée. Puis, nous passons à la contribution avant de présenter l'organisation de la thèse. Le chapitre se termine par la liste des articles publiés.

Contexte

Les systèmes informatiques sont devenus de plus en plus distribués et hétérogènes, rendant la gestion manuelle difficile et sujette aux erreurs. L'approche « informatique autonomique » a été proposée pour surmonter ce problème par la minimisation de l'intervention humaine et la réduction des coûts d'évolution dans les systèmes logiciels complexes.

En conséquent, les systèmes informatiques sont devenus capable de simuler les propriétés des systèmes naturels telles que l'intelligence, la rationalité, la capacité d'apprendre, d'anticiper et de s'adapter automatiquement au contexte. Ces types de systèmes sont appelés Systèmes Auto-Adaptatifs [71, 62]. Donc, les Systèmes Auto-Adaptatifs (SAAs) doivent être sensibles au contexte car ils sont sujets à des changements de contexte inattendus, c'est-à-dire ne peuvent pas être anticipés avant le déploiement. De plus, ces systèmes sont censés gérer l'adaptation au moment de l'exécution en cas de besoin. La construction d'un tel système nécessite la conception et la mise en œuvre de boucles autonomique qui collectent des événements et des mesures, analysent ces données, prennent des décisions et exécutent les actions correspondantes. Dans la littérature, le modèle de référence populaire pour la conception des SAAs dans le contexte de l'informatique autonomique est la boucle de contrôle $MAPE-K$ [1] (Monitor, Analyze, Plan, Execute - Knowledge base). La base de la capacité d'auto-adaptation des SAAs est la prise en charge au moment de l'exécution des propriétés d'auto-* ou d'autogestion. Une propriété est particulièrement intéressante car elle est essentielle dans la réalisation de propriétés autonomiques de haut-niveau : l'auto-adaptation. Le développement rapide de ce domaine a rendu nécessaire de proposer des moyens simples pour concevoir et développer des SAAs tout en assurant la cohérence du système et en répondant à ses besoins. Par conséquent, de nombreux chercheurs ont choisi d'utiliser les approches à base de composants afin de diminuer le coût et le temps de réalisation et d'augmenter la qualité des systèmes informatiques.

Les systèmes multi-agents présentent des caractéristiques importantes à la conception des SAAs, en particulier le couplage lâche, la sensibilité au contexte et la robustesse aux défaillances et aux événements inattendus [114]. Dans une certaine mesure, le couplage lâche et la sensibilité au contexte sont également présents dans les systèmes orientés-services. Mais les SMAs auto-adaptatifs étendent ces mécanismes au comportement de chaque composant de l'agent [114].

Dans ce chapitre, il est nécessaire de fournir un contexte lié à l'informatique autonomique et au SMA avant d'entrer au cœur de cette thèse et de présenter les SAAs et la sensibilité au contexte (voir Chapitre 01), l'auto-adaptation des systèmes à base de composants (voir Chapitre 02) et les SAAs basés sur plusieurs boucles de contrôle (voir Chapitre 03) afin de fournir un contexte générale et d'introduire la terminologie utilisé tout au long de cette

1. Dans ce document, nous utilisons le terme anglais « MAPE-K » car il est le plus fréquemment utilisé dans la littérature.

thèse.

L'informatique autonomique

L'informatique autonomique vise à améliorer les systèmes informatiques par la réduction de l'implication humaine. Le terme « autonome » vient de la biologie. Dans le corps humain, le système nerveux autonome régit notre rythme cardiaque et notre température corporelle, libérant ainsi notre cerveau conscient de la charge de ces fonctions et de bien d'autres fonctions de bas niveau, mais vitales. Sans le système nerveux autonome, nous serions constamment occupés à adapter consciemment notre corps à ses besoins et à l'environnement. De la même manière, les capacités autonomes de l'auto-gestion anticipent les exigences du système de technologie de l'information et résolvent les problèmes avec une intervention humaine minimale [57] [60]. Le concept « Informatique autonomique » a été utilisé pour la première fois par IBM en 2001 pour décrire les systèmes informatiques qui sont auto-gérés [60].

▶ Propriétés auto-*

IBM a suggéré que les systèmes informatiques complexes devraient également avoir des propriétés autonomes, c'est-à-dire être capables de prendre en charge indépendamment les tâches de maintenance et d'optimisation régulières, réduisant ainsi la charge de travail sur les administrateurs système [60]. Ces propriétés comprennent :

- *Auto-configuration (self-configuring).* Le système autonome se configure lui-même en fonction des politiques de haut niveau, représentant par exemple des objectifs fournies par le professionnel de domaine.
- *Auto-réparation (self-healing).* Le système autonome détecte, diagnostique et répare les problèmes résultant des défaillances logicielles ou matérielles. Le système devrait alors faire correspondre la réparation avec des actions correctives, puis refaire le test.
- *Auto-optimisation (self-optimizing).* Le système autonome optimise son utilisation des ressources, cherchant continuellement les moyens d'améliorer leur fonctionnement. Le but est de rendre le système plus efficace en termes de performance ou de coût.
- *Auto-protection (self-protecting).* Le système autonome se protège contre les attaques malveillantes, aussi contre les utilisateurs finaux (Par exemple, la suppression d'un fichier important). Le système doit assurer de manière autonome la sécurité, la confidentialité et la protection des données.

Les propriétés auto-* ont pour but de maintenir les services assurés par le système. Par exemple, l'auto-configuration et l'auto-réparation sont utiles en cas de défaillance, l'auto-optimisation et l'auto-protection sont utilisées en présence des menaces.

Systèmes Multi-Agents

L'objectif de cette section est d'introduire des notions étroitement liées au domaine des SMAs. D'abord nous présentons la définition des SMAs et d'agent. Ensuite, nous détaillons les manières avec lesquelles les SMAs sont conçus pour s'adapter à la dynamique de l'environnement. Puis, nous décrivons les interactions entre les agents qui conduisant à des

SMAs. Aussi, nous présentons l'organisation comme processus de mise en ordre résultant des interactions des agents.

► Généralités sur les Systèmes Multi-Agents

Dans les années 80, le paradigme SMA est apparu issue de l'évolution du domaine de l'intelligence artificielle distribuée. Une multitude de définition de ce paradigme existe dans la littérature, mais y a pas de véritable consensus sur sa définition.

1) **Définition de SMA**

Un SMA est un système qui compose d'un ensemble d'agents logiciels autonomes, interagissant entre eux dans un environnement commun, dans le but de résoudre une tâche commune [120].

Plusieurs éléments sont à prendre en compte pour créer un SMA [38] :

- Les agents et leurs modèles comportementaux,
- L'environnement dans lequel ils sont implantés (spatialement ou non),
- Les interactions et les outils pour les gérer entre les agents,
- L'organisation structurelle des agents de la hiérarchie aux relations simples.

2) **Définition de l'agent**

Le concept « Agent » est utilisé dans des nombreuses applications comme une entité physique ou virtuelle autonome. En intelligence artificiel, l'*agent* est le composant principal des SMAs. Etant donnée la jeunesse du domaine SMA, il existe plusieurs définitions d'un *agent*. Une des définitions les plus populaires est celle donnée par Ferber [45]. Un *agent* est une entité logicielle qui :

- évolue dans un système informatique ouvert ;
- peut communiquer avec d'autres agents ;
- a ses propres objectifs ;
- possède ses propres ressources ;
- n'a qu'une représentation partielle des autres agents ;
- a des capacités (services) qu'il peut offrir à d'autres agents ;
- a un comportement qui tend à satisfaire ses propres objectifs, compte tenu de ses ressources, des représentations et des communications qu'il reçoit.

Donc, un *agent* est une entité virtuelle ou physique qui est capable de percevoir son environnement et d'effectuer des actions afin d'atteindre son propre but. Il peut communiquer avec d'autres *agents* pour les aider ou demander des services. Les principales caractéristiques de l'agent sont : coopération, coordination et négociation [118].

L'agent est fonctionné selon le cycle de vie classique *Perception-Décision-Action* :

- *Perception.* Cette phase permet à l'*agent* de mettre à jour sa représentation de ce qu'il perçoit localement de son extérieur (son environnement et les agents qui l'entourent).
- *Décision.* Durant cette phase, l'*agent* utilise son comportement pour lier les actions disponibles qui lui permettront d'atteindre au mieux ses objectifs compte tenu de ses nouvelles représentations (ou perceptions).

- *Action.* Durant cette phase, l'*agent* effectue effectivement l'action choisie, ce qui implique des changements dans son environnement et les agents qui l'entourent.

Woolridge [119] propose une manière plus précise de distinguer un *agent*, où plusieurs propriétés d'agents peuvent être identifiées :

- *Autonomie* est considérée comme une notion centrale de l'*agent*. Le système est capable de réagir sans l'intervention directe d'humains ou d'autres personnes, où les *agents* contrôlent leurs actions et leurs états internes.
- *Habilité sociale*, où les *agents* interagissent avec d'autres agents avec lesquels il partage la réalisation des tâches.
- *Réactivité*, où les *agents* perçoivent leur environnement et réagissent dans le temps requis aux changements qui se produisent.
- *Pro-activité*, où les *agents* prennent l'initiative et prennent des décisions dans leur environnement grâce à des comportements ciblés distincts.

3) **Environnement**

L'environnement correspond à l'ensemble des entités extérieures au système. Grâce à l'environnement, les *agents* peuvent interagir au travers et peuvent agir dessus. Selon Giunchiglia et al. [50] comme cité dans [102] :

> *«Without an environment, an agent is effectively useless. Cut off from the rest of its world, the agent can neither sense nor act [50].»*

La spécification de l'environnement permet de définir la dynamique du SMA. Deux types d'environnements peuvent être distingués : (1) l'environnement du système qui correspond à l'ensemble des entités extérieures au système, et (2) l'environnement de l'agent qui correspond à tout ce qui est extérieur à lui-même.

4) **Interaction**

La création d'un SMA nécessite une description précise de leurs interactions. L'interaction est la mise en relation d'un agent avec d'autres agents ou avec son environnement par le biais d'un ensemble d'actions réciproques [45].

5) **Organisation**

Les relations créées par les interactions entre agents constituent une organisation qui peut évoluer dans le temps. L'organisation permet de structurer les entités du système [46]. Le processus d'organisation est dynamique et toujours dans un état de réorganisation des entités et des liens qui relient ces entités.

Problématique

Le développement d'un SAA soulève de nombreux défis. Ce système logiciel met en œuvre un mécanisme de contrôle autonome appelé boucle MAPE-K. Une étude de la littérature montre que les conceptions existantes du MAPE-K sont trop générales, c'est-à-dire qu'elles manquent de détails suffisants sur la structure et la dynamique de ses sous-parties. De plus, ces conceptions n'utilisent pas des modèles de conception bien établis.

Lorsque les systèmes sont grands et distribués, il est nécessaire d'utiliser plusieurs boucles de contrôle pour mettre en œuvre la logique d'adaptation de manière correcte et cohérente.

En pratique, différents modèles de composition des boucles de contrôle ont été proposés, basés sur des techniques de centralisation et / ou décentralisation. La centralisation de l'architecture de contrôle peut se heurter à des difficultés telles que : la surcharge de communication, ainsi que lorsque le système est complexe, le temps de réaction est lent. De plus, concevoir et évaluer des architectures de contrôle décentralisées peuvent rencontrer certaines difficultés, car les ingénieurs doivent considérer de nombreux problèmes pour assurer la cohérence des adaptations, ainsi que le coût élevé de la coordination entre les différentes entités décentralisées de la boucle.

Un autre défi pour un SAA à grande échelle est de savoir comment les préoccupations peuvent-elles être séparer entre les différentes boucles de contrôle ?. Selon Weyns et al. [117], la hiérarchie reflète généralement la séparation des préoccupations entre les différentes boucles de contrôle. Le contrôle hiérarchique peut être considéré comme un point intermédiaire entre un contrôle entièrement décentralisé et centralisé. Trouver une solution à ce problème nécessite l'utilisation d'un modèle hiérarchique pour la conception des systèmes afin de faciliter la séparation des préoccupations, permettre un développement fiable et améliorer l'évolutivité du système.

Dans le contrôle hiérarchique, il y a toujours un certain degré de décentralisation. En règle générale, les entités décentralisées nécessitent une coordination pour éviter les conflits entre les boucles de contrôle. De nombreux travaux ont été proposés des modèles de coordination, mais il y a toujours la possibilité de souffrir du coût élevé de la transmission des données. D'autre part, la majorité des systèmes dépendent d'un modèle de composition des boucles de contrôle unique. Compte tenu de la nature évolutive et dynamique des SAAs, il est nécessaire de rendre le sous-système de gestion également capable de modifier sa structure et donc le comportement du système au moment de l'exécution.

Objectif

L'objectif principal de cette thèse est de proposer une approche de composition hiérarchique des boucles de contrôle MAPE-K afin d'améliorer l'efficacité de l'adaptation dans les SAAs.

Ainsi, cette thèse vise les objectifs suivants :

- Développer une boucle de contrôle MAPE-K entière et définir chaque phase séparément pour lever l'ambiguïté sur le principe de fonctionnement de chaque phase et également pour faciliter la conception des SAAs.
- Proposer un modèle de contrôle partiellement centralisé pour éviter des décisions conflictuelles ainsi que des actions inutiles et peut-être redondantes.
- Proposer un modèle de composition hiérarchique des boucles de contrôle, nommé HCLs (Hierarchical Control Loops) pour gérer la complexité de l'auto-adaptation en séparant les préoccupations à l'aide du modèle de composants Fractal sous la forme d'une hiérarchie de boucles MAPE. De plus, nous nous appuyons sur l'apprentissage par renforcement et le modèle de caractéristiques pour explorer des stratégies d'adaptation.

Contributions

Afin d'atteindre notre objectif en mettant en œuvre ce cadre, nous apportons les contributions suivantes :

- **C1 :** Une extension étendue de la boucle de contrôle MAPE-K est proposée. Nous définissons chaque phase séparément pour faciliter le processus de la conception des SAAs. De plus, nous nous appuyons sur un modèle de composant pour créer une boucle basée sur les composants afin de réduire la complexité de la conception, du développement et d'administration de ces systèmes.
- **C2 :** Nous proposons un modèle de contrôle partiellement centralisé pour assurer une prise de décision correcte et cohérente afin de parvenir à une adaptation globale sans conflit. Les composants Monitor, Analyze et Execute sont décentralisées, tandis que les composants Plan et Knowledge sont centralisées. Nous modélisons le composant Plan comme un composant partagé, utilisant un modèle de composant Fractal.
- **C3 :** Une amélioration a été apportée à C2 pour gérer la complexité de l'auto-adaptation qui s'appuie sur le contrôle hiérarchique afin de proposer le modèle HCLs, un modèle de contrôle hiérarchique. Au début, nous nous appuyons sur le modèle de composant Fractal pour construire l'application afin de faciliter la séparation des préoccupations, donc de faciliter l'attachement des entités MAPE à la position appropriée dans la hiérarchie de l'application. De plus, nous nous appuyons sur l'exploration des stratégies d'adaptation utilisant l'apprentissage par renforcement et le modèle de caractéristiques pour améliorer l'évolutivité du système.
- **C4 :** Une démonstration de notre approche est fournie par une étude de cas de gestion d'une connexion Internet sans fil dans un aéroport pour valider l'approche proposée HCLs, en utilisant le modèle de composant auto-adaptatif « SAFRAN ».

Organisation de la thèse

Cette thèse est divisée en deux parties. La première partie qui présente l'état de l'art est composée de trois chapitres. La deuxième partie présente la contribution. Elle comporte trois chapitres.

La première partie de la thèse est organisée comme suit :

- **Chapitre 01 : Systèmes Auto-Adaptatifs et la sensibilité au contexte.** Ce chapitre constitue l'état de l'art des SAAs et la sensibilité au contexte, et résume les concepts de base pour comprendre la problématique et la contribution de cette thèse. Il présente également quelques définitions et la terminologie appliquée tout au long de ce document. À la fin de ce chapitre, nous nous concentrons sur la boucle de contrôle MAPE-K et ses travaux associés.
- **Chapitre 02 : Auto-adaptation des systèmes à base de composants.** Ce chapitre dresse un état de l'art de l'auto-adaptation des systèmes à base de composants. Après un rappel des concepts clés du paradigme à base de composants, nous décrivons la notion de composant logiciel, qui est au cœur de notre travail. Ensuite, nous présentons quelques modèles de composant tels que : Fractal et directMOD. Puis, nous

présentons quelques SAAs à base de composants. Ce chapitre se termine par une comparaison entre ces modèles afin de déterminer le modèle qui est l'une des bases de notre travail.

- ✲ **Chapitre 03 : SAAs basés sur plusieurs boucles de contrôle.** Ce chapitre commence par la présentation d'un ensemble de défis de la recherche affectant les SAAs à boucles de contrôle. Ensuite, nous décrivons les types d'interactions entre les différents composants de la boucle de contrôle MAPE-K. Puis, nous présentons les différents approches de contrôle : centralisé, décentralisé et hiérarchique. Enfin, nous présentons un ensemble de travaux précédemment proposés, en soulignant les défis à considérer qui ont conduit à notre proposition.

La deuxième partie de cette thèse se compose de trois chapitres et elle est consacrée à présenter nos contributions. Cette partie se concentre sur la présentation de nos approches nouvellement développées résolvant la problématique de cette thèse décrite ci-dessus.

- ✲ **Chapitre 04 : Un Modèle à composants de la boucle de contrôle MAPE-K.** Ce premier chapitre de la contribution présente au début un ensemble de caractéristiques importantes du modèle de composant Fractal et sa réflexion sur la flexibilité de la boucle de contrôle. Puis, nous présentons un modèle intégré de la boucle de contrôle MAPE-K à base de composants. En utilisant le modèle de composant Fractal, nous proposons une architecture d'une boucle MAPE-K avec suffisamment de détails sur la structure et la dynamique de ses sous-parties. À la fin de ce chapitre, la visualisation du comportement d'une boucle de contrôle à base de composants sera facile et compréhensible.
- ✲ **Chapitre 05 : Un modèle de contrôle partiellement centralisé.** Ce chapitre décrit un modèle de composition de plusieurs boucles de contrôle partiellement centralisé. Tout d'abord, nous présentons un langage de spécification de modèles MAPE, appelé MSL. Puis, nous présentons un modèle de contrôle partiellement centralisé qui consiste à centraliser les composant Plan et Knowledge, et à décentraliser les composants Monitor, Analyze et Execute. Dans notre modèle, il n'y a qu'une seule instance de composants Plan et Knowledge, nous le modélisons comme des composants partagés. Nous nous appuyons sur le modèle de composant Fractal qui prend en charge le partage de composant. De plus, les composants Execute sont coordonnés les uns avec les autres. À la fin de ce chapitre, nous décrivons l'architecture de ce modèle basé sur le composant et sur le langage MSL.
- ✲ **Chapitre 06 : Une composition hiérarchique des boucles MAPE-K.** Ce chapitre décrit une solution architecturale proposée pour prendre en charge correctement l'auto-adaptation des SAAs. Nous proposons le modèle de composition hiérarchique des boucles de contrôle dans les SAAs. L'approche proposée permet de gérer la complexité de l'auto-adaptation en séparant les préoccupations à l'aide du modèle de composant Fractal sous la forme d'une hiérarchie de boucles MAPE. Où, ces boucles de contrôle doivent interagir et coordonner pour éviter les conflits et apporter certaines garanties d'adaptation. Nous nous sommes donc appuyés sur des stratégies d'adaptation pour l'apprentissage par renforcement en ligne, utilisant le modèle de caractéristiques pour définir l'espace d'adaptation. Enfin, une étude de cas est présentée pour illustrer notre approche dans ce chapitre.

Enfin, la thèse s'achève par une conclusion générale, qui résume l'apport réalisé et des perspectives pouvant être développées par la suite, ouvrant ainsi de nouvelles voies à envisager pour des recherches futures.

Articles publiés

1- Selma Ouareth, Soufiane Boulehouache, and Smaine Mazouzi. *A Survey of Uncertainties in MAPE-K Control Loop.* International Conference on Advanced Technologies, Computer Engineering and Science (ICATCES 2018). ISBN :978-605-9554-17-6. P : 447- 452. 11-12-13 May 2018. Safranbolu, Turkey.

2- Selma Ouareth, Soufiane Boulehouache, and Smaine Mazouzi. *A component-based MAPE-K control loop model for self-adaptation.* In 2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS), pp. 1-7. IEEE, 2018.

3- Soufiane Boulehouache, Selma Ouareth, and Ramdane Maamri. *A WildCAT Based Observable Bayesian Student Model.* International Conference on Cyber Security and Computer Science (ICONCS'18). 18-19-20 October 2018. Safranbolu, Turkey.

4- Soufiane Boulehouache , Selma Ouareth, and Ramdane Maamri. *A self-switching multi-strategic pedagogical agent.* Journal of King Saud University-Computer and Information Sciences 32, no. 4 (2020) : 505-513.

5- Selma Ouareth, Soufiane Boulehouache, and Mazouzi Smaine. *Reliable Composition of MAPE-K Loops in Self-Adaptive Software Systems.* International Journal of Organizational and Collective Intelligence (IJOCI) 10, no. 3 (2020) : 38-54.

6- Selma Ouareth, Soufiane Boulehouache, and Smaine Mazouzi. *An Approach for Composing Multiple Control Loops Hierarchically.* In 2021 International Conference on Theoretical and Applicative Aspects of Computer Science (ICTAACS), pp. 1-5. IEEE, 2021.

7- Selma Ouareth, Soufiane Boulehouache, and Smaine Mazouzi. *Self-Adaptation Through Reinforcement Learning Using a Feature Model.* International Journal of Organizational and Collective Intelligence (IJOCI) 12, no. 4 (2022) : 1-20.

Première partie

Etat de l’art

Chapitre 1

Systèmes Auto-Adaptatifs et la sensibilité au contexte

1.1 Introduction

La complexité croissante des systèmes informatiques actuels, tels que l'informatique omniprésente et l'internet des objets, a encore accru le besoin de Systèmes Auto-Adaptatifs (SAAs). Aujourd'hui, l'un des principaux défis du génie logiciel est le développement des SAAs qui peuvent maintenir la flexibilité du système tout au long de leur cycle de vie [83].

Les SAAs visent à ajuster divers artefacts ou attributs en réponse aux changements du système lui-même et du contexte de système. Le contexte englobe tout ce qui est dans l'environnement d'exploitation qui affecte les propriétés du système et son comportement [97]. Par conséquent, la sensibilité au contexte joue un rôle majeur dans le développement et la mise en œuvre des SAAs.

De nos jours, les SAAs s'appuient sur des boucles de contrôle pour mettre en œuvre l'auto-adaptation. L'une des boucles de contrôle les plus populaires est la boucle de contrôle MAPE-K. Dans ce chapitre, nous commençons par introduire le contexte de notre travail à savoir la sensibilité au contexte et les SAAs. Ensuite, nous présentons l'architecture conceptuelle des SAAs. Puis, nous présentons une taxonomie des SAAs qui nous permet de fournir une vision unifiée de ce domaine émergent. Aprés, nous essayons de mettre en évidence le modèle de référence MAPE-K et les détails de chaque étape séparément. Enfin, nous présentons les modèles de contrôle autonomique précédemment proposés et nous concluons avec une synthèse.

1.2 Sensibilité au contexte

Au regard de la littérature, le *contexte* est un élément clé, car il est au centre des mécanismes d'adaptation prônés par les systèmes sensibles au contexte [81]. Dans cette section nous présentons tout d'abord le concept *contexte*, puis nous présentons les systèmes sensibles au contexte.

1.2.1 Qu'est-ce que le contexte ?

De nombreuses études ont été réalisées pour définir le concept *contexte* dans divers domaines des sciences comme l'économie, l'informatique et la philosophie. Dans cette section, nous présentons quelques unes dans le domaine de l'informatique :

> *«Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [39]. »*

Li et al. [72] donnent une autre définition pour le contexte.

> *«Context is any piece of information that can represent changes of the circum-*

stance (either static or dynamic). Further, it could be useful for understanding the current situation and predicting potential changes [72]. »

Récemment, The Can Do [110] a également proposé la définition suivante :

«Context is the set of possible situations for a set of specific viewpoints [110]. »

D'après ces définitions, le *contexte* fait référence à diverses informations qu'une application peut utiliser pour effectuer une adaptation. Le *contexte* peut inclure une personne, un lieu ou un objet. Il inclut également l'utilisateur et l'application elle-même.

Le contexte peut être divisé en deux parties selon les aspects internes ou externes de l'application [51] :

► Contexte externe

Le contexte externe d'une application peut être divisé en types [2, 30, 68], afin d'aider les concepteurs d'applications à découvrir les éléments de contexte externe les plus susceptibles d'être utiles dans leurs applications. La table 1.1 présente les différents types de contexte externe et quelques éléments de chaque type de contexte.

Types de contexte externe	**Élements de contexte externe**
Contexte Machine	- Occupation des ressources physiques. - URL et URI - Serveurs à proximité
Contexte Environnemental	- Objets (ou personne) à proximité - Climat - Bruit
Contexte Utilisateur	- Localisation - Émotion - Pression artérielle
Contexte Temporel	- Date du Système - Heure du système - Historique des actions

TABLE 1.1: Aperçu des différents types du contexte externe [2, 30, 68].

► Contexte interne

Le contexte interne d'une application est constitué d'informations provenant des composants de l'application, telles que les composants utilisés, les connexions entre les composants et l'état actuel de l'application (valeurs des attributs de l'assemblage de composants).

En effet, le contexte prend une étendue plus large sous le nom de la sensibilité au contexte [15]. La sensibilité au contexte représente la capacité à percevoir l'information contextuelle en observant le système, l'utilisateur et l'environnement afin d'adapter ses fonctionnalités face aux changements d'une façon dynamique et proactive [81]. Selon Schilit et Theimer [100], la sensibilité au contexte implique la capacité d'une application à découvrir et à réagir aux modifications dans l'environnement où se trouve l'utilisateur.

1.2.2 Systèmes sensibles au contexte

Dans ce cadre, un système sensible au contexte est un système ou un composant de système permettant de percevoir la situation de l'utilisateur dans son environnement à un moment donné [110]. Puis, il adapte par conséquent son comportement à la situation en question. De nombreux chercheurs ont proposé des définitions du système sensible au contexte telles que :

> *«if an application has the ability to monitor input from sensing devices and choose the suitable context according to user's need or interests, then it can be labeled as a context-aware application [95].»*

Dey et Abowd [40] donnent une autre définition pour les systèmes sensibles au contexte.

> *«A system is context-aware if it uses context to provide relevant information and / or services to the user, where relevancy depends on the user's task [40].»*

Ces systèmes sont conçus pour analyser en permanence les informations contextuelles [41]. Divers capteurs capturent les données qui créent des informations de contexte. Où, il utilise des logiciels et des matériels pour collecter et analyser les données afin de guider automatiquement les réponses. Il se concentre sur la modélisation et le raisonnement sur des environnements pertinents en utilisant une ontologie, un méta-modèle, ... etc. pour répondre à un changement anticipé d'application ou de configuration [110]. Il existe de nombreux travaux qui ont été réalisés dans le domaine du framework sensible. Dans cette section, nous présentons quelques frameworks sensibles au contexte basés sur l'application sensible au contexte et l'interaction avec la vue contextuelle [110].

- *PersonisAD* [9] est un cadre sensible au contexte qui s'appuie sur la modélisation homogène de toutes les entités pertinentes telles que les personnes, les capteurs, les appareils et les lieux pour prendre en charge des applications ubiquitaires. Cette approche fournit une technique de modèle de contexte qui s'organise sous la forme d'un arbre du modèle de contexte, qui contient des composants du modèle. De plus, le framework PersonisAD apporte également une réponse puissante et cohérente à l'évolution des informations de contexte. Le principal pouvoir de ce cadre est de soutenir la capacité de services ubiquitaires personnalisés, d'intégrer la collecte de divers types de preuves sur l'environnement ubiquitaire et de l'interpréter de manière flexible. La limite de cette étude est le manque de stockage des informations contextuelles. Les

auteurs ne fournissent pas non plus de mécanisme pour résoudre les conflits entre les adaptations générées par le framework PersonisAD.

- *Costa* [27] a proposé une solution intégrée pour le développement d'un système sensible au contexte, se concentrant sur la modélisation du contexte et la plate-forme de gestion du contexte. Leurs abstractions de modélisation de contexte soutiennent les concepteurs d'applications avec des bases conceptuelles appropriées qui sont utilisées pour s'adapter aux exigences d'une application spécifique. Cette approche propose également un modèle pour la spécification d'une situation dans l'application sensible au contexte, et les comportements d'application sensibles au contexte peuvent être présentés comme des règles logiques, appelées règles de contrôle d'événement et d'action. Ils utilisent un composant de processeur de contexte pour gérer les problèmes de contexte, recueillir des informations contextuelles à partir de l'environnement de l'utilisateur et générer des événements de contexte et de situation. Dans cette approche, ils utilisent des composants de contrôleur pour surveiller les règles de conditions et observer les événements via des processeurs de contexte. Lorsque la condition est satisfaite, le composant contrôleur déclenchera des actions sur l'exécuteur d'action. Cette solution peut simplifier la tâche du concepteur en matière de développement d'applications et améliorer la capacité à intégrer plusieurs applications dans le même système. Cependant, cette étude n'indique pas le problème des conflits entre les adaptations d'applications et de solution.
- Achilleo et al. [3] ont proposé une méthodologie basée sur un modèle qui prend en charge la création d'un cadre de modélisation de contexte. Il peut simplifier la tâche de conception et de mise en œuvre de services omniprésents. Leur méthodologie basée sur des modèles offre également un niveau d'automatisation plus élevé dans la génération de logiciels. Ils ont construit un cadre de création de services pervasifs sensible au contexte, *CA-PSCF* (Context-Aware Pervasive Service Creation Framework) qui permet au concepteur de définir un modèle de contexte à un niveau abstrait. La CA-PSCF prend en charge une validation des modèles de contexte pour déterminer les définitions valides et non valides, puis elle rectifie ces définitions non valides avant la phase de mise en œuvre. Cette solution permet aux concepteurs de définir des modèles de contexte indépendants suivant leurs préoccupations et de les réutiliser pour différentes applications.

La communauté du génie logiciel a pris conscience du rôle majeur que joue la sensibilité au contexte dans le développement et la mise en œuvre de SAA capable de s'adapter aux changements importants de l'environnement et des exigences. Dans ce qui suit, nous nous concentrerons plus particulièrement sur les SAAs.

1.3 Systèmes Auto-Adaptatifs

De nombreuses recherches sur les SAAs ont été effectuées et de nombreux travaux sont en cours. Krupitzer et al. [62] et Salehie et al. [97] ont présentés des enquêtes approfondies sur les SAAs en général ainsi que sur les taxonomies pour unifier et améliorer la compréhension des concepts présents dans ce domaine de recherche.

1.3.1 Définition d'un Système Auto-Adaptatif

Parmi plusieurs définitions existantes pour les SAAs (en anglais Self-Adaptive Systems « SASs »), une est fournie par Oreizy et al. [83] comme suit :

> *«Self-adaptive software modifies its own behavior in response to changes in its operating environment [83]. »*

Une définition similaire est proposée par Laddaga [63], qui est :

> *«Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible [63]. »*

A travers leurs définitions, Oreizy et al. [83] et Laddaga [63] s'accordent sur le principe des SAAs comme des systèmes qui modifient leur comportement. Néanmoins, lorsqu'ils examinent les raisons de l'adaptation, ils adoptent des points de vue différents : Oreizy et al. [83] se concentrent sur l'extérieur du système en s'intéressant aux changements de l'environnement, tandis que Laddaga [63] considère l'intérieur du système en s'intéressant au fonctionnement de ce dernier.

Cheng et al. [10] donnent une autre définition pour les SAAs.

> *«self-adaptive system is a system that is able to adjust its behavior in response to their perception of the environment and the system itself. »*

D'après ces définitions [83, 63, 10], nous pouvons exprimer les SAAs comme suit : les SAAs ont la capacité de s'adapter son comportement et sa structure interne en fonction de sa perception de l'environnement, du système lui-même et de ses objectifs. Par exemple, la réponse aux défaillances, la variabilité des ressources disponibles ou la modification des priorités des utilisateurs.

1.3.2 Architecture conceptuelle d'un SAA

L'architecture conceptuelle d'un SAA est décrit par la figure 1.1. Les éléments constitutifs de cette architecture sont : *sous-système de gestion* et *sous-système géré*. Le sous-système de gestion implémente la logique d'adaptation qui gère le sous-système géré [6]. Ce dernier implémente la logique de domaine qui fournit les fonctionnalités du système [117, 6]. Le SAA fonctionne dans un environnement non contrôlable [6]. L'environnement fait partie du monde externe avec laquelle le SAA interagit et dans laquelle les effets du système seront observés et évalués [59]. L'environnement peut représente toutes entités matérielles et logicielles [117]. Par exemple, l'environnement d'un système robotique comprend des entités physiques telles que des caméras externes et des pilotes de logiciels correspondants [117].

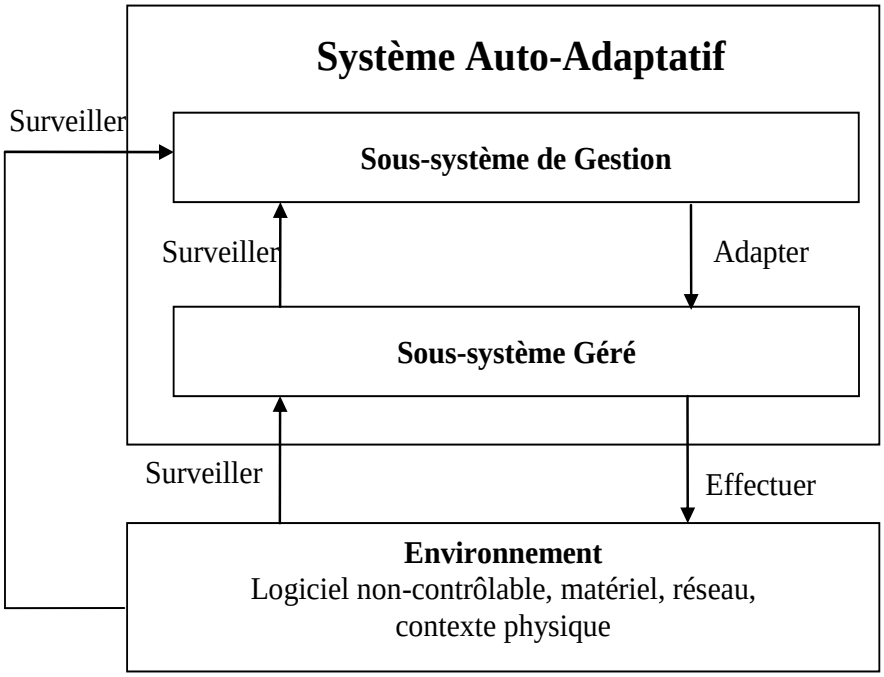


FIGURE 1.1: Architecture conceptuelle d'un SAA [6].

► Sous-Système Géré

Le sous-système géré représente un ensemble de ressources gérées à laquelle un comportement autonome est attribué en le couplant à un sous-système de gestion [57]. Il comprend la logique d'application qui fournit les fonctionnalités essentielles de l'application [117]. Selon Weyns et al. [117], le sous-système géré surveille et affecte l'environnement pour effectuer ses fonctionnalités. En plus, pour prendre en charge les adaptations, le sous-système géré doit prendre en charge la surveillance (en utilisant des capteurs) et l'exécution des adaptations (en utilisant des actionneurs) [117]. Par exemple : dans le cas des robots, la navigation du robot et le transport de la charge sont effectués par le sous-système géré [117].

Dans la littérature, différents termes sont utilisés à la place du concept « Sous-Système Géré ». Pa exemple : élément géré (managed element)[60], couche système (system layer) [49], fonction centrale (core function)[97] et plante contrôlable (controllable plant) [48].

► Sous-Système de Gestion

Le sous-système de gestion gère le sous-système géré [117]. Il comprend la logique d'adaptation qui traite un ou plusieurs préoccupations [117]. La logique d'adaptation implémente une boucle de contrôle qui évolue et adapte la logique d'application [6]. Pour atteindre les objectifs d'adaptation, le sous-système de gestion surveille l'environnement et le sous-système géré pour adapter ce dernier si nécessaire [117]. Par exemple, un robot peut être équipé d'un sous-système de gestion. Ce dernier permet au robot d'adapter sa stratégie

de navigation pour garantir qu'un certain nombre de fonctionnalités sont exécutées dans des conditions de fonctionnement changeantes, telles que la modification de la charge de la tâche ou la réduction de la bande passante pour la communication [117].

Dans la littérature, différents termes sont utilisés à la place du concept « Sous-Système de Gestion ». Exemple : gestionnaire autonome (autonomic manager)[60], couche d'architecture (architecture layer) [49], moteur d'adaptation (adaptation engine)[97] et contrôleur (controller) [48].

