République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab Blida

Faculté des sciences

Département informatique

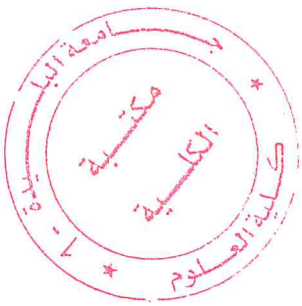
Mémoire de fin d'études

En vue d'obtenir le diplôme de master

Filière : Informatique

Spécialité : Informatique

Option : Ingénier de logiciel



THEME

Meta-heuristique pour la gestion d'évolution d'architectures logicielles

Présenté par :

M. SERIR MOHAMED

M. ATIK YASSINE

Soutenu le : 26 septembre 2018, devant le jury composé de :

M^{me}. OUKID L	Président
M^{me}. ARKAM	Examinateur
M^{me}. GUESSOUM D	Promotrice

Année universitaire : 2017/2018

Résumé

Les systèmes modernes tentent à s'adapter automatiquement pour prendre en considération les changements fréquents de leur environnement d'exécution et ainsi répondre aux qualités de services attendus comme par exemple le temps de réponse. Le but de ce projet de fin d'étude est d'implémenter une nouvelle méthode pour la gestion d'adaptation des architectures logicielles on se base sur les algorithmes métaheuristiques, L'objectif principale de notre proposition est de trouver la structure optimale dynamique d'architecture logicielle. La méthode a été concrétisée par le développement d'un outil pour atteindre la structure optimale. Ce dernier se base sur l'algorithme génétique (AG) afin de trouver et de sélectionner la meilleure stratégie pour adapter l'architecture du système. Une étude de cas a été utilisée sur le système Znn.com pour valider l'outil proposé.

Mots clés : Architectures logicielle dynamique ; Métaheuristiques ; Algorithme génétique ; Stratégie d'adaptation, Znn.com.

Abstract:

Modern systems try to adapt automatically to take into account the frequent changes in their execution environment and thus respond to expected service qualities such as response time. The aim of this end of study project is to implement a new method for the adaptation management of software architectures based on algorithms metaheuristics. The main objective of our proposal is to find the structure optimal dynamic software architecture. The method has been concretized by the development of a tool to reach the optimal structure. The latter is based on the genetic algorithm (AG) to find and select the best strategy to adapt the architecture of the system. A case study was used on the Znn.com system to validate the proposed tool.

Keywords: Dynamic software architectures; Metaheuristics; Genetic algorithm; Adaptation Strategy, Znn.com.

ملخص

تحاول الأنظمة الحديثة التكيف تلقائيًا لمراعاة التغييرات المتكررة في بيئة التنفيذ الخاصة بها وكذلك الاستجابة لصفات الخدمة المتوقعة مثل وقت الاستجابة. يهدف مشروع نهاية الدراسة هذا إلى تطبيق طريقة جديدة لإدارة التكيف في معماريات البرمجيات القائمة على خوارزميات الأدلة العليا، والهدف الرئيسي من اقتراحنا هو إيجاد البنية هندسة البرمجيات الديناميكية المثلى. تم تجسيد هذه الطريقة من خلال تطوير أداة للوصول إلى الهيكل الأمثل. ويستند هذا الأخير على الخوارزمية الجينية (AG) لإيجاد واختيار أفضل استراتيجية لتكييف بنية النظام. تم استخدام دراسة حالة على نظام Znn.com للتحقق من صحة الأداة المقترحة.

الكلمات المفتاحية: معمارية البرمجيات الديناميكية. الأدلة العليا. الخوارزمية الجينية استراتيجية التكيف، Znn.com.

Dédicaces

Avec tout l'amour qui se trouve dans mon cœur,
je dédie mon travail à un grand homme qui m'a quitté a
jamais, et qui me manque
énormément, c'est à mon père qui il est toujours présent dans
mon cœur et mon esprit et que Dieu le bénisse dans son
vaste Paradis.

A Plus chère personne de ma vie, ma mère pour leur amour,
leur confiance et leur orientation tout au long de mes études.

A mes frères, Sofiane, Fethi, Nadjib, et Hakim
qui ont été toujours présents de mes côtés.

A mes chères sœurs et belles sœurs, pour leur
Soutien et encouragement.

A mon binôme Yassine

Et A toute mes chers amis
que j'aime et qui m'aiment

Mohamed

Dédicaces

Je dédie ce modeste travail

A mes chers parents qui m'ont tout donné durant ma vie, et tous les mots de l'univers ne suffisent pas pour les remercier

A mes frères Rafik, Mohamed, Fares et sofien

A mes belles sœurs

pour leur Soutien et encouragement.

A tous les membres de la famille Atik et la famille Idir

A mon binôme Mohamed

A tout qu'ils sont proche de moi

A toute la promotion 2017-2018

Yassine

Abréviation

ADL	*Architecture Description Languages*
C2SADEL	*Software Architecture Description and Evolution Language*
AG	*Algorithme Génétique*
UML	*Unified Modeling Language*
NSGAII	*Non-dominated Sorting Genetic Algorithm II*
EA	*Evolutionary Algorithm*
GUI	*Graphical User Interface*
ME	*Méthode Exacte*
RAM	*Random Access Memory*

Table des matières

Chapitre IV : Implémentation de l'application et tests

Liste des tableaux

Liste des figures

Introduction générale

Les architectures logicielles modélisent un système en termes de composants représentants les fonctionnalités de ce système et des connecteurs décrivant les interactions entre ces composants. Étant donné que l'architecture du logiciel est devenue une partie intégrante du développement de logiciels, la gestion de son évolution est devenue la préoccupation de la plupart des chercheurs en architecture. Plusieurs chercheurs se sont concentrés sur le développement d'approches et de formalismes fournissant des systèmes d'architecture dynamique.

Problématique

La scalabilité des systèmes est un problème classique dans les travaux de recherche de planification de l'évolution sur les architectures logicielles. En effet, les descriptions d'architecture peuvent être considérées comme des graphes (des composants logiciels connectés) dont la taille peut théoriquement être arbitrairement grands. L'établissement de plans d'évolution équivaut donc à explorer toutes les combinaisons d'actions de changement possibles sur ces graphiques pour restaurer des propriétés qui peuvent être considérées comme des contraintes (locales ou globales) sur ces graphiques. Ceci peut être très contraignant dans les systèmes complexes ou à grandes échelles. Le fait de trouver le meilleur plan d'évolution en explorant toutes les possibilités peut prendre un temps considérable, ce qui nuit à la performance des systèmes. De plus, les architectes ont peu d'outils pour les aider à planifier et à exécuter l'évolution des architectures logicielles en tenant en compte du problème de la scalabilité.

Objectif

Il est question dans ce PFE d'implémenter une technique pour aide à la planification de l'évolution des architectures logicielles en utilisant une méta-heuristique pour répondre à la problématique de la scalabilité. La méthode proposée devra prendre en compte plusieurs critères pour choisir un plan d'évolution, ces critères sont définis comme des propriétés non-fonctionnelles (tels que le cout, le temps de réponse, la performance) calculées par une fonction objective. L'approche proposée est validée par une étude de cas : le système Znn.com qui est considéré comme le cas d'étude le plus utilisé dans le domaine des systèmes auto-adaptatifs.

La suite du mémoire est organisée en quatre chapitres :

Le premier chapitre ***: Etat de l'art sur les architectures logicielles*** : nous présentent dans ce chapitre les éléments et les concepts de base des architectures logicielles notamment les architectures logicielles dynamiques.

Le deuxième chapitre : *les méta-heuristiques* **:** nous nous focaliserons sur la notion des métaheuristiques ainsi que leurs caractéristiques et leurs classifications. Ensuite, nous présentons la métaheuristique « algorithme génétique » utilisée au sein de notre projet ainsi que l'application de ces derniers sur les architectures logicielles.

Le troisième chapitre : *Conception du gestionnaire de l'évaluation basée sur l'AG* **:** nous développerons et expliquerons notre approche utilisée pour résoudre le problème de la scalabilité dans architectures logicielles dynamiques en utilisant l'algorithme génétique.

Le quatrième chapitre : *Implémentation de l'application et tests* **:** Nous présentons les éléments d'implémentation et les résultats des tests obtenus de notre approche.

Nous terminons le mémoire par une conclusion général.

Chapitre I :

Etat de l'art sur les architectures logicielles

1. Introduction

Actuellement un grand intérêt est porté au domaine des architectures logicielles. Cet intérêt est motivé principalement par la réduction des couts et des délais de développement de tels systèmes. En effet, l'architecture logicielle permet d'exposer de manière compréhensible et synthétique la complexité d'un système logicielle et de faciliter l'assemblage de composants logiciels [1].nous présenterons dans ce que suit, une introduction à l'architecture logicielle, les principaux concepts et terminologies, ainsi qu'un aperçu des principaux avantages de son utilisation dans le développement logiciel. Enfin nous nous penchons sur l'aspect des architectures logicielles dynamiques.

2. Définition

L'architecture logicielle d'un programme ou d'un système informatique est une représentation du système qui aide à comprendre le comportement du système. Elle sert de modèle pour le système et le projet qui le développe, définissant les tâches à effectuer par les équipes de conception et de mise en œuvre. L'architecture est le principal support des qualités du système telles que la performance, la modifiabilité et la sécurité, dont aucune ne peut être réalisée sans une vision architecturale unificatrice. L'architecture est un artefact pour une analyse précoce afin de s'assurer qu'une approche de conception produira un système acceptable. En construisant une architecture efficace, les risques de conception seront identifiés et peuvent être atténués au début du processus de développement. [2]

Cependant, il n'existe toujours pas une définition universelle du terme« *architectures logicielles* », en effet plusieurs définitions sont proposées dans la littérature mais il est largement accepté qu'une architecture soit définie par un ensemble de composants (ex : filtres, objets, bases de données, serveurs, etc.) ainsi que par la description des interactions entre ceux-ci (ex : appels de procédures, envoi de messages, émission d'événements, etc.).Celle-ci permet de spécifier les caractéristiques d'un système en définissant : quels types de modules sont des composants du système, combien de composants de chaque type peut-il y avoir, comment les composants interagissent, quelles propriétés structurelles et comportementales doivent être respectées. [2]

La description architecturale d'un système informatique permet en outre de spécifier :

- ✓ sa structure : éléments de traitement et leurs interactions (pas de détails d'implémentation).
- ✓ son comportement : fonctionnalités et protocoles de communication, dynamisme, évolution.
- ✓ ses propriétés globales (exemples : sécurité, vivacité…).

3. Les concepts de l'architecture logicielle

L'architecture logicielle est caractérisée par les concepts fondamentaux suivants :

3.1. Le concept de composant

Le composant est une unité de calcul ou de stockage. Il peut être simple ou composite, et sa fonctionnalité peut aller de la simple procédure à une application complète.

Le composant est considéré comme un couple spécification-code : la spécification donne les interfaces, les propriétés du composant ; le code correspond à la mise en œuvre delà spécification par le composant. [2]

Les caractéristiques globales d'un composant sont les suivantes :

3.1.1. L'interface

L'interface d'un composant est la description de l'ensemble des services offerts et requis par le composant sous la forme de signature de méthodes, de type d'objets envoyés et retournés, d'exceptions et de contexte d'exécution. [2]

L'interface est un moyen d'expression des liens du composant ainsi que ses contraintes avec l'extérieur. [2]

3.1.2. Le type

Le type d'un composant est un concept représentant l'implantation des fonctionnalités fournies par le composant. Il s'apparente à la notion de classe que l'on trouve dans le modèle orienté objet. Ainsi, un type de composant permet la réutilisation d'instances de même fonctionnalité soit dans une même architecture, soit dans des architectures différentes. En fournissant un moyen de décrire, de manière explicite, les propriétés communes à un

ensemble d'instances d'un même composant, la notion de type de composant introduit un classificateur qui favorise la compréhension d'une architecture et de sa conception. [2]

3.1.3. La sémantique

La sémantique du composant est exprimée n partie par son interface. Cependant, l'interface, telle qu'elle est décrite ci-dessus, ne permet pas de préciser complétement le comportement du composant. La sémantique doit être enrichie par un modèle plus complet et plus abstrait permettant de spécifier les aspects dynamiques ainsi que les contraintes liées à l'architecture. Ce modèle doit garantir une projection cohérente de la spécification abstraite de l'architecture vers la description de son implantation avec différents niveaux de raffinements. La sémantique d'un composant s'apparente à la notion de type dans le modèle orienté objet. [2]

3.1.4. Les contraintes

Les contraintes définissent les limites d'utilisation des composants. Une contrainte est une propriété devant être vérifiée pour un système ou pour une de ces parties. Si celle-ci est violée, le système est considéré comme incohérent. Elle permet ainsi de décrire de manière explicite les dépendances des parties internes d'un composant comme la spécification de la synchronisation entre composants d'une même application. [2]
L'évolution d'un composant doit être simple et s'effectuer par le biais de techniques comme le sous typage ou le raffinement. Un ADL (*Architecture Description Languages*) doit favoriser la modification de ses propriétés (interface, comportement, implantation) sans perturber son intégration dans les applications déjà existantes. [2]

3.1.5. Les contraintes non fonctionnelles

Les propriétés non fonctionnelles doivent être exprimées à part, permettant ainsi une séparation dans la spécification du composant des aspects fonctionnels (aspects métiers de l'application) et des aspects non fonctionnels ou techniques (aspects transactionnels, de cryptographie, de qualité de service). Cette séparation permet la simulation du comportement d'un composant à l'exécution dès la phase de conception, et de la vérification de la validité de l'architecture logicielle par rapport à l'architecture matérielle et l'environnement d'exécution. [2]

3.2. Le concept de connecteur

Le connecteur modélise un ensemble d'interactions entre composants. Cette interaction peut aller du simple appel de procédure distante aux protocoles de communication.

Tout comme le composant, le connecteur est un couple spécification-code : la spécification décrit les rôles des participants à une interaction ; le code correspond à l'implantation du connecteur. Cependant, la différence avec le composant est que le connecteur ne correspond pas à une unique, mais éventuellement à plusieurs unités de programmation. [2]

Ces caractéristiques sont les suivantes :

3.2.1. L'interface d'un connecteur

L'interface d'un connecteur définit les points d'interactions entre connecteurs et composants. L'interface ne décrit pas des services fonctionnels comme ceux du composant mais s'attache à définir des mécanismes de connexion entre composants. Certains ADLs nomment ces points d'interactions comme étant des rôles. [2]

3.2.2. Les contraintes

Les contraintes permettent de définir les limites d'utilisation d'un connecteur, c'est-à-dire les limites d'utilisation du protocole de communication associé. Une contrainte est une propriété devant être vérifiée pour un système ou pour une de ses parties. Si celle-ci est violée, le système est considéré comme un système incohérent. Par exemple, le nombre maximum de composants interconnectés à travers le connecteur correspond à une contrainte.

La maintenance des propriétés (interface, comportement) d'un connecteur doit lui permettre d'évoluer sans perturber son utilisation et son intégration dans les applications existantes. Il s'agit de maximiser la réutilisation par modification ou raffinement des connecteurs existants. Un ADL donnant la possibilité de définir un connecteur doit donc permettre de le faire évoluer de manière simple et indépendante en utilisant le sous typage ou des techniques de raffinement. [2]

3.3. Le concept d'un rôle

Représente un point d'interaction entre un connecteur et son environnement, il dispose d'un domaine représentant le top ou le bottom de son connecteur, il peut être connecté au port d'un composant ou à un autre rôle avec un autre connecteur. [2]

3.4. Le concept d'un port

Un composant interagit avec son environnement par l'intermédiaire de points d'interactions appelés ports. Les ports (et par conséquent les interfaces) peuvent être fournis ou requis. Un port représente un point d'accès à certains services du composant. Le comportement interne du composant ne doit être ni visible, ni accessible autrement que par ses ports. [2]

4. Les architectures dynamiques

Les systèmes logiciels traitant des applications distribuées dans des environnements changeants nécessitent normalement une supervision humaine pour continuer à fonctionner dans toutes les conditions. Ces tâches de (re) configuration, de dépannage et de maintenance générale entraînent des procédures coûteuses et fastidieuses pendant la phase d'exploitation. Par conséquent, il existe une forte demande pour la réduction de la complexité de gestion, l'automatisation de la gestion, la robustesse et la réalisation de toutes les exigences de qualité souhaitées dans une plage de coûts et de temps raisonnable pendant le fonctionnement. La prise en compte de ces taches au niveau d'architecture logicielle semble être une bonne une réponse à ces demandes. L'architecture logicielle facilite la création de systèmes auto-adaptatifs visant à s'ajuster aux changements pendant son fonctionnement. On parle alors des architectures dynamiques. Les changements peuvent provenir de la nature du système logiciel (causes internes, par exemple, défaillance) ou du contexte (événements externes, par exemple, augmentation des demandes des utilisateurs). [3]

Les architectures dynamiques correspondent à des applications dont les composants sont créés, interconnectés, et supprimés pendant l'exécution. Cette dynamique répond à des contraintes liées à l'adaptabilité d'une application distribuée et à la mobilité de ses utilisateurs. Le caractère dynamique des architectures implique des difficultés supplémentaires pour la description. Pour les architectures statiques, cette description est réalisée via la déclaration des instances des composants, et de liens d'interconnexions. Cette approche est inutilisable puisque la structure même de l'architecture change. [2]

Les modifications des architectures logicielles de manière générale peuvent se produire soit au moment de :

- ✓ La conception,
- ✓ au moment de la pré-exécution
- ✓ ou au moment de l'exécution.

Les architectures dynamiques changent de structures pendant l'exécution du système et donnent une description à propos de ce changement.

La dynamique de l'architecture consiste donc à adapter une architecture pour prendre en compte de nouveaux besoins. Ceci permet de faire passer une architecture d'une configuration C à une configuration C'. Une opération peut être initiée soit par un administrateur, soit par l'application elle-même. [2]

Ces architectures dynamiques sont particulièrement utilisées dans les applications dites « sensibles au contexte » c'est-à-dire les applications qui doivent évoluer suite à des changements de besoins de l'utilisateur. [2]

4.1. Les Types d'architecture logicielle dynamique

Pour soutenir le développement des architectures dynamiques plusieurs chercheurs se sont concentrés sur le développement d'approches et de formalismes fournissant des systèmes d'architecture dynamique.

Suite à ces recherche, différentes type de dynamique sont apparus. Parmi ces types nous trouvons :

- ➢ **Evolution statique**

Le temps d'évolution statique fait référence aux changements effectués pendant la phase de conception du système. Une fois la nouvelle architecture décrite, une nouvelle instance du système est générée et exécutée. Ensuite, une fois qu'un système est implémenté, nous devons l'arrêter pour effectuer des modifications, puis produire un autre système exécutable. Ce genre d'évolution est assez simple à réaliser. C2SADEL (*Software Architecture Description and Evolution Language*) [4] est le premier ADL qui traite de l'évolution statique dans les architectures logicielles.