1.3.3 Taxonomie des SAAs

Différentes taxonomies ont été développées au fil du temps pour capturer les exigences d'adaptation. Krupitzer et al. [62] ont mené une vaste revue de la littérature sur l'auto-adaptation et ont proposé une taxonomie basée sur les résultats de la revue. La taxonomie comprend cinq dimensions : la raison, le temps, la technique, le niveau et le contrôle de l'adaptation (voir la figure 1.2).

- **Raison.** L'identification de la raison d'adaptation est importante pour la décision d'adaptation. Dans un SAA, l'adaptation peut être déclenchée par des changements dans un ou plusieurs éléments du système : (i) un changement dans le contexte, par exemple : l'état d'une variable de contexte a changé, (ii) un changement des ressources techniques, par exemple : un défaut d'un composant matériel ou un défaut logiciel, ou (iii) un changement concernant le(s) utilisateur(s), par exemple : un changement de la composition du groupe d'utilisateurs ou des préférences de l'utilisateur. Il est fondamental d'identifier la raison de l'adaptation et d'en raisonner pour prendre une décision d'adaptation. De plus, les raisons de l'adaptation doivent être incluses dans les aspects de l'auto-adaptation, car elles déterminent les éléments à surveiller.
- **Niveau.** La dimension niveau fait référence au niveau spécifique auquel une adaptation est exécutée sur un système. Le SAA est composé de différents éléments : le(s) élément(s) géré(s) et la logique d'adaptation [117]. Les éléments gérés sont composés de différents niveaux : (i) l'application peut être une application exécutée sur un seul appareil ou une application distribuée divisée en parties d'application s'exécutant simultanément sur différents appareils, (ii) un système logiciel contrôle le matériel (par exemple : ordinateur, robots, ...etc.). il peut être un système d'exploitation ou un middleware (dans le cas de systèmes distribués), (iii) les techniques de communication autonome permettent une adaptation au niveau réseau [42], (iv) les ressources techniques sont du matériel et d'autre ressources gérés telles que des ordinateurs, des smartphones, ...etc., (v) le contexte du système est un niveau supplémentaire en plus des ressources techniques et des logiciels.
- **Temps.** L'aspect temporel fait référence au moment où une adaptation est exécutée par rapport à un changement. Handte et al. [54] divisent les aspects temporels en deux dimensions : (i) l'adaptation réactive qui décrit une situation, dans laquelle un système identifie le besoin d'adaptation avant qu'une baisse de performance ne se produise, et (ii) l'adaptation proactive qui décrit une situation, dans laquelle un système prédit quand le changement va se produire.

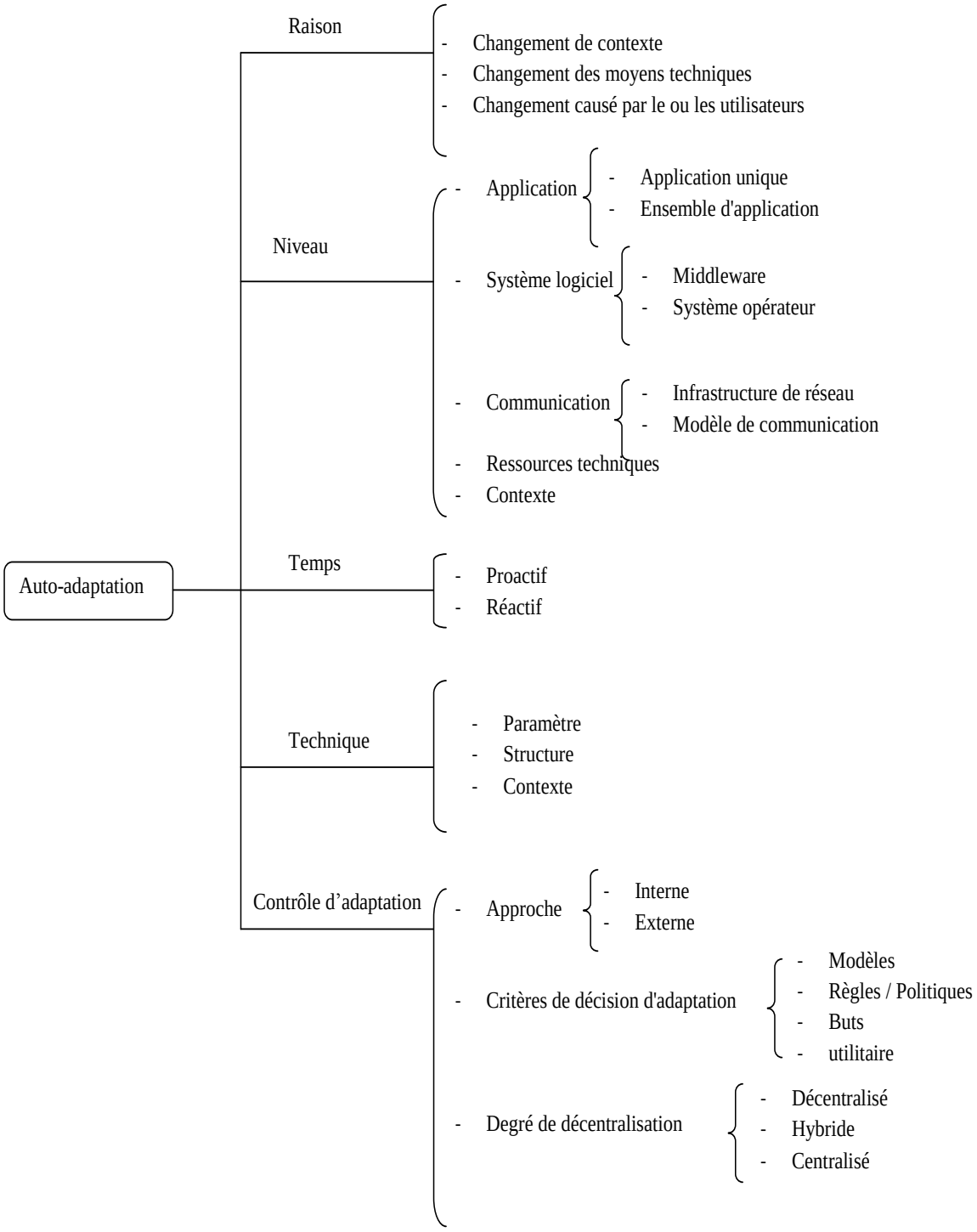


FIGURE 1.2: Taxonomie de l'auto-adaptation [62].

✻ **Technique.** Dans la littérature, différentes techniques d'adaptation peuvent être utilisées pour adapter les éléments gérés : (i) l'adaptation des paramètres permet d'obtenir un comportement du système modifié en ajustant les paramètres du système, (ii) l'adaptation de la structure permet d'échanger des composants défectueux afin d'évi-

ter des pertes de performances, d'améliorer les performances en ajoutant de nouveaux composants ou d'adapter le système à de nouvelles circonstances, ou (iii) l'adaptation au contexte, où les systèmes peuvent modifier le contexte dans lequel les systèmes fonctionnent (selon l'avis de Krupitzer et al. [62]).

- **Contrôle d'adaptation.** Cela implique la manière dont le processus d'adaptation est contrôlé dans les SAAs. Dans cette taxonomie, le contrôle de l'adaptation est divisé comme suit : (i)l'approche d'adaptation peut être interne (c'est-à-dire entrelacée avec les éléments gérés) ou externe (c'est-à-dire séparée les éléments gérés), (ii)les critères de décision d'adaptation permet d'analyser les différentes possibilités d'adaptation afin de décider comment s'adapter, et (iii) le degré de décentralisation est un aspect de la logique d'adaptation qui peut être *centralisé* notamment pour les systèmes avec peu de ressources à gérer, *décentralisé* notamment pour les systèmes avec de nombreux composants à gérer et *hybride* dans le cas de la distribution de la logique d'adaptation aux sous-systèmes.

1.4 Boucle de Contrôle MAPE-K

La boucle de contrôle MAPE-K est l'une des principales approches pour réaliser l'auto-adaptation dans les SAAs. Ce modèle se compose de *capteurs* et d'*effecteurs* ainsi que quatre phases clés : Surveiller (Monitor), Analyser (Analyze), Planifier (Plan), et Exécuter (Execute), complétées d'une base de connaissance (Knowledge), qui permet le transfert des informations entre les autres phases. Les capteurs et les effecteurs font partie du sous-système géré.

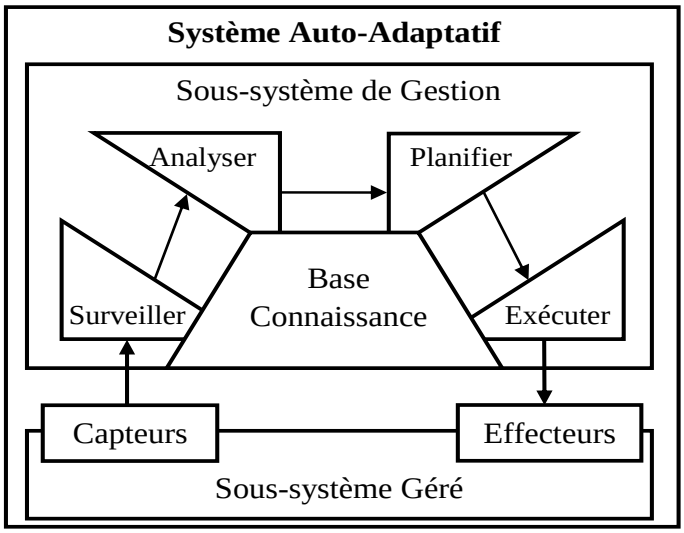


FIGURE 1.3: Boucle de contrôle autonomique MAPE-K [60].

- *Capteurs (Sensors)* permettant la surveillance et la collecte d'informations sur l'environnement et le sous-système géré lui-même [57]. Par exemple : capturer le temps de réponse aux demandes des clients pour un serveur Web [105]. Selon Ballagny [11], ces capteurs peuvent vérifier les valeurs des paramètres fonctionnels tel que le taux

de perte de paquets sur un réseaux [96], l'introduction de nouveaux composants [91], les ressources, ... etc. Les capteurs peuvent être n'importe quel composant logiciel ou matériel [57].

- *Effecteurs (Effectors)* souvent appelés actionneurs, permettant d'apporter des modifications au sous-système géré [57]. En d'autres termes, les effecteurs sont utilisés pour agir sur le système [11]. Par exemple : changer les paramètres de configuration dans un serveur Web [105, 101].

La figure 1.3 montre l'architecture de la boucle de contrôle MAPE-K. Les phases de cette boucle autonomique sont définies comme suit :

1.4.1 Surveiller (Monitor)

La phase de surveillance [1] consiste à collecter les données qui caractérisent le comportement des ressources gérées [70], via des capteurs [57]. Ces données peuvent inclure des informations de topologie, des métriques, des paramètres de propriétés de configuration, ... etc. [26]. En outre, les données collectées comprennent des informations sur la configuration des ressources gérées, leur état, la capacité offerte et le débit [26]. De plus, ces données accumulées doivent être traitées avant les corréler et les stocker [26, 18]. En règle générale, le processus de pré-traitement peut inclure le filtrage et l'agrégation [18]. Ensuite, la corrélation exige à déterminer les symptômes associés à la combinaison particulière de données pré-traitées [26], puis les stocker dans la base de connaissance (Knowledge) pour référence future [18]. Finalement, ces symptômes sont transmis à la phase d'analyse (Analyze) [26].

Mohamed et al. [80] supposent différents types de surveillance :

- *Surveillance de propriété modifiée (Property Changed Monitoring).* La ressource surveillée envoie des notifications chaque fois qu'une propriété surveillée est modifiée.
- *Surveillance des appels de méthode (Method Call Monitoring).* La ressource surveillée envoie des notifications chaque fois qu'une des méthodes du service est appelée.
- *Surveillance du temps d'exécution (Execution Time Monitoring).* La ressource surveillée notifie le temps d'exécution chaque fois qu'une invocation de service s'est produite.

D'autre part, Huebscher et McCann [57] identifient deux modes de surveillances dans les systèmes autonomiques :

▶ *Surveillance Active*

Dans le cas où le sous-système géré ne fournit pas des mécanismes pour surveiller son état interne et ses métriques, il est nécessaire de mettre en place des capteurs associés au sous-système géré pour permettre l'obtention des métriques pertinentes [47]. Par exemple, la modification et l'ajout de code à la mise en œuvre de l'application ou du système d'exploitation pour capturer la fonction ou les appels système [57].

1. Il est désigné par le symbole M

Lorsque l'élément géré n'offre pas le mécanisme pour surveiller son état interne et ses métriques, il est nécessaire de modifier l'élément géré d'une certaine manière pour permettre l'obtention de métriques pertinentes.

► ***Surveillance Passive***

Le sous-système géré fournit les mécanismes permettant de prendre en charge la surveillance des données de mesure [47]. La surveillance passive d'un système informatique peut être facilement effectuée à travers les interfaces du système d'exploitation [57, 47]. Par exemple : dans Linux, la commande *top* renvoie des informations sur l'utilisation du processeur par chaque processus et la commande *vmstat* renvoie les statistiques d'utilisation de la mémoire et du processeur [57].

La collecte d'informations distingue deux types : la *collecte instantané* qui peut s'effectuer à chaque occurrence d'événement et la *collecte par échantillonnage* qui peut s'effectuer de manière périodique [93].

1.4.2 Analyser (Analyze)

La phase d'analyse [2] reçoit les symptômes sous forme d'entrées fournies par la phase de surveillance. La phase d'analyse fournit des mécanismes permettant d'analyser ces symptômes [26]. Les moteurs de règles et les moteurs de corrélation peuvent être utilisés pour analyser les données contenues dans les symptômes afin d'extraire des situations qui nécessitent une adaptation [11]. Dans le cas où les modifications sont nécessaires, la phase d'analyse génère des demandes de changement. Logiquement, ces demandes de changement doivent ensuite être traitées par la phase de planification (Planifier) [26]. Par exemple, la nécessité d'établir une modification peut se produire lorsque la fonction d'analyse détermine qu'une stratégie n'est pas respectée [26].

1.4.3 Planifier (Plan)

La phase de planification [3] agit comme un module de décision [14], utilisant des algorithmes dont le rôle est de déterminer un plan d'action [69]. Sur la base des demandes de changement fournies par la phase d'analyse, la phase de planification produit la liste des actions d'adaptation à réaliser [11]. Au sens le plus large, la planification crée ou sélectionne un plan de changement approprié pour l'appliquer au sous-système géré [26]. Ce plan peut prendre de nombreuses formes, allant d'une commande unique à un flux de travail complexe [26]. De plus, le plan de changement reconnaît ce qui doit être modifié pour satisfaire aux exigences et comment effectuer cette modification [5]. La phase de planification est responsable à sélectionner le meilleur état possible pour atteindre les objectifs du système

2. Il est désigné par le symbole A
3. Il est désigné par le symbole P

[47]. En règle générale, ce plan de modification doit être transmis à la phase d'exécution (Execute) [26].

Différents techniques de décision automatique sont utilisés, on trouve les tables de décision et les approches basées sur des calculs d'utilité, les systèmes experts, les algorithmes de satisfaction de contraintes, les réseaux de neurones, les approches floues (contrôle et raisonnement) ou des combinaisons [23].

Plusieurs approches peuvent être utilisées pour prendre des décisions d'adaptation : C'est un ensemble de règles qui génère directement des plans d'adaptation à partir de combinaisons de certains événements

► *Politiques ECA (ECA policies)*

C'est un ensemble de règles qui produisent directement des plans d'adaptation à partir des combinaisons de certains événements [57]. La politique ECA *when Event if Condition do Action*, ça signifie que lorsqu'un événement spécifique se produit et, si une condition est vérifiée, un ensemble d'actions à exécuter [47]. L'exécution d'action permet d'atteindre un nouvel état [11]. Selon Ferreira [47], l'utilisation des politiques ECA peut se produire des conflits entre des règles différentes. Dans ce cas, les décideurs doivent connaître les détails de bas niveau de la fonction système [111] pour éviter les conflits. Par exemple, le système SAFRAN (Self-Adaptive FRActal CompoNents) qui utilise ce type de politique [34].

► *Politiques de buts (Goal policies)*

Les politiques de buts sont plus élaborées que les politiques ECA, car elles spécifient seulement l'état souhaité, et laissent au système la tâche de trouver comment atteindre cet état [47]. A partir d'un état courant, le système peut déterminer les actions qui permettent d'atteindre les états-cibles [11], en utilisant les connaissances sur le sous-système géré [47]. Parfois, la phase de planification peut être incapable de déterminer comment le système s'adapte, auquel cas il est impossible d'atteindre l'état souhaité [47].

► *Politiques de fonction d'utilité (Utility function policies)*

Les fonctions d'utilité résolvent le problème des politiques ECA en définissant un niveau quantitatif de satisfaction à chaque état [57]. A partir de l'état courant du système, la fonction d'utilité calcule l'indice de satisfaction pour chaque état cible, puis les états-cibles sont ordonnés en fonction de l'indice de satisfaction [11]. Le problème des fonctions d'utilité est qu'elles sont difficiles à définir, car chaque aspect affectant la fonction doit être quantifié [47].

1.4.4 Exécuter (Execute)

La phase d'exécution [4] est responsable de la mise en œuvre du plan des actions générées par la phase de planification [11]. En d'autres termes, c'est l'implémentation du plan d'actions en envoyant des commandes aux effecteurs de le sous-système géré [70]. Les actions d'adaptation peuvent être modifiées l'état d'une ou de plusieurs ressources géré [26], comme par exemple la modification d'un paramètre de configuration. De plus, ces actions peuvent être appliquées à l'environnement, par exemple l'ajout de ressources pour un cluster de serveurs, même si ces actions sont généralement très spécifiques au domaine de le sous-système géré [47]. L'exécution de l'adaptation est réalisée à travers la deuxième partie du mécanisme de réflexion qui est l'intercession (appelé aussi introaction), c'est la capacité du SAA à adapter sa propre configuration ou sa propre structure [23]. Par exemple, dans une architecture basé sur des composants, l'exécution d'actions implique l'ajout, la suppression ou le remplacement de composants, ainsi que la possibilité de modifier les liaisons entre composants [23].

1.4.5 Base de connaissance (Knowledge base)

La base de connaissance [5] contient les données utilisées par les quatre phases du sous-système de gestion (Surveiller, Analyser, Planifier et Exécuter) [26]. Ces connaissances comprend des données telles que les informations de topologie, les journaux historiques, les mesures, les symptômes et les politiques [26]. Les méthodes généralement utilisées pour représenter les connaissances dans les systèmes autonomiques sont [57] :

► *Concept d'utilité (Concept of Utility)*

Le fonctionnement d'un système exprime son utilité en tant que mesure d'éléments tels que la quantité de ressources disponibles pour l'utilisateur, la qualité, la fiabilité ou la précision de la ressource [57]. Par exemple, dans le cas d'un système d'approvisionnement en ressources, l'utilité découle de la consommation électrique en tant que partie des coûts de fonctionnement [84, 104].

► *Techniques bayésiennes (Bayesian Techniques)*

Les techniques bayésiennes sont basées sur des techniques probabilistes permettant de choisir le meilleur parmi un certain nombre de services ou d'algorithmes [57]. Par exemple, Guo [53] montre comment utiliser les réseaux bayésiens pour déterminer le meilleur algorithme autonomique.

4. Il est désigné par le symbole E
5. Il est désigné par le symbole K

► ***Apprentissage par renforcement (Reinforcement learning)***

L'apprentissage par renforcement consiste à apprendre des politiques en essayant des actions dans différents états et en examinant les conséquences de chaque action [108]. Grâce à l'apprentissage par renforcement, il est possible d'extraire des connaissances sur le sous-système géré [47]. La figure 1.4 montre le processus d'apprentissage par renforcement, qui se compose d'un agent et d'un environnement. L'agent effectue l'action **a** à exécuter au pas de temps **t** dans l'état (State) d'environnement **s**. Ensuite, l'agent reçoit une récompense (Reward) **r** et l'état suivant **s'** de l'environnement.

Le principal avantage de cette méthode est qu'elle ne nécessite pas de modèle du sous-système géré [74, 44]. Cependant, il souffre d'une faible évolutivité [57]. Pour éviter ce problème, les chercheurs proposent d'intégrer les stratégies d'exploration basées sur le modèle de caractéristiques [7] dans des algorithmes d'apprentissage par renforcement. Les algorithmes d'apprentissage par renforcement les plus couramment utilisés [107] sont Q-learning et SARSA. Q-Learning et SARSA utilisent des algorithmes d'apprentissage par renforcement sans modèle. Ils sont similaires dans les stratégies d'exploitation et diffèrent dans les stratégies d'exploration.

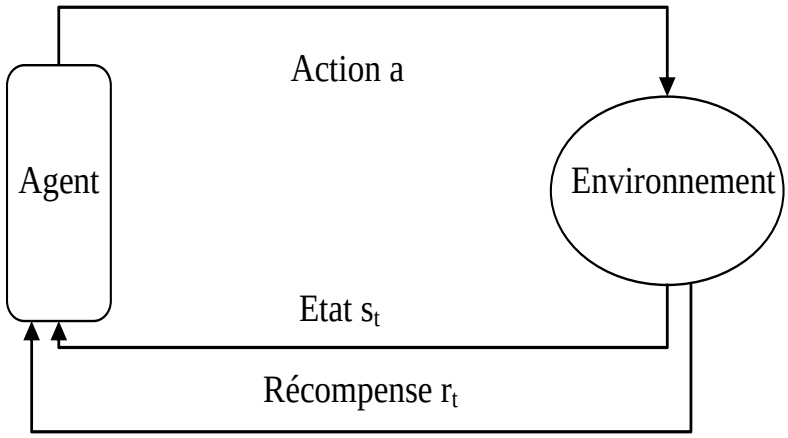


FIGURE 1.4: Processus d'apprentissage par renforcement [108].

Q-learning [113] est un algorithme d'apprentissage par renforcement sans modèle, proposé par Watkins. Cet algorithme fournit un moyen pratique à un agent d'apprendre à partir d'une sélection d'actions aléatoires pour assurer des performances optimales. L'objectif du Q-learning est d'estimer la valeur de qualité des paires *action-état*. La fonction de qualité d'apprentissage appelée « Q » représente une estimation de la récompense accumulée que l'algorithme obtient. La limitation de l'algorithme Q-Learning est de savoir comment choisir l'action appropriée dans un état spécifique. En conséquence, certaines politiques [122] sont apparues qui facilitent la sélection de l'action appropriée, telles que : Boltzmann, greedy policy et ϵ-greedy.

SARSA (State–Action–Reward–State–Action) [94] est un algorithme d'apprentissage

d'une politique de processus de décision de Markov, utilisé dans le domaine de l'apprentissage par renforcement automatique. Le principe de SARSA est met à jour la table **Q** avec les échantillons (**S, A, R, S'**) générés par la politique actuelle. (**S', A'**) est le prochain état et la prochaine action dans les échantillons de transition. Après avoir atteint **S'**, il prendra l'action **A'** et utilisera **Q(S', A')** pour mettre à jour la valeur **Q**. La différence entre ces deux algorithmes est que SARSA choisit une action suivant la même politique actuelle et met à jour ses valeurs **Q** alors que Q-learning choisit l'action qui donne la valeur **Q** maximale pour l'état, c'est-à-dire, il suit une politique optimale.

1.5 Modèles de contrôle autonomique

Dans la littérature, les auteurs présentent des efforts intéressants pour améliorer les modèles formels existants du gestionnaire autonome afin de parvenir à l'auto-adaptation. Nous concentrons dans cette section sur les travaux les plus pertinents. Puis, nous dressons une synthèse de ces modèles.

1.5.1 Autonomic Control Loop

Bobson et al. [42] ont proposé d'utiliser le mécanisme Autonomic Control Loop (ACL) pour atteindre l'auto-adaptabilité, la sensibilité au contexte et la proactivité [18]. Cette boucle de contrôle comprend quatre activités principales : Collecter (Collect), Analyser (Analyse), Décider (Decide) et Agir (Act). La phase *Collect* est responsable de la collecte d'informations à partir de diverses sources, y compris les capteurs de réseau traditionnels, les flux de rapports, le contexte d'appareil et le contexte d'utilisateur de niveau supérieur. La phase *Analyse* permet d'analyser les informations collectées pour construire un modèle de la situation évolutive à laquelle sont confrontés le réseau et ses services. La phase *Decide* s'appuie sur ce modèle pour générer des décisions d'adaptation. La phase *Act* permet d'exécuter les décisions d'adaptation via le réseau et seront potentiellement rapportées aux utilisateurs ou aux administrateurs. L'impact des décisions peut ensuite être collecté pour informer le prochain cycle de contrôle. Par la suite, IBM [60] a proposé une boucle de contrôle MAPE-K qui exécute des tâches similaires à celles décrites dans l'ACL : surveiller (Collect), analyser (Analyse), planifier (Decide) et exécuter (Act) pris en charge par une base de connaissance. Récemment, MAPE-K est le paradigme de référence de base pour la conception des SAAs dans le contexte de l'informatique autonomique. Sur cette base, nous avons fourni une présentation détaillée de la boucle de contrôle MAPE-K dans la section 1.4.

1.5.2 FRAMESELF

Ben Alaya et Monteil [14] ont proposé l'approche FRAMESELF, qui est une architecture autonome générique basée sur des modèles de décision. L'architecture FRAMESELF est basée sur la référence d'architecture autonome d'IBM [60]. Dans cette approche, les phases de surveillance, d'analyse, de planification et d'exécution fonctionnent comme des

systèmes experts pour imiter la capacité de prise de décision des experts humains. Ces modules sont conçus pour résoudre des problèmes complexes en raisonnant sur des connaissances, comme un expert. Chaque module est divisé en deux parties : une constante indépendante du système telle que le moteur d'inférence et une variable telles que le modèle de la base de connaissance. Les auteurs ont exprimé chaque phase par une question dont la réponse indique le principe de fonctionnement de chaque phase. Le rôle de la phase de surveillance est de répondre à la question « Que se passe-t-il ? », tandis que la phase d'analyse se concentre sur la question « Que faire ? ». La phase de planification se concentre sur la question « Comment faire ? ». Enfin, la phase d'exécution doit répondre à la question « Comment cela se fait-il ? ». Ben Alaya and Monteil [14] ont utilisé le diagramme des composants UML pour décrire les quatre phases.

1.5.3 Modèle Formel de la Boucle MAPE-K

Iglesia et Weyns [58] ont présenté une modélisation étendu de la boucle de contrôle MAPE-K. En particulier, les auteurs présentent un ensemble des modèles formellement spécifiés pour modéliser les différents composants d'une boucle de contrôle MAPE-K, basés sur des réseaux d'automates temporisés, où ils ont montré la réutilisabilité de la boucle. Les modèles fournissent des connaissances de conception réutilisables, qui peuvent être considérées comme un langage spécifique à un domaine qui permet une modélisation et une vérification rigoureuses des comportements des boucles de contrôle pour un domaine cible d'applications distribuées dans lequel l'auto-adaptation est utilisée pour gérer les ressources pour la robustesse et les exigences d'ouverture via l'ajout et la suppression de ressources du système.

1.5.4 Architecture Fonctionnelle d'un Composant de Surveillance

Banouar et al. [12] ont proposé et implémenté un modèle d'architecture fonctionnelle pour un composant de surveillance (Monitor component) qui permettant l'observation de métriques et la détection en temps réel des symptômes liés à la dégradation de la qualité de service (Quality of Service, QoS). Les auteurs ont utilisés la description UML pour décrire le composant de surveillance fonctionnelle de haut niveau permettant le niveau de maturité visé. Les principaux composants fonctionnels de cette architecture sont les suivants :

- Le composant « Monitoring-Manager » est responsable de la configuration et le déploiement des différents composants et l'interaction entre eux. Ce composant assure aussi l'interface avec l'administrateur.

- Le composant « QoS-Configurator » permet de configurer la QoS requise pour les applications. Il interagit avec la base de connaissance pour la récupération de l'accord de niveau de service et la description des capteurs appropriés. Ce composant interagit aussi avec le composant «*Events-Collector*» pour la spécification des métriques à collecter en fonction de l'accord de niveau de service.

- Le composant « Events-Collector » est responsable de la collecte des métriques par interaction avec les capteurs correspondants. La description de cette interaction est précisée par le composant «*QoS-Configurator*» qui interroge la base de connaissance. L'interaction entre le composant «*Events-Collector*» et les capteurs intégrés dans l'entité gérée peut se faire via une requête/réponse périodique ou via une méthode d'écoute avec des notifications des capteurs lors d'un changement.

- Le composant « Events-Processor » traite les événements qui arrivent en appliquant des modèles définis stockés dans la base de connaissance, il peut s'agir d'événements de filtrage (*Events-Filter*), de corrélation (*Events-Correlator*) ou d'agrégation (*Events-Aggregator*). Dès qu'un pattern est détecté, il le notifie au composant «*Monitoring-Manager*».

- Le composant « Symptoms-Generator » est invoqué par le composant « *Monitoring-Manager* » lorsqu'un ou plusieurs événements correspondent à un modèle défini. Il génère ensuite un symptôme adéquat corrélé à la non-satisfaction des exigences de QoS via le catalogue de symptômes inclus dans la base de connaissance. Il envoie ensuite la description du symptôme associé au composant d'analyse.

1.5.5 Synthèse

Dans cette section, nous avons parcouru les travaux les plus pertinents sur la modélisation des gestionnaires autonomiques.

kephart et al. [60] ont proposé la boucle de contrôle MAPE-K. Cette boucle est devenue un modèle de référence car la majorité des travaux de recherche dans le domaine du SAA est basée sur cette boucle de contrôle. Même les conceptions améliorées ont conservé les mêmes composants de base (surveillance, analyse, planification, exécution et base de connaissance) et ont fourni l'infrastructure pour ces composants. Parmi ces travaux, le modèle formel de la boucle MAPE-K proposé par Iglesia et Weyns [58] qui nous a donné une vision plus clair sur la structure interne et le comportement de chaque composant séparément. D'autre part, le modèle FRAMESELF exprime également chaque composant séparément, mais la structure interne du composant « Knowledge » n'est pas modéliser dans ce modèle. D'autres travaux portent sur la modélisation de l'une des composants de gestionnaire autonomique. L'architecture fonctionnelle d'un composant de surveillance proposée par Banouar et al. [12] nous a fourni uniquement la structure du composant de surveillance, mais avec suffisamment de détails sur sa structure interne.

Peu de travaux ont utilisé des modèles de conception bien établis pour modéliser tout ou une partie du gestionnaire autonomique. Le modèle FRAMESELF utilise un diagramme de composant UML pour décrire les différents composants de gestionnaire autonomique. Les auteurs dans [58] ont proposé un modèle formel de la boucle MAPE-K basé sur des réseaux d'automates temporisés. En revanche, d'autres travaux ont été présentés des architectures informelle et ne sont pas basés pas sur des modèles ou des paradigmes de conception bien établis [18, 60].

1.6 Conclusion

Dans ce chapitre, nous avons commencé par une brève introduction au SAA et la sensibilité au soi et au contexte. Puis, nous avons présenté l'architecture conceptuelle des SAAs. Ensuite, nous avons présenté une taxonomie proposée par Krupitzer et al. [62] pour capturer les exigences d'adaptation. Aprés, nous avons mis en évidence le modèle de référence MAPE-K. Cette boucle sera utilisée par la suite comme référence afin de proposer une structure détaillée du MAPE-K, c'est-à-dire de définir en détails l'architecture fondamentale de la boucle de contrôle MAPE-K. Enfin, nous avons présenté des modèles de contrôle autonomique et une synthèse de ces modèles.

Dans le chapitre suivant, nous allons introduire l'auto-adaptation des systèmes à base de composants afin de diminuer le coût et le temps de réalisation et d'augmenter la qualité des systèmes informatiques.

Chapitre 2

Auto-adaptation des systèmes à base de composantss

Like a jigsaw puzzle : you have to make the pieces fit without getting out the scissors.

– Dr. Karl Maurer

2.1 Introduction

Les approches à base de composantss et d'agent sont complémentaire [1]. Ils ont des points communs où un agent réactif est équivalent à un composant [1]. Cependant, ils possèdent leurs principales caractéristiques qui ne sont pas partagées entre eux. Le manque de réutilisation et la composition des composants sont des limites de l'approche à base d'agent [73, 99]. L'approche à base de composantss tente de réduire les coûts et les délais de développement. Il permet aussi de faciliter l'évolution et la maintenance des applications.

Dans ce chapitre, nous présentons quelques définitions générales liées aux approches à base de composantss, nous intéressons aux abstractions particulières que sont un composant et ses services, le cycle de vie logiciel à base de composantss et la composition de composants. Ensuite, nous introduisons le modèle de composant en général, puis nous nous limitons à deux modèles de composant qui sont le modèle Fractal et le modèle directMOD. Enfin, nous étudions quelques SAAs à base de composants qui sont : SAFRAN, MaDcAr et K-Component, puis nous dressons une comparaison entre eux.

2.2 Les composants logiciels

Le concept composant « Component » a été proposé par McIlroy [75], où il a implémenté une infrastructure de l'idée de composant sous UNIX. Au début des années 90, le développement basé sur les composants est apparu. Ce paradigme de développement est devenu très populaire en raison des caractéristiques prometteuses telles que la réduction de la complexité et la réduction du temps de développement. Dans ces dernières années, de nombreux modèles de composant ont été apparu, où la plupart de ces modèles concentrent sur certains points spécifique de ce paradigme.

2.2.1 Définition d'un composant logiciel

Depuis l'émergence du paradigme à base de composants, une multitude de définitions du concept « *composant* » ont été proposées. Nous pouvons en citer quelques-uns :

Sametinger [98] mis l'accent sur la réutilisation dans sa définition :

> *«Reusable software components are self-contained, clearly identifiable artefacts that describe and/or perform specific functions and have clear interfaces, appropriate documentation and a defined reuse status [98]. »*

Nous citons également cette définition de Clemens et al. [25], très couramment acceptée :

> *«A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties [25]. »*

Heineman et Councill [28] donnent une autre définition pour le composant logiciel :

> *«A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard [28]. »*

D'après ces définitions [28, 25, 98], le composant est une entité logiciel conforme à un modèle de composant. Il expose des interfaces qui spécifient les services. Un composant peut être composé avec un autre composant ou un autre élément logiciel, et il peut être déployé indépendamment sans modification selon une norme de composition.

Toutes ces définitions font apparaître une grande diversité des modèles proposés pour décrire les composants.

2.2.2 Service de composant

Au sein d'une application, le rôle du composant est de fournir des services à d'autres composants. Le type d'échange de services entre composants s'exprime au travers de couples *client-serveur*, où le composant considère les autres composants d'une part comme des serveurs fournissant des services et d'autre part comme des clients ayant besoin de ses services.

Bosch et al. [17] sont définis le service comme suit :

> *«A service is a running instance (all the way down to supporting hardware and infrastructure) that has specific quality attributes, such as availability, while a component needs to be deployed, installed, loaded, instantiated, composed, and initialized before it can be put to actual use [17]. »*

2.2.3 Cycle de vie logiciel à base de composants

Le cycle de vie logiciel à base de composants est le processus qui mettant l'accent sur les règles métier, la modélisation, la conception, la construction, les tests continus, le déploiement, l'évolution et la réutilisation et la maintenance [28].

Selon Oussalah [89], le cycle de vie logiciel à base de composants comprend généralement les activités suivantes :

- *Analyse des besoins en termes de composants.* Ce processus consiste à étudier et comprendre les caractéristiques et les contraintes essentielles du système. Il crée ensuite un modèle abstrait de l'application où ses exigences sont satisfaites. Le modèle abstrait de l'application comprend des abstractions de haut niveau de composants logiciels. De plus, les services fournis par un système logiciel aident à identifier les sous-systèmes et leurs principaux composants. Notez que, cette phase ne concerne pas les détails fonctionnels des composants. Le produit final de cette étape est une description graphique ou textuelle d'un modèle abstrait de l'application.
- *Conception des types de composants.* La conception s'appuie sur la définition des besoins de l'étape précédente pour déterminer les types de composant. Au cours de

cette étape, il est nécessaire de spécifier les interfaces, les propriétés et les contrats des composants. Cette étape utilise également certains mécanismes qui sont :

- L'adaptation consiste à ajuster des composants pour qu'ils deviennent appropriés.
- L'intégration d'un composant à l'application en cours de développement.
- La recherche consiste à trouver des composants qui répondent à des besoins spécifiques.
- La sélection de composant le plus proche des besoins parmi les composants candidats.