➢ **Evolution dynamique**

Evolution dynamique également connue sous le nom d'évolution active, d'évolution en temps réel, de changement en ligne, de mise à jour dynamique, de changement dynamique, d'évolution du temps de fonctionnement, etc. systèmes critiques. Les changements dynamiques d'une architecture peuvent être planifiés au moment de la spécification de l'architecture ou non planifiés [4]. Dans le premier cas, les changements susceptibles de se produire pendant l'exécution du système doivent être connus au départ, ils doivent donc être spécifiés lors de la description de l'architecture. L'évolution non planifiée n'impose aucune restriction au moment de la spécification de l'architecture sur les types de changements autorisés [5].

➢ **Dynamique d'implémentation**

Ce type de dynamique permet de modifier la mise en œuvre d'un composant, c'est-à-dire la manière avec laquelle un composant a été développé(le code du composant) sans changer ni les interfaces ni les connexions. Ce type de changement permet de faire évoluer un composant d'une version à une autre (voir figure I.1) [6].

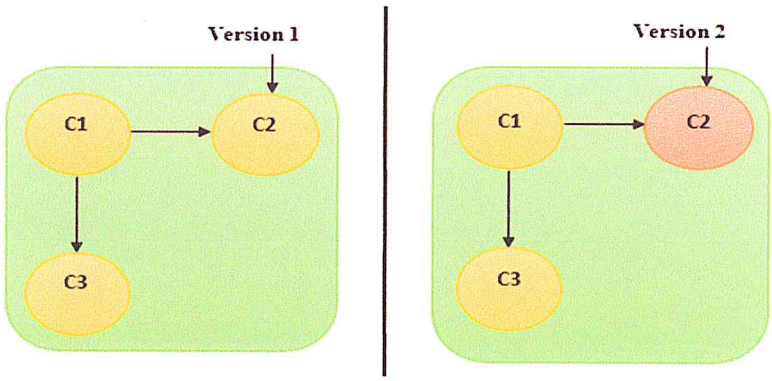


Figure I.1: Exemple de changement d'implémentation

➢ **Dynamique d'interfaces**

Comme nous l'avons détaillé précédemment, un composant fournit ses services à travers des interfaces. La dynamique d'interfaces consiste modifier les services fournis par un composant au biais de ses interfaces. Ceci, se traduit soit par la modification de l'ensemble des services fournis soit par la modification de la signature d'un service (voir figure I.2) [6].

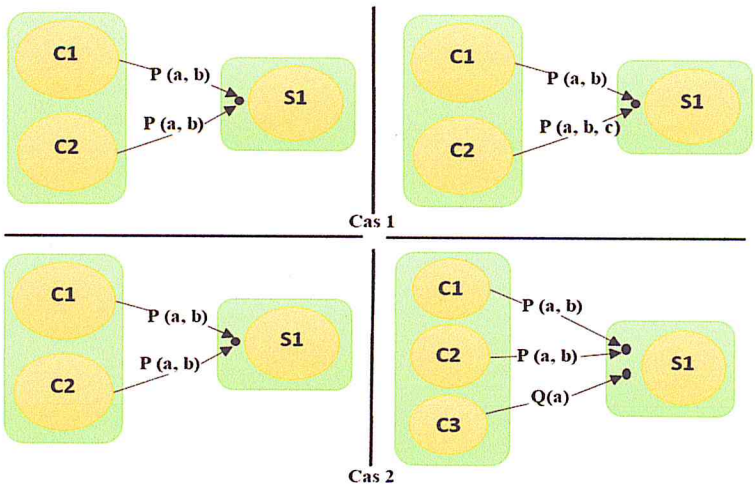


Figure I.2: Exemple de changement d'interface

➢ **Dynamique de géométrie**

Elle appelée aussi dynamique de localisation ou encore de migration. La dynamique de géométrie modifier la distribution géographique d'une application. En effet, ce type de dynamique altère uniquement l'emplacement des composants. Ainsi, un composant peut changer de localisation en migrant d'un site vers un autre. Comme le montre la figure I.3, le composant C4 est déplacé du site 2 vers le site 3 [6].

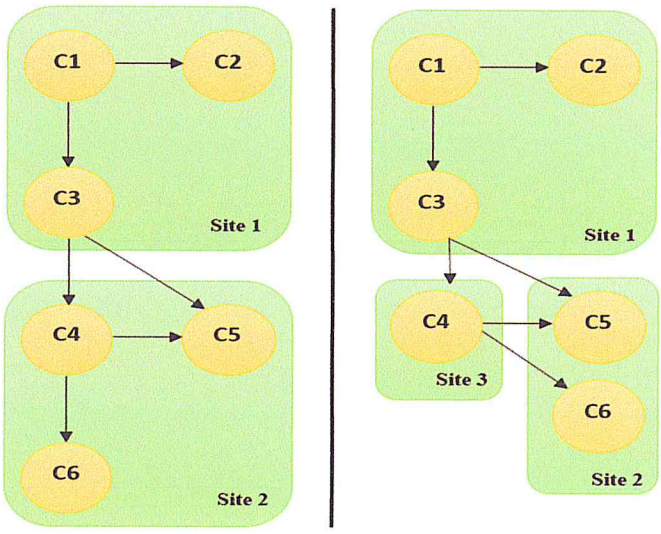


Figure I.3: Exemple de changement géométrique

➢ **Dynamique de structure**

Elle modifier la topologie de l'application. C'est-à-dire dans ce type de dynamique on s'intéresse uniquement au changement de la structure de l'application en termes de composants et de connexions. Cette dynamique se réalise à l'aide de quatre opérations de base [7] : ajout d'un composant, suppression d'un composant, ajout d'une connexion, suppression d'une connexion. Comme le montre la figure I.4, la connexion T1 redirigée vers la composant S1 [6].

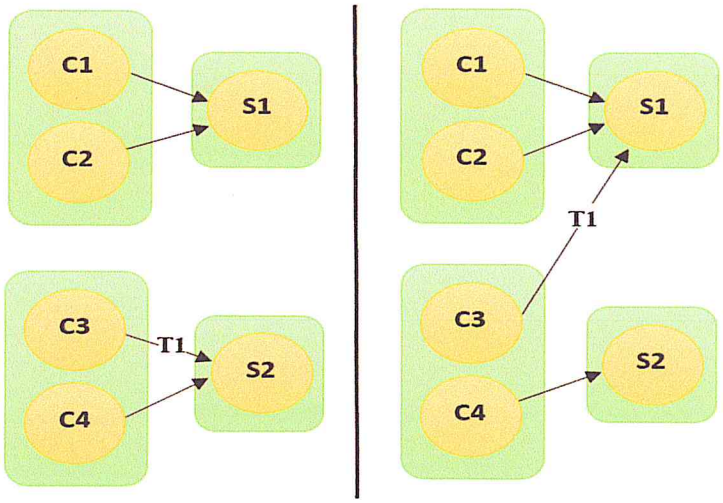


Figure I.4: Exemple de changement de structure

4.2. Les avantages par rapport aux architectures statiques

➢ **La correction**

Dans certains cas, nous remarquons que l'application en cours d'exécution n'agit pas correctement comme prévu. La solution consiste à identifier le composant d'application qui pose le problème et à le remplacer par une nouvelle version supposée correcte. Cette nouvelle version fournit la même fonctionnalité que l'ancienne ; il corrige simplement ses défauts. [8]

➢ **L'adaptabilité**

Même si l'application s'exécute correctement, il arrive que l'environnement d'exécution, tel que le système d'exploitation, les composants matériels ou d'autres applications ou données dont elle dépend, change. L'architecture doit s'adapter et évoluer soit en ajoutant de nouveaux composants / connexions ou en remplaçant des composants / connexions par d'autres. [8]

➢ **La perfection**

L'objectif de ce type d'adaptation est d'améliorer les performances du système. Par exemple, nous réalisons que l'implémentation d'un composant n'est pas optimisée, nous avons donc décidé de remplacer l'implémentation du composant. Un autre exemple d'un composant qui reçoit beaucoup de demandes et ne peut pas les satisfaire. Pour éviter la dégradation des performances de l'application, nous réduisons la charge de ce composant en installant un autre composant qui partage sa tâche. [8]

➢ **Evolution**

Lors du développement de l'application, certaines fonctionnalités ne sont pas prises en compte. Avec l'évolution des besoins des utilisateurs, l'application doit être étendue avec de nouvelles fonctionnalités. Cette extension peut être réalisée en ajoutant un ou plusieurs composants / connexions pour assurer les nouvelles fonctionnalités ou même conserver la même architecture de l'application et simplement étendre les composants existants. [8]

➢ **La scalabilité**

En raison de la taille et de la complexité croissante des systèmes logiciels, l'évolutivité est un problème important, ce qui signifie que l'application est capable de réagir automatiquement à une demande croissante de ressources disponible. [8]

La principale préoccupation des techniques de conception de systèmes logiciels est de pouvoir faire face à ce problème important d'évolution de l'architecture logicielle et de soutenir toutes ces différentes catégories d'évolution que nous avons citées plus haut. [8]

5. Conclusion

Nous avons présenté dans ce chapitre le contexte global dans lequel s'intègre ce travail de recherche : le domaine des architectures logicielles. Pour cerner la notion d'architectures logicielles nous avons présenté les principaux concepts et terminologies de architecture logicielles notamment les architectures logicielles dynamiques, l'étude de cette terminologie nous a permis de mettre en lumière les différentes définitions et notion que nous jugeons nécessaires pour aborder notre problématique.

Chapitre II :

Les méta-heuristiques

1. Introduction

Durant ces dernières années, plusieurs métaheuristiques ont prouvé leur efficacité pour la résolution de divers problèmes dans différents domaines. Ce chapitre est consacré à l'étude des métaheuristiques notamment l'algorithme génétique que nous avons utilisé dans notre approche. Nous présenterons d'une manière détaillée l'algorithme génétique en mettent l'accent sur quelques exemples d'application de cet algorithme dans le domaine de l'architecture logicielle.