* *Implantation des types de composants.* Sur la base d'un langage dédié par un modèle de composant spécifique, et prendre en compte les spécifications du composant, les parties fonctionnelles du type de composant sont implantées.
* *Paquetage et diffusion des implantations de composants.* Au cours de cette étape, les composants sont archivés dans des packages, où chaque package contient des éléments spécifiques au déploiement, qui sont : le type du composant, son implémentation et un ensemble de descripteurs utilisés dans le déploiement.
* *Assemblage.* Il consiste à assembler des composants, où le résultat de cette opération doit être un composant afin qu'il puisse être assemblé avec d'autres composants. Pour décrire cet assemblage, des langages de description de l'architecture (ADL) peuvent être utilisés. Les ADL sont basés sur les concepts de composant, connecteur et configuration qui permettant de décrire l'architecture d'une application. Les connecteurs sont utilisés pour créer les connexions entre les composants.
* *Déploiement.* Ce processus vise à préparer le système, c'est-à-dire à intégrer un composant dans son environnement après avoir extrait les éléments du package. L'instanciation du composant est effectuée sur la base d'un ensemble de descripteurs. Ce dernier permet de vérifier l'adéquation de la structure d'accueil avec leurs spécifications.
* *Test et maintenance.* Le processus de test aide à identifier les erreurs et aussi à déterminer la qualité du logiciel. La maintenance permet d'ajuster le logiciel par la modification d'un composant existant ou la création d'un nouveau.

2.2.4 Composition des composants

La composition consiste à combiner deux composants logiciels ou plus donnant un nouveau comportement de composant à un niveau d'abstraction différent [28]. Les propriétés de comportement d'un composite sont déterminées par les composants combinés (sous-composants) et par la manière dont ils sont combinés [28]. La composition permit de définir comment les composants peuvent être composés pour créer une structure plus grande et comment un producteur peut remplacer un composant par un autre qui existe déjà dans la structure [28]. De plus, les ingénieurs peuvent créer des infrastructures de composants logiciels spécifiques à un domaine dont les composants interagissent pour obtenir le comportement souhaité du système [28].

Belloir [13] distingue deux types de relation de composition :

- *La composition horizontal.* Elle représente l'échange de services entre deux composants dans le même espace d'adressage. La composition horizontale permet de résoudre les dépendances d'un composant en faisant correspondre ses interfaces requises (Required interfaces) avec les interfaces fournies (Provided interfaces) d'un d'autre composant.
- *La composition verticale ou hiérarchique.* Elle concerne l'application d'un assemblage entre un composant dans un espace d'adressage faible (sous-composant) et un autre composant dans un espace d'adressage plus forte (composite). Le composite peut offrir un point d'accès unique aux services fournis par ses sous-composants. De plus, il peut résoudre ses dépendances en interne. Par exemple : Fractal est un modèle de composant hiérarchique [21].

La composition soit horizontale ou verticale consiste de façon traditionnelle à une composition structurelle. Une des conséquences négatives de la composition verticale est le risque d'augmentation du couplage entre les composants. Cependant, ce type offre un avantage c'est que le remplacement d'un sous-composant par un autre n'aura d'effet direct que sur le composite dont il fait partie. Contrairement à la composition horizontale, le remplacement d'un composant par un autre aura un effet direct sur les autres composants du même espace d'adressage.

2.3 Modèle de composant

Un modèle de composant définit des critères spécifiques pour l'interaction et la composition des composants [28]. Au sens le plus large, le modèle de composant spécifie comment créer un composant, il standardise la structure, les interactions et les principes de composition auxquels chaque composant doit respecter pour être conforme au modèle [28]. De plus, le modèle de composant peut définir des mécanismes de personnalisation décrivant comment les composants peuvent être étendus sans modification [28]. D'après Councill et Heineman [28], un modèle de composant fonctionne sur deux niveaux. Premièrement, le modèle de composant définit comment construire et représenter un composant. Deuxièmement, le modèle de composant peut appliquer un comportement global sur la manière dont un ensemble de composants communique et interagit les uns avec les autres [28].

Selon Lau et Wang [67], le modèle de composants se caractérise par :

- La *syntaxe* des composants, ils sont définis de façon graphique ou textuelle, servant à décrire la structure du composant indépendamment de la syntaxe d'implémentation.
- La *sémantique* des composants, c'est-à-dire la définition des interfaces, services fournis/requis, opérations, ... etc.
- La *composition* des composants, détaillant la manière d'assembler les composants.

Un ensemble de modèles permet de décrire un système sous différents angles [11], c'est-à-dire différents modèles de composant peuvent être intégrés dans le même système.

Dans la littérature du paradigme à base de composants, il existe de nombreux modèles de composant dans les domaines industriels telles que : EJB [79] et CCM (CORBA Component Model) [112], et dans les domaines académiques telles que : Fractal [21] et SOFA (SOFtware

Appliance) [92]. Cependant, dans le cadre de ce manuscrit, on se limite à deux modèles qui sont le modèle Fractal et le modèle directMOD. Ce dernier est l'un des modèles de composant les plus récents par rapport aux autres modèles.

2.3.1 Fractal

Fractal [20, 29] est un modèle de composant extensible, flexible et hiérarchique développé par le consortium OW2 [1]. Outre sa prise en charge de la hiérarchisation des composants, Fractal prend en charge un ensemble de fonctionnalités intéressantes telles que : Partage de composants et d'aspects via des contrôleurs de composants.

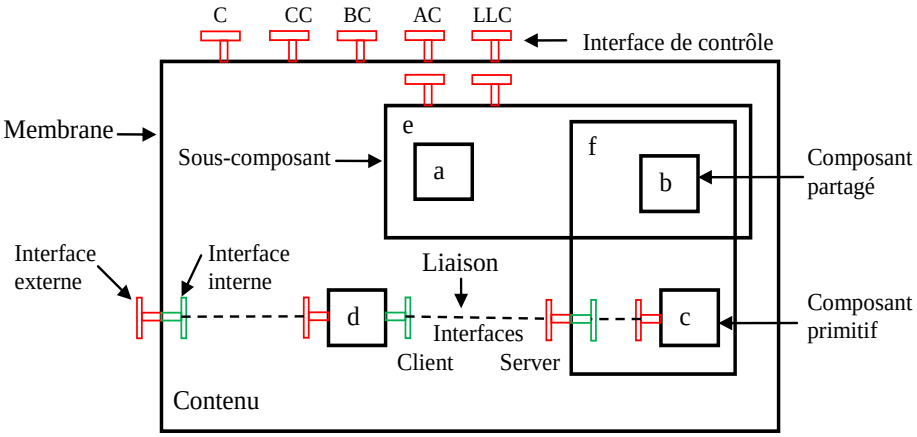


FIGURE 2.1: Anatomie d'un composant Fractal.

La figure 2.1 présente un exemple simple d'application Fractal. Un composant Fractal [20] a une *membrane* et un *contenu*. La membrane expose les interfaces (représentant par des formes en « T ») répertoriant les opérations. On distingue deux types d'interface qui sont l'interface *serveur* qui fournit les opérations et l'interface *client* qui nécessite des opérations. La membrane expose également des interfaces de contrôle standard qui fournissent des informations sur la configuration et la composition interne du composant. Il existe cinq interfaces de contrôle principales comme suit [19, 34] : l'interface *Component* (C) permet la découverte de l'ensemble des interfaces d'un composant, l'interface *ContentController* (CC) permet l'ajout et la suppression de sous-composants de composite, l'interface *BindingController* (BC) permet la création et la dissociation des liaisons, l'interface *AttributesController* (AC) permet d'obtenir et de modifier les attributs du composant, et l'interface *LifeCycleController* (LCC) permet de démarrer et d'arrêter correctement un composant.

Un composant Fractal peut être *primitif* ou *composite* [19]. Un composant primitif cache son contenu, il peut être défini par la spécification des interfaces fournies, des interfaces requises et de la classe qui implémente ce composant, par exemple les composants **a**, **b**, **c** et

1. Un consortium international indépendant à but non lucratif dédié au développement d'une infrastructure de code logiciel open source pour les systèmes d'information middleware.

d sont des composants primitive. Un composant composite est constitué d'un nombre arbitraire de composants (appelés sous-composants), il ne cache pas son contenu, par exemple les composants **e** et **f** sont des composants composites. Il est défini par la spécification des interfaces fournies, des interfaces requises, des sous-composants et des liaisons de ces sous-composants avec le composant composite lui-même [19]. Enfin, un composant peut être partagé, ce qui peut être utilisé simultanément par plusieurs composants, par exemple le composant **b** est un composant partagé entre les composants composites **f** et **e**.

ADL (Architecture Description Language) Fractal [20] est un langage de description d'architecture qui permet, comme son nom l'indique, de décrire l'architecture d'un composant Fractal quelle que soit son implémentation.

2.3.2 directMOD

directMOD [65] est un modèle de composant dont l'objectif est de permettre une reconfiguration évolutive de l'assemblage au moment de l'exécution et d'avoir de bonnes propriétés d'ingénierie logicielle, c'est-à-dire la réutilisation et la séparation des préoccupations, même lorsque la transformation de l'assemblage est impliquée. Pour atteindre ces objectifs, directMOD repose sur deux idées principales.

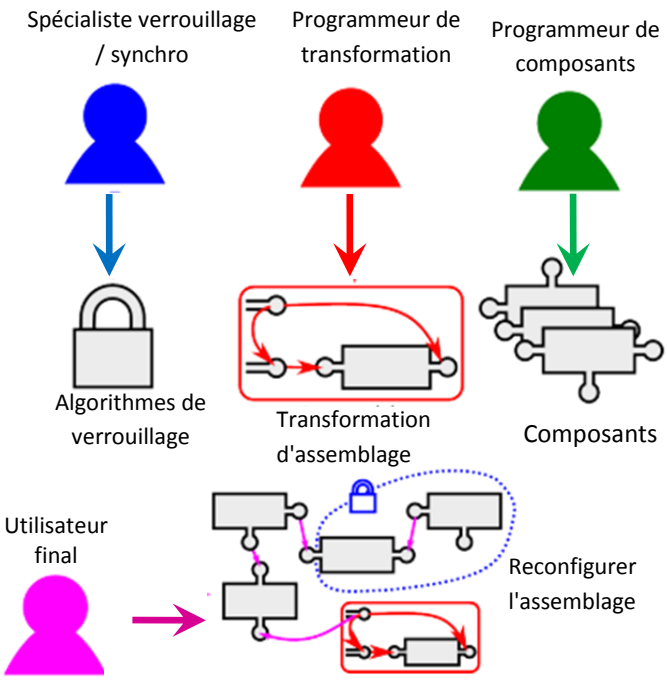


FIGURE 2.2: Un aperçu de la séparation des préoccupations, appliquée par directMOD [65].

Tout d'abord, il adopte l'approche de l'état de repos, c'est-à-dire que pour reconfigurer une partie de l'assemblage, cette partie doit être arrêtée/verrouillée. Cependant, pour

atteindre l'évolutivité, l'unité de verrouillage (c'est-à-dire les plus petites parties d'un assemblage qui peuvent être verrouillées) est définie arbitrairement au niveau de l'assemblage et n'est pas imposée par le modèle. Cela signifie que des ports individuels peuvent être verrouillés ainsi que des blocs complets d'assemblage en fonction de l'application. De cette façon, le verrouillage peut être adapté à chaque application et peut être arbitrairement distribué (pour une évolutivité maximale) ou centralisé (par exemple, pour une cohérence facile). Nous appelons ces unités verrouillables des domaines. Deuxièmement, étant donné que la reconfiguration se produit sur des parties verrouillées de l'assemblage, le fait que ces parties soient arbitraires pourrait entraîner des problèmes d'utilisation d'un code de reconfiguration écrit indépendamment avec un schéma de verrouillage donné. Pour contourner ce problème, les transformations (c'est-à-dire le code de la reconfiguration) sont définies indépendamment et connectées aux domaines au niveau de l'assemblage à l'aide d'un dispositif appelé adaptateurs de transformation. Les adaptateurs de transformations permettent de connecter une transformation à un sous-ensemble spécifique d'un domaine.

Le modèle de composant directMOD définit quatre rôles possibles pour l'utilisateur et le programmeur qui sont présentés dans la figure 2.2. Les programmeurs de composants écrivent des composants traditionnels, les programmeurs de transformation écrivent des transformations à l'aide du modèle et les programmeurs de domaine implémentent les domaines et ils sont responsables de tout le code de verrouillage. Enfin, les utilisateurs finaux peuvent tout assembler en utilisant des connexions de composants traditionnels, des domaines pour la logique de verrouillage et des adaptateurs de transformation pour brancher les transformations.

2.4 SAAs à base de composantss

Dans cette section, nous présentons les principaux travaux proposant des SAAs basés sur les composants logiciels. Ensuite, nous comparons ces travaux à l'aide d'un ensemble de critères d'évaluation afin de déterminer le modèle qui correspond à notre contribution.

2.4.1 SAFRAN

Le modèle de composant SAFRAN (Self-Adaptive FRActal compoNents) [33] est une extension du modèle de composants Fractal (présenté dans la section 2.3.1.) visant à supporter le développement des composants auto-adaptatifs (voir la figure 2.3). Il prend en charge l'adaptation structurelle logique et l'adaptation comportementale grâce au paramétrage et à la redirection des appels de service.

Les politiques de SAFRAN sont basées sur des règles d'Event-Condition-Action (ECA). (1) Les « Events » utilisent des informations envoyées par des capteurs indépendants du système (ces capteurs sont fournis par la bibliothèque WildCat). WildCAT [35] est un framework Java extensible pour faciliter la création d'applications sensibles au contexte. WildCAT fournit un modèle dynamique simple mais puissant pour représenter le contexte d'exécution d'une application. Son but est de faciliter la création d'applications sensibles au contexte, (2) la « Condition » est spécifiée avec le sous-langage FPath dédié [36], qui per-

met de naviguer dans l'architecture du système et de sélectionner les éléments sur lesquels appliquer les reconfigurations, (3) les « Actions » sont spécifiées avec le langage FScript qui permet de manipuler les composants (modification des valeurs d'attributs, ajout de méta-composants, ...etc.) et l'architecture (connexion / déconnexion de composants, ajout / suppression de sous-composants, ... etc). Le langage FScript [37] permet de définir des scripts de reconfiguration complexes qui peuvent modifier la structure et la configuration d'une application Fractal. Le programme FScript consiste en un ensemble de définitions de fonctions et d'actions.

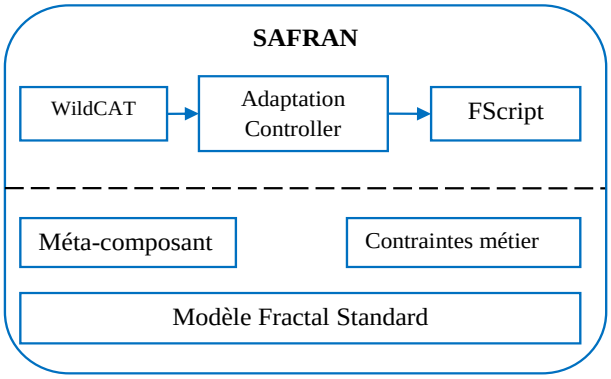


FIGURE 2.3: Les différents éléments constituants le système SAFRAN [33].

2.4.2 MaDcAr

MaDcAr (Model for automatic and Dynamic component Assembly reconfiguration) est un modèle abstrait pour la reconfiguration dynamique de l'assemblage des composants [51]. MaDcAr fournit une solution uniforme pour automatiser à la fois la construction d'applications à partir de rien et l'adaptation des assemblages de composants existants. La figure 2.4 montre un moteur d'assemblage MaDcAr qui inclut un solveur de contraintes pour calculer automatiquement les décisions d'assemblage appropriées. Les composants disponibles fournissent un ensemble de composants à assembler. La description d'application est la spécification des propriétés fonctionnelles et extra-fonctionnelles des assemblages valides. La politique d'assemblage pilote les décisions d'adaptation. Le gestionnaire de contexte fait référence à la fois aux informations sur les ressources matérielles (consommation CPU, bande passante réseau, ... etc.) et à l'état de l'application (c'est-à-dire les valeurs d'attribut des composants assemblés).

Les composants, la description de l'application et le moteur peuvent être réutilisés pour construire différentes applications. Un moteur d'assemblage MaDcAr doit être construit comme un cadre spécialisé pour traiter un modèle de composant donné. En principe, cette spécialisation nécessite l'implémentation d'un petit ensemble d'opérations pour gérer les composants : la connexion / déconnexion de deux composants, l'activation / désactivation d'un composant et l'importation / exportation d'un état de composant. Une description d'application est composée d'un ensemble de configurations alternatives et d'une spécifi-

cation de transfert d'état. La description de l'application et la politique d'assemblage sont spécifiées en termes de contraintes.

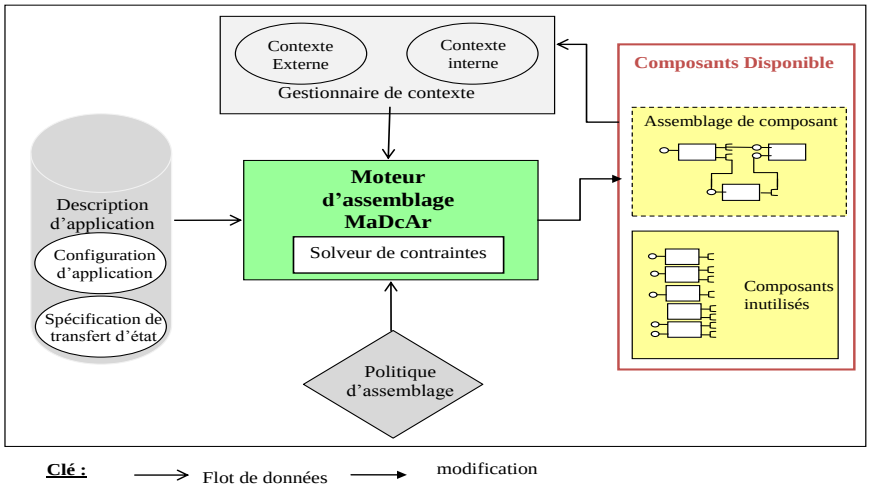


FIGURE 2.4: Entrées-Sorties de MaDcAr [51].

2.4.3 K-Component

K-Component [43] fournit une architecture de construction d'un SAA capable de surveiller l'état de son fonctionnement et de son environnement, et d'effectuer une adaptation conditionnelle de sa structure ou de son comportement.

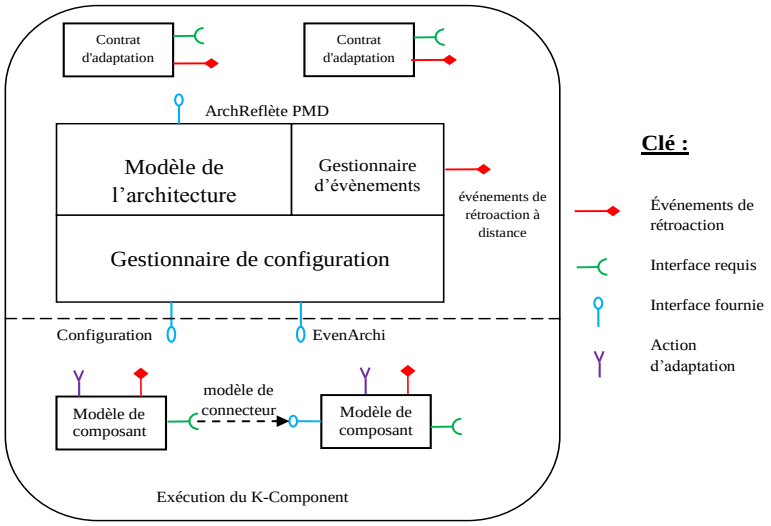


FIGURE 2.5: Structure d'un K-Component [43].

La figure 2.5 présente la structure d'un K-Component qui correspond à un nœud d'un

système distribué. Chaque nœud dispose de sa propre boucle de contrôle comprenant : (i) Un modèle de l'architecture qui est un graphe des liaisons entre les composants au sein du K-Component ainsi que les liaisons des composants vers des composants appartenant à d'autres K-Components, (ii) Un gestionnaire de configuration qui gère un ensemble de « contrats d'adaptation » qui s'exécutent en parallèle des composants, et (iii) Un gestionnaire d'évènements qui permet aux K-Components de communiquer par envoi d'évènements, indépendamment des communications entre les composants qu'ils hébergent.

2.4.4 Comparaison

Dans cette section, nous présentons d'abord un ensemble de critères que nous proposons d'utiliser dans la comparaison entre SAFRAN, MaDcAr et K-Component qui sont présentés ci-dessus. La réalisation de la majorité des critères proposés facilite la sélection du modèle approprie pour cette étude.

- **Ouverture.** Le système doit être ouvert afin de pallier les défauts de la conception initiale. Où, de nouveaux comportements peuvent être introduits ainsi que de nouvelles politiques et entités réalisant les boucles de contrôle du système. Ces éléments peuvent être ajoutés par l'administrateur système ou par un autre système logiciel. Malheureusement, l'ouverture n'est pas supportée par tous les types de systèmes, notamment les systèmes embarqués dont l'occupation mémoire sont très limitée et les systèmes enfouis pour lesquels il est difficile d'introduire de nouvelles entités.
- **Hiérarchie.** Différents composants peuvent être encapsulés par un composant composite. La conception d'un système à base d'un modèle hiérarchique consiste à créer un composant de granularité plus importante à partir de sous-composants. Le modèle de composition hiérarchique s'appuie sur la théorie mathématique de la relation « Tout-Partie », où un composant de forte granularité est construit à partir d'un ou plusieurs composants de granularité plus faible [13].
- **Instanciation de composant.** Plusieurs instances de composant peuvent apparaître dans le même système. Ceci est important lorsqu'un composant définit une ou plusieurs variables d'état qui sont mises à jour lorsque les points de jonction sont atteints.
- **Partageabilité.** Le composant peut être partagé par plusieurs composites, c'est-à-dire qu'il peut être utilisé simultanément par plusieurs composants. Le partage peut modéliser le partage des ressources et des préoccupations transversales. La partageabilité a un impact sur la disponibilité des services rendus par un sous-composant. Le composant partagé peut nécessiter d'être configuré d'une certaine manière lorsqu'il interagit avec un composite et d'être configuré d'une autre manière lorsqu'il interagit avec un autre composite [13].
- **Séparation des préoccupations.** Ce processus consiste à diviser une application en parties, dans ce cas des « composants ». Chaque partie gère un petit nombre de « préoccupations » spécifiques [64]. La séparation des préoccupations facilite l'écriture de parties d'applications, ce qui signifie moins de soucis implique moins de complexité. De plus, ce processus facilite la réutilisation, ce qui signifie moins de dépendances.

Modèle	Système	Hiérarchie	Instantiation de composant	Partagibilité	Séparation des préoccupations
MaDcAr	Ouvert		✓		✓
K-Component	Fermé		✓		?
SAFRAN (extension de Fractal)	Ouvert	✓	✓	✓	✓

TABLE 2.1: Critères d'évaluation des SAAs à base de composantss.

La table 2.1 fournit une comparaison des différentes critères du SAA à base de composants, où le symbole «✓» signifie que le critère est supporté et «?» signifie que le critère est indéterminé.

SAFRAN prend en compte les critères sélectionnés dans cette comparaison. Il propose une infrastructure d'adaptation très ouverte et basée sur des concepts de haut niveau, telle que les politiques d'adaptation. Le degré d'ouverture de l'infrastructure d'adaptation de SAFRAN est important, car elle est facilement configurable à travers une politique d'adaptation, un langage de reconfiguration ainsi que un gestionnaire de contexte qui est extensible. Le degré d'ouverture de MaDcAr est aussi important, car le processus d'adaptation de MaDcAr est configurable à travers une politique d'assemblage qui spécifie le contexte, le moment et les conditions de déclenchement des adaptations, ainsi que les composants concernés par l'opération de réassemblage et l'ordonnancement des étapes de réassemblage. Cependant, K-Component est borné par les comportements et les adaptations définis lors de la phase de conception.

Parmi les SAAs à base de composantss discutés ci-dessus, seul SAFRAN (extension de Fractal) prend en charge la hiérarchisation et a la particularité d'être introspectable et permettre le partage de composant. En revanche, MaDcAr, K-Component et SAFRAN prennent en charge l'instanciation de composant.

SAFRAN offre une bonne séparation des préoccupations en isolant le code générique et réutilisable des adaptations spécifiques à l'application, en l'occurrence ici des configurations. Aussi, MaDcAr encourage la séparation des préoccupations en isolant les spécifications d'applications et les implémentations. Cependant, K-Component ne donne aucune information sur la séparation des préoccupations.

2.5 Conclusion

Dans ce chapitre, nous avons présenté un état de l'art sur les notions de base relatives à l'ingénierie des composants et quelques SAAs à base de composantss connus tels que SAFRAN, MaDcAr et K-Component. Nous avons ensuite comparé entre eux et mis en évidence les critères d'évaluation que chacun apporte afin de choisir le SAA approprié pour ce travail qui est le SAFRAN. Les critères les plus importantes de SAFRAN (extension du Fractal) sont : le partage de composant, la composition hiérarchique, l'ouverture du système et la séparation des préoccupations.

Fractal sera utilisé par la suite comme un modèle de conception établi pour concevoir une architecture détaillée de boucle de contrôle MAPE-K basé sur les composants. De plus, ce modèle sera aussi utilisé pour assurer la séparation des préoccupations entre les différents boucle de contrôle sous la forme d'une hiérarchie de boucles MAPE.

Chapitre 3

SAAs basés sur plusieurs boucles de contrôle

L'intelligence, c'est la faculté d'adaptation

– André Gide

3.1 Introduction

Lorsque les systèmes sont grands et distribués, une seule boucle de contrôle peut ne pas être suffisante pour réaliser la logique d'adaptation [117]. Par conséquent, l'introduction de plusieurs boucles de contrôle dans les SAAs est la solution intuitive [117]. Cependant concevoir et déployer des SAAs avec plusieurs boucles de contrôle présente un défi majeur, car il est nécessaire de concevoir une architecture de contrôle capable d'effectuer les adaptations correcte en temps voulu et d'assurer les objectifs du système. De nombreux efforts [117][109][125][125] sont effectués dans la littérature pour proposer des modèles de conception et de développement des SAAs qui soient simples à mettre en place, assurent la cohérence du système et répondent aux besoins des SAAs.

Dans ce chapitre, nous commençons par introduire les défis de la recherche affectant les SAAs avec plusieurs boucles de contrôle. Puis, nous présentons les différents types d'interaction entre les composants MAPE-K. Ensuite, nous présentons les trois approches de contrôle des SAAs, trouvées dans la littérature. Enfin, nous étudions quelques modèles de contrôle, puis nous dressons une synthèse de ces modèles.

3.2 Les défis de la recherche affectant les SAAs à boucles de contrôle

Malgré les efforts considérables qui ont été déployés dans différents domaines de recherche sur la réalisation des SAAs, certains défis concernant les capacités et la construction des SAAs restent ouverts. Dans cette section, nous nous concentrons sur les défis qui affectent le fonctionnement des SAAs basé sur plusieurs boucles de contrôle.

3.2.1 La coordination de plusieurs boucles de contrôle

Plusieurs boucles de contrôle dans un SAA décentralisé nécessitent un modèle de coordination efficace pour éviter les conflits de décision entre les boucles de contrôle. Le problème se pose de savoir comment coordonner plusieurs boucles de contrôle. Gueye et al. [52] ont proposés une approche de coordination de gestionnaires, basée sur l'utilisation des modèles réactifs et des techniques de contrôle discret pour la spécification de la coordination et le modèle à composants pour la mise en œuvre. Yusof et al. [123] ont proposés un modèle de coordination qui consiste en une combinaison de négociation et de synchronisation qui peut fournir une flexibilité pour le cycle d'adaptation. En particulier, la négociation permet aux gestionnaires autonomes de résoudre tout conflit en vue de trouver un accord mutuel. Pendant ce temps, la synchronisation garantit que l'exécution se produit au moment requis pour éviter tout conflit supplémentaire.

Cependant, lors de l'utilisation de la coordination entre toutes ou la plupart des entités de la boucle, le système peut souffrir du coût élevé du transfert de données entre différentes entités. De plus, la scalabilité des systèmes peut être difficile dans ce cas. D'autre

part, l'absence totale de coordination ou des sélections invalides d'entités de la boucle de contrôle nécessitant une coordination peuvent introduire des décisions incohérentes. Par conséquent, il est également nécessaire de savoir quelles entités doivent être coordonnées.

3.2.2 La séparation des préoccupations entre différentes boucles de contrôle

La séparation des préoccupations est un principe de conception qui permet de maîtriser la complexité d'un système. Ce principe s'est incarné successivement dans toutes les techniques de modélisation du point de vue de la recherche de mécanismes de séparation des préoccupations de plus en plus efficaces, tels que : l'approche modulaire, l'approche à base d'objets et de composants, et plus récemment l'approche à base d'aspects [90]. Les approches à base d'objet ou de composants permettent de décomposer un système complexe en différents éléments plus simples à appréhender, mais cette décomposition est unique, et correspond à l'une des préoccupations présentes dans le système [33]. L'idée générale est qu'il faut éviter de regrouper différentes préoccupations dans la conception ou le code. Lorsque les préoccupations sont bien séparées, il y a plus de possibilités de mise à niveau, de réutilisation et de développement indépendant des modules [66]. L'objectif global de la séparation des préoccupations est d'établir un système bien organisé où chaque partie remplit un rôle significatif et intuitif tout en maximisant sa capacité à s'adapter au changement.

La séparation des préoccupations est un défi majeur pour gérer la complexité de l'auto-adaptation. Alors, le problème est de savoir comment séparer les préoccupations entre les différentes boucles de contrôle afin de faciliter l'attachement des entités de la boucle de contrôle à la position appropriée dans la hiérarchie de l'application.

3.2.3 La boucle de contrôle MAPE-K statique vs. dynamique

Selon Zavala et al. [124], les SAAs existants reposent sur des boucles de contrôle statiques, où il est impossible de modifier la structure et le comportement de la boucle de contrôle au moment de l'exécution. D'autre part, la plupart des SAAs s'appuient sur un modèle unique de composition des boucles de contrôle. La nature de SAA peut nécessiter l'intégration de différents modèles de composition des boucles de contrôle. Chaque partie du système peut dépendre d'un modèle de composition différent afin de répondre aux objectifs des parties du système en particulier et à l'objectif global du système en général. Cependant, les modèles existants peuvent ne pas répondre aux exigences des parties du système. Ainsi, il est préférable de rendre les boucles de contrôle dynamiques et capables de modifier leur structure et donc leur comportement au moment de l'exécution en réponse aux changements du système et de l'environnement. Où, chaque partie du système sera basée sur un modèle de composition des boucles de contrôle différent et est susceptible de changer au moment de l'exécution. Enfin, cela pourrait conduire à l'émergence de nouveaux modèles de composition des boucles de contrôle.

3.2.4 Les conflits de décision

Les conflits de décision surviennent lorsqu'il y a un désaccord et une contradiction dans les objectifs de l'adaptation, car cela peut conduire à une adaptation incohérente. Selon Sylla et al. [109], les sources de conflit dans le gestionnaire autonome résident dans la phase de planification et d'exécution. Le fait d'avoir plusieurs entités de planification peut provoquer des conflits entre eux. Les planificateurs entrent en conflit lorsqu'ils essaient d'effectuer des interactions impliquant le même composant. En général, les conflits à la phase de planification reflètent les conflits à la phase de l'exécution. Pour éviter les conflits, Sylla et al. [109] ont proposé de coordonner les gestionnaires autonomes en combinant la négociation et la synchronisation lors des phases de planification et d'exécution. Cependant, la coordination souffre de certaines difficultés comme présenté dans la section ci-dessus qui peuvent conduire à des conflits.

3.3 Types d'interaction entre composants MAPE-K

L'interaction modélise la communication entre composants MAPE-K. Il est caractérisé par quatre types d'interaction, à savoir sous-système géré-gestion, inter-composant, intra-composant et Lire-écrire [32].

3.3.1 Sous-système géré-gestion (Managing-managed subsystem)

Les interactions de type *Managing-managed subsystem* modélisent l'interaction entre les composants de surveillance et le sous-système géré à des fins de surveillance, ainsi qu'entre les composants d'exécution et le sous-système géré pour effectuer des adaptations [117]. Cependant, il existe des modèles de composition de MAPE dans lesquels certain composants de surveillance et d'exécution n'interagissent pas avec le sous-système géré. Par exemple, dans le modèle de contrôle hiérarchique proposé par Weyns et al. [117], les composants de surveillance et d'exécution de la couche inférieure interagissent directement avec le sous-système géré, et les composants de surveillance et d'exécution des couches de niveau supérieur interagissent avec le groupe de composants MAPE des couches sous-jacentes.

3.3.2 Inter-composant (Inter-component)

Les interactions de type *Inter-component* suivent le flux logique d'une boucle de contrôle MAPE-K [116]. En d'autres termes, inter-composant modélise les interactions entre différents types de composants MAPE-K [32]. Par exemple : l'interaction entre les composants Surveillance-Analyse, Analyse-Planification et Planification-Exécution.

3.3.3 Intra-composant (Intra-component)

Intra-component modélise les interactions entre les composants MAPE-K du même type [32]. Par exemple : soit deux boucles MAPE-K_1 et MAPE-K_2, l'interaction entre Surveillance$_1$ - Surveillance$_2$, Analyse$_1$ - Analyse$_2$, Planification$_1$ - Planification$_2$ et Exécution$_1$ - Exécution$_2$. Ce type d'interaction est connu sous le nom de *coordination*. Donc, la *coordination* est utilisée lorsque des composants du même type interagissent les uns avec les autres issus de différentes boucles de contrôle. Par exemple, deux composants de planification "**P**" qui doivent négocier l'un avec l'autre pour offrir une flexibilité dans la prise de décision, et deux composants d'exécution "**E**" qui doivent synchroniser les actions d'une adaptation en cours pour éviter tout conflit d'exécution entre les boucles de contrôle MAPE-K.

3.3.4 Lire-écrire (Read-write)

Lire-écrire représentent l'interaction entre un composant MAPE et la base de connaissance (Knowledge base) [32]. Par exemple : l'interaction entre les composants (Surveillance et Base_de_connaissance), (Analyse et Base_de_Connaissance), (Planification et Base_de_Connaissance) et (Exécution et Base_de_Connaissance).

3.4 Approches de contrôle

En principe, les activités d'adaptation dans un SAA (surveillance, analyse, planification et exécution) peuvent être centralisées ou décentralisées, indépendamment du mode de déploiement des logiciels des sous-systèmes gérés et de gestion [116]. Elles ont été suggérées pour faciliter le comportement autonome dans les SAAs [56].

3.4.1 Contrôle centralisé

La centralisation implique un seul composant existe pour l'une des activités d'auto-adaptation qui remplit sa fonction [116]. Elle permet un contrôle global du comportement [121].

► Limites de la centralisation

Le contrôle centralisé montre ses limites dans certains cas :

- ✲ La centralisation souffre de problèmes d'évolutivité [121].
- ✲ Quand la taille du système augmente, le temps de réaction sera lent [82].
- ✲ Centralisé un composant peut imposer une surcharge de communication importante [116].

3.4.2 Contrôle décentralisé

La décentralisation au niveau des quatre activités implique un type de contrôle dans lequel plusieurs composants responsables d'une des activités d'auto-adaptation effectuent leurs fonctionnalités localement, mais en coordination avec des pairs [116]. Le contrôle décentralisé est crucial pour les exigences de qualité telles que la résilience, la robustesse et l'évolutivité [115]. Il distingue deux types :

- *Contrôle partiellement décentralisé.* On peut de dire aussi contrôle partiellement centralisé, car dans ce type les deux extrêmes « centralisation et décentralisation » sont présentées dans le même modèle. Par exemple : le modèle Maître / Esclave est partiellement décentralisé, car les composants de surveillance et d'exécution sont décentralisées, tandis que les composants d'analyse et de planification sont centralisées [116].
- *Contrôle totalement décentralisé.* Ce type de contrôle consiste à décentraliser les quatre activités MAPE, c'est-à-dire chaque activité d'auto-adaptation effectue leur fonctionnalité en coordination avec leur pair. Par exemple : le modèle de contrôle coordonné est totalement décentralisé, car les quatre composants de la boucle de contrôle MAPE sont décentralisées [116].

▶ Les avantages de la décentralisation

En résumé, la décentralisation présente beaucoup d'avantages tels que :

- La décentralisation peut également contribuer à améliorer la robustesse car il n'y a pas de point de défaillance unique [116].
- La décentralisation du contrôle peut être la seule option dans les cas où aucune entité n'a la connaissance ou l'autorité nécessaire pour coordonner les adaptations sur un ensemble de sous-systèmes gérés [116].
- La décentralisation permet d'améliorer la flexibilité des SAAs, et de réaliser la vision informatique autonome [60].

▶ Les inconvénients de la décentralisation

Il existe également un certain nombre d'inconvénients potentiels du contrôle décentralisé [116] :

- Le contrôle décentralisé peut poser des problèmes pour assurer la cohérence des adaptations à cause d'erreurs de communication.
- Le coût pour parvenir à un consensus sur les mesures d'adaptation appropriées peut être élevé, et cela en termes de communication et / ou de calendrier.
- Dans le cas où la coordination est nécessaire entre les composants MAPE de nombreux nœuds, l'évolutivité peut être compromise.
- Lorsque la communication est manquée entre plusieurs composants responsables d'une des activités d'auto-adaptation, il peut y avoir des conflits entre les objectifs

des différentes activités. Par exemple : le modèle de partage d'informations proposé par Weyns et al. [117], il n'y a pas de communication entre les activités de planification. Dans ce cas, les décisions peuvent être en conflit les unes avec les autres, ce qui entraîne une adaptation perpétuelle du système.

Entre la *centralisation* et la *décentralisation*, il existe une variété de manières d'organiser les quatre activités d'auto-adaptation.

3.4.3 Contrôle hiérarchique

Le contrôle hiérarchique est un bon compromis pour éviter la dégradation des performances d'un système complètement décentralisé, le problème de cohérence des adaptations à cause d'erreurs de communication, ainsi que le coût élevé pour parvenir à un consensus sur les mesures et la complexité d'une solution centralisée. Dans le contrôle hiérarchique, il y a toujours un degré de décentralisation mais conserve également un degré de centralisation grâce à une couche de contrôle de niveau supérieur qui gère les unités de niveau inférieur [31].

► Les avantages de la contrôle hiérarchique

En conséquence, le contrôle hiérarchique présente beaucoup d'avantages tels que :

- ✲ Le contrôle hiérarchique peut être appliqué via plusieurs hiérarchies, chacune ayant son propre effet.
- ✲ Le contrôle hiérarchique permet d'améliorer la fiabilité globale du système.

3.5 Modèles de contrôle

Dans la littérature, les auteurs ont proposé une sélection de modèles de contrôle. Nous distinguons dans cette section deux catégories : les modèles de contrôle décentralisés et les modèles de contrôle hiérarchiques. Puis, nous dressons une synthèse de ces modèles.

3.5.1 Modèles de contrôle décentralisés

Weyns et al. [117] ont proposé une sélection de modèle MAPE qui modélisent différents types d'interaction de boucle MAPE et avec différents degrés de décentralisation.

- ✲ **Le modèle de contrôle coordonné**, où chaque boucle de contrôle doit se coordonner avec ses pairs pour prendre une décision commune sur la manière de s'adapter. Cette approche consiste à décentraliser les quatre activités MAPE. Au sens le plus large, tous les composants **M**, **A**, **P** et **E** de chaque boucle coordonnent leur fonctionnement avec les pairs correspondants des autres boucles. Par exemple : les composants **A** échangent des informations pour prendre une décision sur la nécessité

d'une adaptation. Les composants MAPE de différents boucles MAPE peuvent interagir avec des pairs pour partager des informations particulières ou pour coordonner leurs actions.

- Contrairement au modèle de contrôle coordonné, **le modèle de partage d'information** consiste à coordonner les composants **M** de surveillance uniquement. Seuls les composants **M** communiquent entre eux. Tandis que les composants **A**, **P** et **E** de chaque boucle fonctionnent indépendamment de leurs pairs. Les informations collectées sur le statut des systèmes gérés sont partagées entre les boucles MAPE, ce qui permet une analyse, une planification et une exécution locales des adaptations sans coordination supplémentaire.
- **Le modèle Maître / Esclave** consiste à créer une relation hiérarchique entre un composant *Maître* (les composants **A** et **P**) responsable de l'analyze et de la planification des adaptations et de multiple composants *Esclave* (les composants **M** et **E**) responsable de la surveillance et de l'exécution. Le modèle est constitué de : (1) un nombre arbitraire d'instances du groupe avec un composant **M** et un composant **E**, (2) une seule instance du groupe avec un composant **A** et un composant **P**.
- **Le modèle de planification régionale**, où différents parties de systèmes faiblement couplées souhaitent réaliser des adaptations locales, c'est-à-dire des adaptations au niveau d'un région, ainsi que des adaptations qui traversent les limites des différentes parties, c'est-à-dire des adaptations entre régions. Les adaptations dans une région ont certains objectifs d'adaptation locaux, tandis que les adaptations entre régions peuvent viser un objectif d'utilité global. Chaque région a un planificateur régional, c'est le composant **P**. Un planificateur régional recueille les demandes de changement nécessaires auprès des sous-systèmes sous-jacents sous sa supervision pour planifier les adaptations. Les planificateurs régionaux interagissent les uns avec les autres pour coordonner les adaptations qui couvrent plusieurs régions. Dans chaque région, ce modèle est constitué de : (1) un nombre arbitraire d'instances du groupe avec les composants **M**, **A** et **E**, et (2) une seule instance du groupe contient uniquement un composant **P**. Pour chaque région, les composants locaux **M** surveillent l'état des sous-systèmes gérés et l'environnement, puis les composants locaux **A** analysent les informations collectées et signalent les résultats d'analyse au planificateur régional associé. Le planificateur régional peut alors décider d'effectuer une adaptation locale au niveau d'une région ou les planificateurs régionaux peuvent interagir les uns avec les autres pour planifier des adaptations couvrant les régions. Une fois que les planificateurs se sont mis d'accord sur un plan, ils peuvent appliquer les adaptations en activant les composants locaux **E** des groupes de composants respectifs impliqués dans l'adaptation.

3.5.2 Modèles de contrôle hiérarchiques

Quelques modèles de la composition hiérarchique des boucles de contrôle sont proposés dans la littérature.

- Weyns et al. [117] ont proposé **un modèle de contrôle hiérarchique** qui structure la logique d'adaptation comme une hiérarchie de boucles MAPE. Les boucles peuvent

fonctionner à différentes échelles de temps et gérer différents ressources avec différentes localités. Cependant, les boucles de contrôle doivent interagir et coordonner les actions pour éviter les conflits. Ce modèle est représenté une hiérarchie de trois niveaux : un niveau inférieure, plusieurs niveaux intermédiaires et un niveau supérieur. Les composants **M** (composant de surveillance) et **E** (composant d'exécution) des groupes abstraits de niveau inférieure interagissent directement avec le sous-système géré. Tandis que, les composants **M** (composant de surveillance) et **E** (composant d'exécution) des groupes abstraits de niveau supérieur interagissent avec les groupes situés dans les niveaux inférieures.

- Sylla et al. [109] ont proposé **un modèle hiérarchique de boucle de contrôle**. Les boucles de contrôle sont conçues avec un ou plusieurs niveaux de hiérarchie. La conception de boucles de contrôle hiérarchiques permet d'envisager de grands environnements. Cela permet également la réutilisation. Le contrôleur d'une boucle peut être réutilisé, sans modification, pour construire un autre contrôleur avec un niveau supérieur de hiérarchie. Ce modèle repose sur des boucles autonomes génériques qui combinent le contrôle basé sur des automates et l'exécution transactionnelle.
- **HIIC** (Hierarchical inter-intra collaborative) proposé par Edith et al. [125], qui combine et étend les schémas hiérarchiques et collaboratifs décrits par Weyns et al. [117]. HIIC permet de prendre en charge des boucles de contrôle avec un degré variable de décentralisation ainsi que des boucles capables d'interagir avec d'autres boucles. L'interaction est modélisée la communication entre les entités MAPE-K peut communiquer de même nature (par exemple, l'entité de surveillance avec d'autres entités de surveillance), et avec une ou plusieurs entités MAPE-K d'un autre type (par exemple, l'entité de surveillance avec deux entités d'analyse).

3.5.3 Synthèse

Après une étude et une analyse de la littérature existante, il a été constaté qu'il existe un ensemble de défis que nous allons présenter sous forme de points comme suit :

- L'absence totale de coordination ou la sélection invalide d'entités nécessitant une coordination peut introduire des problèmes, par exemple un conflit dans la décision.
- Le risque d'erreurs de communication pouvant conduire à prendre des décisions contradictoires avec l'état réel du système.
- Le surcoût de communication important lors de la centralisation de certaines entités.
- La gestion de la coordination entre les entités MAPE décentralisées peut être un défi majeur pour les ingénieurs, en particulier le coût élevé de la coordination. De plus, dans les cas où une coordination est nécessaire entre les entités MAPE de nombreuses boucles, l'évolutivité du système peut être compromise.
- Dans le cas du contrôle hiérarchique, les entités des boucles de contrôle aux niveaux inférieurs de la hiérarchie n'ont pas de communication directe entre elles.
- La plupart des SAAs actuels reposent sur MAPE-K statique, où la structure et le comportement des entités MAPE-K doivent être définis au stade de la conception, c'est-à-dire qu'il est impossible d'apporter des modifications à la boucle lors de l'exécution.

* La complexité des systèmes peut rendre difficile la séparation des préoccupations entre les différentes boucles de contrôle. La séparation des préoccupations aide à éviter les conflits inutiles et aide à minimiser les compétences et les connaissances requises pour rédiger des parties individuelles de la candidature.

Enfin, nous nous appuierons sur ces points pour apporter une contribution prenant en compte ces défis.

3.6 Conclusion

Ce chapitre et les deux premiers chapitres forment ensemble les fondements théoriques de la thèse. Dans ce chapitre, nous avons présenté un état de l'art sur les SAAs basé sur plusieurs boucles de contrôle. Tout d'abord, nous avons présenté les défis principaux qui affectent les SAAs avec plusieurs boucles de contrôle. Puis, nous avons mis en évidence les types d'interaction entre les composants MAPE-K. Ensuite, nous avons présenté les approches de contrôle, qui sont : contrôle centralisé, contrôle décentralisé et contrôle hiérarchique. Vue l'objectif de cette thèse, le contrôle hiérarchique est un bon compromis pour gérer la complexité de l'auto-adaptation en raison de bon équilibre apporté par le contrôle hiérarchique entre la lourdeur d'un contrôle décentralisé et la rigidité d'un contrôle centralisé. Cependant, la décomposition hiérarchique des préoccupations d'adaptation et l'affectation de ces préoccupations aux différentes boucles de contrôle pourraient être difficiles à réaliser. Enfin, nous avons présenté une sélection d'approches de composition des boucles de contrôle. De plus, nous avons terminé en présentant un ensemble de défis auxquels sont confrontés ces modèles proposés dans la littérature.

Deuxième partie

Contributions

Chapitre 4

Un Modèle à composants de la boucle de contrôle MAPE-K

Every new beginning comes from some other beginning's end

- Seneca Roman philosopher, mid-1st century AD

4.1 Introduction

Bien que la boucle de contrôle MAPE-K soit plus populaires pour la conception des SAAs, ses conceptions actuelles sont très générales. Où, nous avons remarqué que ses sous-parties ne fournissent pas suffisamment de détails sur l'architecture fondamentale de la boucle de contrôle MAPE-K. De plus, les conceptions existantes n'utilisent pas de modèles de conception bien établis ce qui nous a poussé a proposer un modèle de la boucle de contrôle basé sur le modèle de composant Fractal. Ce dernier fournit les caractéristiques requises telles que le partage de composants et la hiérarchisation des composants. Aussi, il simplifie la réutilisation et la maintenance.

Dans ce qui suit, nous présentons les caractéristiques importantes du Fractal et sa réflexion sur la flexibilité de la boucle de contrôle. Ensuite, nous présentons un modèle de la boucle de contrôle MAPE-K à base des composant Fractal. Enfin, nous discutons les améliorations de ce modèle dans le domaine de SAA.

4.2 Les caractéristiques importantes du Fractal et sa réflexion sur la flexibilité de la boucle de contrôle

Fractal fournit un ensemble de caractéristiques qui lui permettent d'être un formalisme de représentation approprié pour construire une boucle de contrôle. Notez que, Fractal est un modèle général qui n'est pas spécifique à un domaine d'application. Les caractéristiques importantes sont :

4.2.1 Partage de composants

La possibilité de partager un composant permet à un composant d'être utilisé par plusieurs autres composants. Dans la littérature, différents modèles de composition des boucles de contrôle ont été proposés basés sur les techniques de centralisation et/ou décentralisation des entités de la boucle. Les entités centralisées peuvent être modélisées comme un composant partagé. Par exemple, dans le modèle Master/Slave proposé par Weyns et al. [117], il existe une instance de composant Master (qui inclut les entités **A** et **P**) et plusieurs instance de composants slaves (qui incluent les entités **M** et **E**). Dans ce cas, le composant Master peut être modélisé comme un composant partagé.

4.2.2 Hiérarchie des composants

Fractal utilise une approche hiérarchique afin d'augmenter la robustesse et la flexibilité d'un système à base de composants. La hiérarchie reflète généralement une séparation des préoccupations entre les différentes boucles de contrôle.

4.2.3 Réutilisation

Grâce à la réutilisation, les composants MAPE peuvent être utilisés encore et encore dans différentes boucles, car ils peuvent fonctionner sur une abstraction plus élevée et n'ont pas besoin d'être personnalisés pour chaque cas d'utilisation [61]. Enfin, Fractal est un modèle général, dynamique et extensible [20] qui nous donne toutes les caractéristiques de base dont nous avons besoin sans les rendre plus complexes que nécessaire. Fractal nous permet de nous concentrer sur notre problématique spécifique, la modélisation de MAPE-K.

4.2.4 Couplage / Cohésion

Un couplage fort entre différents composants peut rendre les systèmes complexes, car ces systèmes seront difficiles à modifier et à adapter. Les approches à base de composants permettent de minimiser les couplages afin de réduire les interventions de maintenance et d'augmenter le taux de réutilisation. Le modèle de composant Fractal fournit des mécanismes pour réduire le couplage entre les composants. Cela fournit une séparation forte entre l'implémentation du composant et ses interfaces. Une fois les modifications appliquées à une partie particulière du système, le couplage faible peut assurer un faible effet de ces modifications sur les autres parties du système [33, 22].

Le couplage et la cohésion sont corrélés car une cohésion forte se produire un couplage faible. Dans les approches à base de composants, la cohésion est un concept qui fait référence aux taux de connectivité dans les relations entre les composants, c'est-à-dire le taux d'utilisation des attributs d'une classe par les méthodes de la même classe. Une forte cohésion peut garantir la fiabilité, la portabilité et donc la qualité du système. Par conséquent, une forte cohésion et un faible couplage sont des propriétés souhaitables pour une conception de composant bien définie [22]. Ainsi, la conception de la boucle de contrôle MAPE-K basée sur un modèle de composants rend les composants de cette boucle en cohésion forte et en couplage faible.

4.3 MAPE-K à base de composants Fractal

La boucle de contrôle MAPE-K présente les avantages que lui confère le modèle Fractal : flexibilité et dynamicité de sa structure et de son comportement. Le composant MAPE-K est un composant composite Fractal dont le contenu contient les cinq sous-composants à l'origine de son nom : Monitor, Analyze, Plan, Execute et Knowledge [85].

La figure 4.1 présente une modélisation des cinq processus de MAPE-K, de leurs sous-processus et de leurs relations d'interdépendance, en tant que composants, sous-composants et liens entre eux, respectivement. Aussi, nous essayons de préciser ce qui se passe réellement dans chaque partie de cette boucle.

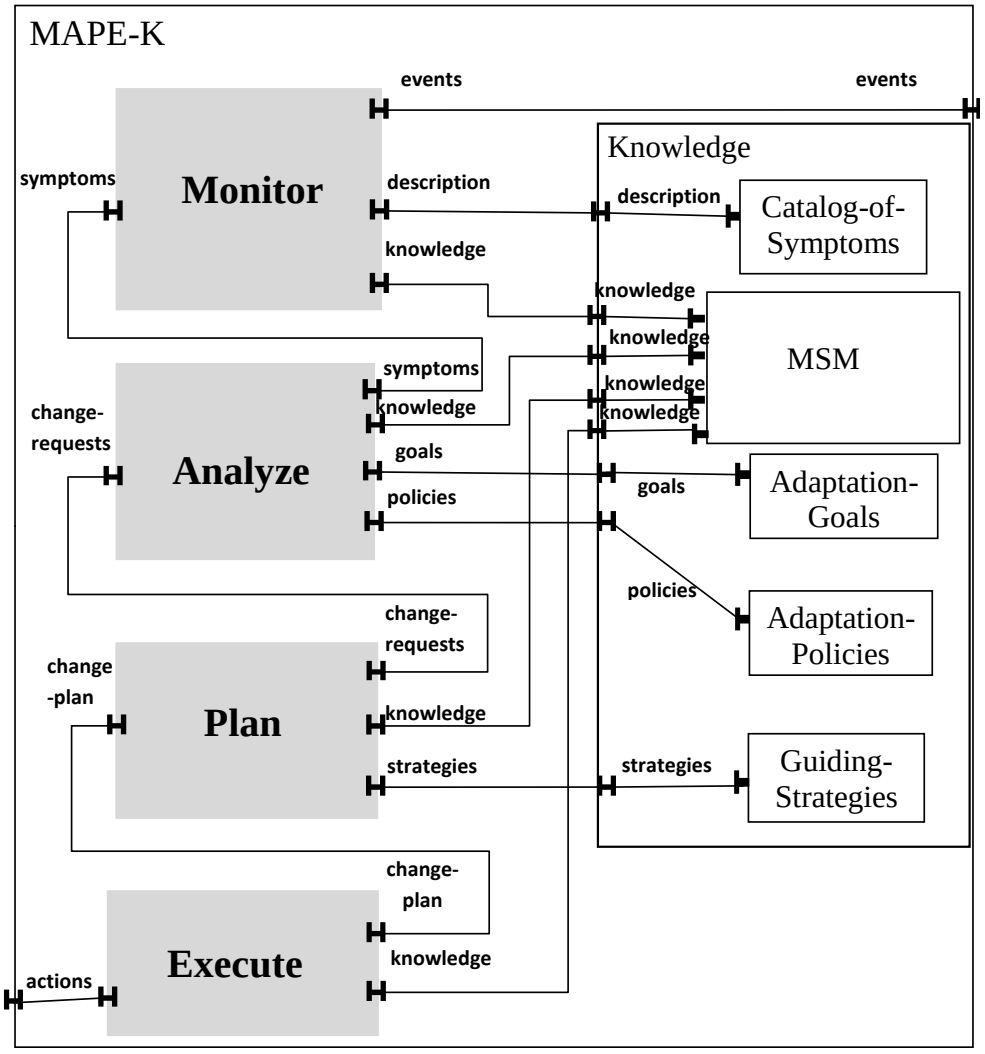


FIGURE 4.1: Modèle MAPE-K à base de composants.

4.3.1 Composant base de Connaissance (Knowledge base Component)

Le composant « Knowledge » est un composant composite Fractal (voir la Figure 4.1). Il représente les informations de gestion qui peuvent être partagées entre les quatre composants de la boucle MAPE-K (Monitor, Analyze, Plan, and Exectue). Ces composants peuvent lire et enrichir la base de connaissance, où il permet également l'échange d'informations entre les composants mentionnés. Son contenu contient les sous-composants suivants :

- ***Le sous-composant « MSM ».*** MSM signifie Managed-System-Model. Ce composant fournit une abstraction du système géré. Où, il représente des informations pertinentes sur les ressources du sous-système géré. Tels que l'état des ressources (utilisées ou non), ... etc. Il fournit une interface appelée « knowledge » qui permet d'obtenir des informations sur les ressources dans le sous-système géré.
- ***Le sous-composant « Catalog-of-Symptoms ».*** Ce composant est utilisé pour générer des symptômes adéquats. Il fournit une interface appelée « description » qui

permet de générer les symptômes.

- ***Le sous-composant « Adaptation-Goals ».*** Ce composant définit les objectifs d'adaptation qui doivent être réalisés par les sous-composants du MAPE-K. Il fournit une interface appelée « goals » qui permet d'analyser les symptômes.
- ***Le sous-composant « Adaptation-Policies ».*** Ce composant contient les politiques saisies par l'administrateur, pour guider les quatre composants de la boucle de contrôle. Les politiques d'adaptation sont les règles utilisées pour déterminer des actions à entreprendre. Ces politiques d'adaptation peuvent être des politiques ECA, des politiques de buts ou des politiques de fonction d'utilité. Il fournit une interface appelée « policies » qui permet de valider les politiques appropriées.
- ***Le sous-composant « Guiding-Strategies ».*** Ces stratégies sont construites en phase de conception pour ordonner l'exécution des actions. C'est un ensemble de contraintes d'ordonnancement. Il fournit une interface appelée « strategies » qui permet de générer un plan de changement.

4.3.2 Composant de Surveillance (Monitor Component)

Le composant de surveillance (Monitor) est un composant composite Fractal. Il représente la première étape du MAPE-K (voir la Figure 4.2). Lorsque les capteurs capturent des événements du système géré et éventuellement de l'environnement, ils appellent le composant de surveillance (Monitor) pour collecter des événements, prétraitent ces événements, puis génèrent un symptôme adéquat pour mettre à jour le *MSM* (situé dans le composant *Knowledge*). Enfin, envoyez-les au composant *Analyze*. Pour détecter l'occurrence d'événements dans le contexte d'exécution de l'application, nous pouvons utiliser le système WildCat comme exemple [35].

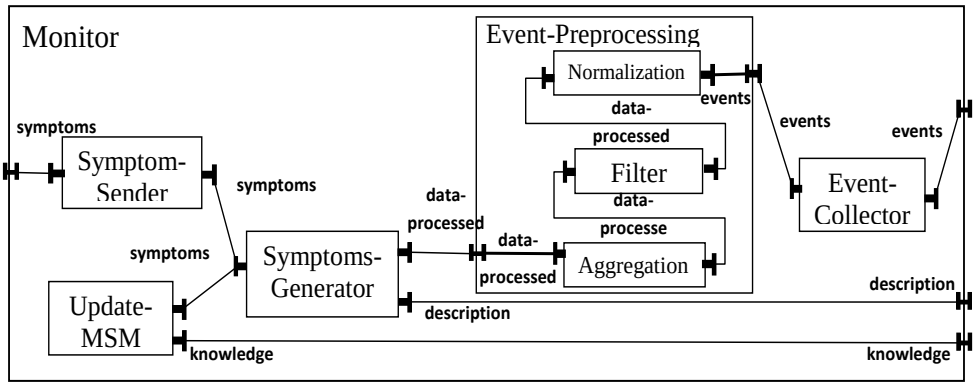


FIGURE 4.2: Composant de surveillance.

Le composant *Monitor* expose une interface fournie et une interface requise. Son contenu contient les sous-composants suivants :

- ***Le sous-composant « Event-Collector ».*** Ce composant est responsable de collecter les événements, il utilise l'interface interne fournie appelée « events » du composant MAPE-K pour recevoir des événements.

- ***Le sous-composant « Event-Preprocessing ».*** Ce composant est responsable du traitement des événements arrivants. Il expose une interface requise appelée « events » et une interface fournie appelée « data-processed ». Son contenu contient les sous-composants suivants :
  - *Normalization.* Ce sous-composant est responsable de transformer, étendre et formater les événements capturés dans un format standardisé. Le but est de les rendre cohérents afin de faciliter le processus de traitement. Il utilise l'interface fournie appelée « events » du sous-composant « Event-Collector » pour recevoir les événements collectés, puis les a normalisés.
  - *Filter.* Ce sous-composant détermine les événements qui sont pertinentes pour la plateforme de gestion actuelle. De plus, le sous-composant *Filter* consiste à ignorer les événements relatifs aux systèmes situés en aval d'une ressource défaillante. Il utilise l'interface fournie appelée « data-processed » du sous-composant « Normalization » pour recevoir des données normalisées, puis les filtre.
  - *Aggregation.* Ce sous-composant prend un ensemble de données en entrée qui peuvent être doublons du même événement, et produit une seule donnée en conséquence. Ces erreurs peuvent se produire en raison de l'instabilité du réseau. Le sous-composant *Aggregation* utilise l'interface fournie appelée « data-processed » du sous-composant « Filter » pour recevoir les données filtrées, puis les agréger
- ***Le sous-composant « Symptoms-Generator ».*** Ce composant est responsable de génèrer un symptôme adéquat via le catalogue de symptômes (inclus dans le composant *Knowledge*). Il utilise l'interface fournie « data-processed » du sous-composant « Event-Preprocessing » pour recevoir les données traitées, puis génère des symptômes. Et pour ce faire, le sous-composant « Symptoms-Generator » utilise l'interface « description » du sous-composant « catalog-of-symptoms » pour générer certains symptômes.
- ***Le sous-composant « Update-MSM ».*** Ce composant est responsable de mettre à jour le MSM via le nouvel état de ressource qui le détecte. Il utilise l'interface fournie « symptoms » du sous-composant « Symptoms-Generator » pour recevoir les symptômes, puis met à jour le *MSM*. Pour ce faire, le sous-composant « Update-MSM » utilise l'interface « knowledge » du sous-composant MSM pour mettre à jour certaines informations sur les ressources.
- ***Le sous-composant « Symptom-Sender ».*** Ce composant permet d'envoyer les symptômes générés au composant *Analyze*. Il utilise l'interface fournie « symptoms » du sous-composant « Symptoms-Generator » pour recevoir les symptômes.

4.3.3 Composant d'Analyse (Analyze Component)

Le composant d'analyse (Analyze) est un composant composite Fractal (voir la Figure 4.3). Il reçoit les symptômes fournis par le composant de surveillance (Monitor). Il permet d'analyser les données contenues dans les symptômes, selon les objectifs d'adaptation pour

déterminer si la situation actuelle doit être modifiée. Si c'est le cas, il sélectionne les composants adaptatifs d'intérêt. Ensuite, il vérifie les politiques d'adaptation existantes (situées dans le composant *Knowledge*) attachées à ces composants adaptatifs. Ensuite, il identifie les politiques d'adaptation nécessaires pour adapter les ressources du sous-système géré. Lorsque les politiques d'adaptation sont sélectionnées, le composant d'analyse génère une demande de changement qui les envoie au composant de planification. Il est important de noter que, l'application basée sur des composants Fractal est considérée comme un assemblage de composants standards et de composants adaptatifs. Où, les politiques d'adaptation sont attachées aux composants adaptatifs.

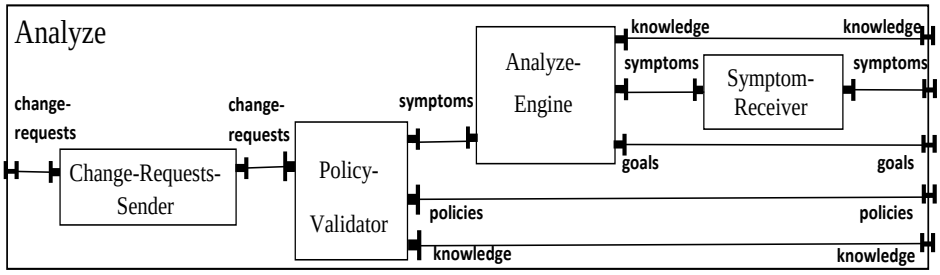


FIGURE 4.3: Composant d'analyse

Le composant *Analyze* expose une interface fournie et une interface requise. Son contenu contient les sous-composants suivants :

- ***Le sous-composant « Symptom-Receiver ».*** Ce composant reçoit les symptômes du composant « Monitor ». Il utilise l'interface fournie « symptoms » du composant « Monitor » pour recevoir les symptômes.
- ***Le sous-composant « Analyze-Engine ».*** Ce composant est responsable d'analyser les données en fonction du composant *MSM* et *Adaptation-Goals*. Ensuite, détermine la situation doit être changée. Il utilise l'interface fournie « symptoms » du sous-composant «Symptom-Receiver» pour recevoir les symptômes, puis les analyses. Pour ce faire, le sous-composant « Analyze-Engine » utilise l'interface « goals » du sous-composant « Adaptation-Goals » pour analyser les symptômes.
- ***Le sous-composant « Policy-Validator ».*** Ce composant permet de valider une politique d'adaptation, en fonction du « Adaptation-Policies ». Ensuite, détermine les demandes de changement. Il utilise l'interface fournie «symptoms» du sous-composant « Analyze-Engine » pour recevoir les symptômes interprétés, puis les politiques validées. Pour ce faire, le sous-composant « Policy-Validator » utilise l'interface « policies » du sous-composant « Adaptation-Policies » pour valider les politiques.
- ***Le sous-composant « Change-Requests-Sender ».*** Ce composant permet d'envoyer la demande de changement générée au composant de planification (Plan). Il utilise l'interface fournie « change-requests » du sous-composant « Policy-Validator » pour recevoir les demandes de modification.

4.3.4 Composant de Planification (Plan Component)

Le composant de planification (Plan) est un composant composite Fractal (voir la Figure 4.4). Il reçoit les demandes de changement fournies par le composant d'analyse (Analyze). Il consiste à traiter les politiques d'adaptation contenues dans les demandes de changement en fonction de l'état actuel du système (vérification MSM), afin de déterminer les actions possibles, et de générer un plan de changement selon Guiding-Strategies (situées dans le composant *Knowledge*), il l'envoie au composant *Execute*.

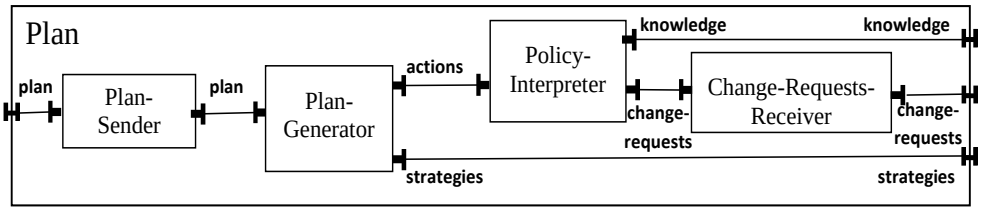


FIGURE 4.4: Composant de planification

Le composant *Plan* expose une interface fournie et une interface requise. C'est un composite qui contient les sous-composants suivants :

- ***Le sous-composant « Change-Requests-Receiver ».*** Ce composant reçoit les demandes de changement du composant *Analyze*. Il utilise l'interface fournie « change-request » du composant *Analyze* pour recevoir les demandes de changement.

- ***Le sous-composant « Policy-Interpreter ».*** Ce composant permet de traiter une politique d'adaptation. Il vérifie le *MSM* pour évaluer la condition qui concerne l'état du système. Si l'évaluation retourne vrai, l'action correspondante sera mise dans le plan. Il utilise l'interface fournie « change-requests » du sous-composant « Change-Requests-Receiver » pour recevoir les demandes de changement, puis les interprètes. Pour ce faire, le sous-composant « Policy-Interpreter » utilise l'interface « knowledge » du sous-composant *MSM* pour interpréter les politiques.

- ***Le sous-composant « Plan-Generator ».*** Ce composant reçoit la séquence d'actions en tant qu'entrée fournie par le sous-composant « Policy-Interpreter ». Il vérifie « Guiding-Strategies », puis génère un plan de changement en réponse. Il utilise l'interface fournie « actions » du sous-composant « Policy-Interpreter » pour recevoir des actions, puis génère un plan. Pour ce faire, le sous-composant « Plan-Generator » utilise l'interface « strategies » du sous-composant « Guiding-Strategies » pour générer le plan de changement.

- ***Le sous-composant « Plan-Sender ».*** Ce composant permet d'envoyer le plan d'action généré au composant *Execute*. Il utilise l'interface fournie « plan » du sous-composant « Plan-Generator » pour recevoir le plan de changement.

4.3.5 Composant d'Exécution (Execute Component)

Le composant d'exécution (Execute) est un composant composite Fractal (voir la Figure 4.5). Il reçoit le plan de changement fourni par le composant de planification « Plan ». Il est responsable de l'exécution du plan d'action. Avant de lancer l'exécution, il convertit le plan en flux de travail. Ensuite, il consulte le sous-composant *MSM* (situé dans le composant *Knowledge*) pour avoir les droits et les moyens de les exécuter. Par exemple : la disponibilité des effecteurs.

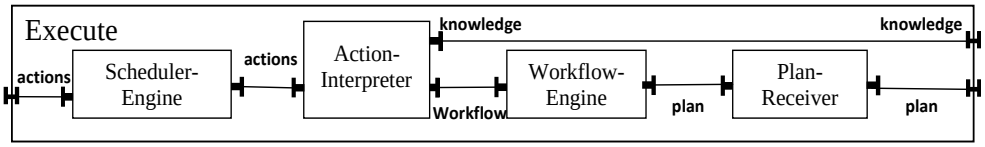


FIGURE 4.5: Composant d'exécution

Le composant *Execute* expose une interface fournie et une interface requise. C'est un composite qui contient les sous-composants suivants :

- ***Le sous-composant « Plan-Receiver ».*** Ce composant reçoit le plan de changement du composant *Plan*. Il utilise l'interface fournie « plan » du composant d'exécution pour recevoir le plan.
- ***Le sous-composant « Workflow-Engine ».*** Ce composant convertit le plan en un flux de travail. Il utilise l'interface fournie « plan » du sous-composant « Plan-Receiver » pour recevoir le plan, puis les convertit en flux de travail.
- ***Le sous-composant « Action-Interpreter ».*** Ce composant se concentre sur le traitement des actions reçues et vérifie si les effecteurs ont les droits et les moyens de les exécuter. Il utilise l'interface fournie « workflow » du sous-composant « Workflow-Engine » pour recevoir le flux de travail, puis l'interprète. Pour ce faire, le sous-composant « Action-Interpreter » utilise l'interface « knowledge » du sous-composant MSM pour traiter les actions reçues.
- ***Le sous-composant « Scheduler-Engine ».*** Pour les actions simultanées, ce composant est responsable de détermine l'action à exécuter ensuite pour équilibrer la charge du système. Il utilise l'interface fournie « actions » du sous-composant « Action-Interpreter » pour recevoir des actions, puis détermine l'action à exécuter ensuite pour équilibrer la charge du système.

4.4 Discussion

Après avoir présenté un modèle de la boucle de contrôle MAPE-K à base de composants Fractal. Nous pensons qu'il serait intéressant d'enrichir ce modèle avec suffisamment de détails à chaque étape de cet boucle afin de faciliter la conception des SAAs. Selon la nature hiérarchique du Fractal, ce modèle offre l'opportunité vers le contrôle hiérarchique qui permet de gérer la complexité de l'auto-adaptation.

Dans ce chapitre, il était aussi nécessaire de parler du principe de la séparation des préoccupations. Cela pose un défi majeur aux chercheurs dans l'étape de conception des SAAs basé sur plusieurs boucles de contrôle. Dans ce cas, comment les préoccupations peuvent-elles se séparer entre différentes boucles de contrôle ?. L'une des technique efficaces pour faciliter la séparation des préoccupations, on trouve le paradigme à base de composants, comme nous nous sommes appuyés dans cette étude sur le Fractal.

Enfin, l'idée principale est d'appliquer les principes de la séparation des préoccupations et de développement à base de composants pour rendre la stratégie d'adaptation des logiciels modulaire et dynamique.

4.5 Conclusion

Dans ce chapitre, nous avons présenté un aperçu de notre modèle de la boucle de contrôle à base des composants Fractal. Ce modèle facilitera le processus de conception des SAAs. Nous avons présenté les caractéristiques importantes du Fractal et sa réflexion sur la flexibilité de la boucle de contrôle. Ensuite, nous avons décrit une boucle de contrôle MAPE-K entière et nous avons définit chaque phase séparément afin de déterminer ce qui se passe réellement dans chaque phase de cette boucle.

Enfin, ce modèle lève l'ambiguïté sur le principe de fonctionnement de chaque phase car nous fournissons une visualisation compréhensible du comportement de cette boucle en préparation du chapitre suivant, dans lequel nous nous concentrerons sur la composition de la boucle de contrôle.

Chapitre 5

Un modèle de contrôle partiellement centralisé

Important thing in science is not so much to obtain new facts as to discover new ways of thinking about them.

– William Bragg

5.1 Introduction

Au cours de la dernière décennie, le processus de conception des SAAs a connu une évolution significative. En effet, parvenir à l'auto-adaptation via plusieurs boucles de contrôle reste l'un des défis majeurs dans les SAAs auxquels les ingénieurs peuvent être confrontés. Le coût élevé et la difficulté d'évolutivité lors de l'utilisation de la coordination entre toutes ou la plupart des entités de la boucle. De plus, le surcoût de communication important lors de la centralisation de certaines entités. Et aussi, le conflit dans la décision lors d'une décentralisation partielle ou totale. Dans ce chapitre, nous présentons un nouveau modèle de contrôle partiellement centralisé pour parvenir à une adaptation globale sans conflit. Généralement, le problème de conflit survient au cours des étapes de planification et d'exécution [123]. En conséquence, nous présentons un modèle de contrôle partiellement centralisé. Où, nous centraliserons les composants *Plan* et *Knowledge* pour assurer des décisions correctes et cohérentes. De plus, nous coordonnerons les composants *Execute* pour assurer un temps d'exécution raisonnable des multiples opérations d'adaptation.

Dans ce qui suit, nous commençons par introduire le langage MSL qui permet de définir les différents modèles MAPE. Ensuite, nous présentons notre modèle de contrôle partiellement centralisé. Puis, nous décrivons l'architecture de ce modèle à base des composants Fractal. Enfin, nous terminons par la définition de ce modèle en MSL.