2. Définition de métaheuristique

Une métaheuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficile (souvent issus des domaines de la recherche opérationnelle, de l'ingénierie ou de l'intelligence artificielle) pour lesquels on ne connaît pas de méthode classique plus efficace.

Les métaheuristiques sont généralement des algorithmes stochastiques itératifs, qui progressent vers un optimum global, c'est-à-dire l'extremum global d'une fonction, par échantillonnage d'une fonction objectif. Elles se comportent comme des algorithmes de recherche, tentant d'apprendre les caractéristiques d'un problème afin d'en trouver une approximation de la meilleure solution.

Il existe un grand nombre de métaheuristiques différentes. Ces méthodes utilisent cependant un haut niveau d'abstraction, leur permettant d'être adaptées à une large gamme de problèmes différents.

3. Les différents algorithmes de métaheuristique

Nous pouvons différencier deux types de métaheuristiques : à base de solution unique et les métaheuristiques à base de population de solutions comme montre la Figure II.1

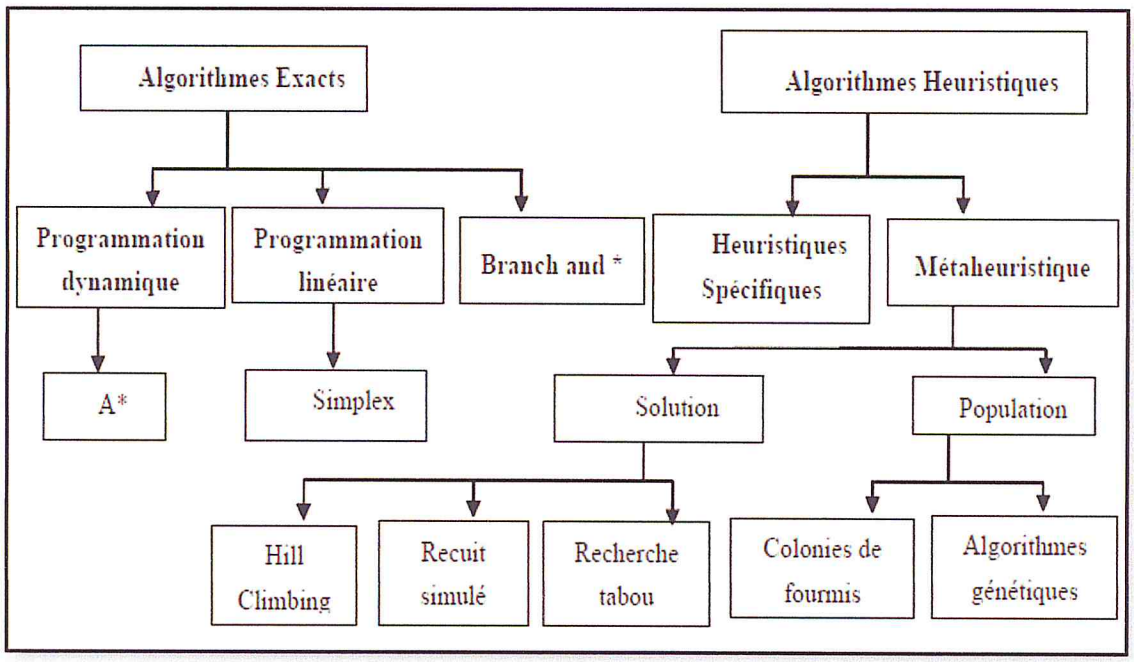


Figure II.1 : Classification des méthodes de résolution de problèmes d'optimisation [9]

Dans notre projet on s'intéresse aux algorithmes génétiques car elles sont plus efficaces et simple à implémenter.

4. Algorithmes génétiques

4.1. Principe

Les algorithmes génétiques ont été mis au point par John Holland de l'université du Michigan aux États-Unis dans les années 60. Ils ont ensuite été raffinés par De Jong et popularisés par Goldberg et Goldberg et Holland. C'est les plus connus des algorithmes évolutionnaires ; ils favorisent l'utilisation du croisement comme principal opérateur de recherche. Il utilise cependant la mutation avec un faible pourcentage de probabilité, et une méthode de sélection de type probabiliste dans laquelle la probabilité de sélection est proportionnelle à la fonction d'adaptation de l'individu. Suivant, le type de codage utilisé, on distingue deux types de représentation des individus : codage binaire et le codage réal qui est le plus utilisé. [10]

L'algorithme génétique est organisé en plusieurs étapes et fonctionne de manière itérative. L'algorithme génétique le plus simple introduit par Holland. Celui-ci met en œuvre différents opérateurs qui seront décrits dans la section suivante. Mais avant, il est nécessaire de définir quelque termes de base rencontrés dans la littérature [11] :

- **Individu** : solution potentielle du problème.
- **Chromosome** : solution potentielle du problème sous une forme codée (forme de chaîne de caractères).
- **Population** : ensemble fini d'individus (de solution).
- **Gène** : partie élémentaire (caractère) non divisible d'un chromosome.
- **Fonction fitness** : Terme anglo-saxon qui désigne la fonction d'évaluation d'un individu. Cette fonction est liée à la fonction à optimiser et permet de définir le degré de performance d'un individu (donc d'une solution).

4.2. Operateurs génétiques

4.2.1. Population initiale

Le choix d'une population de N individus P(0) = {X1, . . . Xn} se fait, en général, par des tirages uniformes dans l'espace de recherche E en veillant éventuellement à ce que les individus produits respectent les contraintes. Par ailleurs, si on dispose d'informations à priori sur le problème indiquant une région où l'on est sûr de trouver l'optimum, il parait bien évidemment naturel d'ajouter manuellement de bonnes solutions dans la population initiale, tout en assurant une diversité suffisante de la population. La population initiale peut aussi être le résultat d'une évolution précédente. [12]

4.2.2. Evaluation

L'évaluation consiste à mesurer la performance de chaque individu de la population. Il utilise pour cela la fonction fitness(Fit). C'est une fonction réelle positive qui reflète la force de l'individu. Un individu ayant une grande valeur fitness représente une bonne solution au problème, alors qu'un individu ayant une faible valeur fitness représente une solution médiocre.

Dans le cas d'un problème de minimisation, la fonction fitness peut être obtenue en utilisant une des formulations suivantes [13] :

$$Fit(x) = 1/F_{obj}(x) \quad (1)$$

$$Fit(x) = -F_{obj}(x) \quad (2)$$

4.2.3. Sélection

La sélection des individus sur lesquels on applique le croisement fait intervenir la fonction fitness à optimiser (Fit). Pour cette phase supposons connu les individus de la population au temps t et la fonction d'évaluation (Fit). Il faut trouver une manière de choisir les individus répondant le mieux au problème. Il s'agit donc de privilégier les individus ayant un score au-dessus de la moyenne, et de pénaliser ceux en dessous de cette moyenne [14]. Plusieurs approches sont possibles parmi eux :

a) La sélection par roulette [15]

Les parents sont sélectionnés en fonction de leurs performances. Meilleur est le résultat codé par un chromosome, plus grandes sont ses chances d'être sélectionné. Il faut imaginer une sorte de roulette de casino sur laquelle sont placés tous les chromosomes de la population, les places accordées à chacun des chromosomes étant en relation avec sa valeur d'adaptation. Ensuite, la bille est lancée et s'arrête sur un chromosome (Figure II.2). Les meilleurs chromosomes peuvent ainsi être tirés plusieurs fois et les plus mauvais ne jamais être sélectionnés. Cela peut être simulé par l'algorithme.

La sélection par roulette rencontre des problèmes lorsque la valeur d'adaptation des chromosomes varie énormément. Si la meilleure fonction d'évaluation d'un chromosome représente 90% de la roulette alors les autres chromosomes auront très peu de chance d'être sélectionnés et on arriverait à une stagnation de l'évolution.

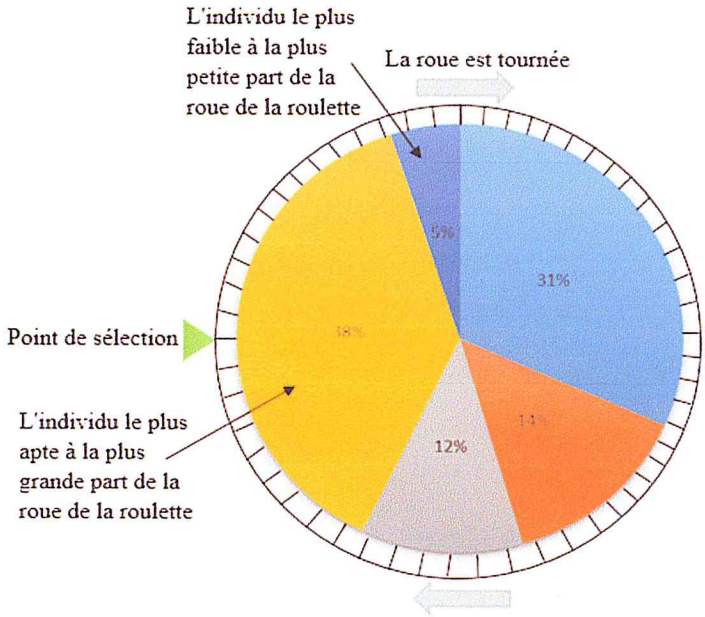


Figure II.2 : Illustration de la roulette de sélection [1]

b) Sélection par rang [15]

La sélection par rang trie d'abord la population par aptitude. Chaque chromosome se voit associé un rang en fonction de sa position. Le plus mauvais chromosome aura le rang 1, le suivant 2, et ainsi de suite jusqu'au meilleur chromosome qui aura le rang N (pour une population de N chromosomes). La sélection par rang d'un chromosome est la même que par roulette, mais les proportions sont en relation avec le rang plutôt qu'avec la valeur de l'évaluation. Avec cette méthode de sélection, tous les chromosomes ont une chance d'être sélectionnés. Cependant, elle conduit à une convergence plus lente vers la bonne solution. Ceci est dû au fait que les meilleurs chromosomes ne diffèrent pas énormément des plus mauvais.

c) Sélection Steady-State [15]

L'idée principale est qu'une grande partie de la population puisse survivre à la prochaine génération. L'algorithme génétique marche alors de la manière suivante :

I. A chaque génération sont sélectionnés quelques chromosomes (parmi ceux qui ont le meilleur coût) pour créer des chromosomes fils.