5.2 Langage MSL

MSL (MAPE Specification Language) [8] est un langage qui permet de définir et d'instancier des modèles MAPE et de donner une sémantique à certains points de variation sémantique de la notation graphique équivalente pour le modèle MAPE. MSL adopte les mêmes concepts de modélisation de la notation graphique MAPE présentés par Weyns et al. [117], mais il utilise une notation textuelle parce que les notations textuelles devraient mieux évoluer que les notations visuelles avec l'augmentation de la taille de la conception du système [77]. Par exemple, la notation graphique utilisée par Weyns et al. [117] ne permet pas de faire la distinction entre un système géré unique et un système composé de différents sous-systèmes, alors que cela est possible dans MSL.

5.2.1 Exigences du langage MSL

Pour concevoir le langage MSL, les auteurs [8] se sont appuyés sur les principes de conception logicielle et de l'architecture DSL [1]. Où, ils ont identifié l'ensemble d'exigences comme des dimensions de conception clès du language MSL. Ces exigences sont :

- **Objectif du modèle.** Le DSL devrait être utilisé comme interface de modélisation pour fournir un modèle concis et simple de la structure composite des boucles MAPE-

1. Domain-Specific Language : un langage informatique spécialisé dans un domaine d'application particulier.

K interactives. Donc, l'objectif principal du modèle est de servir une compréhension partagée de la structure des boucles MAPE-K entre les différents espaces de solutions et les frameworks (ou les plateformes logicielles associés connectés au Framework centré sur DSL).

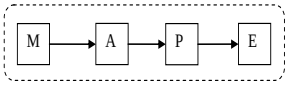
- **Séparation des préoccupations.** Introduire des abstractions de langage qui permettent de décomposer la logique d'adaptation d'un SAA en préoccupations d'adaptation distinctes en les structurant en différentes boucles de contrôle MAPE avec le moins de chevauchement de fonctionnalités possible.
- **Principe de la moindre connaissance (ou loi de Demeter).** La loi de Demeter permet de fournir un logiciel a tendance à être plus maintenable et adaptable. Où, une boucle ou un composant MAPE ne doit pas connaître les détails internes des autres boucles ou composants MAPE.
- **Minimiser la conception en amont.** Introduire des constructions de langage pour concevoir ce qui est requis, évitant ainsi trop d'efforts de conception prématurément.
- **Conception orientée modèle.** Combiner des modèles individuels de boucles MAPE dans une structure composée hétérogène de boucles MAPE pour faciliter le développement constructif de la couche d'adaptation du système.
- **Composabilité du modèle.** A un certain point de la conception, les modèles de boucles MAPE-K obtenus par séparation de l'adaptation sont liés à la demande de composition du modèle. À cette fin, le principal problème consiste à assurer la capacité de composer des boucles MAPE-K interférentes pour former des sous-systèmes de gestion fiables sans violer les exigences d'adaptation souhaitées, et sans réduire la fiabilité résultante.

5.2.2 Notation de modélisation MSL

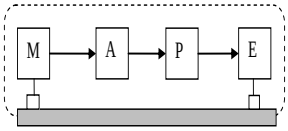
Les éléments de modélisation de base de MSL pour définir un modèle MAPE en termes de groupes abstraits de composants MAPE, de sous-systèmes gérés (c'est-à-dire un ensemble d'éléments ou parties observables d'un sous-système géré) et leurs interactions avec les multiplicités sont rapportés dans la table 5.1. Pour chaque élément, la notation graphique correspondante adoptée par Weyns et al. [117] et la notation textuelle en MSL [8] sont indiquées.

L'une des ambiguïtés de la représentation en boucle MAPE proposée par Weyns et al. [117] est liée à l'interprétation des interactions [1,*], [*,1] et [*,*] entre plusieurs composants de différents groupes MAPE. En effet, il n'est pas clair si l'interaction et la communication doit s'établir entre tous les composants impliqués ou seulement certains d'entre eux. Par conséquent, en MSL, en plus de la multiplicité standard **1**, nous permettons la spécification de la sémantique voulue de * au moyen des multiplicités : *-ALL, *-SOME et *-ONE (voir la table 5.2). Lorsqu'elles sont utilisées comme multiplicité source de l'interaction, ces multiplicités signifient respectivement que le groupe cible doit recevoir la communication de tous les groupes en interaction, d'au moins l'un d'entre eux, ou d'un seul d'entre eux.

Les éléments MAPE	Notation graphique	Notation MSL

Groupe abstrait AGrp de composants MAPE		Group AGrp {
		components M, A, P, E
		}

Système géré		system Sys
Système géré divisé en sous-systèmes Sys1, Sys2, etc.		system Sys1 system Sys2 ...

Gestion de l'interaction gérée entre un groupe abstrait AGrp et un System Sys		system Sys
		Group AGrp {
		managedSys Sys
		components M, A, P, E
		}

Gestion de l'interaction gérée entre un groupe abstrait AGrp et un système Sys1, Sys2, etc.		system Sys1
		system Sys2
		...
		Group AGrp {
		managedSys Sys1, Sys2
		components M, A, P, E
		}

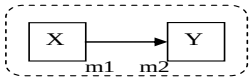
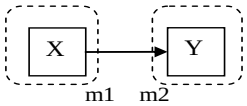
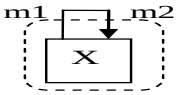
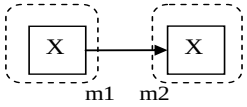
Interaction inter-composants dans un groupe abstrait AGrp		interaction AGrp.X ->AGrp.Y [m1,m2]
Interaction inter-composants entre deux groupes abstraits AGrp1 et AGrp2		interaction AGrp1.X ->AGrp2.Y [m1,m2]
Interaction intra-composant dans un groupe abstrait AGrp		interaction AGrp.X ->AGrp.X [m1,m2]
Interaction intra-composant entre deux groupes abstraits AGrp1 et AGrp2		interaction AGrp1.X ->AGrp2.X [m1,m2]

TABLE 5.1: Notation MSL pour définir les modèles MAPE [8].

Concepts	**Signification**
Interaction Centralisée	Grp.X→Grp.Y[1, 1]
Interaction Décentralisée	Grp1.X→Grp2.Y[m1 , m2]
Sémantique du point de variation m1, m2	*-ALL : à tous les composants associés. *-SOME : à un sous-ensemble de composants. *-ONE : à un composant.

TABLE 5.2: Les interactions en MSL [8].

5.3 Un modèle de contrôle partiellement centralisé

En effet, plusieurs modèles ont émergé des expériences des auteurs avec la construction des SAAs. Ils ont proposé différents modèles de boucles MAPE qui interagissent avec différents degrés de décentralisation. Cependant, ces modèles précédemment proposés ne représentent pas toutes les configurations possibles. Conformément à nos recherches sur la construction des SAAs, nous avons proposé une nouvelle configuration de contrôle partiellement centralisé.

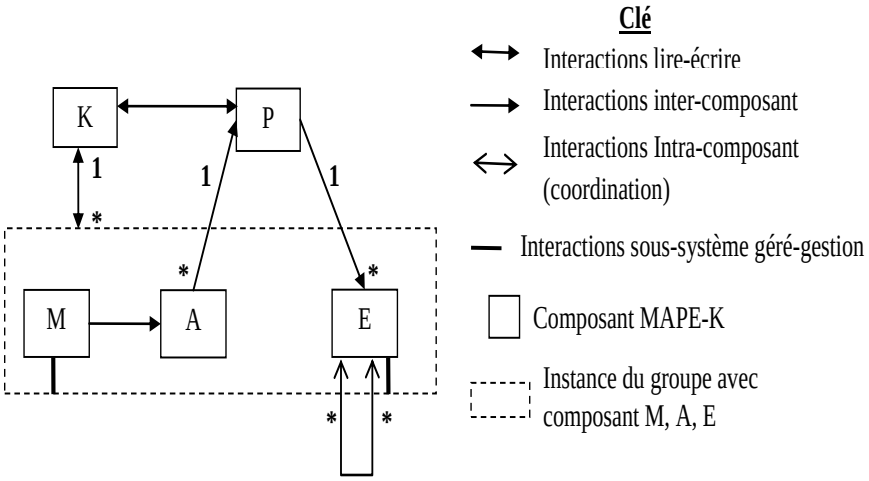


FIGURE 5.1: Modèle de contrôle partiellement centralisé.

La figure 5.1 montre un modèle de composition partiellement centralisé de boucles MAPE-K [87]. Nous intégrons les techniques de centralisation et de décentralisation afin de bénéficier les avantages des deux. Ce modèle se compose d'une instance unique des composants **P** et **K** (où, **P** et **K** sont centralisés) et d'un nombre arbitraire d'instances du groupe avec les composants **M**, **A** et **E** (où, les composants **M**, **A** et **E** sont décentralisés). Le composant partagé **K** est disponible dans ce pattern. Pour qu'il puisse être utilisé par toutes les instances du groupe avec les quatre composants d'une boucle MAPE. Dans ce modèle, nous distinguons quatre types d'interaction, à savoir : les interactions *lire-écrire*, les interactions *inter-composant*, les interactions *intra-composant* (coordination) et les interactions

sous-système géré-gestion.

Le composant **K** est un aspect important de la conception des boucles MAPE-K interactives. Ce composant est partagé entre les différents composants MAPE des différentes boucles. Suivant le modèle MAPE-K à base de composants (présenté dans le Chapitre 04), les quatres composants *Monitor* (M), *Analyze* (A), *Plan* (P) et *Execute* (E) sont en interactions de type *lire-écrire* avec le sous-composant *MSM* (voir la figure 4.1). Où, *MSM* représente les informations pertinents sur les ressources du sous-système géré. Ainsi, au lieu d'échanger ces informations via des canaux de coordination entre chaque composant et ses pairs, ils peuvent y accéder via des interactions *lire-écrire* entre le composant **K** et les différentes instances des composants MAPE. Donc, le composant **K** a un impact sur la cohérence de l'adaptation globale.

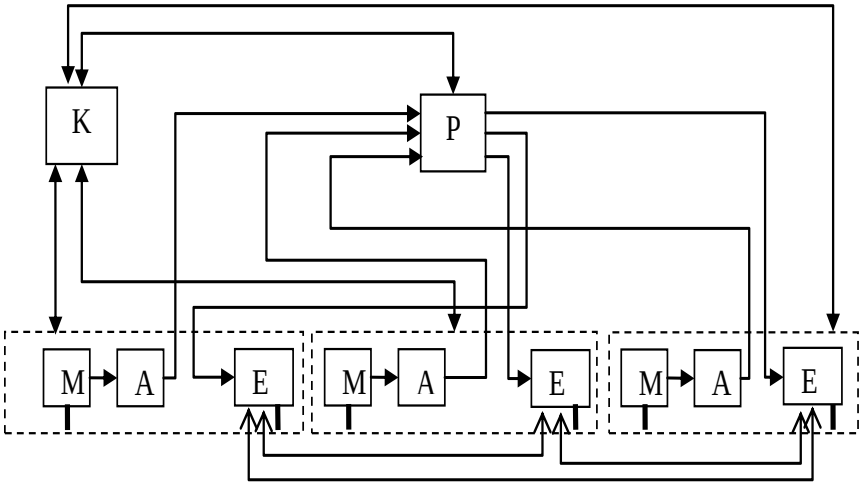


FIGURE 5.2: Instance du modèle dans une configuration concrète.

La figure 5.2 illustre le modèle pour une configuration concrète. Ce diagramme d'instance montre une instanciation concrète de notre modèle avec trois boucles de contrôle MAPE-K. Dans ce qui suit, nous présentons les trois caractéristiques du modèle de contrôle partiellement centralisé comme suit :

5.3.1 Décentralisation des composants M et A

Dans le modèle de contrôle partiellement centralisé, le composant **M** fonctionne indépendamment de ses pairs. Cela est dû au fait que les opérations effectuées par le composant **M** n'ont pas besoin d'être coordonnées avec ses pairs. Ces opérations comme présentées dans la section 4.3.2 sont : collecter les évènements, prétraiter ces évènements (Normaliser, Filtrer et Agréger), générer des symptômes utilisant **catalog-of-symptoms** et mettre à jour le **MSM**. Donc, chaque composant **M** ne peut pas avoir d'impact sur le fonctionnement de ses pairs.

Chaque composant **M** peut interagir avec le composant **K**. L'interaction *lire-écrice* entre

les composants **M** et **K** permet de : (1) lire les descriptions de **catalog-of-symptoms** pour générer les symptômes. Dans ce modèle, chaque composant **M** et ses pairs peuvent utiliser le même catalogue de symptômes, et (2) écrire les informations du sous-système géré dans **MSM**. Ainsi, les informations collectées sur l'état du sous-système géré sont disponibles dans le sous-composant partagé **MSM**, ce qui peut conduire à une analyse locale.

Le principe de fonctionnement du composant **M** peut aider le composant **A** à travailler indépendamment de ses pairs. Semblable à composant **M**, les opérations effectuées par le composant **A** n'ont pas besoin d'être coordonnées avec ses pairs. Ces opérations comme présentées dans la section 4.3.3 sont : analyser les symptômes en fonction du **MSM** et **Adaptation-Goals** et valider les politiques d'adaptation en fonction du **MSM** et **Adaptation-Policies**. L'analyse des symptômes d'une ressource et la validation des politiques d'adaptation dans son fonctionnement peuvent nécessiter des informations pour d'autres ressources. Ceci est possible grâce à **MSM** qui fournit des informations pertinentes au sous-système géré.

Chaque composant **A** peut communiquer avec le composant **K**. L'interaction *lire-écrire* entre les composants **A** et **K** permet de : (1) lire les objectifs et les politiques d'adaptation depuis les sous-composants **Adaptation-Goals** et **Adaptation-Policies**, respectivement. Dans ce modèle, chaque composant **A** et ses pairs utilisent les mêmes objectifs et politiques d'adaptation, et (2) lire certaines informations sur l'état du système depuis le sous-composant partagé **MSM**, qui est collecté par plusieurs composants **M**. Par conséquent, les composants **M** et **A** doivent être effectuées localement à chaque ressource géré.

Enfin, les communications entre les composants **M** et **K** et entre les composants **A** et **K** permettent aux composants **M** et **A** d'agir localement sans besoin de coordination.

5.3.2 Centralisation du composant P

Le composant **P** reçoit les demandes de changement de nombre arbitraire des composants **A** pour planifier les adaptations. Notre modèle consiste à centraliser le composant de la planification (**P**), qui s'inspire du système nerveux central (SNC) qui prend les informations sensorielles du corps. Ensuite, le cerveau (une entité du SNC) utilise ces informations sensorielles, et prend en charge les décisions nécessaires pour maintenir l'homéostasie du corps et assurer sa survie.

Les opérations effectuées par le composant **P** comme présentées dans la section 4.3.4 sont : l'interprétation des politiques d'adaptation en fonction de MSM et la génération d'un plan de changement en fonction du **Guiding-strategies**. La centralisation de ces opérations permet d'éviter les conflits de décision et les problèmes de coordination tels que les coûts élevés et les erreurs de communication qui peuvent survenir en cas de décentralisation et de coordination des composants **P**. De plus, la centralisation du composant **P** peut faciliter la mise en œuvre d'un plan d'adaptation efficace visant à atteindre des objectifs et des garanties globales.

L'objectif derrière la centralisation de composant **P** est d'atteindre certains critères importants dans la génération d'un plan cohérent qui ne peut pas être atteint dans le cas où

le composant **P** est décentralisé, même coordonné. Ces critères sont : (1) l'ordre des actions d'adaptation en termes de temps, de priorité, et (2) l'adaptation optimale, en sélectionnant les actions appropriées du point de vue du système global.

5.3.3 Coordination des composants E

Ce qui caractérise ce modèle est que chaque composant **E** coordonne ses opérations avec ses pairs des autres boucles. Au cours de cette activité, la synchronisation doit être effectuée pour s'assurer que l'exécution de l'adaptation est gérée correctement.

Suivant le modèle MAPE-K à base de composants (présenté dans le Chapitre 04), chaque composant **E** reçoit un plan de changement et applique ces opérations : (i) convertir le plan en un flux de travail contenant un ensemble des actions, puis (ii) traiter ces actions. Grâce à la coordination, les composants **E** peuvent coordonner l'exécution en terme d'occupation des effecteurs disponibles et l'ordre des actions également en termes de temps et de priorité. De plus, grâce à la coordination, le sous-composant **Schedular-Engine** peut contrôler l'exécution des actions simultanées grâce à l'échange de messages entre différents composants **E**.

La coordination permet l'échange de messages pour synchroniser les actions d'adaptation. Le transfert du message pour synchroniser l'exécution n'a pas besoin d'un coût élevé. L'exécution décentralisée permet d'exécuter leurs fonctionnalités localement, mais en coordination avec ses pairs. Du côté positif, la charge d'exécution est répartie sur l'arbitraire de composant **E**. Dans le cas de systèmes distribués à grande échelle, l'exécution décentralisée peut être la meilleure option.

5.4 Modélisation textuelle du modèle de contrôle partiellement centralisé

Dans cette section, nous introduisons le modèle de contrôle partiellement centralisé en utilisant le modèle de composant Fractal et le Langage MSL. Puis, nous présentons une comparaison de ces deux outils de modélisation.

5.4.1 Modèle de contrôle partiellement centralisé à base de composants

Pour illustrer la validité du modèle de contrôle partiellement centralisé, nous avons choisi comme exemple les véhicules électriques. La véhicule électrique fait référence à tout véhicule qui utilise des moteurs électriques pour la propulsion. Généralement, il est alimenté par une batterie de stockage. Le véhicule électrique peut être, par exemple, un vélo électrique, une voiture électrique ou un avion électrique. Ce domaine est très intéressant pour de nombreux chercheurs qui étudient des cas spécifiques dans ce domaine.

Par exemple : économies d'énergie pour les véhicules en stationnement afin d'améliorer la connectivité VANET [106] technique intelligente pour prédire l'intention du conducteur [16].

Dans cette étude, nous choisissons une voiture électrique (Electric car, EC). Les EC peuvent être considérés comme une combinaison de différents sous-systèmes. Chacun de ces sous-systèmes interagit les uns avec les autres pour faire fonctionner l'EC, et plusieurs technologies peuvent être utilisées pour faire fonctionner les sous-systèmes. En raison de la complexité du système, nous présentons un sous-système pour valider notre proposition avec deux boucles de contrôle MAPE.

La figure 5.3 montre l'application EC basée sur Fractal. Les composants **MAPE-K1** et **MAPE-K2** sont des instances du modèle MAPE-K basé à base de composants présenté dans le Chapitre 04. Le composant **Knowledge** n'est pas modélisée dans la Figure 5.3 en raison des liens fréquents entre les composants **MAPE-K1** , **MAPE-K2** et le composant **Knowledge**. Cependant, il reste le même principe utilisé dans le Chapitre 04.

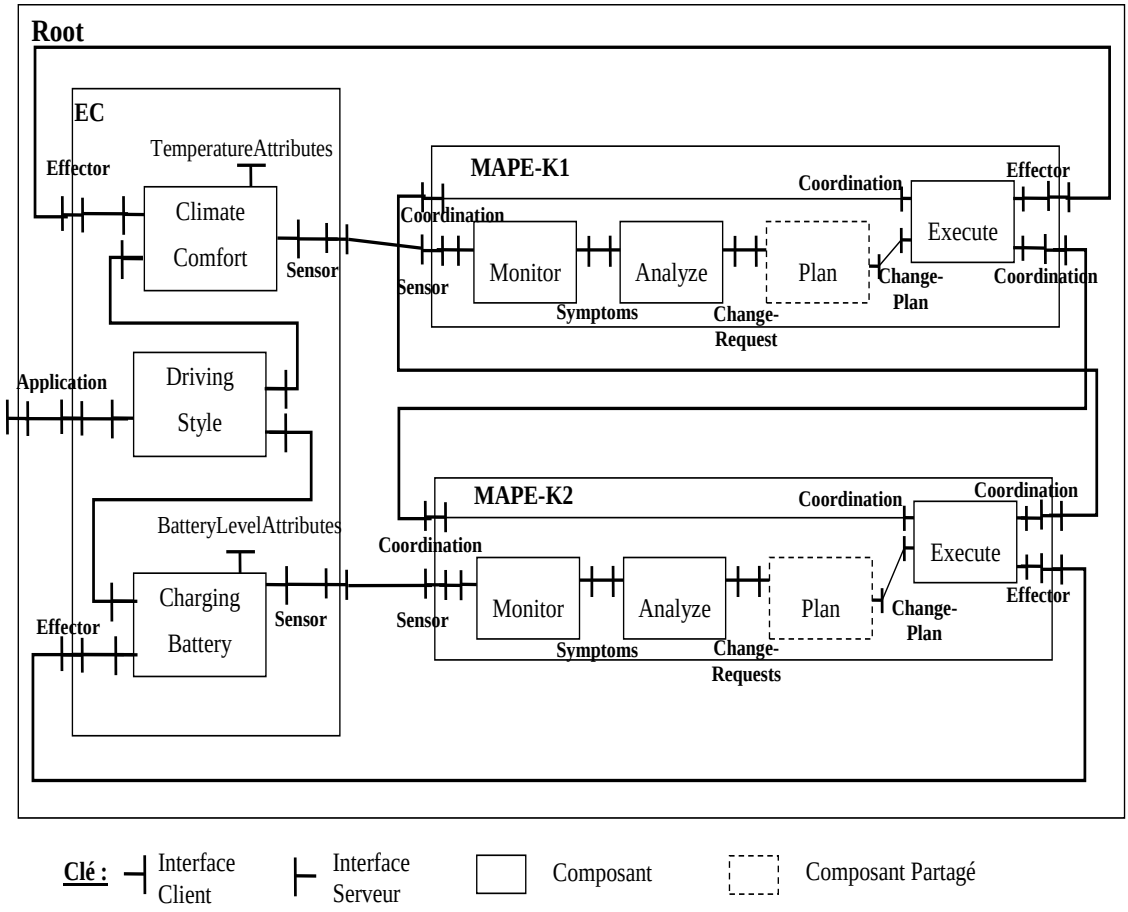


FIGURE 5.3: Application EC basée sur le Fractal.

Fractal prend en charge la notion du partage de composant (Shared Component), c'est-à-dire le composant qui a plusieurs super-composants directs [36]. C'est une fonctionnalité puissante [36] qui nous a fait choisir Fractal. Dans notre modèle, il n'y a qu'une seule

instance du composant **Plan**, nous le modélisons comme un composant partagé par deux composites **MAPE-K1** et **MAPE-K2**.

Les composants adaptatifs **ClimateComfort** et **ChargingBattery** ont des interfaces d'attribut de type *TemperatureAttributes* et *BatteryLevelAttributes*, respectivement. L'interface *TemperatureAttributes* fournit deux méthodes pour obtenir et définir l' attribut ***Temperature***. Tandis que, L'interface *BatteryLevelAttributes* fournit deux méthodes pour obtenir et définir l' attribut ***BatteryLevel***. Chaque instance du composant **Monitor** fonctionne indépendamment et il en va de même pour le composant **Analyze**. Le composant **Plan** est partagé par deux composites **MAPE-K1** et **MAPE-K2** afin de recevoir les demandes de changement des composants **Analyze** pour planifier les adaptations. Enfin, les composants **Execute** sont coordonnés leurs opérations pour synchroniser l'exécution des actions d'adaptation.

Le code 1 montre une description ADL Fractal de l'application EC présentée à la figure 5.3. Il représente trois composants abstraits **EC**, **MAPE-K1** et **MAPE-K2**. Le composant **Plan** est partagé par deux composites **MAPE-K1** et **MAPE-K2**. La dernière section du code montre les liaisons entre ces composants.

```
<definition name="Application.Root">

<interface name=" ap" role="server" signature="Application.App"/>

<component name=" EC" definition="Application.EC" />
<component name="MAPE-K1" definition="Application.MAPE-K" />
<component name="MAPE-K2" definition="Application.MAPE-K" >
        <component name="Plan" definition="MAPE-K1/Plan" />
</component>

<binding client="this.ap" server="EC.ap" />
<binding client="EC.sensor1" server="MAPE-K1.sensor1" />
<binding client="EC.sensor2" server="MAPE-K2.sensor2" />
<binding client="MAPE-K1.effector1" server="EC.effector1" />
<binding client="MAPE-K2.effector2" server="EC.effector2" />
<binding client="MAPE-K1.coordination1" server="MAPE-K2.cor" />
<binding client="MAPE-K2.coordination2" server="MAPE-K1.coordination1" />

</definition>
```

Code 1 : Description ADL Fractal de l'application EC.

Le Code 2 fournit la description ADL Fractal du composant **MAPE-K**. La cardinalité pour l'interaction de type *coordination* entre les composants **E**, comme le montre la figure 5.1 est [*,*]. Dans le Fractal, les deux interfaces qui sont appelés *coordination1* et *coordination2*, dont les rôles sont respectivement *server* et *client*. Leur cardinalité est *collection*, ce qui signifie que la communication peut être établie entre tous les composants **E**.

Le Code 3 fournit la description ADL Fractal du composant **Plan**. La cardinalité pour

```
<definition name="Application.MAPE-K">

<interface name="sensor" role="server" signature="Application.Sensor"/>
<interface name="coordination1" role="server" cardinality="collection" signature="Application.Coordination"/>
<interface name="effector" role="client" signature="Application.Effector"/>
<interface name="coordination2" role="client" cardinality="collection" signature="Application.Coordination"/>

<component name="Monitor" definition="Application.Monitor" />
<component name="Analyze" definition="Application.Analyze" />
<component name="Plan" definition="Application.Plan" />
<component name="Execute" definition="Application.Execute" />
<component name="Knowledge" definition="Application. Knowledge " />

<binding client="this.sensor" server="Monitor.sensor" />
<binding client="Monitor.symptoms" server="Analyze.symptoms" />
<binding client="Analyze.changeRequest" server="Plan.changeRequest" />
<binding client="Plan.changePlan" server="Execute. changePlan " />
<binding client="Execute.effector" server="this.effector" />
<binding client="this.coordination1" server="Execute.coordination1" />
<binding client="Execute.coordination2" server="this.coordination2" />

</definition>
```

Code 2 : Description ADL Fractal du composant MAPE-K.

```
<definition name="Application.Plan">

<interface name="changeRequest" role="server" cardinality="collection" signature="Application.ChangeRequests"/>
<interface name="changePlan" role="client" cardinality="collection" signature="Application.ChangePlan"/>

<content class="Application.Plan"/>

</definition>
```

Code 3 : Description ADL Fractal de composant Plan.

l'interaction de type *inter-composant* entre les composants **A** et **P** et entre les composants **P** et **E**, comme le montre la figure 5.1 est [*,1] et [1,*], respectivement. Dans le Fractal, les deux interfaces qui sont appelés *changeRequests* et *changePlan*, dont les rôles sont respectivement *server* et *client*. Leur cardinalité est *collection*, ce qui signifie que la communication peut être établie entre les différents composants **A** avec la composant **P**. De plus, la communication peut être établie entre le composant **P** avec les différents composants **E**.

En conséquence, lorsque l'on décrit la déclaration d'une interface de composant ayant l'expression « cardinality="collection" », cela signifie que ce composant peut communiquer avec de nombreux autres composants qui lui sont associés via cette interface. En revanche, déclarer une interface de composant qui ne comporte pas l'expression « cardinality="collection" » signifie que ce composant ne peut communiquer qu'avec un seul composant qui lui est associé via cette interface.

Afin d'illustrer la validation du modèle de contrôle partiellement, nous nous référons aux scénarios suivants.

► Scenario 1 : Climate Comfort

La température est surveillée par un capteur, nommé *TemperatureSensor*. Si la température dépasse la température souhaitée (dans cet exemple 21°C), le système l'adaptera en effectuant l'action *Adapt_Temperature*. Par exemple, lorsque le capteur de température capte 30°C, le système diminue la température de 4 à 5°C (c'est-à-dire qu'elle devient 25°C). Dans ce scénario, une seule boucle **MAPE-K1** est utilisée.

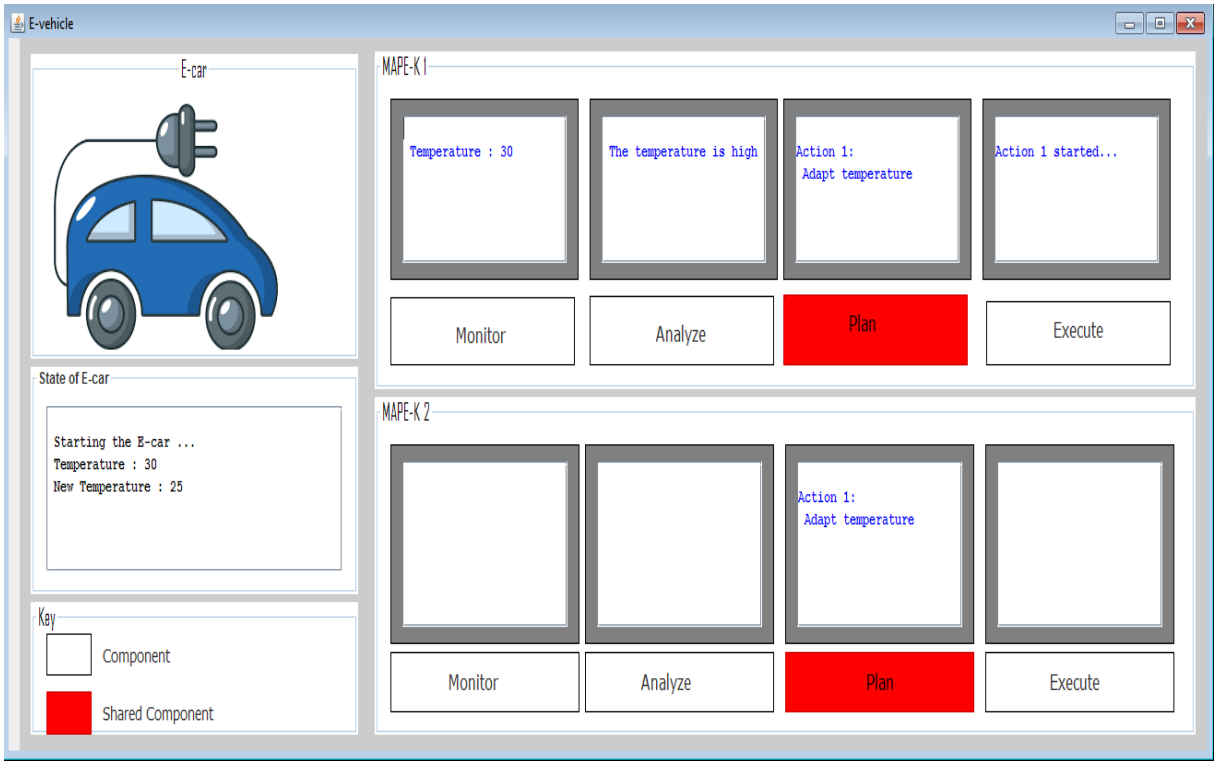


FIGURE 5.4: Scénario de confort climatique.

La figure 5.4 montre le premier scénario, l'auto-adaptation de la température de l'EC. Il représente toutes les étapes de la boucle de régulation : (i) Surveiller la température, (ii) L'analyser, puis (iii) générer un plan de changement, enfin (iv) effectuer une action d'adaptation.

► Scénario 2 : Confort climatique / Chargement de la batterie

Le système capture simultanément deux événements : (i) un changement de température via le capteur, nommé *TemperatureSensor*, (ii) et la signalisation de la batterie via un autre capteur, nommé *BatteryLevelSensor*. Après en analysant ces événements, le composant partagé **Plan** génère le plan de changement approprié. Il faut d'abord charger la batterie, puis

la température va s'adapter (car l'adaptation de la température nécessite de l'électricité). La coordination des composants **Execute** assure un temps d'exécution raisonnable de ces actions.

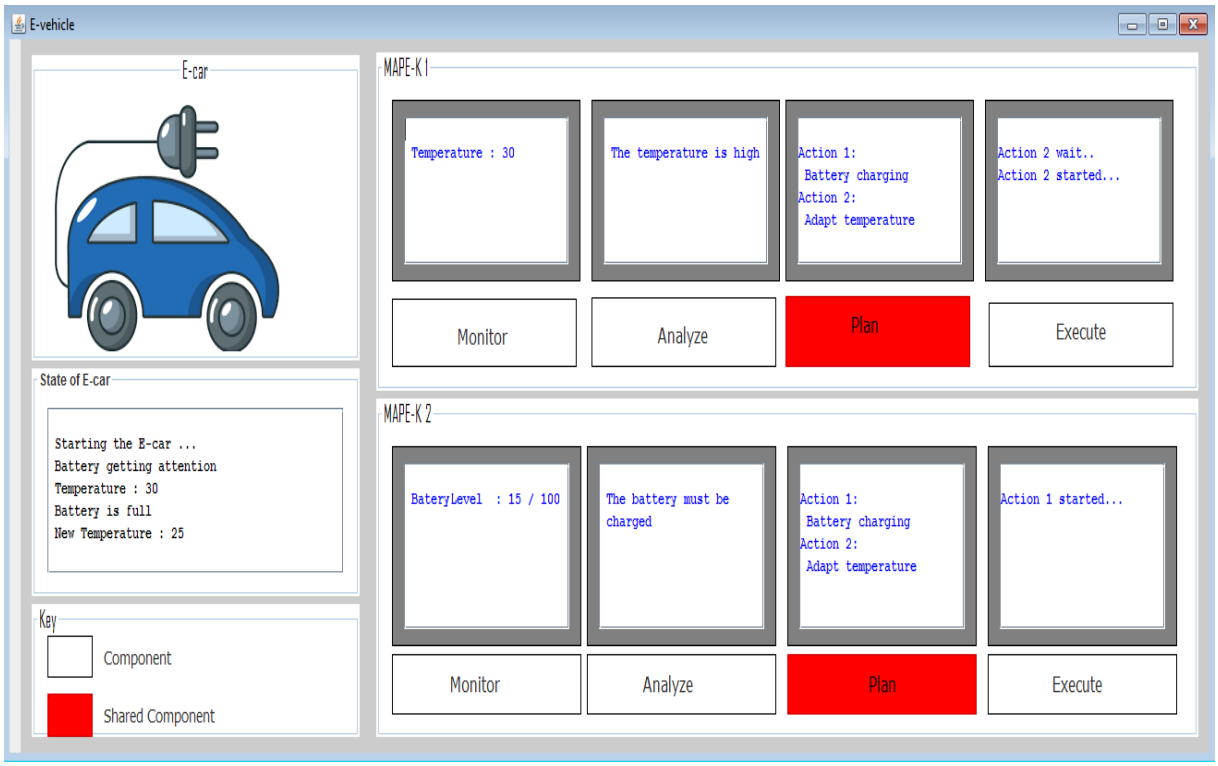


FIGURE 5.5: Scénario confort climatique / charge batterie.

La figure 5.5 montre le deuxième scénario, l'auto-adaptation de l'état de l'EC. Il représente toutes les étapes de MAPE-K1 et MAPE-K2 : (i) Surveillance de la température/niveau de la batterie, (ii) Analyse de la température/niveau de la batterie, puis (iii) génération d'un plan de changement, enfin (iv) réalisation d'actions d'adaptation avec coordination de l'exécution.

En conséquence, le modèle de contrôle partiellement centralisé assure une décision cohérente grâce à la décentralisation du composant **Plan** et une exécution dans un temps raisonnable grâce à la décentralisation et la coordination des composants **Execute**. De plus, la centralisation du composant **Plan** facilite la création d'un plan d'adaptation qui répond à certains critère tels que : (i) L'ordre d'action en terme de temps et de priorité, et (ii)L'adaptation optimale en sélectionnant les actions appropriées du point de vue du système globale. Aussi, limiter la coordination aux composants **Execute** peut aider à réduire le coût de la coordination, réduisant ainsi le taux d'erreur de communication.

D'autre part, en termes de comparaison entre la modélisation textuelle et graphique fournie par le modèle de composant Fractal, ADL Fractal permet d'être plus précis que la notation graphique du Fractal. Comme nous l'avons observé dans la schéma représentatif d'un modèle de contrôle partiellement centralisé basé sur les composants (voir Figure 5.3), nous n'avons pas été en mesure de représenter le composant **Knowledge** en raison des liens fréquents entre les composants **MAPE-K1** , **MAPE-K2** et le composant **Knowledge**.

Ainsi, la représentation graphique rend l'augmentation de la taille de la conception difficile et invisible.

5.4.2 Modèle de contrôle partiellement centralisé en MSL

C'est vrai que nous pouvons définir et instancier des modèles MAPE utilisant le Fractal. Mais avec l'avènement du MSL ces dernières années, nous pouvons définir ces modèles et donner une sémantique à certains points de variation. Le Code 4 rapporte la définition MSL d'un modèle de contrôle partiellement centralisé avec deux boucles de contrôle MAPE-K. Ici, le système géré est divisé en deux sous-systèmes *Sys1* et *Sys2*. La gestion de l'interaction est gérée entre le groupe abstrait bottom et les sous-systèmes.

```
abstrat pattern PartiallyCentralizedMAPE-K {

system Sys1
system Sys2

group bottom {
managedSys Sys1, Sys2
components M, A, E
}

group high {
components P, K
}

interaction bottom.M ->bottom. A [1, 1]
interaction bottom.A ->high.P [*-SOME, 1]
interaction high.P ->bottom.E [1,*-SOME]
interaction bottom.M ->high. K [*-SOME, 1]
interaction high.K ->bottom.M [1, *-SOME]
interaction bottom.A ->high.K [*-SOME, 1]
interaction high.K ->bottom.A [1, *-SOME]
interaction high.P ->high.K [1, 1]
interaction high.K ->high.P [1, 1]
interaction bottom.E ->high.K [*-SOME, 1]
interaction high.K ->bottom.E [1, *-SOME]
interaction bottom.E ->bottom.E [*-ALL, *-ALL]

}
```

Code 4 : Une définition de modèle de contrôle partiellement centralisé dans MSL.

Le Code 4 présente la définition de modèle de contrôle partiellement centralisé dans MSL. On peut résumer les interactions comme suit :

- La communication d'un composant **M** (cardinalité =1) avec un composant **A** (cardinalité =1).
- La communication d'au moins un composant **A** (cardinalité =*-SOME) avec le composant **P** (cardinalité =1).
- La communication d'un composant **P** (cardinalité =1) avec au moins un composant **E** (cardinalité =*-SOME).
- La communication d'au moins un composant **M** (cardinalité =*-SOME) avec le composant **K** (cardinalité =1). (la même chose pour les composants **A** et **E**).
- La communication d'un composant **K** (cardinalité =1) avec au moins un composant **M** (cardinalité =*-SOME). (la même chose pour les composants **A** et **E**)
- La commination entre les composants **E** (cardinalité =*-ALL).

Nous avons utilisé la cardinalité *-SOME pour identifier la communication entre les composants **A** -> **P** et **P**->**E**. En effet, le composant **A** ne s'exécute que lorsqu'il reçoit des symptômes de **M**. Avoir plusieurs composants **A** ne signifie pas nécessairement qu'ils s'exécutent tous simultanément. Par conséquent, le composant **P** reçoit des demandes de changement d'au moins un composant **A**. La même chose pour la communication **P** -> **E**. De plus, la communication *-ALL entre les composants **E** signifie que chaque composant **E** doit communique avec ses pairs afin de synchroniser l'exécution du plan de changement.

En conclusion, MSL est un langage moderne et intéressant. L'une des difficultés pour les concepteurs des SAAs est la modélisation de sous-système de gestion. À notre avis, MSL est la première étape vers la création d'un outil intégré pour modéliser rapidement et facilement les SAAs.

5.4.3 Comparaison

Dans cette section, nous présentons une comparaison de la modélisation avec Fractal et MSL dans un ensemble de points comme suit :

- Le Fractal est un modèle général qui n'est pas spécifique à un domaine d'application. Où, le Fractal offre la possibilité de modéliser le sous-système gestion et le sous-système géré comme indiqué dans le Code 1. En revanche, le MSL est un langage de modélisation spécifique des modèles MAPE, c'est-à-dire qu'il permet la modélisation de sous-système de gestion uniquement. Dans ce cas, il est nécessaire d'intégrer un autre modèle ou langage pour modéliser le sous-système géré.
- La modélisation textuelle peut être meilleure que la modélisation graphique à mesure que la taille de conception du système augmente. Cependant, la modélisation graphique a l'avantage de faciliter la visualisation et la compréhension d'une conception particulière. En tant qu'ingénieurs dans le domaine de l'informatique, nous utilisons souvent d'abord la modélisation graphique, puis nous passons à la modélisation textuelle comme nous l'avons fait dans la section 5.4.1
- Pour exprimer les liaisons entre les différents composants de la boucle de contrôle, le langage MSL utilise le concept *interaction*, tandis que le modèle Fractal utilise le concept *binding*. La différence entre ces deux concepts est que le MSL peut exprimer

la multiplicité en trois sens : *-ALL : le groupe cible doit recevoir la communication de tous les groupes en interaction, *-SOME : le groupe cible doit recevoir la communication d'au moins l'un d'entre eux et *-ONE : le groupe cible doit recevoir la communication d'un seul d'entre eux. En revanche, le Fractal n'a pas cette caractéristique, car il exprime la multiplicité en général.

5.5 Conclusion

Dans ce chapitre, nous avons présenté un modèle de contrôle partiellement centralisé. Tout d'abord, nous avons présenté le langage MSL qui permet de définir et d'instancier des modèles MAPE. Puis, nous avons détaillé le modèle proposé dans ce chapitre, qui se compose d'une seul instance des composants **P** et **K**(ici, Plan et Knowledge sont centralisés) et un nombre arbitraire d'instances du groupe avec des composants **M**, **A** et **E** (ici, Monitor, Analyze et Execute sont décentralisés). Ensuite, nous nous sommes appuyés sur le modèle de composant Fractal pour modéliser le modèle proposé. Étant donné que Fractal prend en charge la notion du partage de composant, nous avons modélisé les composants **P** et **K** comme des composants partagés. Le composant **E** se coordonne avec ses pairs via les liaisons entre les composants **E**. Enfin, nous avons définit le modèle proposé à l'aide de Fractal et MSL afin de donnée une sémantique aux différentes composants du modèle.

Chapitre 6

Une composition hiérarchique des boucles MAPE-K

La volonté trouve, la liberté choisit.
Trouver et choisir, c'est penser

– Victor Hugo

6.1 Introduction

Dans la littérature, de nombreux chercheurs [117, 109, 124] ont choisi de s'appuyer sur le contrôle hiérarchique pour améliorer la fiabilité globale du système. Cependant, le contrôle hiérarchique a été confronté à un ensemble de défis qui affectent les SAAs basé sur plusieurs boucles de contrôle, comme décrit dans la section 3.2. En conséquence, nous présentons HCLs (Hierarchical Control Loops), un modèle de boucles de contrôle hiérarchiques.

Ce chapitre est divisé en trois parties. La première partie concerne la présentation de l'architecture du modèle HCLs. Nous décrivons d'abord la notion de *Composant Adaptatif* qui joue un rôle important dans l'affectation des boucles de contrôle. Puis, nous distinguons deux types de changement (structure et paramètres) que le HCLs peut supporter. Ensuite, nous fournissons une description des types d'adaptation à différent niveau de la hiérarchie, à savoir l'*Adaptation Locale*, l'*Adaptation Régionale* et l'*Adaptation Supérieure*. La deuxième partie concerne l'utilisation de l'apprentissage par renforcement pour soutenir l'auto-adaptation dans le modèle HCLs. Nous présentons d'abord l'intégration de l'apprentissage par renforcement dans le modèles MAPE-K à base de composants. Puis, nous présentons l'exploration d'actions d'adaptation, basée sur le modèle de caractéristiques. Ensuite, nous décrivons l'algorithme étendu de Q-learning pour explorer les stratégies d'adaptation. La dernière partie porte sur la présentation d'une étude de cas sur le service de connexion internet sans fil pour les passagers à l'aéroport. Trois scénarios seront présentés pour vérifier l'efficacité de HCLs.

6.2 HCLs : un modèle de boucles de contrôle hiérarchiques

La conception du modèle HCLs [86] permet d'envisager un système à grande échelle. La figure 6.1 montre l'architecture des SAAs, où plusieurs boucles de contrôle sont conçues avec un ou plusieurs niveaux hiérarchiques. Le contrôle hiérarchique peut être appliqué via plusieurs hiérarchies, chacune ayant son propre effet. Nous basons sur le modèle de composant Fractal pour réaliser cette structuration hiérarchique d'une application.

L'utilisation du Fractal pour construire une application permet d'expliciter l'architecture d'une application, et de favoriser le découplage de ses différentes fonctionnalités. Fractal combine le caractère hiérarchique et dynamique, où : (1) *Dynamique* : il est possible de configurer l'application en cours d'exécution en modifiant son architecture, et donc indirectement son comportement. (2) *Hiérarchique* : une application est vue comme une hiérarchie de composants car Fractal intègre la notion de composant composite (c'est-à-dire, un composant qui contient au moins un sous-composant).

6.2.1 Importance du Composant Adaptatif dans HCLs

L'entité centrale de HCLs est la notion de Composant Adaptatif (CA). SAA est inclus un ensemble de CA. Un composant primitif peut être un composant régulier ou un CA. Le point commun entre ceux-ci est la nature du composant, deux sont des composants primi-

tifs. En revanche, seul le CA est attaché à la boucle de contrôle MAPE-K. Chaque CA a la capacité de s'adapter avant que le composite qui l'utilise puisse s'interfacer avec ses opérations. Les CAs sont principalement utilisés lorsqu'il est nécessaire d'adapter différentes parties du système en fonction de l'objectif global du système. Ce type de composant peut modifier son comportement actuel en fonction de l'état actuel du sous-système géré. Enfin, CA est une caractéristique puissante dans cette étude, car ils nous permettent de créer un comportement autonome.

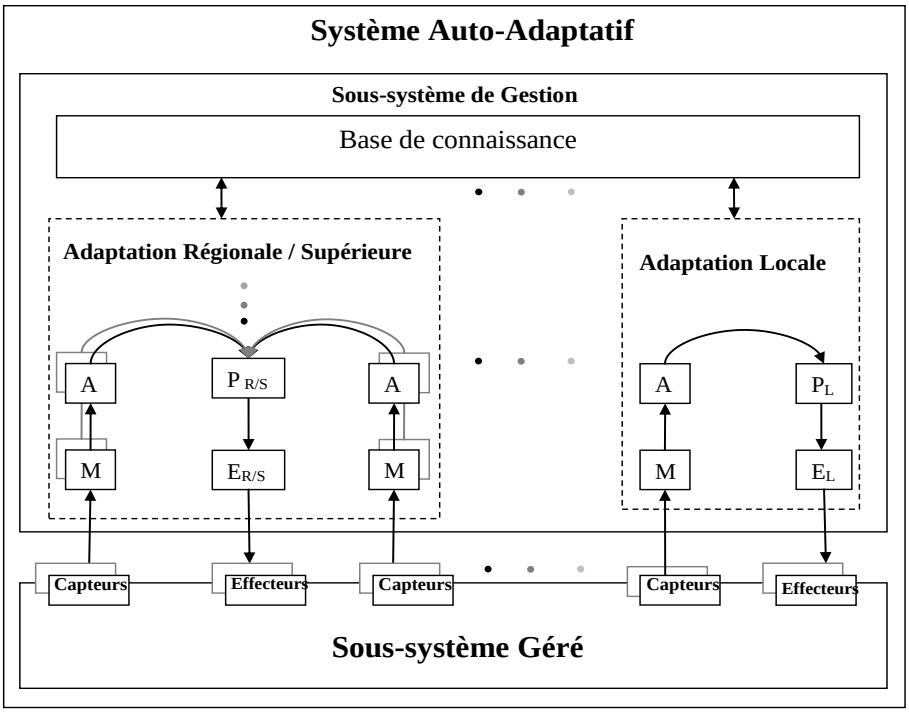


FIGURE 6.1: Modèle de boucles de contrôle hiérarchiques (HCLs).

6.2.2 Type de changement dans HCLs

Le HCLs est capable de prendre en charge l'adaptation des paramètres et de la structure des ressources du sous-système géré. La table 6.1 montre ces deux types de changement.

► Adaptation des paramètres

Un paramètre est une propriété utilisé pour configurer l'état d'un composant sans utiliser les liaisons existantes entre les différents composants. Par conséquent, l'adaptation des paramètres (Attributs) indique une adaptation via le changement des valeurs des paramètres. L'adaptation se fait en ajustant les paramètres en réponse à des changements dans le système lui-même ou dans l'environnement. Comme l'ajout/ la suppression d'un attribut ou la modification des valeurs d'un attribut.

L'adaptation des paramètres est une forme d'adaptation du comportement, c'est-à-dire qu'elle permet d'obtenir un comportement modifié du système après avoir ajusté les paramètres du système. Par exemple, dans un système basé sur des règles qui déterminent le nombre de serveurs nécessaire en fonction de la charge de travail des serveurs, lorsque ces serveurs atteignent un certain taux de charge, il est nécessaire d'ajouter de nouveaux serveurs puis de leur transférer dynamiquement des responsabilités.

Le modèle HCLs repose sur les interface de contrôle fournies par le modèle de composant Fractal, qui ont été présentés dans la section 2.3.1. En particulier, l'adaptation des paramètres peut être réalisée en s'appuyant sur l'interface de contrôle « AttributeController » qui permet d'obtenir et de modifier l'attribut (paramètre) du composant comme indiqué dans la table 6.1.

Type d'adaptation	Adaptation Locale	Adaptation Régionale	Adaptation Supérieure
Type de changement	Paramètre	Structure	
Interface de contrôle (Fractal)	AttributeController	ContentController BindingController	
Action correspondant à la modification	Added_Attribute Removed_Attribute Change_Attribute	Added_Component Removed_Component Bind Unbind	
Condition	L'adaptation ne franchit pas les frontières du CA	L'adaptation ne franchit pas les frontières de la région	L'adaptation franchit les frontières de la région

TABLE 6.1: Aperçu des différents types d'adaptation.

► Adaptation de la structure

L'adaptation de la structure ou l'adaptation de composition (selon la définition de McKinley et al. [76]) permet un échange dynamique de la structure du système au moment de l'exécution. Par conséquent, il est possible de remplacer les composants défectueux afin d'éviter des pertes de performances, d'améliorer les performances en ajoutant de nouveaux composants ou d'adapter le système à de nouvelles conditions.

Comme mentionné dans la section ci-dessus, Fractal fournit plusieurs interfaces de contrôle parmi lesquelles on retrouve l'interface « ContentController » qui permet de changer des composants, de créer un nouveau composite de composants ou de supprimer / ajouter des composants. L'interface de contrôle « BindingController » permet également de modifier le lien entre les composants, c'est-à-dire qu'il permet de créer/supprimer un lien primitif entre deux interfaces de composants comme indiqué dans la table 6.1.

6.2.3 Types d'adaptation dans HCLs

Avant de présenter les types d'adaptation supportés par le modèle HCLs, il convient de noter que la conception de la structure interne de SAA basée sur le modèle HCLs est décrite à l'aide du modèle Fractal, et implémentée à l'aide du framework SAFRAN [33], une extension de Fractal. De plus, les composants **M**, **A**, P_x et E_x ,où **x** $\in$ **L, R, S** (L : Locale, R : Régionale et S : Supérieure) sont en fait des composants Fractal, où nous nous appuyons sur le modèle MAPE-K à base de composants présenté dans le Chapitre 04. Pour clarifier l'architecture du HCLs, nous modélisons le sous-système de gestion comme des composants Fractal et les différentes entités de la boucle de contrôle comme des entités sous forme des cercles. Aussi, le composant *Knowledge* (**K**) n'est pas représentée pour la même raison, qui est la clarté de l'architecture présenté.

Dans cette section, nous fournissons une description détaillée de chaque type d'adaptation pris en charge par le modèle HCLs comme suit :

► Adaptation Locale

Chaque CA du logiciel peut réaliser une Adaptation Locale (AL) à l'aide d'une seule boucle de contrôle, où l'adaptation ne traverse pas les limites de CA. Les boucles de la couche inférieure incluent les entités *Monitor (M), Analyze (A), Local Planning (P_L) et Local Executing (E_L).* L'AL fonctionne à une échelle de temps courte assurant une adaptation dans le temps. L'AL permet l'adaptation des paramètres du CA, c'est-à-dire qu'il doit contrôler et modifier uniquement le paramètre (attribut) comme indiqué dans la table 6.1. La figure 6.2 montre une instanciation de HCLs avec une seule boucle de contrôle. Les entités *M, A,* P_L et E_L ont été rattachées à CA nommée **CA12**

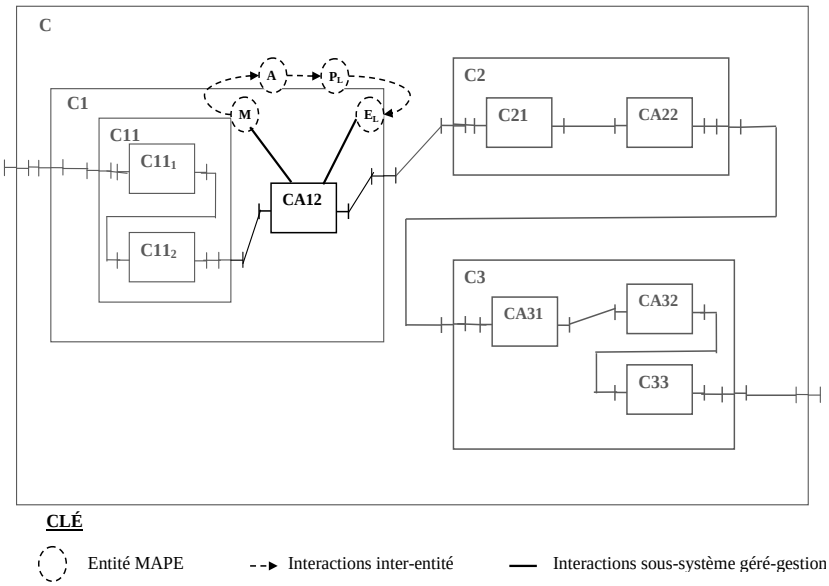


FIGURE 6.2: Aperçu de l'Adaptation Locale.

Le composant adaptatif **CA12** possède également une interface « AttributeController » qui fournit des méthodes pour obtenir (méthode *get()*) et définir (méthode *set(...)*) le(s) attribut(s) du composant **CA12**. HCLs s'appuie sur WildCat pour collecter les valeurs d'attribut surveillées qui caractérisent le comportement de composant **CA12**.

L'entité **M** surveille l'état du composant adaptatif **CA12** via des capteurs fournis par la bibliothèque WildCat. Ici, l'entité **M** représente le composant de surveillance (Monitor) du modèle MAPE-K à base de composants introduit dans le Chapitre 04. Dans le composant *Monitor* se trouve le sous-composant *Event-Collector* où il permet de collecter de nouvelles valeurs pour le(s) attribut(s) surveillé(s). Puis, le sous-composant *Event-Preprocessing* traite ces données par l'application des opérations de normalisation, filtre et agrégation. Ensuite, le sous-composant *Symptoms-Generator* génère les symptômes avec la préservation de type d'événement comme suit : *Type_événement : symptôme*. Enfin, le sous-composant *Update-MSM* met à jour le MSM via la nouvelle valeur de(s) attribut(s) surveillé(s).

L'entité **A** analyse les symptômes envoyés par l'entité **M**. Le sous-composant *Symptom-Receiver* du composant **Analyze** reçoit les symptômes. Puis, le sous-composant *Analyze-Engine* analyse ces symptômes et détermine la situation doit être changée. Ensuite, le sous-composant *Policy-Validator* vérifie les politiques d'adaptation et détermine les demandes de changement. Enfin, le sous-composant *Change-Requests-Sender* envoie les demandes de changement générée au composant de planification locale P_L car le type d'événement dans ce cas concerne l'adaptation de paramètres ou d'attributs.

L'entité P_L est responsable de déterminer le plan de changement local à exécuter. Le sous-composant *Change-Requests-Receiver* du composant *Plan* (indique l'entité P_L) reçoit les demandes de changement envoyés par le composant *Analyze*. Puis, le sous-composant *Policy- Interpreter* gère les politiques d'adaptation afin d'interprèter les demandes de changement. Ensuite, le sous-composant *Plan-Generator* génère un plan de changement locale en réponse aux demandes de changement. Enfin, le sous-composant *Plan-Sender* envoie le plan d'action généré au composant d'exécution locale E_L.

L'entité E_L est responsable de contrôler l'exécution d'un plan de changement. Le sous-composant *Plan-Receiver* du composant *Execute* (indique l'entité E_L) reçoit le plan d'action envoyés par le composant **Plan**. Puis, le sous-composant *Workflow-Engine* convertit le plan en un flux de travail. Ensuite, le sous-composant *Scheduler-Engine* traite les actions reçues et vérifie si l'exécuteur a les droits et les moyens de les exécuter. Enfin, le sous-composant *Scheduler-Engine* détermine l'action à exécuter tout en maintenant l'équilibre du système et contrôle l'exécution d'action à l'aide de l'interface de contrôle « AttributeController ».

Dans l'AL, le plan de changement contient un ensemble d'actions qui ont appliqué la modification nécessaire comme le montre dans la table 6.1, où : *Added_Attribute* : Apparition d'un attribut dans le contexte, *Removed_Attribute* : Disparition d'un attribut dans le contexte, ou/et *Change_Attribute* : Modification de la valeur d'un attribut passant de l'ancienne valeur à la nouvelle valeur.

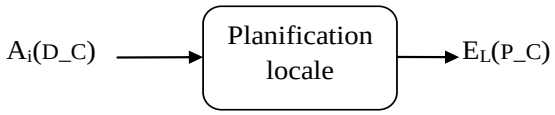


FIGURE 6.3: Entrée/sortie du composant de la planification locale.

La figure 6.3 montre le processus d'entrée/sortie du composant de la planification locale P_L. IL reçoit en entrée les demandes de changement (D_C) depuis un seul composant d'analyse (A_i), où le *i* représente l'identifiant de composant. D'autre part, le processus de la planification locale a comme sortie le plan de changement (P_C) envoyée à composant d'exécution (E_L) qui appartient au même groupe de composants abstraits MA$P_L E_L$.

► Adaptation Régionale

La détermination de la région pour le logiciel est basée sur l'architecture de l'application. Selon la nature hiérarchique du Fractal, le système est considéré comme un assemblage de composants. Le composant composite (CC) du logiciel peut réaliser l'Adaptation Régionale (AR). Le CC est contient un nombre arbitraire de sous-composants primitifs dans le même espace d'adressage, ou un nombre arbitraire de sous-composants primitifs et CC à condition que seul les CC contenant au moins un CA peut être considéré comme une région. L'AR ne traverse pas les frontières de la région. Plusieurs couches intermédiaires sont possibles. La figure 6.4 montre une instanciation de HCLs avec deux boucles de contrôle. Les entités **M** et **A** ont été rattachées à CA nommée **AC31** et **AC32**, tandis que P_R et E_R ont été rattachés à la région (CC) nommée **C3**.

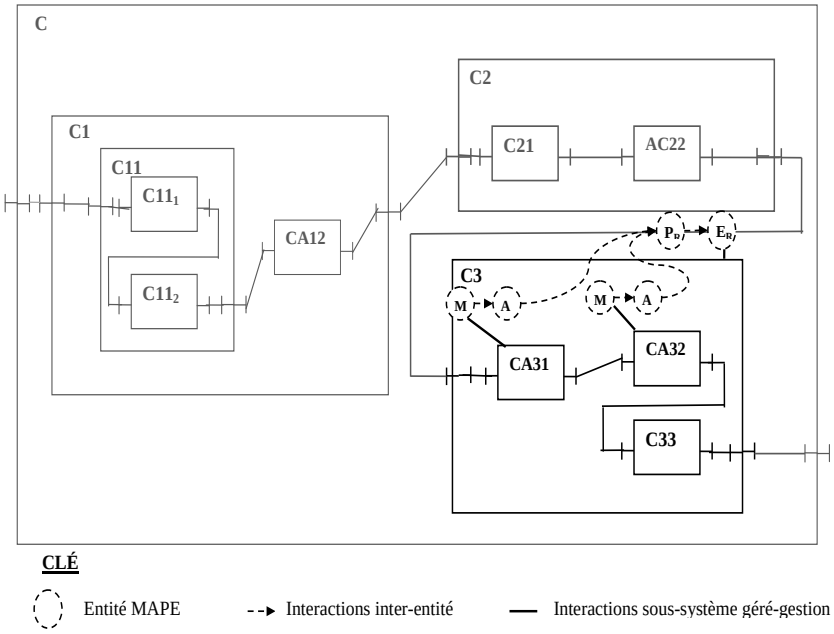


FIGURE 6.4: Aperçu de l'Adaptation Régionale.

Les composants adaptatifs **CA31** et **CA32** possèdent également des interfaces de contrôle nommée : « ContentController » et « BindingController ». HCLs s'appuie sur WildCat pour collecter les données qui caractérisent le comportement des ressources gérées, dans cet exemple les composants **CA31** et **CA32**.

Dans la région, chaque entité **M** surveille les changements du CA via des capteurs fournis par la bibliothèque WildCat. Ici, chaque entité **M** représente une instance de composant *Monitor*. Dans le composant *Monitor* se trouve le sous-composant *Event-Collector* où il permet de collecter les événemets lié au changement structurel tel que l'apparition ou la disparition d'une ressource (composant) dans le contexte, et la connexion ou la déconnexion des interfaces de composants. Puis, le sous-composant *Event-Preprocessing* traite ces données par l'application des opérations de normalisation, filtre et agrégation. Ensuite, le sous-composant *Symptoms-Generator* génère les symptômes avec la préservation de type d'événement comme suit *Type_événement : symptôme*. Enfin, le sous-composant *Update-MSM* met à jour le MSM avec le nouvel état des ressources gérées.

L'entité **A** analyse les symptômes envoyés par l'entité **M**. Le sous-composant *Symptom-Receiver* du composant **Analyze** reçoit les symptômes. Puis, le sous-composant *Analyze-Engine* analyse ces symptômes et détermine la situation doit être changée. Ensuite, le sous-composant *Policy-Validator* vérifie les politiques d'adaptation et détermine les demandes de changement. Enfin, le sous-composant *Change-Requests-Sender* envoie les demandes de changement générée au composant de planification régionale P_R car le type d'événement dans ce cas concerne l'adaptation de la structure comme indiqué dans la table 6.1.

L'entité P_R est responsable de déterminer le plan de changement régional à exécuter. Le sous-composant *Change-Requests-Receiver* du composant *Plan* (indique l'entité P_R) reçoit les demandes de changement envoyés par le composant *Analyze*. Puis, le sous-composant *Policy-Interpreter* gère les politiques d'adaptation afin d'interprèter les demandes de changement. Ensuite, le sous-composant *Plan-Generator* génère un plan de changement régionale en réponse aux demandes de changement. Enfin, le sous-composant *Plan-Sender* envoie le plan d'action généré au composant d'exécution régionale E_R.

L'entité E_R est responsable de contrôler l'exécution d'un plan de changement. Le sous-composant *Plan-Receiver* du composant *Execute* (indique l'entité E_R) reçoit le plan d'action envoyés par le composant **Plan**. Puis, le sous-composant *Workflow-Engine* convertit le plan en un flux de travail. Ensuite, le sous-composant *Scheduler-Engine* traite les actions reçues et vérifie si l'exécuteur a les droits et les moyens de les exécuter. Enfin, le sous-composant *Scheduler-Engine* détermine l'action à exécuter tout en maintenant l'équilibre du système et contrôle l'exécution d'action à l'aide des interfaces de contrôle « ContentController » et « BindingController » en fonction du type d'action à exécuter.

Le plan de changement contient un ensemble d'actions, qui sont : **Added_Component** : Ajouter un nouveau sous-composant au composite (région) **C3**, **Removed_Component** : Retirer un sous-composant du composite **C3**, **Bind** : Etablire une connexion entre les interfaces des sous-composants du composite **C3**, ou/et **Unbind** : Dissocier une connexion entre les interfaces des sous-composants du composite **C3**.

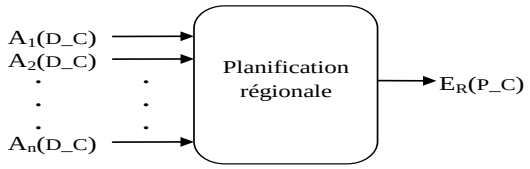


FIGURE 6.5: Entrées/sortie du composant de la planification régionale

La figure 6.5 montre le processus des entrées / sortie du composant de la planification régionale. IL reçoit en entrées les demandes de changement (D_C) depuis un ou plusieurs composants *Analyze* (A_i), où le i représente l'identifiant de composant. D'autre part, le composant de la planification régionale (P_R) a comme sortie le plan de changement (P_C) envoyée à un seul composant d'exécution régionale (E_R) qui appartient au même groupe de composants abstraits MA$P_R E_R$.

► Adaptation Supérieure

L'Adaptation Supérieure (AS) se caractérise par sa capacité à recevoir plusieurs demandes de changement en entrée. L'AS traverse les frontières de la région. Elle permet d'adapter le composant quelle que soit sa position dans la hiérarchie, c'est-à-dire permet de gérer ces actions : ajout/suppression de composants, et connexion/déconnexion entre les composants quelle que soit leur position dans la hiérarchie de l'application. La figure 6.6 montre une instanciation de HCLs avec trois boucles de contrôle. Les entités **M** et **A** ont été rattachées à CA, ici nommée **AC12**, **AC22**, **AC31**, où P_S et E_S ont été rattachés au composite qui est à la base de la création de tous les composants du sous-système.

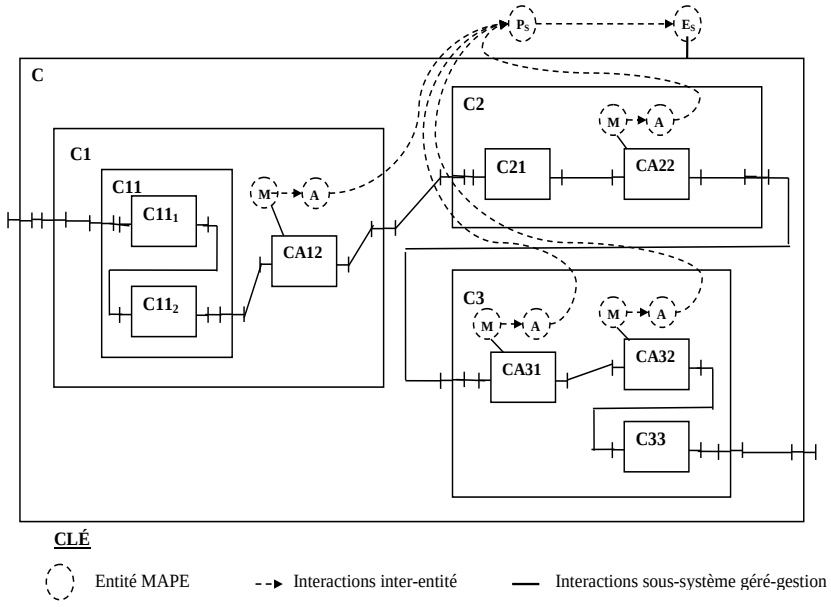


FIGURE 6.6: Aperçu de l'Adaptation Supérieure.

En général, le principe de fonctionnement des entités **M, A,** P_S et E_S est le même que pour le type adaptation régionale. Ici c'est l'entité **A** qui fait la différence entre ces deux types d'adaptation. Selon les demandes de changement soumises par l'entité **A** qui nécessitent des changements qui vont au-delà des frontières de la région, ce qui nécessite l'application d'une AS. L'entité **A** envoie des demandes de changement à l'entité P_S. Par exemple, les demandes de changement de l'entité **A** pour le composant **CA21** nécessitent de créer une liaison entre **CA12** et **CA31**. Ici, l'interface de contrôle « BindingController » du composite **C1** ne peut pas établir cette changement car il se trouve en dehors du composite ou de la région. Alors que l'interface « BindingController » de composite **C** peut établir la connexion entre **CA12** et **CA31**.

Le processus des entrées / sortie du composant de la planification supérieure est similaire au processus de planification régionale, comme illustré à la figure 6.5. IL reçoit en entrées les demandes de changement (D_C) depuis un ou plusieurs composants *Analyze* (A_i), où le *i* représente l'identifiant de composant. D'autre part, le processus de la planification supérieure a comme sortie le plan de changement (P_C) envoyée à un seul composant d'exécution supérieure (E_S) qui appartient au même groupe de composants abstraits MA$P_S E_S$.

6.3 HCLs basé sur l'apprentissage par renforcement

Dans cette section, nous présentons d'abord le modèle de caractéristiques. Puis, nous présentons l'intégration de l'apprentissage par renforcement dans le modèle MAPE-K à base de composants [88]. Ensuite, nous présentons l'adaptation de l'exploration d'action basée sur le modèle de caractéristiques. Enfin, nous décrivons les stratégies d'exploration dans l'algorithme Q-learning.

6.3.1 Modèle de caractéristiques

Le modèle de caractéristiques [7] (Feature Model, en anglais) est une approche de représentation des caractéristiques et des variables d'un produits logiciels. Il fournit une aide efficace au raisonnement pour les phases de développement logiciel, qu'il s'agisse de la spécification des besoins d'une application, de la conception, de la production, de la configuration ou des tests.

Le modèle de caractéristiques permet de lister les caractéristiques sous forme d'arbre hiérarchique, où chaque relation entre le *parent* et leurs *fils* peut être obligatoire, facultative, alternative (XOr) ou à choix multiple (Or). De plus, il existe d'autres contraintes supplémentaires qui spécifient la relation entre différentes caractéristiques telles que les relations d'*inclusion* et d'*exclusion*.

- **Obligatoire (Mandatory).** Lorsque la relation entre une caractéristique et ses sous-caractéristiques est obligatoire, la sous-caractéristique est sélectionnée chaque fois que son parent respectif est sélectionné. Cette relation est également valide en sens inverse, c'est-à-dire que la caractéristique parent est sélectionnée chaque fois que la

sous-caractéristique est sélectionnée.

- ✲ **Facultative (Optional).** Lorsque la relation entre une caractéristique et ses sous-caractéristiques est facultative, la sous-caractéristique est sélectionnée si son parent respective est sélectionné, mais il est possible de sélectionner la caractéristique parent sans sélectionner la sous-caractéristique.
- ✲ **Choix multiple (Or).** La relation à choix multiple entre une caractéristique et ses sous-caractéristiques signifie qu'une ou plusieurs sous-caractéristiques doivent être sélectionnées si la caractéristique parent est sélectionnée.
- ✲ **Choix alternatif (XOr).** Lorsque la relation entre une caractéristique et ses sous-caractéristiques est alternative, chaque fois que la caractéristique parent est sélectionnée, l'une des sous-caractéristiques doit être sélectionnée. À l'inverse, aucune des sous-caractéristiques n'est sélectionnée lorsque la caractéristique parent n'est pas sélectionnée.

En plus des relations entre une caractéristique et ses sous-caractéristiques, le modèle de caractéristiques fournit d'autre type de relations transversales, c'est-à-dire des relations inter-arbres qui spécifient l'incompatibilité entre les caractéristiques.

- ✲ **Inclusion (Require).** La relation d'inclusion « **A** requires **B** » signifie que la sélection de la caractéristique **A** implique la sélection de la caractéristique **B** dans un produit.
- ✲ **Exclusion (Exclude).** La relation d'exclusion « **A** excludes **B** » signifie que les caractéristiques **A** et **B** ne peuvent pas faire partie de la même combinaison.

Ces relations peuvent être représentées sous forme d'expressions logique en utilisant la formule propositionnelle comme montré dans la table 6.2. Où, "**p**" présente la caractéristique parent, et les sous-caractéristiques sont présentés par "**a**" et "**b**".

Relation	**Formule propositionnelle**
Obligatoire	$(p \rightarrow a) \wedge (a \rightarrow p)$
Facultatif	$a \rightarrow p$
Choix multiple (Or)	$p \rightarrow (a_1 \vee ... \vee a_n) \wedge (a_1 \rightarrow p) \wedge ... \wedge (a_n \rightarrow p)$
Choix alternatif (XOr)	$(p \rightarrow alt\ (a_1, ..., a_n)) \wedge (a_1 \rightarrow p) \wedge ... \wedge (a_n \rightarrow p)$
Inclusion	$a \rightarrow b$
Exclusion	$a \rightarrow \neg\ b$

TABLE 6.2: Formule propositionnelle des relations du modèle de caractéristiques [7].

Chen et al. [24] ont exploité le modèle de caractéristiques et l'algorithme évolutionnaire multi-objectif pour améliorer le processus de prise de décision d'adaptation à l'exécution. Où, ils ont utilisé le modèle de caractéristiques comme connaissance du domaine pour guider la recherche et étendre davantage l'algorithme évolutionnaire multi-objectif, offrant une plus grande chance de trouver de meilleures solutions.

Récemment, Metzger et al. [78] ont combiné les modèles de caractéristiques avec l'apprentissage par renforcement en ligne pour explorer l'espace d'adaptation des SAAs. Les auteurs montrent que l'exploitation de la structure hiérarchique et de la différence des modèles de caractéristiques sur différents cycles conduit à une accélération de la convergence du processus d'apprentissage.

En conclusion, les modèles de caractéristiques peuvent être utilisés pour déterminer l'espace d'adaptation des SAAs. Où, chaque action d'adaptation est représentée par une combinaison de caractéristiques.