II. Les chromosomes les plus mauvais sont retirés et remplacés par les nouveaux.

III. Le reste de la population survie à la nouvelle génération.

[1] http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php/

4.2.4. Croissement (L'hybridation)

Le croisement est l'opérateur principal AG (*Algorithme Génétique)*. Son rôle consiste à choisir aléatoirement deux individus parents parmi ceux sélectionnés pour les combiner et créer deux nouveaux individus enfants. Il joue un rôle important sur la convergence de l'AG, en lui permettant de concentrer une partie de la population autour des meilleurs individus. À chaque type de codage correspondent différents types de croisement. Pour les AG avec un codage binaire, on utilise le croisement à un point (chaque bit caractérisant le nouvel individu est celui de l'un de ses « parents » choisi aléatoirement), à multipoints et le croisement uniforme. Selon Deb, les AG avec un codage réel peuvent utiliser la méthode de croisement linéaire (la valeur de l'individu enfant est une combinaison linéaire de celle de ses parents), naïf, mélangé ou encore binaire simulé. [10]

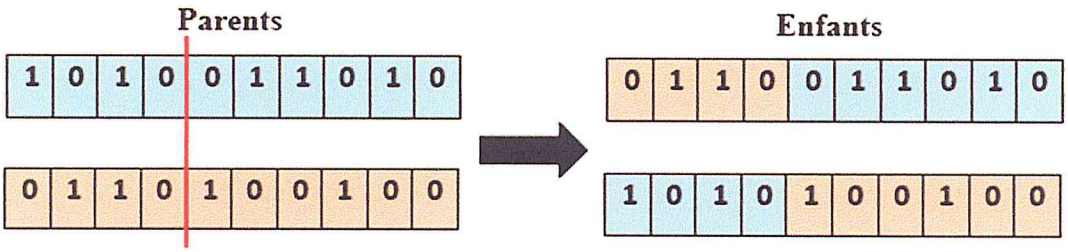


Figure II.3 : Représentation schématique du croisement à un point

4.2.5. Mutation

Les AG utilisent l'opérateur de mutation pour conserver la diversité de la population et évite en particulier à l'AG de converger vers un optimum local. Pour les AG avec un codage binaire, elle consiste à changer la valeur d'un ou plusieurs bits de l'individu de la population parent d'une valeur 1 à la valeur 0, ou réciproquement, avec une probabilité de mutation p_m fixée. Concernant les AG avec un codage réel, il existe différentes méthodes de mutation telles que la mutation aléatoire, la mutation non uniforme, la mutation distribuée normalement et la mutation polynomiale. [10]

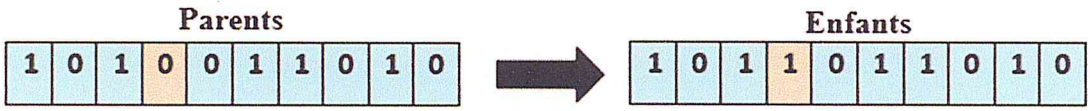


Figure II.4 : Représentation schématique d'une mutation dans un chromosome

La figure II.5 suivante illustre le processus d'optimisation développé par les algorithmes génétiques :

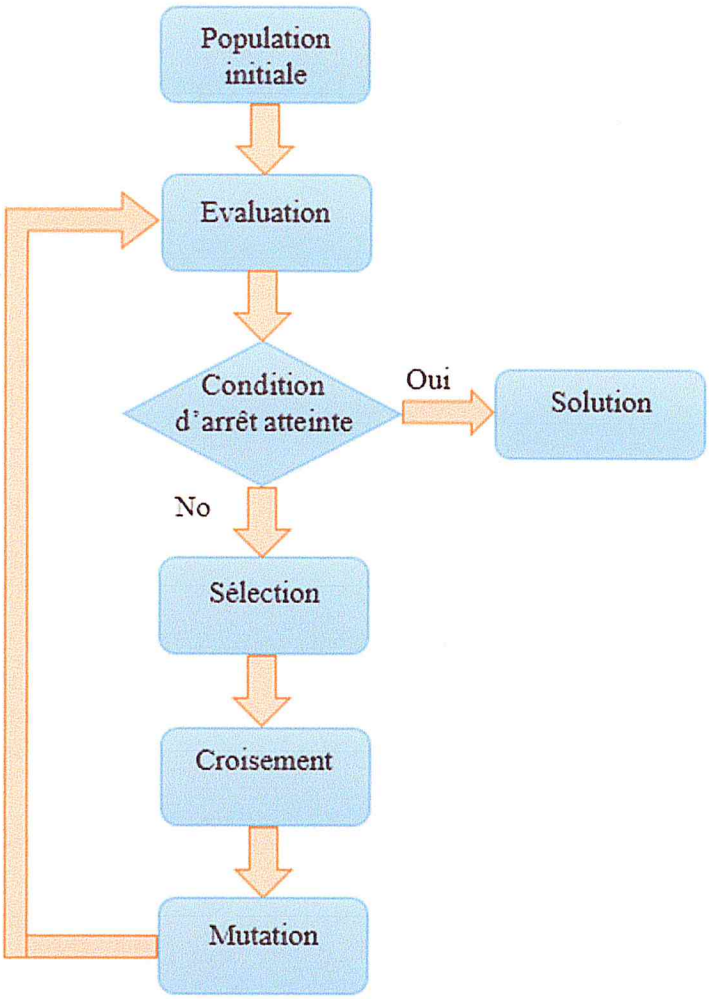


Figure II.5 : Organigramme d'un algorithme génétique [16]

5. Exemples d'application de l'algorithme génétique dans les architectures logicielles

Il existe plusieurs domaines dans l'ingénierie logicielle dans lesquels des algorithmes de recherche ont été implémentés avec succès. Nous allons maintenant présenter un bref aperçu de quelques exemples d'application des AG dans le domaine des architectures logicielles afin de comprendre et de s'inspirer de ces applications pour les adaptés dans notre proposition.

5.1. Synthèse génétique de l'architecture logicielle [17]

Le premier travail de recherche que nous avons étudié est celui de la synthèse de l'architecture logicielle, elle peut être vue comme un problème combinatoire, elle consiste à déterminer comment trouver une configuration optimale de modèles pour un système. Le problème majeur dans la synthèse d'architecture logicielle automatisée est la représentation des exigences fonctionnelles.

5.1.2. Objectif du travail

Le but principal de ce travail est de développer une approche pour la synthèse de l'architecture logicielle en utilisant les AG. A partir des exigences de base (responsabilités) du système présenté sous forme du graphe de dépendances qui est ensuite donné en entrée à un algorithme génétique, ce qui produira une suggestion pour l'architecture du système donné sous la forme d'un diagramme de classes UML (*Unified Modeling Language*).

5.1.3 Codage génétique de l'architecture

Le codage utilisé dans ce travail est comme suit :

Un chromosome est un ensemble de responsabilités donné. Cette instance de Chromosome contient toutes les informations concernant les responsabilités données dans l'entrée. Il existe deux types de données concernant chaque responsabilité **ri**. Premièrement, il y a l'information de base donnée en entrée. Ceci contient les responsabilités **Ri** = {rk, rk + 1, ..., rm} en fonction de **ri**, son nom **ni**, type **di** (fonctionnel ou données), fréquence **fi**, taille du paramètre **pi**, temps d'exécution **ti**, coût d'appel **ci** et variabilité **vi**. Deuxièmement, il y a l'information sur la place de la responsabilité dans l'architecture : la classe **Ci** = {C_{i1}, C_{i2}, ..., C_{iv}} à laquelle elle appartient, l'interface **Ii** qu'elle implante, le répartiteur **Di** utilise, les responsabilités **RDi⊂ Ri** qui l'appelle à travers le répartiteur, et le design pattern **Pi**, il est une partie de. Le répartiteur reçoit un champ distinct par opposition aux autres motifs pour des raisons d'efficacité. La Figure II.6 représente la structure d'un supergène.

Ri	ni	Di	Fi	Pi	Ti	ci	vi	Ci	Ii	Di	RDi	Pi

Figure II.6 : Supergène SG_i pour la responsabilité r_i

Le chromosome réel est formé en collectant simplement tous les super gènes. La figure II.7 illustre un chromosome avec **m** responsabilités.

SG_1	SG_2	.....	SG_{m-1}	SG_m

Figure II.7 : Chromosome

5.1.4. Population initiale

Une population initiale est d'abord produite, où seulement les structures de base, telles que la division de classes et les interfaces sont choisies au hasard. Quatre cas particuliers sont insérés : toutes les responsabilités étant dans la même classe, toutes les responsabilités étant différentes classes, toutes les responsabilités ayant leur propre interface, et toutes les responsabilités étant autant groupées à la même interface que la division de classe permet.