6.3.2 Intégration de l'apprentissage par renforcement dans le modèle MAPE-K à base de composants

La figure 6.7 montre l'intégration de l'apprentissage par renforcement dans le modèle MAPE-K basé sur les composants. Dans ce cas, nous avons proposé de créer un nouveau composant composite **RL-MAPE-K**, dans lequel nous réutilisons les sous-composants existants dans le modèle MAPE-K à base de composants qui sont : *Monitor*, *Analyse*, *Plan*, *Execute* et *Knowledge*. Puis, nous avons intégré les composants du processus d'apprentissage par renforcement entre les composants d'analyse (Analyze) et de planification (Plan), et modélisé les interactions de ce processus avec la base de connaissance (Knowledge).

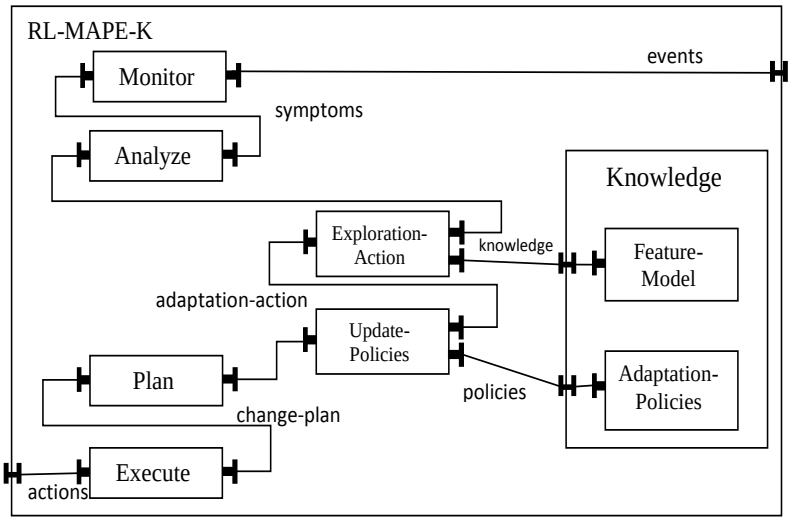


FIGURE 6.7: Intégration de l'apprentissage par renforcement dans le modèle MAPE-K à base de composants.

Comme le montre la figure 4.1, le composant *Knowledge* contient les sous-composants suivants : MSM, Catalog-of-Symptoms, Adaptation-Goals, Adaptation-Policies et Guiding-Strategies. Dans la figure 6.7, nous nous sommes concentrés sur l'ajout d'un nouveau sous-composant *Feature-Model* au composant composite *Knowledge* et sur l'identification

du sous-composant *Adaptation-Policies* car ces deux sous-composants concernent l'intégration de l'apprentissage par renforcement dans la boucle MAPE-K à base de composants. En ce qui concerne le reste des sous-composants, il reste présent avec le même principe de fonctionnement comme décrit dans le Chapitre 04.

Les composants « **Exploration-Action** » et « **Update-Policies** » sont ajoutés au composite **MAPE-K**. Où, le sous-composant « **Exploration-Action** » est responsable de l'exploration de l'action d'adaptation à l'aide de « **Feature-Model** » et des connaissances reçues du composant **Analyze**. Le composant « **Update-Policies** » est responsable de la mise à jour des politiques incluses dans « **Adaptation-Policies** » à l'aide de l'action reçue du composant « **Exploration-Action** ».

6.3.3 Adaptation de l'exploration d'action basée sur le modèle de caractéristiques

En raison de la nature évolutive des SAAs, l'espace d'adaptation est toujours dans un état dynamique. Les stratégies d'exploration traversent le modèle de caractéristiques pour structurer l'espace d'adaptation du SAA. Cela aide à explorer la prochaine action d'adaptation. L'action d'adaptation est représentée par une combinaison de caractéristiques valide qui spécifie la configuration d'exécution cible du système.

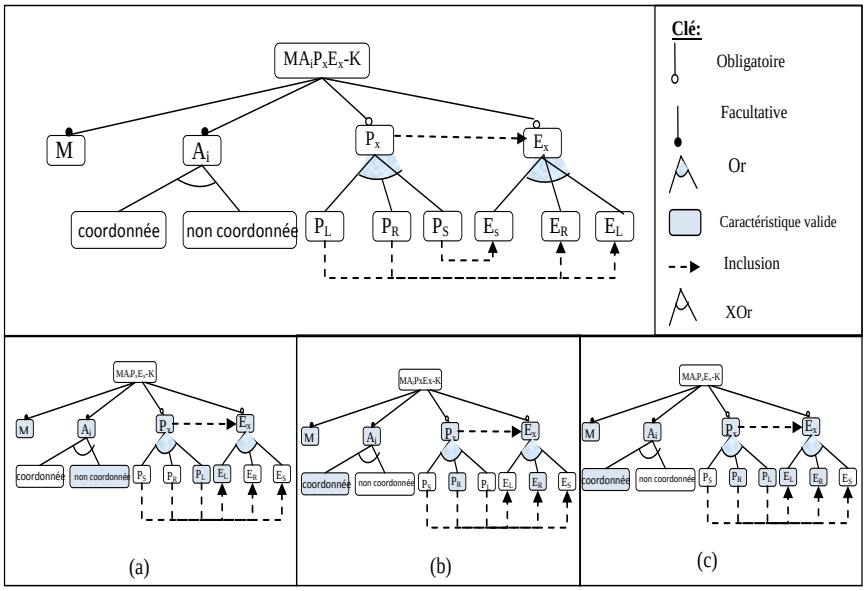


FIGURE 6.8: En haut : modèle de caractéristiques MAPE-K, en bas : exemples illustratifs

Ce travail est basé sur l'exploration de stratégies pour déterminer les configurations possibles sur les gestionnaires autonomes. Ces stratégies portent sur l'adaptation de la coordination entre les différents composants *Analyze* (A). De plus, la sélection du type d'adap-

tation approprié (AL, AR ou AS) est prise en considération, pour assurer une adaptation correcte sans conflit et sans coût élevé.

La figure 6.8 dans la partie supérieure présente le modèle de caractéristiques de la boucle de contrôle MAPE-K, où i $\in \mathbb{N}$: représente le nombre de composants *Analyze* (A) existant dans le système, et x $\in$ {L, R, S} présente le type d'adaptation à appliquer.

La caractéristique *M* (c'est-à-dire Monitor) est obligatoire et est toujours sélectionnée. La caractéristique A_i (c'est-à-dire Analyze) est également toujours sélectionnée et comporte deux sous-caractéristiques (coordonnée et non coordonnée), où vous devez sélectionner l'une de ces deux caractéristiques. La caractéristique « coordonnée » signifie que le composant *Analyze* A_i doit être coordonné avec d'autre(s) composant(s) *Analyze*, et « non coordonnée » signifie que le composant A_i n'a pas besoin d'être coordonné avec un autre composant *Analyze*.

Les caractéristiques P_x et E_x sont optionnelles ou facultatives car parfois le composant Analyze *A* après leur analyse constate que l'état détecté ne nécessite pas d'adaptation. Cependant, la sélection de la caractéristique P_x nécessite de sélectionner la caractéristique E_x. Chaque caractéristique P_x et E_x a trois sous-caractéristiques « P_L, P_R, P_S » et « E_L, E_R, E_S », respectivement. La relation « Inclusion » signifie ici l'obligation de sélection des caractéristiques *P* et *E* de même type. Par exemple, la sélection de P_L implique la sélection de E_L (voir figure 6.8 (a)).

La figure 6.8 en bas montre trois exemples (a, b, c) d'exploration de stratégies, où la combinaison de caractéristiques colorées en gris donne une action d'adaptation. L'exploration commence toujours par la racine en raison de l'existence de plusieurs boucles de contrôle MAPE-K.

6.3.4 Stratégies d'exploration dans l'algorithme Q-learning

L'algorithme proposé « Exploration Strategies into Q-learning » représente l'algorithme étendu de Q-learning. Au départ, il faut initialiser l'état du système (voir ligne 10) avec une action d'adaptation a_0. Où, a_0 est l'action d'adaptation déterminée dans la phase de conception en fonction des besoins et du point de vue du concepteur.

Cette étape peut être une référence pour mettre à jour les connaissances sur la base de la meilleure action suivante possible. Après l'initialisation, la fonction **getExploratedAction** renvoie la prochaine action d'adaptation (voir ligne 13) à plusieurs reprises. Après la sélection de l'action suivante, la procédure **Q-learningMain** exécute cette action (ligne 14), puis observe la récompense (ligne 15). Enfin, il permet de mettre à jour la fonction d'apprentissage **Q** et l'état du système **S** (ligne 16-17). Les stratégies d'exploration sont intégrées dans l'algorithme Q-learning dans la fonction **getExploratedAction** qui sélectionne la prochaine action adaptative du gestionnaire autonome tout en faisant une comparaison entre l'exploration et l'exploitation. Nous nous sommes basés sur le modèle de caractéristiques, et en particulier sur les relations entre les différentes caractéristiques.

Le modèle de caractéristiques est caractérisé par la racine de l'arbre (*parent*) et un ensemble de feuilles (*fils*). Chaque racine a un ensemble de feuilles. La fonction **getExplora-**

Algorithm 1: Exploration Strategies into Q-learning Algorithm

```
S = {s0, s1, s2, ..., st};          ▷ Espace d'état, où t représente un pas de temps discret.
A = {a0, a1, a2, ..., an};          ▷ Espace d'action d'adaptation, où n ∈ ℕ
M                                   ▷ Modèle de caractéristiques
Q : S × A → R;                      ▷ Fonction d'apprentissage
T : S × A → S;                      ▷ Fonction de transition d'état
R : S × A → R;                      ▷ Fonction de récompense
α ∈ [0,1];                          ▷ Taux d'apprentissage, généralement α =0,1
γ ∈ [0,1];                          ▷ Facteur de réduction
Procedure Q-LearningMain(M, S, A,α,γ) :
    st = Intialize_FM(M,a0, s0);                    // Initialiser l'espace
    repeat
        root ← getRoot(M);
        ai=getExploratedAction(M,root);             // Sélection de l'action
         suivante
        st+1 ← T(st, ai);                           // Recevoir le nouvel état
        Rt+1 ← R(st, ai);                           // Recevoir la récompense
        Q(st,ai) ← Q(st,ai) α . [ Rt+1 + γ . maxa Q(st+1,ai) - Q(st,ai)];   // Mettre à
         jour le facteur Q associé à l'état st et à l'action ai
        st ← st+1 ;                                 // Mettre à jour l'espace
    until last time step
Function Intialize_FM(a, s) :
    for i := 1 to N do
                    // N représente le nombre de boucles de contrôle
        set ⟨ Feature ⟩ a ← { M, Ai, coordinate, PX, PL, PR, PS, EX, EL, ER, ES };
         // Sélection de l'action initiale
        st+1 ← T(st,a);                             // Recevoir le nouvel état
        st ← st+1 ;
    return St
Function getExploratedAction(M,root) :
    set ⟨ Feature ⟩ children ← getDescands(root);
    for each child of the set of children do
        if relation(root, child) = "Mandatory" then
            a ← addFeature(a, child);
        if relation(root, child) = "Optional" then
            if random() = 1 then
                a ← addFeature(a, child);
        if requireRelation(M, child) ≠ " ∅ " and (child ∉ a) then
            a ← addFeature(a, child);
        if relation(root, child) = "Or" and root ∈ a and child ∉ a then
            child ← randomSeletChild(root);
            a ← addFeature(a, child);
        if getDescands(child) ≠ ∅ then
            a=getExploratedAction(M,child);
    return a
```

tedAction vérifie la relation entre la *racine* et la *feuille*. S'il est *Mandatory* (Obligatoire), la caractéristique (feuille) sera ajoutée à l'action. Si la relation est *Optional* (Facultative), la décision de choisir est aléatoire. Dans ce cas, si *random()=1*, la caractéristique (feuille) sera ajoutée à l'action, sinon cette caractéristique ne sera pas ajoutée. La fonction **requireRelation** permet de vérifier si cette sous-caractéristique est demandé par une autre sous-caractéristique. Si tel est le cas, il sera ajouté à l'action. Si la relation entre la racine et la feuille est *Or* (Ou), la fonction **randomSelectChild** renvoie une ou plusieurs sous-caractéristiques. Enfin, la fonction **getExploratedAction** s'arrête lorsque le dernier fils est atteint.

La table 6.3 représente l'espace d'action d'adaptation, où x signifie que la caractéristique est activée. Les stratégies d'exploration dans les algorithmes de Q-Learning permettent d'explorer des stratégies et de déterminer la fonction de qualité d'apprentissage **Q** afin de sélectionner les configurations possibles. A travers ce tableau, nous concluons qu'il existe 14 configurations possibles pour former différentes boucles de contrôle. Ce qui distingue le modèle HCLs basée sur l'apprentissage par renforcement est que le système peut contenir plus d'une configuration. Où chaque sous-système peut utiliser l'un des configurations représentées dans la table 6.3.

N	M	A	Px			Ex		
			PL	PR	PS	EL	ER	ES
1	x	non coordonné	x			x		
2	x	coordonné	x			x		
3	x	non coordonné	x	x		x	x	
4	x	coordonné	x	x		x	x	
5	x	non coordonné	x	x	x	x	x	x
6	x	coordonné	x	x	x	x	x	x
7	x	non coordonné		x	x		x	x
8	x	coordonné		x	x		x	x
9	x	non coordonné			x			x
10	x	coordonné			x			x
11	x	non coordonné		x			x	
12	x	coordonné		x			x	
13	x	non coordonné	x		x	x		x
14	x	coordonné	x		x	x		x

TABLE 6.3: Espace d'action d'adaptation.

6.4 Étude de cas

Dans cette section, nous présentons l'étude de cas et décrivons trois scénarios pour valider notre proposition. L'étude de cas concerne le service de connexion Internet sans fil aux passagers dans un aéroport [4, 103]. L'objectif de ce système est de fournir un accès

Internet gratuit aux passagers. Hannousse [55] présente l'architecture des composants de cette application. Selon Hannousse [55], il existe trois types d'utilisateurs qui sont : (1) les passagers utilisent le nombre de leurs billets d'avion valides pour se connecter au réseau, (2) les passagers utilisent le nombre de leurs billets d'avion fréquents pour se connecter au réseau avec un temps d'accès du temps maximum prévu par les billets d'avion de l'utilisateur, et (3) les utilisateurs réguliers qui n'ont pas de billets ou qui ont déjà terminé le temps prévu par les billets. Dans ce cas, l'utilisateur peut créer un compte temporaire et acheter du temps de connexion.

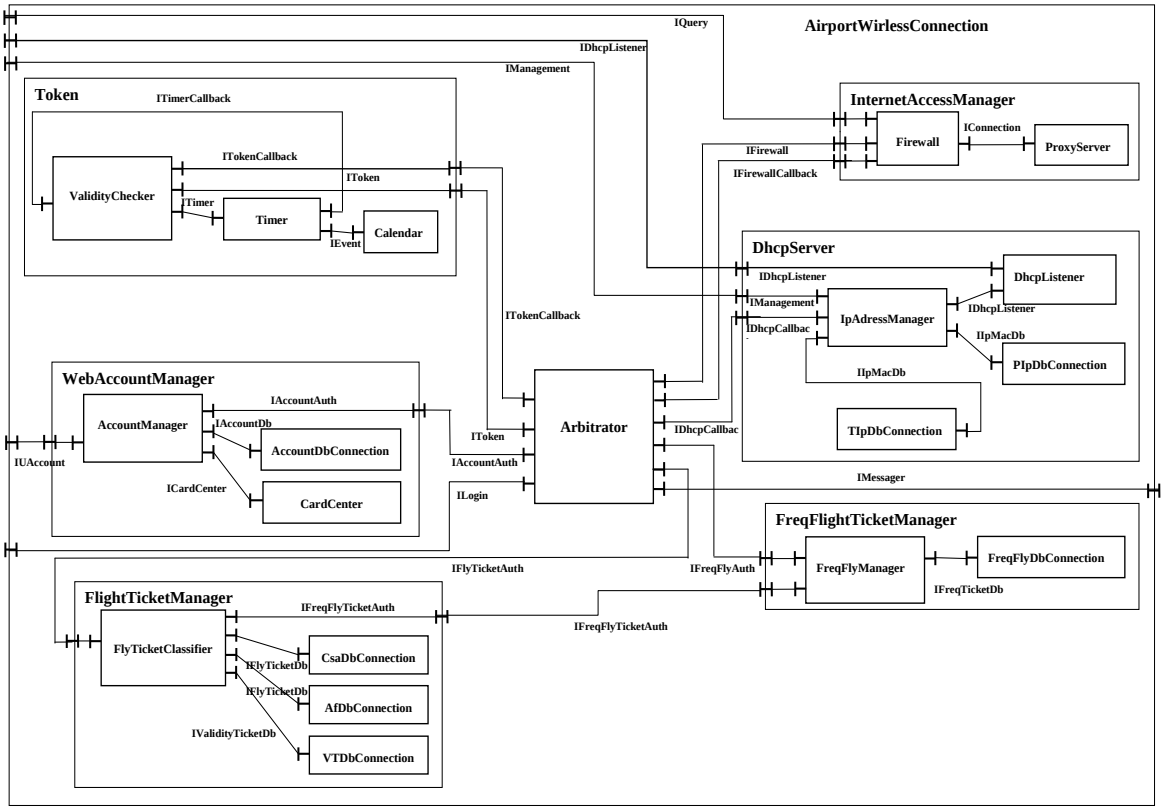


FIGURE 6.9: Application d'accès sans fil à l'aéroport basée sur le Fractal.

- Le composant « Arbitrator » est le gestionnaire du système d'accès à la connexion internet sans fil de l'aéroport.
- Le composant « InternetAccessManager » est responsable de l'activation de la communication pour l'utilisateur.
- Le composant « DhcpServer » est responsable de la connexion au système sans fil de l'aéroport.
- Le composant « Token » » est responsable de la modélisation d'une session utilisateur.
- Le composant « WebAccountManager » se charge de créer un compte temporaire et d'acheter du temps d'accès.
- Le composant « FlightTicketManager » est responsable de la gestion des billets d'avion.
- Le composant « FreqFlightTicketManager » est responsable de la gestion des billets d'avion fréquents.

6.4.1 Implémentation

Dans cette section, nous ajoutons quelques modifications à l'architecture du système. La figure 6.9 montre l'application d'accès Internet sans fil à l'aéroport basée sur Fractal. L'objectif est d'introduire trois scénarios nécessitant une auto-adaptation afin d'illustrer la validité de notre approche.

► Scénario 1 : Adaptation Locale

Dans cette architecture, nous avons ajouté un composant appelé « Calendar ». Ce composant est chargé d'indiquer tous les événements de l'année. Qu'il s'agisse de fêtes nationales, internationales ou religieuses. Le composant « Timer » utilise l'interface fournie "*IEvent*" du composant « Calendar » pour recevoir des événements (voir la Figure 6.9). Le composant « ValidityChecker » est chargé de démarrer le composant « Timer » de la session et de recevoir des informations du composant « Timer » sur le temps de session lorsqu'il s'est écoulé. D'autre part, le composant « ValidityChecker » informe à son tour le composant « Arbitrator ». Les jours fériés, le système propose d'offrir un temps bonus aux utilisateurs selon leur type. Par exemple, 15 minutes pour les passagers avec des billets d'avion fréquents, 10 minutes pour les passagers avec des billets d'avion valides et 5 minutes pour les utilisateurs réguliers. Le composant « Timer » possède également une interface de contrôle « AttributeController » de type « TBonusAttribute », qui fournit deux méthodes pour obtenir et définir l'attribut « TBonus » du composant « Timer ».

Le composant « Timer » est un CA. Lorsque le composant « Timer » reçoit un événement, le composant **M** collecte les événements et les prétraite. L'implémentation du composant **M** doit inclure l'opération de surveillance à l'aide de WildCat. Le composant **A** vérifie les politiques d'adaptation et génère des demandes de changement. Ensuite, les composants « Exploration-Action » et « Update-Policies » appliquent l'algorithme Q-learning basé sur les stratégies d'exploration afin de trouver à travers les expériences la fonction de qualité d'apprentissage **Q**. Après avoir trouvé les valeurs **Q**, le système sélectionne l'action qui donne la plus grande utilité attendue. Dans ce cas, c'est le composant P_L qui les demandes de changement et détermine le plan de changement local. Le composant P_L vérifie le type de client précédemment enregistré pour décider du temps bonus à offrir. Enfin, le composant E_L réinitialise la minuterie pour le temps bonus correspondant (exécutant l'action *adaptTimer*). La figure 6.10 montre l'auto-adaptation de la minuterie pour différents types d'utilisateurs.

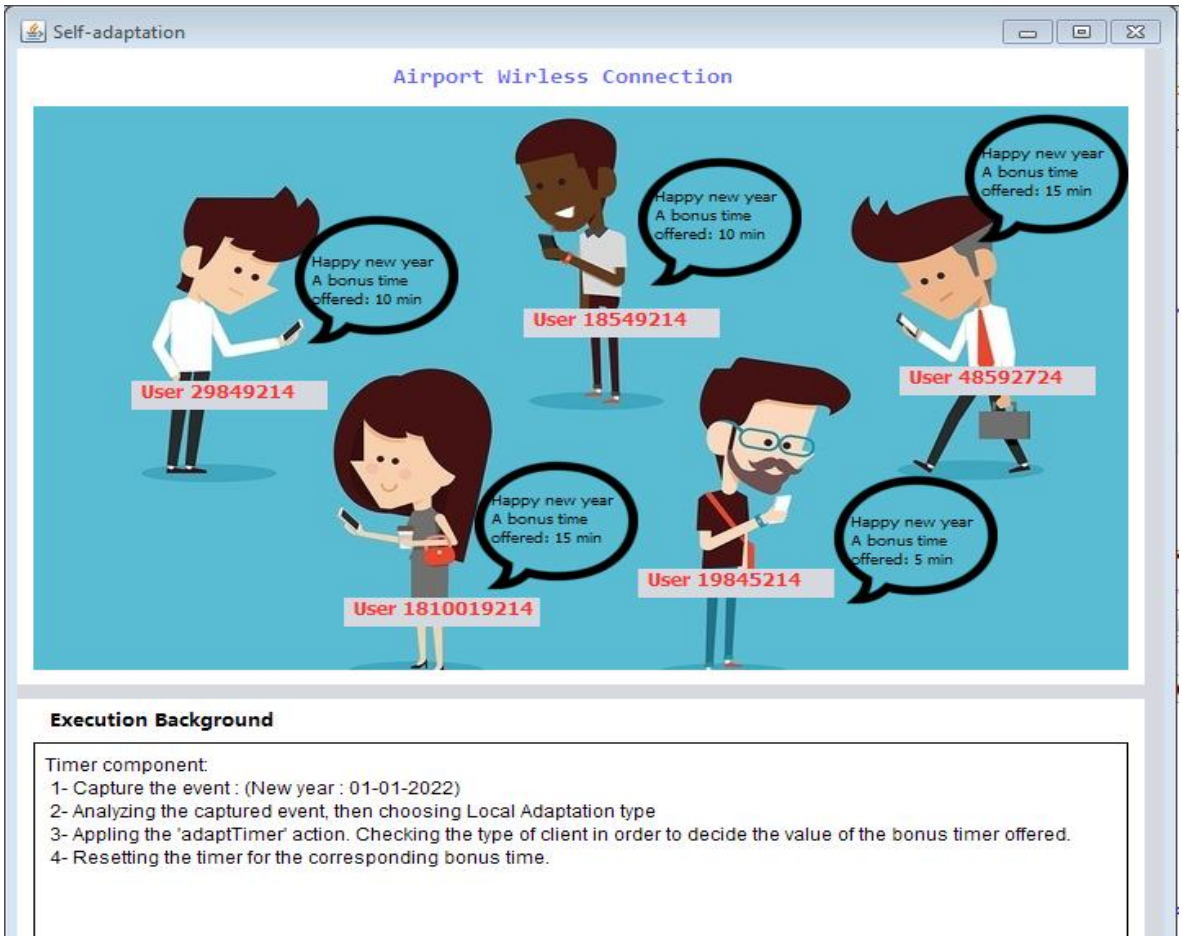


FIGURE 6.10: Scénario d'adaptation locale.

► Scénario 2 : Adaptation Régionale

Le système de connexion Internet sans fil de l'aéroport offre l'avantage de protéger les mineurs (âge <= 19 ans) des dangers d'Internet. Le composant « Firewall » est un CA et fonctionne pour bloquer les connexions Internet non autorisées. D'autre part, il envoie les adresses IP activées au composant « ProxyServer ». Une fois les utilisateurs connectés, il envoie des requêtes au composant « InternetAccessManager » (utilisez l'interface *IQuery*). Le composant « InternetAccessManager » transmet les requêtes des utilisateurs à composant « Arbitrator ». Pour accéder aux informations de l'utilisateur, par exemple l'âge de l'utilisateur, le composant « Firewall » envoie une requête au composant « Arbitrator », en utilisant l'interface *IFirewall*. Le composant « Arbitrator » envoie une requête aux composants « FlightTicketManager », « FreqFlightTicketManager » et « WebAccountManager » en utilisant respectivement les interfaces *IFlyTicketAuth*, *IFreqFlyAuth* et *IAccoutAuth*. Le composant « Firewall » reçoit les informations utilisateur envoyées par le composant « Arbitrator » à l'aide de l'interface *IFirewallCallback*. Le composant « Filter » est facultatif et n'apparaît que lorsque l'utilisateur est mineur. Le composant « Firewall » envoie la connexion IP demandée au composant « Filter » afin de vérifier si la connexion IP de-

mandée est dans la liste noire ou non. Notez que le composant « InternetAccessManager » contient deux sous-composants : « Firewall » et « ProxyServer » avant l'application de l'auto-adaptation.

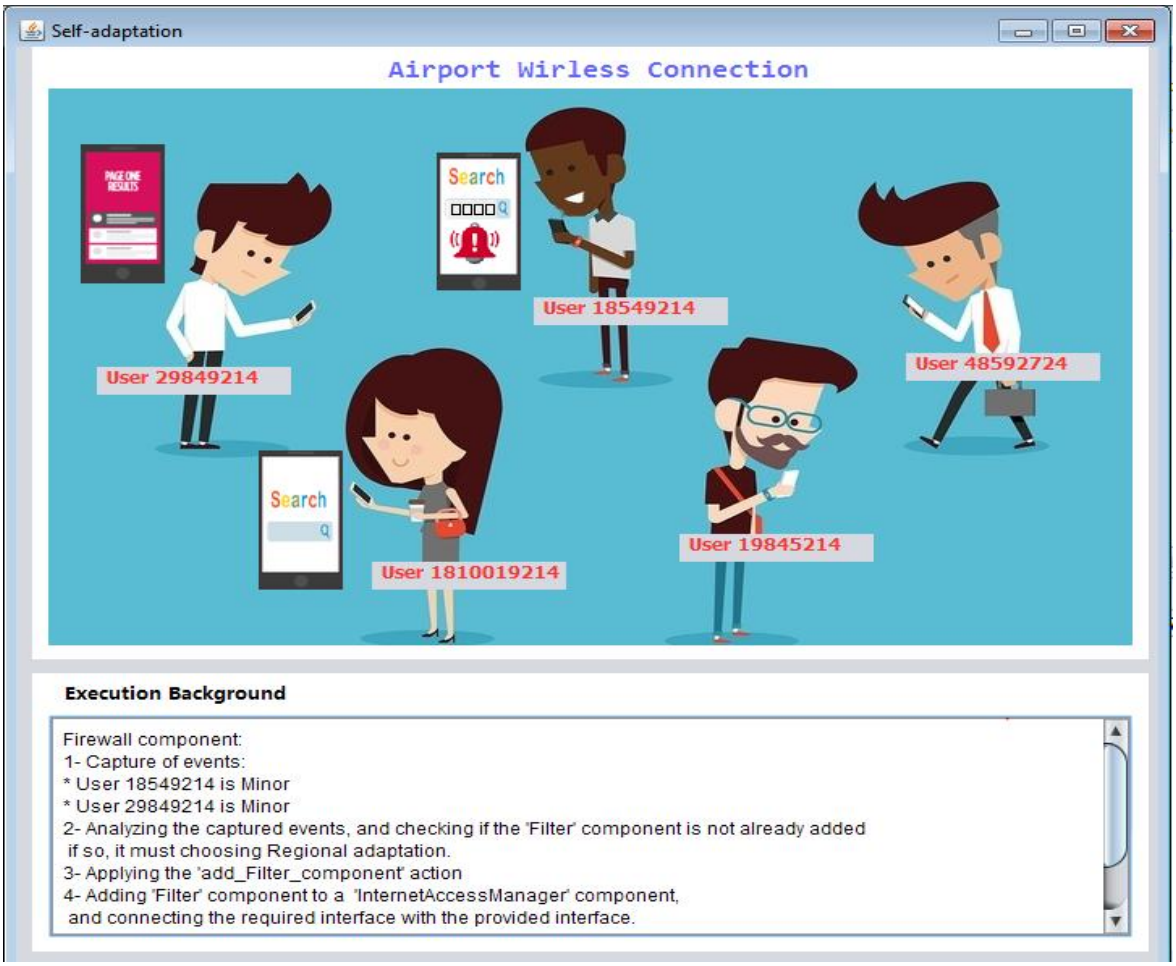


FIGURE 6.11: Scénario d'adaptation régionale.

Lorsque le composant « Firewall » reçoit une requête sur l'interface *IQuery*, le composant **M** collecte les données capturées à l'aide de WildCat. De plus, il prétraite et génère des symptômes. Le composant **A** reçoit les symptômes et les analyse. Puis, il vérifie si le composant « Filter » n'est pas déjà ajouté, si c'est le cas, il doit ajouter le composant « Filter » dans le composite « InternetAccessManager » (voir la figure 6.12). Ensuite, les composants « Exploration-Action » et « « Update-Policies » appliquent l'algorithme Q-learning basé sur les stratégies d'exploration afin de trouver à travers les expériences la fonction de qualité d'apprentissage **Q**. Après avoir trouvé les valeurs **Q**, le système sélectionne l'action qui donne la plus grande utilité attendue. Dans ce cas, c'est le composant P_R qui reçoit les demandes de changement et détermine le plan de changement régional qui contient l'action d'adaptation *add_Filter_component* :

— **add_component**(InternetAccessManager, Filter). Cette action permet d'ajouter un nouveau sous-composant « Filter » à un composite « InternetAccessManager ».

— **bind**(IFilter, IFilter). L'action **bind** permet de connecter l'interface requise avec l'interface fournie.

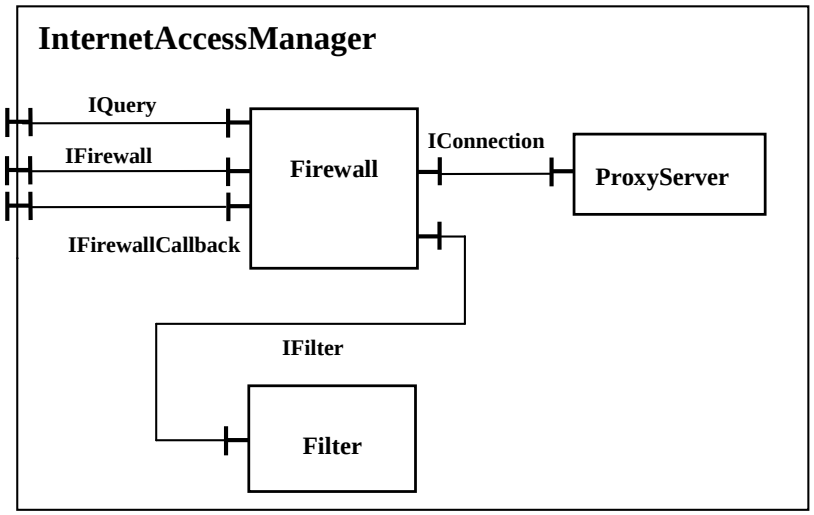


FIGURE 6.12: Composite InternetAccessManager basé sur le Fractal après l'application d'adaptation.

Enfin, le composant E_R contrôle l'exécution de l'action adaptative *add_Filter_component*. La figure 6.11 montre l'auto-adaptation de la structure du composite « InternetAccessManager ».

► Scénario 3 : Adaptation Supérieure

Le composant « CustomToken » est facultatif et n'apparaît que lorsque les utilisateurs réguliers créent un compte. Le composant « AccountManager » est un CA, et il fonctionne pour mettre à jour le temps de connexion autorisé de l'utilisateur dans le « WebAccountManager ». Les utilisateurs réguliers peuvent créer un compte temporaire (Géré par le composant « AccountManager »), et acheter du temps d'accès en utilisant l'interface *IUAccount* du composant « AirportWirelessConnection ». Le composant « AccountDbConnection » est chargé de gérer le temps consommé. Notez que le composite « Token » contient trois sous-composants : « ValidityChecker », « Timer » et « Calendar » avant l'application de l'auto-adaptation.

Lorsque le composant « AccountManager » reçoit une demande de création de compte temporaire, le composant **M** collecte les données capturées à l'aide de WildCat. De plus, il prétraite et génère des symptômes. Le composant **A** reçoit les symptômes et les analyse. Il vérifie si le composant « CustomToken » n'est pas déjà ajouté, si c'est le cas, il doit ajouter le composant « CustomToken » dans la région « WebAccountManager ». Ensuite, les composants « Exploration-Action » et « Update-Policies » appliquent l'algorithme Q-learning

basé sur les stratégies d'exploration afin de trouver à travers les expériences la fonction de qualité d'apprentissage **Q**. Après avoir trouvé les valeurs **Q**, le système sélectionne l'action qui donne la plus grande utilité attendue. Dans ce cas, c'est le composant P_S qui reçoit les demandes de changement et détermine le plan de changement supérieur qui contient l'action d'adaptation *add_CustomToken_ component.*

FIGURE 6.13: Scénario d'adaptation supérieure.

- **add_component**(Token, CustomToken) : Cette action permet d'ajouter un nouveau sous-composant « CustomToken » à un composite « Token ».
- **bind**(ICustomCallback, ICustomCallback) **bind**(IAccount, IAccount) : L'action **bind** permet de connecter l'interface requise avec l'interface fournie.

Enfin, le composant E_S contrôle l'exécution de l'action adaptative *add_CustomToken_ component.* La figure 6.13 montre l'auto-adaptation de la structure du composite « Airport-WirlessConnection », et plus précisément, des composites « Token » et « WebAccount-

Manager ». La figure 6.14 montre l'architecture Fractal des composites « Token » et « WebAccountManager » après l'application de l'adaptation.

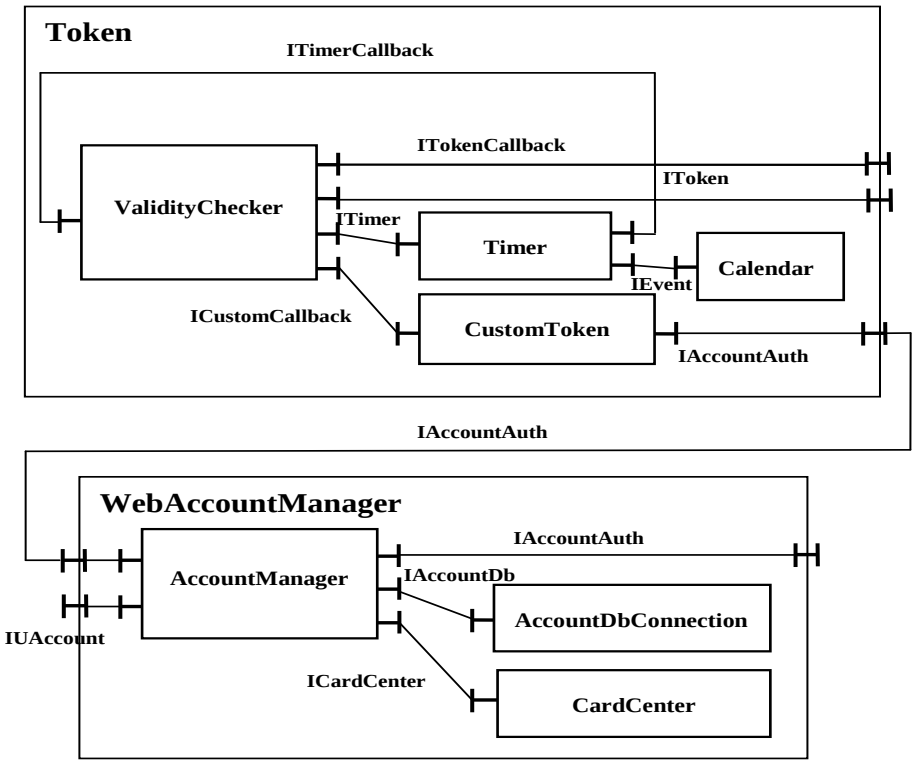


FIGURE 6.14: Composites Token et WebAccountManager basés sur le Fractal après l'application d'adaptation.