5.1.5. Fonction de fitness

La fonction de fitness est basée sur des métriques logicielles largement utilisées [18], dont la plupart proviennent de la suite des métriques introduite par Chidamber et Kemerer [19]. Ces métriques ont été utilisées comme point de départ pour la fonction de fitness. w_i est le poids pour la subfitness respective sf_i, la fonction fitness f (x) pour le chromosome x peut être exprimé comme suit :

$$f(x) = w_1 * sf_1 - w_2 * sf_2 + w_3 * sf_3 - w_4 * sf_4 - w_5 * sf_5 \qquad (3)$$

Ici, sf1 mesure la modifiabilité positive, sf2 négatif modifiabilité, efficacité positive sf3, efficacité négative sf4 et complexité de sf5.

5.1.6. Résultat

Ce travail montre que la synthèse de l'architecture logicielle à comportement régulier est possible en utilisant des algorithmes génétiques avec un nombre raisonnable de générations. En particulier, l'utilisation de schémas de structuration de niveau supérieur, comme les styles architecturaux et les modèles de conception, semble assez bien cadrer avec le processus génétique.

5.2. Une approche pour la découverte évolutive des architectures logicielles [20]

La deuxième approche étudiée concerne la découverte de blocs fonctionnels d'un système donnée.

5.2.1. Problématique

Souvent, les ingénieurs logiciels doivent procéder à l'analyse architecturale à partir d'un ensemble de blocs fonctionnels en quels consiste un système afin de le migrer ou d'étendre ses fonctionnalités. Cela pourrait être une tâche difficile lorsque la conception du système sous-jacent a été corrompue en raison de changements dans les exigences. Dans ces cas, les ingénieurs doivent consacrer leur temps et leurs efforts, avec leur propre expérience comme seule garantie, à la découverte manuelle de ces blocs fonctionnels.

5.2.2. Objectif

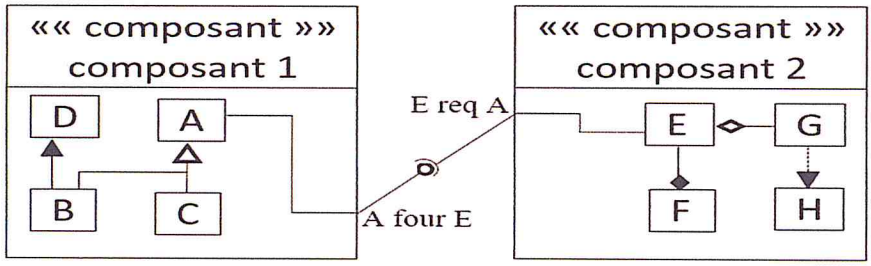
L'objectif de Ce travail est d'explorer la conception d'un algorithme évolutif (EA) pour la découverte de l'architecture sous-jacente des systèmes logiciels. Et la recherche d'une approche générique évolutive pour aider les ingénieurs logiciels à prendre leurs décisions et l'amélioration de leurs conceptions. Le résultant est une architecture représentée avec un diagramme de composants UML 2.

5.2.3. Encodage des solutions

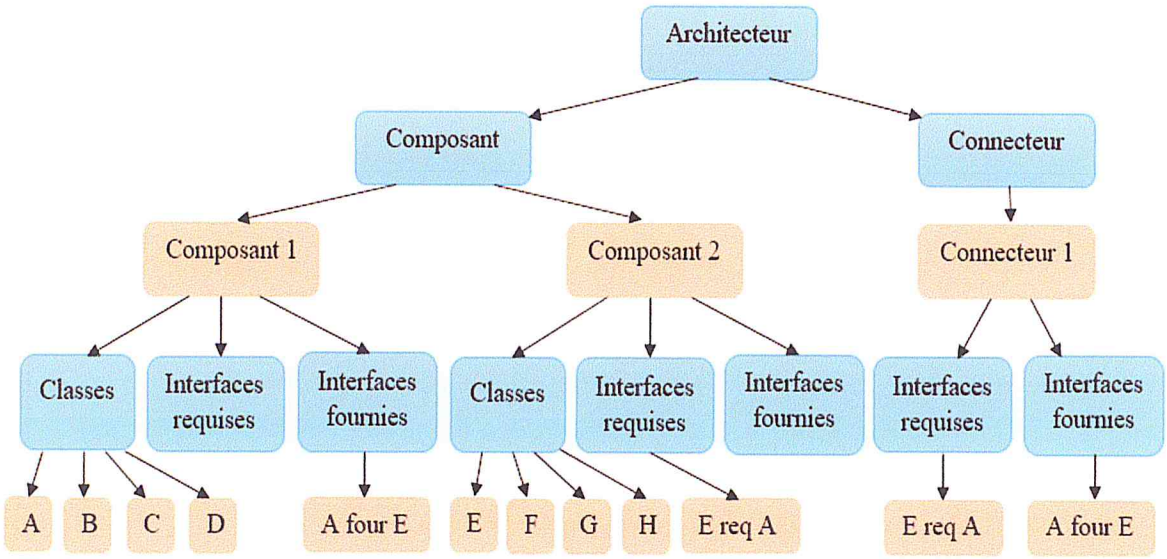
Le codage utilisé est comme suit :

Les composants, les interfaces, les connecteurs et les éléments internes présentent une composition hiérarchique. Les classes et leurs relations peuvent constituer un composant dont la spécification complète nécessite la définition de ses interfaces fournies et requises. Les connecteurs peuvent être divisés en interfaces qu'ils lient. Ensuite, le mappage d'un diagramme de composants dans une structure arborescente est réalisable comme le montre la figureII.8, où les nœuds d'ombrage constituent la trame de la solution, constituée par les artefacts obligatoires apparaissant dans tout modèle architectural. Le reste des nœuds représente les éléments qui peuvent être différents d'une solution à l'autre, c'est-à-dire un certain nombre de composants et de connecteurs ainsi que la distribution des classes et des interfaces entre eux. Plus précisément, le nœud racine, Architecture, représente le diagramme de composants constitué d'un ensemble de composants et de connecteurs. Chaque composant est défini par un nœud Component en fonction de ses classes internes et de ses interfaces. De même, chaque connecteur est décrit par la paire d'interfaces requises et fournies qu'il relie.

Comme ce sont des éléments composés, ils sont représentés comme des nœuds non terminaux. Enfin, les classes et les interfaces constituent les nœuds terminaux.



(a) Phénotype



(b) Génotype

Figure II.8 Le processus de cartographie phénotype / génotype

5.2.4. Population initiale

L'espace de recherche est constitué de toutes les combinaisons possibles de distribution de classes parmi les composants, identifiant également ses interfaces et les connecteurs. Ces groupes de classes candidats et la façon dont les interfaces et les connecteurs en sont déduits doivent également garantir que l'architecture correspondante représente une solution valide.

Tout d'abord, un nombre aléatoire de composants est sélectionné entre un minimum et un maximum. Les valeurs par défaut sont définies sur un minimum de deux et un maximum de n composants, n'étant le nombre de classes dans le modèle d'entrée. La limite supérieure

garantit qu'aucun composant vide ne sera généré. Ensuite, chaque classe est assignée à un composant, assurant que chaque composant a au moins une classe. Après cette affectation initiale, les autres contraintes détaillées sont omises, ce qui permet un processus d'initialisation plus rapide, l'idée principale est que ces individus invalides seront progressivement éliminés au fil des générations.

5.2.5. Fonction de fitness

Une fois que les trois métriques de conception ont été définies et calculées (ICD, ERP, GCR) par écrivain, la fonction de condition physique peut être calculée comme une agrégation de classement, où les meilleures valeurs sont les plus faibles et, par conséquent, la forme physique globale doit être minimisée. r renvoie la position de classement d'un individu spécifique.

$$\text{Fitness}_{ind} = \begin{cases} r(ICD_{ind}) + r(ERP_{ind}) + r(GCR_{ind}) & \text{si ind est valide} \\ \#\text{individus} * \#\text{métriques} + 1 & \text{si ind est invalide} \end{cases} \quad (4)$$

ICD : la densité de couplage intra-modulaire.

ERP : pénalité de relations externes.

GCR : la métrique des groupes / composantes.

5.2.6. Résultat

Le travail étudié montre que l'algorithme évolutif fournit à l'architecte logiciel des informations précieuses qui pourraient être utilisées pour analyser les forces et les faiblesses de la structure du système, pour reconsidérer certaines décisions de conception et explorer différentes configurations pour atténuer les risques.

5.3. Sélection des services Web à base d'algorithmes génétiques multi-objectifs [10]

La dernière approche étudiée est celui de la sélection des services web.

5.3.1. Problématique

Aujourd'hui le besoin d'un client est devenu de nature multi objectif (minimiser les coûts, minimiser les délais, augmenter le taux de service…). De plus, comme les services Web avec des fonctionnalités similaires sont censés être fournis par des fournisseurs concurrents, Il sera donc nécessaire, de fournir un outil d'aide à la décision.

De ce fait, découvrir un service web qui nous intéresse est une chose, alors que découvrir le service web le plus adéquat en est une autre. La qualité de service dans le cas des services web se mesure à l'aide de plusieurs métriques dont les métriques de performance, de disponibilité et de fiabilité. Une recherche sur internet nous permet certainement de trouver plusieurs services web qui remplissent ces critères. Mais lequel sera le plus adéquat, le moins cher, le plus luxueux, le plus rapide, etc., bref, le meilleur ? Lequel de ces services sera le plus disponible, le plus fiable ? Lequel aura un temps réponse acceptable ? Il devient ainsi nécessaire de choisir les services web pertinents parmi ceux trouvés et de fixer des critères pour choisir les meilleurs.

5.3.2. Objectif

Leur travail consiste à développer une application basée sur une approche d'optimisation multi-objective évolutionnaires pour la sélection des meilleurs services web en fonction de QOWS (Quality of Web Service).

5.3.3. Codage (Chromosome)

Le codage utilisé dans ce travail est un codage réel.

5.3.4. Fonction de fitness

La fonction de fitness est partitionnée en en fonction-cout, fonction-latence, fonction-disponibilité, fonction-fiabilité, fonction-réputation et calcule leurs résultats par un système décisionnel.

5.3.5. Résultat

Dans ce travail, ils ont présenté les concepts de l'optimisation multi-objectifs, puis les algorithmes évolutionnaires (AEs) comme étant les meilleurs algorithmes de résolution de problèmes d'optimisation multi-objectifs. Ensuite, on a décrit l'algorithme génétique

NSGAII, qui est utilisé dans ce travail comme algorithme de référence pour les problèmes d'optimisation multi-objectifs.

5.4. Discussion

Après avoir étudié les trois travaux d'application d'AG dans les architectures logicielles nous avons constaté que les travaux ont abordé différentes problématiques et ils n'ont pas abordé la problématique de la scalabilité des systèmes. Cette problématique a été traitée dans notre projet de fin d'étude. Le tableau ci-dessus résume le résultat d'étude de ces travaux.

Tableau II.1 : Résultat comparatif des approches étudié

Titre	Objectif	Cas d'étude	Scalabilité Oui/Non	Codage	Référence
Synthèse génétique de l'architecture logicielle	Synthèse de l'architecture logicielle du système donné sous la forme d'un diagramme de classes UML	Système domestique intelligent	Non	Réel	[17]
Une approche pour la découverte évolutive des architectures logicielles	Découverte de l'architecture sous-jacente des systèmes logiciels représentée avec un diagramme de composants UML 2	/	Non	Réel	[20]
Sélection des services Web à base d'algorithmes génétiques multi-objectifs	Sélection des meilleurs services web en fonction de QOWS	/	Non	Réel	[10]
Notre approche	Planification de l'évolution des architectures logicielles à grandes échelle	Znn.com	Oui	Binaire	/

6. Conclusion

Ce chapitre nous a permis d'acquérir les connaissances de bases sur les métaheuristiques notamment les AG. Nous avons tout d'abord présenté les principes de bases des AG ensuite nous avons passé en revue quelques travaux les plus significatifs basés sur les AG pour résoudre une problématique donnée dans le domaine des architectures logicielles définis.

Chapitre III :

Conception du gestionnaire de l'évaluation basée sur l'AG

1. Introduction

Après l'étude que nous avons menée sur la conception des problèmes à l'aide des AG qui consiste à se concentrer sur deux grandes parties : le codage du chromosome et la fonction de fitness. Nous nous s'intéressons dans ce chapitre à la spécification du problème de l'évolution des systèmes logiciels, en plus particulier à un système très connu, il s'agit du système znn.com [21] sur lequel nous avons appliqué notre approche basée sur l'AG.

2. Rappel de la problématique

Un système autoadaptatif est un système qui ajuste et change automatiquement son comportement et/ou son architecture pendant son fonctionnement en fonction de ses besoins pour à s'ajuster aux changements de son environnement perçus comme des événements externes (augmentation des demandes des utilisateurs par exemple). Bien que cela puisse être accompli dans certaines situations, cette caractéristiques reste toujours hors de portée pour certains systèmes tel que les systèmes à grande échelle (i.e. des systèmes avec un très grand nombre d'entités), où ce genre de systèmes fait face à des difficultés majeures pour trouver l'architecture optimale capable de répondre aux changements, les architectes de ces systèmes vont consacrer tout leur temps et leurs efforts à la découverte manuelle de la conception optimale, car une méthode exacte qui parcourt toute les solutions afin de retrouver la meilleure architecture risque de consommer une grande puissance d'exécution. Pour remédier à cette problématique, notre solution est d'utiliser une méthode approchée basée sur l'AG pour trouver la meilleure stratégie d'adaptation en minimisant le temps nécessaire pour le faire. Nous appliquons notre approche sur un cas d'étude qui est très utilisé comme exemple de système auto-adaptatif. La section suivante présente ce système.

3. Présentation du cas d'étude Znn.com [21]

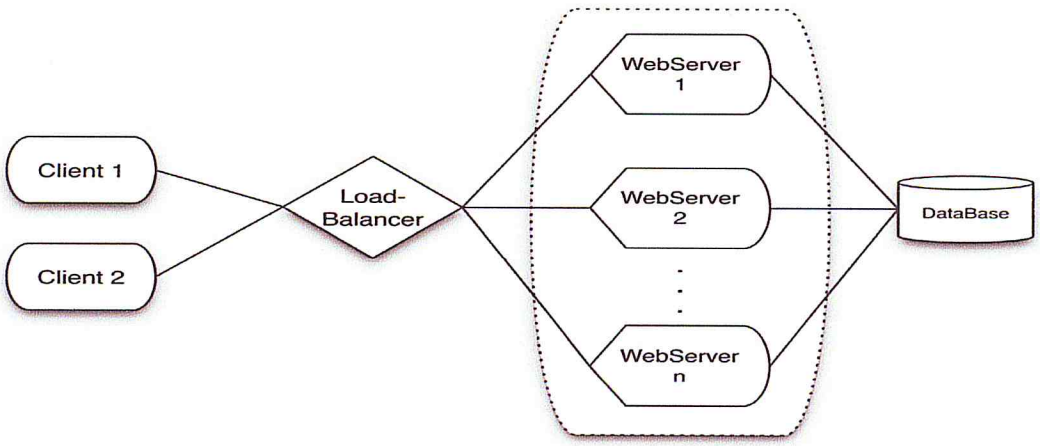


Figure III.1 : Architecteur Znn.com

Afin d'illustrer notre approche nous présentons le système Znn.com [21] .Znn.com est un site de nouvelles qui diffuse du contenu multimédia à ses clients. Du point de vue architectural, Znn.com est un système client-serveur basé sur le Web qui se conforme à un style N-tier. Comme illustré dans la Figure III.1. Znn.com utilise un balanceur de charge pour équilibrer les demandes sur un pool de serveurs répliqués, dont la taille est ajustée dynamiquement pour équilibrer l'utilisation du serveur et le temps de réponse du service. Un ensemble de processus client envoie des demandes de contenu sans état à l'un des serveurs. Supposons que nous puissions surveiller le système pour des informations telles que la charge du serveur et la bande passante des connexions serveur-client. Supposons en outre que nous pouvons modifier le système, par exemple, pour ajouter plus de serveurs au pool ou pour changer la qualité du contenu.

Les objectifs commerciaux de Znn.com consistent à diffuser du contenu d'information à ses clients dans un délai de réponse raisonnable tout en maintenant le coût du pool de serveurs dans son budget de fonctionnement. De temps en temps, en raison d'événements très populaires, Znn.com subit des pics dans les demandes de nouvelles qu'il ne peut pas servir de manière adéquate, même à la taille maximale du pool. Pour éviter les latences inacceptables, Znn.com choisit de proposer un contenu textuel minimaliste pendant ces heures de pointe, au lieu de fournir à ses clients un service inégalé. Les administrateurs système de Znn.com adaptent le système en utilisant deux actions : ajuster la taille du pool de serveurs ou changer de mode de contenu. Lorsque le système est soumis à une charge élevée, les administrateurs système peuvent augmenter la taille du pool de serveurs jusqu'à ce qu'un maximum déterminé par les coûts soit atteint, à partir duquel l'administrateur système fait basculer les serveurs pour servir le contenu textuel.

La décision d'adaptation est déterminée par des observations du temps de réponse moyen global par rapport à la charge du serveur. Plus précisément, quatre stratégies d'adaptations sont possibles, et le choix dépend non seulement des conditions du système, mais aussi des objectifs de l'entreprise :

- Dégrader mode du contenu du serveur.
- améliorer le mode du contenu du serveur.
- Augmenter la taille du pool de serveurs.
- Décrémenter la taille du pool de serveurs

4. Codage du chromosome proposé

Pour appliquer notre solution basée sur l'AG, l'architecture logicielle d'un système est représentée comme un chromosome constitué de gènes. Nous avons suivi l'idée supergène, introduite par Outi Räihä et al [15]. Dans la représentation chromosomique traditionnelle, chaque chromosome se compose de plusieurs gènes, dont chacun a un champ. Un supergène, a donc plusieurs champs pour stocker des données. Prenant cette idée comme point de départ, il est assez simple de placer toutes les informations concernant un serveur dans un supergène. Un supergène SG_i contient deux types d'informations. Tout d'abord, il y a l'information donnée en entrée pour l'état serveur ES_i. Deuxièmement, il y a l'information concernant la fidélité Fi (le mode du contenu) du serveur SR_i. Un supergène SG_i est représenté sous forme d'un vecteur binaire. La figure III.2 représente la structure d'un supergène proposée.

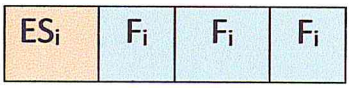


Figure III.2 : Supergene SG_i pour le serveur SR_i

- l'état serveur ES_i prendre deux valeurs $\begin{cases} 0 & \text{si le serveur inactif} \\ 1 & \text{si non} \end{cases}$

- la fidélité F_i prendre les valeurs suivantes :

Si le serveur inactif

- 000

Si non

- 001 pour mode textuel
- 010 pour mode image
- 011 pour mode image haut qualité
- 100 pour mode vidéo
- 101 pour mode vidéo haut qualité

Le chromosome réel est formé en collectant simplement tous les supergènes. La Figure III.3 illustre un chromosome avec n serveur.

SG_1				SG_2				.....	SG_n			
ES_1	F_1	F_1	F_1	ES_2	F_2	F_2	F_2	.....	ES_n	F_n	F_n	F_n

Figure III.3 : La structure du chromosome proposée

5. La population initiale

La population initiale est considérée comme un point de départ pour l'AG. Les générations futures sont au fur et à mesure améliorées afin d'accéder aux meilleures solutions, en appliquant les opérateurs génétiques.