6.4.2 Résultats

Après avoir appliqué un ensemble de simulations au modèle proposé. Dans cette section, nous présentons nos résultats.Nous commençons par le diagramme circulaire qui illustre le taux d'application de chaque type d'adaptation comme le montre dans la figure 6.15. On note que le type AL occupe 50% car la majorité des auto- adaptations appliquées à un SAA sont des adaptations de paramètres. Cet état est le plus souvent utilisé pour adapter l'état d'un sous-système sans utiliser les connexions entre les composants qu'il contient. L'adaptation des paramètres est une forme d'adaptation comportementale car elle permet d'obtenir un comportement modifié du système après avoir ajusté les paramètres du système. Le type d'adaptation AR occupe 38%, car l'adaptation de la structure s'appliquent davantage aux composants situés dans le même espace d'adressage que l'AS, car les composants contenus dans la même région sont fortement connectés les uns aux autres. De plus, il existe plusieurs niveaux d'application d'AR grâce à la hiérarchie des applications.

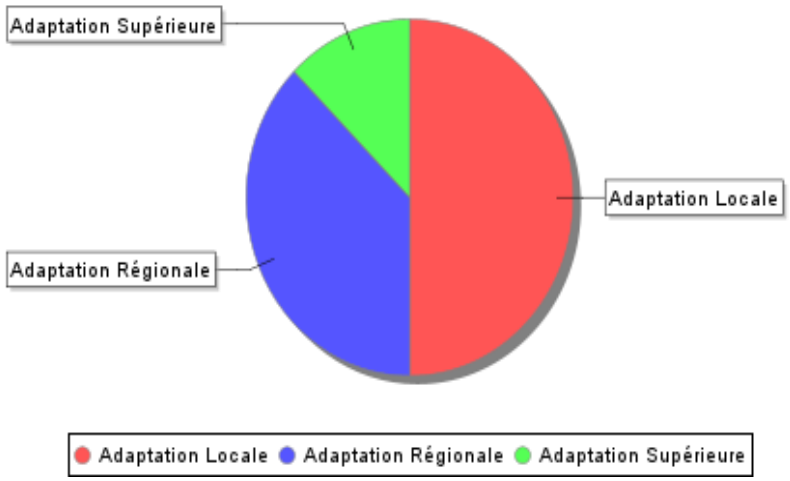


FIGURE 6.15: Diagramme circulaire du taux d'application de chaque type d'adaptation.

La courbe de la figure 6.16 montre l'application de HCLs avec et sans l'apprentissage par renforcement. Dans le cas de l'application de HCLS sans l'intégration de l'apprentissage par renforcement, le système est basé sur un seul modèle MAPE proposé dans la phase de conception. Pour l'application de HCLs avec intégration de l'apprentissage par renforcement, le système peut être basé sur plusieurs modèles de configuration MAPE. Alors que l'espace d'adaptation est toujours dans un état dynamique, c'est-à-dire que chaque sous-système peut avoir un modèle de configuration MAPE. Le nombre de modèle MAPE est susceptible de changer à chaque itération en raison de la nature évolutive des SAAs.

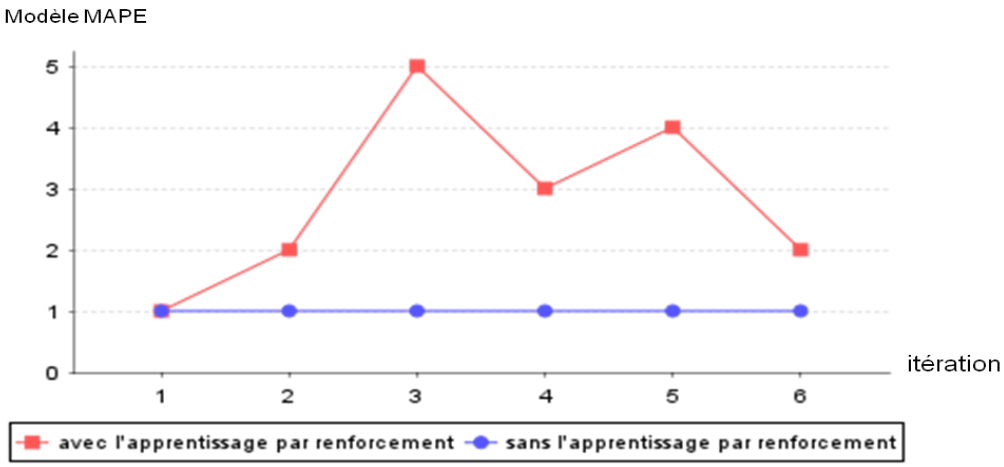


FIGURE 6.16: Courbe de comparaison pour l'application de HCLs avec et sans apprentissage par renforcement.

D'après les résultats analysés dans cette thèse et notre expérience à SAFRAN, nous pouvons dire que HCLs est une proposition prometteuse pour soutenir le contrôle hiérar-

chiques dans les SAAs. De plus, il démontre également la faisabilité de l'intégration de l'apprentissage par renforcement en ligne, utilisant le modèle de caractéristiques pour définir un espace d'adaptation.

6.5 Conclusion

Dans ce chapitre, nous avons présenté HCLs, une proposition architecturale qui prendre en charge la composition hiérarchique des entités MAPE dans les SAAs. HCLs prend en charge l'adaptation de la structure et des paramètres à trois couches, qui sont : l'Adaptation Locale, l'Adaptation Régionale et l'Adaptation Supérieure. HCLs s'appuyé sur des stratégies d'exploration pour l'apprentissage par renforcement en ligne, utilisant le modèle de caractéristiques pour définir un espace d'adaptation. Le modèle HCLs a été évaluées dans un domaine de connexion Internet sans fil pour les passagers à l'aéroport. Les résultats de l'évaluation démontrent la faisabilité et la validité de HCLs utilisant SAFRAN. Cette évaluation a présenté les avantages suivants : (i) un modèle qui permet de réaliser une adaptation globale sans conflit, (ii) facilite la séparation des préoccupations et la reconfiguration à l'exécution grâce à SAFRAN, et (iii) une composition sans coordination complexe.

Conclusion générale et perspectives

Dans ce chapitre, nous revenons sur les considérations générales du travail présenté dans cette thèse et présentons les contributions de cette étude. Nous identifions également les domaines de recherche sur lesquels nous travaillerons à l'avenir.

Conclusion

Dans cette thèse, nous nous sommes concentrés sur la partie contrôle autonome des SAAs, et en particulier sur la boucle de contrôle MAPE-K (Monitor, Analyze, Plan, Execute et Knowledge base). La structure de cette boucle est très basique et même les conceptions étendues de cette boucle sont très génériques. De plus, les conceptions existantes n'utilisent pas de modèles de conception bien établis. Par conséquent, nous avons trouvé nécessaire d'améliorer cette boucle qui facilitera le processus de conception des SAAs. Dans un premier temps, nous avons proposé un modèle de la boucle de contrôle basé sur le modèle de composant Fractal grâce à certaines caractéristiques importantes que le Fractal fournit, dont les plus importantes sont le partage des composants et la hiérarchie des composants. Le modèle MAPE-K à base de composant Fractal a montré le reflet de ces caractéristiques sur la flexibilité de la boucle de contrôle.

Dans le cas où le système est complexe et à grande échelle, plusieurs boucles de contrôle peuvent être nécessaires pour gérer l'adaptation du système. Les chercheurs proposent un ensemble de modèles pour la composition des boucles de contrôle avec différents degrés de décentralisation. Dans un seconde lieu, nous avons proposé une nouvelle combinaison possible répondant à certaines exigences. Graçe à Fractal, nous avons modélisé le composant centralisé *Plan* comme un composant partagé pour éviter les conflits de décision. Les composants *Monitor, Analyze* et *Execute* sont décentralisés. De plus, le composant *Execute* se coordonne avec ses pairs pour s'assurer que l'exécution d'adaptation est synchronisée. En raison de la nature évolutive des SAAs, il existe d'autres défis affectent le fonctionnement des SAAs, tels que : la distribution des actions d'adaptation peut imposer un surcoût de communication important, le composant Plan représente un point de défaillance unique et une surcharge de communication importante est effectué sur le composant Plan. Par conséquent, nous avons amélioré ce modèle en passant du contrôle partiellement centralisé au contrôle hiérarchique. En dernier lieu, nous avons proposé un modèle de composition hiérarchique pour les boucles de contrôle, HCLs. Dans ce modèle, nous avons mis l'accent sur les trois principaux défis du contrôle hiérarchique, à savoir : la séparation des préoccupations, l'adaptation des composants de la boucle de contrôle MAPE-K et les problèmes de coordination dans le cas où le système repose sur plusieurs boucles de contrôle. Le modèle HCLs est capable de prendre l'adaptation des paramètres et de la structure. De plus, trois types d'adaptation pouvant être appliqués par HCLs : (i) Adaptation locale qui ne franchit pas les limites du composant adaptatif, où chaque composant adaptatif du logiciel peut réaliser une adaptation locale. Ce type prend en charge l'adaptation des paramètres, c'est-à-dire qu'il doit contrôler et modifier uniquement les paramètres, (ii) Adaptation régionale, où la région est déterminée en fonction de l'architecture du système. Seul le composant

qui contient au moins un composant adaptatif peut être considéré comme une région, (iii) Adaptation supérieure qui traverse les frontières de la région. Elle permet d'adapter le composant quelle que soit sa position dans la hiérarchie. Le modèle HCLs est basé sur des stratégies d'exploration pour l'apprentissage par renforcement en ligne, utilisant le modèle de caractéristiques pour définir un espace d'adaptation. Il offre la possibilité de modifier la structure et le comportement des boucles au moment de l'exécution. En outre, il est permis de choisir le type d'adaptation approprié. Le modèle HCLs a été évalué dans un domaine de connexion sans fil pour les passagers de l'aéroport. Les résultats de l'évaluation démontrent la faisabilité et la validité de HCLs utilisant SAFRAN.

Perspectives

Dans cette thèse, nous avons présenté notre vision et notre solution pour construire des SAAs à boucles de contrôle hiérarchiques. Cependant, il reste encore certains aspects où notre travail doit être complété. Dans cette section, nous présentons les perspectives qui nous paraissent les plus pertinentes pour la poursuite de ce travail.

- La prochaine étape pour l'approche HCLs serait d'introduire un processus de référence initial pour gérer l'incertitude dans les SAAs. Nous mènerons une enquête centrée sur la capacité du la SAA à faire face aux changements imprévus et à modéliser l'incertitude, et sur les principaux défis qui limitent cette capacité. Des exemples de telles incertitudes sont les perturbations de l'environnement, les variations des lectures des capteurs et les changements dans les besoins des utilisateurs.
- L'implémentation de cette approche proposée peut être difficile en raison de la multiplicité des langages utilisés en programmation. Par conséquent, il est nécessaire de trouver ou de suggérer un outil ou un simulateur qui facilite le processus d'application de cette approche afin qu'il devienne facile de développer et de simuler l'approche.
- L'internet des objets (En anglais Internet of Things « IoT ») est un domaine d'application très pertinent pour l'auto-adaptation, telles que les véhicules intelligents et la maison intelligente. À l'avenir, nous nous concentrerons sur les systèmes intelligents dans le domaine de l'Internet des objets qui prennent en charge les systèmes avec plus d'autonomie grâce aux propriétés d'auto-adaptation et d'auto-organisation.
- MSL est langage récent et important pour définir et instancier des modèles MAPE. À l'avenir, nous nous concentrerons sur ce langage pour créer un outils qui permet de concevoir des SAAs de manière simple et avec de bons résultats.
- Enfin, en ce qui concerne l'évaluation de HCLs, elle peut être étendue avec des expériences dans d'autres domaines d'application et réalisée à la fois dans des environnements de simulation et réels.

Bibliographie

[1] Nour Aboud. "Service-Oriented Integration of Component and Organizational MultiAgent Models". Thèse de doct. Thèse de doctorat, Pau, 2012.

[2] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith et Pete Steggles. "Towards a better understanding of context and context-awareness". In : *International symposium on handheld and ubiquitous computing*. Springer. 1999, p. 304-307.

[3] Achilleas Achilleos, Kun Yang et Nektarios Georgalas. "Context modelling and a context-aware framework for pervasive service creation: A model-driven approach". In : *Pervasive and Mobile Computing* 6.2 (2010), p. 281-296.

[4] Jiri Adamek, Tomas Bures, Pavel Jezek, Jan Kofron, Vladimir Mencl, Pavel Parizek et Frantisek Plasil. *Component reliability extensions for fractal component model*. 2007.

[5] Mohamed Amoud et Ounsa Roudies. "MaPE-K-Based Approach for Security@ Runtime". In : *Software Science, Technology and Engineering (SWSTE), 2016 IEEE International Conference on*. IEEE. 2016, p. 138-140.

[6] Jesper Andersson, Luciano Baresi, Nelly Bencomo, Rogerio De Lemos, Alessandra Gorla, Paola Inverardi et Thomas Vogel. "Software engineering processes for self-adaptive systems". In : *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, p. 51-75.

[7] Paolo Arcaini, Angelo Gargantini et Paolo Vavassori. "Generating tests for detecting faults in feature models". In : *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE. 2015, p. 1-10.

[8] Paolo Arcaini, Raffaela Mirandola, Elvinia Riccobene et Patrizia Scandurra. "MSL: A pattern language for engineering self-adaptive systems". In : *Journal of Systems and Software* 164 (2020), p. 110558.

[9] Mark Assad, David J Carmichael, Judy Kay et Bob Kummerfeld. "PersonisAD: Distributed, active, scrutable model framework for context-aware services". In : *International Conference on Pervasive Computing*. Springer. 2007, p. 55-72.

[10] Cheng B et al. "Software Engineering for Self-Adaptive Systems: A Research Roadmap." In : *Berlin Heidelberg, Lecture Notes in Computer Science* 5525 (2009).

[11] Cyril Ballagny. "MOCAS: un modèle de composants basé états pour l'auto-adaptation". Thèse de doct. Thèse de doctorat, Université de Pau et des Pays de l'Adour, 2010.

[12] Yassine Banouar, Saad Reddad, Codé Diop, Christophe Chassot et Abdellah Zyane. "Monitoring solution for autonomic Middleware-level QoS management within IoT systems". In : *2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA)*. IEEE. 2015, p. 1-8.

[13] Nicolas Belloir. “Composition logicielle basée sur la relation Tout-Partie”. Thèse de doct. Ph.D. thesis, Université de Pau et des Pays de l’Adour, 2004.

[14] Mahdi Ben Alaya et Thierry Monteil. “FRAMESELF: A generic autonomic framework for self-management of distributed systems-Application on the self-configuration of M2M architecture using semantic and ontology”. In : *International Conference on Collaboration Technologies and Infrastructures (IEEE WETICE 2012),* 2012.

[15] Yazid Benazzouz. “Découverte de contexte pour une adaptation automatique de services en intelligence ambiante”. Thèse de doct. Thèse de doctorat, Saint-Etienne, EMSE, 2011.

[16] Lech Birek, Adam Grzywaczewski, Rahat Iqbal, Faiyaz Doctor et Victor Chang. “A novel Big Data analytics and intelligent technique to predict driver’s intent”. In : *Computers in Industry* 99 (2018), p. 226-240.

[17] Jan Bosch, Clemens Szyperski et Wolfgang Weck. “Ws5. The Eighth International Workshop on Component-Oriented Programming (WCOP 2003”. In : *Ce rapport résume les contributions faites par les auteurs des positions papers acceptés aussi bien celles faites par tous les participants du workshop* (2003).

[18] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè et Mary Shaw. “Engineering self-adaptive systems through feedback loops”. In : *Software engineering for self-adaptive systems*. Springer, 2009, p. 48-70.

[19] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma et Jean-Bernard Stefani. “An open component model and its support in java”. In : *International Symposium on Component-based Software Engineering*. Springer. 2004, p. 7-22.

[20] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma et Jean-Bernard Stefani. “The fractal component model and its support in java”. In : *Software: Practice and Experience* 36.11-12 (2006), p. 1257-1284.

[21] Eric Bruneton, Thierry Coupaye et Jean-Bernard Stefani. “Recursive and dynamic software composition with sharing”. In : *Proceedings of the 7th ECOOP International Workshop on Component-Oriented Programming (WCOP’02)*. Citeseer. 2002.

[22] Pierre Caserta. “Analyse statique et dynamique de code et visualisation des logiciels via la métaphore de la ville: contribution à l’aide à la compréhension des programmes.” Thèse de doct. Thèse de doctorat, Université de Lorraine, 2012.

[23] Franck Chauvel. “Méthodes et outils pour la conception de systèmes logiciels auto-adaptatifs”. Thèse de doct. Thèse de doctorat, Université de Bretagne Sud, 2008.

[24] Tao Chen, Ke Li, Rami Bahsoon et Xin Yao. “FEMOSAA: Feature-guided and knee-driven multi-objective optimization for self-adaptive software”. In : *ACM Transactions on Software Engineering and Methodology (TOSEM)* 27.2 (2018), p. 1-50.

[25] Szyperski Clemens, Gruntz Dominik et Murer Stephan. “Component software: Beyond object-oriented programming”. In : *Addison-Westley: Boston, MA, USA* (1998).

[26] Autonomic Computing et al. “An architectural blueprint for autonomic computing”. In : *IBM White Paper* 31 (2006), p. 1-6.

[27] P Dockhorn Costa. “Architectural support for context-aware applications: from context models to services platforms”. In : (2007).

[28] Bill Councill et George T Heineman. “Definition of a software component and its elements”. In : *Component-based software engineering: putting the pieces together* (2001), p. 5-19.

[29] Thierry Coupaye, Vivien Quéma, Lionel Seinturier et Jean-Bernard Stefani. *Le système de composants Fractal*. 2006.

[30] Joëlle Coutaz et Gaëtan Rey. “Foundations for a Theory of Contextors”. In : *Computer-Aided Design of User Interfaces III*. Springer, 2002, p. 13-33.

[31] Yuan-Shun Dai, Tom Marshall et XH Guan. “Autonomic and dependable computing: Moving towards a model-driven approach”. In : *Journal of Computer Science* 2.6 (2006), p. 496-504.

[32] Mirko D’Angelo, Mauro Caporuscio et Annalisa Napolitano. “Model-driven engineering of decentralized control in cyber-physical systems”. In : *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*. IEEE. 2017, p. 7-12.

[33] Pierre-Charles David. “Développement de composants Fractal adaptatifs: un langage dédié à l’aspect d’adaptation”. Thèse de doct. Thèse de doctorat, Université de Nantes, 2005.

[34] Pierre-Charles David et Thomas Ledoux. “An aspect-oriented approach for developing self-adaptive fractal components”. In : *International Conference on Software Composition*. Springer. 2006, p. 82-97.

[35] Pierre-Charles David et Thomas Ledoux. “WildCAT: a generic framework for context-aware applications”. In : *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*. 2005, p. 1-7.

[36] Pierre-Charles David, Thomas Ledoux, Marc Léger et Thierry Coupaye. “FPath and FScript: Language support for navigation and reliable reconfiguration of Fractal architectures”. In : *annals of telecommunications-annales des télécommunications* 64.1 (2009), p. 45-63.

[37] Pierre-Charles David, Thomas Ledoux et al. “Safe dynamic reconfigurations of fractal architectures with fscript”. In : *Proceeding of Fractal CBSE Workshop, ECOOP*. T. 6. Citeseer. 2006.

[38] Yves Demazeau. “From interactions to collective behaviour in agent-based systems”. In : *In: Proceedings of the 1st. European Conference on Cognitive Science. Saint-Malo*. Citeseer. 1995.

[39] Anind K Dey. “Understanding and using context”. In : *Personal and ubiquitous computing* 5.1 (2001), p. 4-7.

[40] Anind K Dey et Gregory D Abowd. “Providing architectural support for building context-aware applications”. In : (2000).

[41] Remus-Alexandru Dobrican et Denis Zampunieris. “Moving towards distributed networks of proactive, self-adaptive and context-aware systems: a new research direction?” In : *The International Journal on Advances in Networks and Services* 7.3 & 4 (2014), p. 262-272.

[42] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt et Franco Zambonelli. “A Survey of Autonomic Communications. ACM Transactions on Autonomous and Adaptive Systems”. In : 1.2 (2006), p. 223-259.

[43] Jim Dowling. “The decentralised coordination of self-adaptive components for autonomic distributed systems”. Thèse de doct. PhD thesis, Dept of Computer Science, Trinity College, Dublin,Ireland, 2004.

[44] Jim Dowling, Raymond Cunningham, Eoin Curran et Vinny Cahill. “Building autonomic systems using collaborative reinforcement learning”. In : *The Knowledge Engineering Review* 21.3 (2006), p. 231-238.

[45] Jacques Ferber. *Les systèmes multi-agents, vers une intelligence collective*. Edition Inter-editions, 1995.

[46] Jacques Ferber, Olivier Gutknecht et Fabien Michel. “From agents to organizations: an organizational view of multi-agent systems”. In : *International workshop on agent-oriented software engineering*. Springer. 2003, p. 214-230.

[47] João Ferreira. “A-OSGi: a framework to support the construction of autonomic OSGi-based applications”. Thèse de doct. Thèse de doctorat, Technical University of Lisbon, 2009.

[48] Antonio Filieri, Henry Hoffmann et Martina Maggio. “Automated design of self-adaptive software with control-theoretical formal guarantees”. In : *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, p. 299-310.

[49] David Garlan, S-W Cheng, A-C Huang, Bradley Schmerl et Peter Steenkiste. “Rainbow: Architecture-based self-adaptation with reusable infrastructure”. In : *Computer* 37.10 (2004), p. 46-54.

[50] Fausto Giunchiglia, James Odell et Gerhard Weiss. “Agent-oriented software engineering III”. In : *Proceedings of the Second International Workshop (AOSE-2002). Lecture Notes in Computer Science. Springer-Verlag*. 2003.

[51] Guillaume Grondin, Noury Bouraqadi et Laurent Vercouter. “Madcar: an abstract model for dynamic and automatic (re-) assembling of component-based applications”. In : *International Symposium on Component-Based Software Engineering*. Springer. 2006, p. 360-367.

[52] Soguy Mak Karé Gueye, Noël de Palma et Eric Rutten. “Component-based autonomic managers for coordination control”. In : *International Conference on Coordination Languages and Models*. Springer. 2013, p. 75-89.

[53] Haipeng Guo. “A bayesian approach for automatic algorithm selection”. In : *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI03), Workshop on AI and Autonomic Computing, Acapulco, Mexico*. 2003, p. 1-5.

[54] Marcus Handte, Gregor Schiele, Verena Matjuntke, Christian Becker et Pedro José Marrón. “3PC: System support for adaptive peer-to-peer pervasive computing”. In : *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 7.1 (2012), p. 1-19.

[55] Abdelhakim Hannousse. “Aspectualizing component models: implementation and interferences analysis”. Thèse de doct. Thèse de doctorat, Ecole des Mines de Nantes, 2011.

[56] Tomasz Haupt. “Towards mediation-based self-healing of data-driven business processes”. In : *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*. IEEE. 2012, p. 139-144.

[57] Markus C Huebscher et Julie A McCann. “A survey of autonomic computing—degrees, models, and applications”. In : *ACM Computing Surveys (CSUR)* 40.3 (2008), p. 7.

[58] Didac Gil De La Iglesia et Danny Weyns. “MAPE-K formal templates to rigorously design behaviors for self-adaptive systems”. In : *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 10.3 (2015), p. 1-31.

[59] Michael Jackson. “The Meaning of Requirements”. In : *Annals of Software Engineering archive* 3 (1997), p. 5-31. url : http://dl.acm.org/citation.cfm?id=590564.590577.

[60] Jeffrey O Kephart et David M Chess. “The vision of autonomic computing”. In : *Computer* 1 (2003), p. 41-50.

[61] Christian Krupitzer, Felix Maximilian Roth, Sebastian Vansyckel et Christian Becker. “Towards reusability in autonomic computing”. In : *2015 IEEE International Conference on Autonomic Computing*. IEEE. 2015, p. 115-120.

[62] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele et Christian Becker. “A survey on engineering approaches for self-adaptive systems”. In : *Pervasive and Mobile Computing* 17 (2015), p. 184-206.

[63] Robert Laddaga. “Active software”. In : *International Workshop on Self-Adaptive Software*. Springer. 2000, p. 11-26.

[64] Vincent Lanore. “On Scalable Reconfigurable Component Models for High-Performance Computing”. Thèse de doct. Thèse de doctorat, Ecole normale supérieure de lyon-ENS LYON, 2015.

[65] Vincent Lanore et Christian Pérez. “A reconfigurable component model for HPC”. In : *2015 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE)*. IEEE. 2015, p. 1-10.

[66] Philip A Laplante. *What every engineer should know about software engineering*. CRC Press, 2007.

[67] Kung-Kiu Lau et Zheng Wang. “Software component models”. In : *IEEE Transactions on software engineering* 33.10 (2007).

[68] Stéphane Lavirotte, Diane Lingrand et Jean-Yves Tigli. “Définition du contexte: fonctions de coût et méthodes de sélection”. In : *Proceedings of the 2nd French-speaking Conference on Mobility and Ubiquity Computing*. 2005, p. 9-12.

[69] Marc Léger. “Fiabilité des reconfigurations dynamiques dans les architectures à composants”. Thèse de doct. Thèse de doctorat, École Nationale Supérieure des Mines de Paris, 2009.

[70] Marc Léger, Thomas Ledoux et Thierry Coupaye. “Reliable dynamic reconfigurations in the fractal component model”. In : *Proceedings of the 6th international workshop on Adaptive and reflective middleware: held at the ACM/IFIP/USENIX International Middleware Conference*. ACM. 2007, p. 3.

[71] Rogério de Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel et al. “Software engineering for self-adaptive systems: A second research roadmap”. In : *Software engineering for self-adaptive systems II*. Springer, 2013, p. 1-32.

[72] Xin Li, Martina Eckert, José-Fernán Martinez et Gregorio Rubio. “Context aware middleware architectures: Survey and challenges”. In : *Sensors* 15.8 (2015), p. 20570-20607.

[73] Jürgen Lind. “Relating Agent Technology and Component Models”. In : *AgentLab* (2001). url : http://www.agentlab.de/documents/Lind2001e.pdf.

[74] Michael L Littman, Nishkam Ravi, Eitan Fenson et Rich Howard. “Reinforcement learning for autonomic network repair”. In : *null*. IEEE. 2004, p. 284-285.

[75] M Douglas McIlroy, J Buxton, Peter Naur et Brian Randell. “Mass-produced software components”. In : *Proceedings of the 1st international conference on software engineering, Garmisch Pattenkirchen, Germany*. 1968, p. 88-98.

[76] Philip K McKinley, Seyed Masoud Sadjadi, Eric P Kasten et Betty HC Cheng. “Composing adaptive software”. In : *Computer* 37.7 (2004), p. 56-64.

[77] Santiago Meliá, Cristina Cachero, Jesús M Hermida et Enrique Aparicio. “Comparison of a textual versus a graphical notation for the maintainability of MDE domain models: an empirical pilot study”. In : *Software Quality Journal* 24.3 (2016), p. 709-735.

[78] Andreas Metzger, Clément Quinton, Zoltán Ádám Mann, Luciano Baresi et Klaus Pohl. "Feature model-guided online reinforcement learning for self-adaptive services". In : *International Conference on Service-Oriented Computing*. Springer. 2020, p. 269-286.

[79] Sun Microsystems. "Enterprise JavaBeans Specification Version 3.0". In : (2005).

[80] Mohamed Mohamed, Djamel Belaid et Samir Tata. "Adding monitoring and reconfiguration facilities for service-based applications in the cloud". In : *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*. IEEE. 2013, p. 756-763.

[81] Salma Najar. "Adaptation des services sensibles au contexte selon une approche intentionnelle". Thèse de doct. Thèse de doctorat, Université Panthéon Sorbonne ParisI, 2014.

[82] Vivek Nallur et Rami Bahsoon. "A decentralized self-adaptation mechanism for service-based applications in the cloud". In : *IEEE Transactions on Software Engineering* 39.5 (2013), p. 591-612.

[83] Peyman Oreizy, Michael M Gorlick, Richard N Taylor, Dennis Heimhigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S Rosenblum et Alexander L Wolf. "An architecture-based approach to self-adaptive software". In : *IEEE Intelligent Systems and Their Applications* 14.3 (1999), p. 54-62.

[84] Takayuki Osogami, Mor Harchol-Balter et Alan Scheller-Wolf. "Analysis of cycle stealing with switching times and thresholds". In : *Performance Evaluation* 61.4 (2005), p. 347-369.

[85] Selma Ouareth, Soufiane Boulehouache et Smaine Mazouzi. "A component-based mape-k control loop model for self-adaptation". In : *2018 3rd International Conference on Pattern Analysis and Intelligent Systems (PAIS)*. IEEE. 2018, p. 1-7.

[86] Selma Ouareth, Soufiane Boulehouache et Smaine Mazouzi. "An Approach for Composing Multiple Control Loops Hierarchically". In : *2021 International Conference on Theoretical and Applicative Aspects of Computer Science (ICTAACS)*. IEEE. 2021, p. 1-5.

[87] Selma Ouareth, Soufiane Boulehouache et Mazouzi Smaine. "Reliable Composition of MAPE-K Loops in Self-Adaptive Software Systems". In : *International Journal of Organizational and Collective Intelligence (IJOCI)* 10.3 (2020), p. 38-54.

[88] Selma Ouareth, Soufiane Boulehouache et Mazouzi Smaine. "Self-Adaptation Through Reinforcement Learning Using a Feature Model". In : *International Journal of Organizational and Collective Intelligence (IJOCI)* 12.4 (2022), p. 1-20.

[89] Mourad Oussalah. *Ingénierie des composants: concepts, techniques et outils*. Vuibert informatique, 2005.

[90] David L Parnas. “On the criteria to be used in decomposing systems into modules”. In : *Pioneers and their contributions to software engineering*. Springer, 1972, p. 479-498.

[91] An Phung-Khac, Antoine Beugnard, Jean-Marie Gilliot et Maria-Teresa Segarra. “Model-driven development of component-based adaptive distributed applications”. In : *Proceedings of the 2008 ACM symposium on Applied computing*. ACM. 2008, p. 2186-2191.

[92] Frantisek Plásil, Dusan Balek et Radovan Janecek. “SOFA/DCUP: Architecture for component trading and dynamic updating”. In : *Proceedings. Fourth International Conference on Configurable Distributed Systems (Cat. No. 98EX159)*. IEEE. 1998, p. 43-51.

[93] Carlos Hernan Prada Rojas. “Une approche à base de composants logiciels pour l’observation de systèmes embarqués”. Thèse de doct. Thèse de doctorat, Université Grenoble Alpes, 2011.

[94] Gavin A Rummery et Mahesan Niranjan. *On-line Q-learning using connectionist systems*. T. 37. Citeseer, 1994.

[95] N Ryan. “Mobile computing in a fieldwork environment: Metadata elements”. In : *Project working document, version 0.2* (1997).

[96] Seyed Masoud Sadjadi, Philip K McKinley, RE Kurt Stirewalt et Betty HC Cheng. “Generation of self-optimizing wireless network applications”. In : *Autonomic Computing, 2004. Proceedings. International Conference on*. IEEE. 2004, p. 310-311.

[97] Mazeiar Salehie et Ladan Tahvildari. “Self-adaptive software: Landscape and research challenges”. In : *ACM transactions on autonomous and adaptive systems (TAAS)* 4.2 (2009), p. 14.

[98] Johannes Sametinger. *Software engineering with reusable components*. Springer Science & Business Media, 1997.

[99] Silvia Schiaffino et Analıa Amandi. “User–interface agent interaction: personalization issues”. In : *International Journal of Human-Computer Studies* 60.1 (2004), p. 129-148.

[100] Bill N Schilit et Marvin M Theimer. “Disseminating active map information to mobile hosts”. In : *IEEE network* 8.5 (1994), p. 22-32.

[101] JP Bigus DA Schlosnagle. “A toolkit for building multiagent autonomic systems”. In : *IBM Systems Journal, Vole* 41.3 (2002), p. 350-371.

[102] Giovanna Di Marzo Serugendo, Marie-Pierre Gleizes et Anthony Karageorgos. *Self-organising software: From natural to artificial adaptation*. Springer Science & Business Media, 2011.

[103] Ondřej Šerỳ et František Plášil. “Slicing of component behavior specification with respect to their composition”. In : *International Symposium on Component-Based Software Engineering*. Springer. 2007, p. 189-202.

[104] Vivek Sharma, Arun Thomas, Tarek Abdelzaher, Kevin Skadron et Zhijian Lu. "Power-aware QoS management in web servers". In : *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*. IEEE. 2003, p. 63-72.

[105] Roy Sterritt, Barry Smyth et Martin Bradley. "Pact: personal autonomic computing tools". In : *Engineering of Computer-Based Systems, 2005. ECBS'05. 12th IEEE International Conference and Workshops on the*. IEEE. 2005, p. 519-527.

[106] Gang Sun, Mingyue Yu, Dan Liao et Victor Chang. "Analytical exploration of energy savings for parked vehicles to enhance VANET connectivity". In : *IEEE Transactions on Intelligent Transportation Systems* 20.5 (2018), p. 1749-1761.

[107] Richard S Sutton et Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[108] Richard S Sutton, Andrew G Barto et al. *Reinforcement learning: An introduction*. MIT press, 1998.

[109] Adja Ndeye Sylla, Maxime Louvel, Eric Rutten et Gwenaël Delaval. "Design framework for reliable multiple autonomic loops in smart environments". In : *2017 International conference on cloud and autonomic computing (ICCAC)*. IEEE. 2017, p. 131-142.

[110] Can Do The. "A context manager for solving conflicts between designer's viewpoints". Thèse de doct. Thèse de doctorat, Ubiquitous Computing. COMUE Université Côte d'Azur, 2019.

[111] William E Walsh, Gerald Tesauro, Jeffrey O Kephart et Rajarshi Das. "Utility functions in autonomic systems". In : *Autonomic Computing, 2004. Proceedings. International Conference on*. IEEE. 2004, p. 70-77.

[112] Nanbor Wang, Douglas C Schmidt et Carlos O'Ryan. "Overview of the CORBA component model". In : *Component-based software engineering: putting the pieces together*. 2001, p. 557-571.

[113] Christopher JCH Watkins et Peter Dayan. "Q-learning". In : *Machine learning* 8.3 (1992), p. 279-292.

[114] Danny Weyns et Michael Georgeff. "Self-adaptation using multiagent systems". In : *IEEE software* 27.1 (2009), p. 86-91.

[115] Danny Weyns, Sam Malek et Jesper Andersson. "On decentralized self-adaptation: lessons from the trenches and challenges for the future". In : *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. ACM. 2010, p. 84-93.

[116] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese et Karl M Göschka. "On patterns for decentralized control in self-adaptive systems". In : *Software Engineering for Self-Adaptive Systems II*. Springer, 2012, p. 76-107.

[117] Danny Weyns, Bradley Schmerl, Vincenzo Grassi, Sam Malek, Raffaela Mirandola, Christian Prehofer, Jochen Wuttke, Jesper Andersson, Holger Giese et Karl M Göschka. "On patterns for decentralized control in self-adaptive systems". In : *Software Engineering for Self-Adaptive Systems II*. Springer, 2013, p. 76-107.

[118] Michael Wooldridge. *An introduction to multiagent systems*. John wiley & sons, 2009.

[119] Michael Wooldridge et Nicholas R Jennings. "Agent theories, architectures, and languages: a survey". In : *International Workshop on Agent Theories, Architectures, and Languages*. Springer. 1994, p. 1-39.

[120] Michael Wooldridge, Nicholas R Jennings et David Kinny. "The Gaia methodology for agent-oriented analysis and design". In : *Autonomous Agents and multi-agent systems* 3.3 (2000), p. 285-312.

[121] Jochen Wuttke, Yuriy Brun, Alessandra Gorla et Jonathan Ramaswamy. "Traffic routing for evaluating self-adaptation". In : *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press. 2012, p. 27-32.

[122] Jun Yan, Haibo He, Xiangnan Zhong et Yufei Tang. "Q-learning-based vulnerability analysis of smart grid against sequential topology attacks". In : *IEEE Transactions on Information Forensics and Security* 12.1 (2016), p. 200-210.

[123] Ahmad Izzuddin Yusof, Azlan Ismail et Siti ZZ Abidin. "Improving coordination of decentralized self-adaptive system in Multi Cloud environments". In : *2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*. IEEE. 2015, p. 30-35.

[124] Edith Zavala, Xavier Franch, Jordi Marco et Christian Berger. "HAFLoop: An architecture for supporting Highly Adaptive Feedback Loops in self-adaptive systems". In : *Future Generation Computer Systems* 105 (2020), p. 607-630.

[125] Edith Zavala, Xavier Franch, Jordi Marco, Alessia Knauss et Daniela Damian. "SACRE: Supporting contextual requirements' adaptation in modern self-adaptive systems in the presence of uncertainty at runtime". In : *Expert Systems with Applications* 98 (2018), p. 166-188.