Une bonne initialisation aide d'une manière considérable l'évolution du processus. On trouve plusieurs mécanismes pour générer une population initiale. Parmi ces mécanismes : le tirage aléatoire, les heuristiques ou une combinaison de solution heuristique et l'aléatoire. Dans notre approche, nous allons générer la population initiale de façon aléatoire uniforme. Pour assurer la diversité dans la population initiale, un ensemble de chromosomes a été définis, dont les premier supergènes sont initialisés par : état d serveur actif en mode vidéo, et les autres supergènes prennent des modes différents de manière aléatoire.

6. La fonction de fitness

La fonction de fitness est utilisée pour désigner une fonction qui sert de critère pour déterminer la meilleure solution à un problème d'optimisation (les meilleurs individus de la population). Le but du problème d'optimisation est alors de maximiser cette fonction jusqu'à l'optimum.

Dans notre système nous sommes concentrés sur les trois critères suivants :

- Critère 1 : la minimisation du temps réponse
- Critère 2 : la maximisation de la fidélité
- Critère 3 : la minimisation du cout (le nombre de serveurs actives)

La fonction de fitness $f(X)$ est définit comme étant la somme pondérée de ces trois objectifs (critères), les poids W_i sont présentés sous forme de nombres de types réels associés à chaque critère, et qui traduisent leurs importances relativement à l'exigence de l'utilisateur. La somme de ces poids égal à 1. La fonction fitness $f(X)$ pour le chromosome X de N supergènes peut être exprimée comme si :

$$f(X) = \sum_{i=1}^{N} -W_1 * T_i - W_2 * C_i + W_3 * F_i \qquad (5)$$

Où :

- ✓ *Ti* : le temps de réponse est l'intervalle de temps entre la réception de la requête par serveur *i* et la transmission du résultat.
- ✓ C_i : le coût ou le prix d'exécution est la redevance que l'architecte de logiciel doit payer pour utiliser le serveur *i.*
- ✓ *Fi* : la fidélité est le débit moyen que serveur *i* est invoqué avec une certaine qualité de service.

7. Normalisation des critères

La valeur de la fonction fitness peut être négative en raison de ses critères de qualité multiples négatifs (les objectifs non fonctionnels), nous pouvons éviter de tomber dans cette situation en normalisant les critères pour ne pas avoir une valeur négative. La formule est expliquée en détail dans la section suivante :

7.1. Matrice de limites

En utilisant des critères de qualité, nous définirons une matrice de limites ML. ML donne une plage de valeurs acceptable pour chaque critère et son poids. Chaque colonne représente un critère, représentant respectivement la valeur minimale acceptable, puis la valeur maximale acceptée et enfin son poids.

$$ML = \begin{pmatrix} Q_1^{min} & Q_2^{min} & Q_3^{min} \\ Q_1^{max} & Q_2^{max} & Q_3^{max} \\ w_1 & w_2 & w_3 \end{pmatrix}$$

7.2. La fonction de fitness normalisée

Les serveurs offrant un code fonctionnel similaire se distinguent par leur qualité définie par les propriétés non fonctionnelles. Nous nous concentrons sur trois critères de qualité couramment utilisés : temps de réponse Q1, coût Q2, fidélité Q3.

Puisque Les trois caractéristiques de la qualité ayant des normes différentes, nous devons les normaliser sur la même échelle dans [0.1].

Supposons que $Q(S) = X(S) \cup Y(S)$ où $X(S) = \{Q_3(S)\}$ un ensemble de critères de qualité positifs et $Y(S) = \{Q_1(S), Q_2(S)\}$ ensemble de critères de qualité négatifs. On note une meilleure qualité de X par des valeurs plus élevées calculées par l'équation (6) et une meilleure qualité

de Y par des valeurs plus faibles. Les valeurs des critères de qualité de X (respectivement Y) sont normalisées à X' (respectivement Y') comme :

$$X_i'(S) = \begin{cases} \dfrac{X_i(S) - Q_i^{min}}{Q_i^{max} - Q_i^{min}} & Si\ \ Q_i^{max} - Q_i^{min} \neq 0 \\ 1 & Si\ non \end{cases} \qquad i=3 \qquad (6)$$

$$Y_j'(S) = \begin{cases} \dfrac{Q_j^{max} - Y_j(S)}{Q_j^{max} - Q_j^{min}} & Si\ \ Q_i^{max} - Q_i^{min} \neq 0 \\ 1 & Si\ non \end{cases} \qquad j=1,2 \qquad (7)$$

La fonction fitness *f* est calculée par une simple somme pondérée de valeurs des critères normalisées :

$$f(S) = \sum_{\forall i | Qi \in X} W_i * X_i'(S) + \sum_{\forall j | Qj \in y} W_j * Y_j'(S) \qquad (8)$$

8. L'impact de stratégies

Le poids des attributs qui définissent l'importance relative des dimensions de qualité est indiquée dans le tableau III.1 et on peut remarquer que le temps de réponse est deux fois plus important que le coût ou la fidélité (performance) du système.

Tableau III.1 : Attributs de poids

Dimensions de qualité	Pourcentage
Temps de réponse moyen (TM)	50%
Fidélité moyen (FM)	25%
Coût moyen(CM)	25%

Les impacts sur les dimensions de qualité pour chaque stratégie sont spécifiés dans le tableau III.2.

Tableau III.2 : Impacts sur les dimensions de qualité pour chaque stratégie

Stratégie	TM	FM	CM
ajouter un serveur	-20%	0%	+1.0
Décharger un serveur	+20%*r	0%	-1.0
Augmenter la fidélité	+5%*r	+2%*r	+0,1
réduction de fidélité	-5%*r	-2%*r	-0,1

Il est à noter que dans notre approche, la charge des serveurs (RAM) a un impact sur la dimension qualité (coût et temps de réponse) à la fois en basse fidélité et en inclusion de stratégies de serveur. Pour ce faire, un ratio est spécifié comme le montre le tableau III.3. Les valeurs qui tombent en points intermédiaires sont extrapolées linéairement. Ce ratio est assigné pour provoquer une augmentation ou une diminution (en fonction de la charge du serveur) de l'impact des deux stratégies sur le coût et le temps de réponse. On peut en déduire que plus la charge du serveur est proche de 100%, plus l'impact sur les deux stratégies est plus important.

Tableau III.3 : Ratio de l'impact de la charge sur les stratégies

Valeur (%)	Ratio
100	1
90	0,9
80	0,8
70	0,7
60	0,6
50	0,5
40	0,4
30	0,3
20	0,2
10	0,1

9. Les opérateurs génétiques

9.1. La sélection

La sélection, est un opérateur génétique utilisé dans les algorithmes génétiques pour sélectionner des solutions potentiellement utiles pour la recombinaison. Il existe plusieurs méthodes de sélection, telles que la sélection par rang ou la sélection par la méthode de l'élitisme ou Steady-State etc.

Dans notre approche nous allons utiliser une stratégie de sélection par roulette (Roue de loterie) cette sélection consiste à dupliquer chaque individu (chromosome) C_i proportionnellement à la valeur de la fonction d'adaptation (fitness). Ainsi même les individus les plus faibles ont une chance de survivre.

Si la population des individus est de la taille égale à N, alors la probabilité de sélection d'un individu (C_i) est :

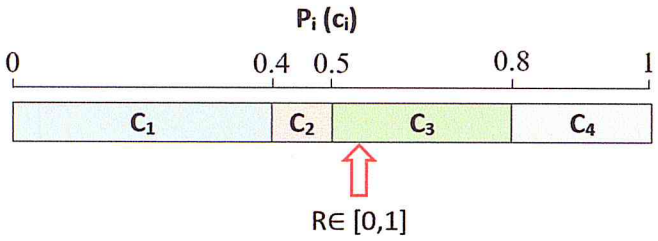
$$P_i\,(C_i) = \frac{f(Ci)}{\sum_{j=1}^{N} f(Cj)} \qquad (9)$$

Où : $P_i\,(C_i)$ est la probabilité de sélection de l'individu xi ;

$f\,(C_i)$ est la fonction de fitness associée à cet individu ;

$\sum f\,(C_j)$ représente la somme des fonctions de fitness de tous les N individus de la population.

La valeur P_i *(Ci)* est dans la gamme [0 à 1]. Cette valeur agit un peu comme la variance de la fitness, dans le sens où plus cette variance est forte et plus la sélection agit de manière efficace. En utilisant cette probabilité de sélection, on peut présenter une roue de loterie sous forme un intervalle délimité entre 0 et 1. Chaque individu de la population occupe une section de l'intervalle proportionnellement à son probabilité de sélection et qui indique aléatoirement quel individu peut se reproduire.



Chromosome C_i	Probabilité P_i (ci)
C_1	0.4
C_2	0.1
C_3	0.3
C_4	0.2

Figure III.4 : Exemple de sélection d'un seul individu

La figure III.4 ci-dessus pourrait être imaginé semblable à une roulette. Où une proportion de l'intervalle est affectée à chacune des individus (chromosomes) possibles en fonction de leur valeur de probabilité de sélection. Ensuite, une sélection aléatoire est faite de la même façon que la roue de la roulette est tournée. Où Un nombre aléatoire uniforme R appartient à l'intervalle [0,1] est choisi et ce nombre donne un individu sur laquelle les autres opérateurs génétiques s'appliqueront. Cela correspond à la boule de roulette tombant dans la section spécifiée pour la l'individu avec une probabilité proportionnelle à sa largeur.

9.2. Le croisement uniforme

Après le choix des deux parents par la méthode de sélection citée ci-dessus on va faire le croisement entre ces deux parents pour obtenir un enfant chromosome.

Les scientifiques ont créé plusieurs méthodes de croisement, mais dans notre cas on utilise le croisement par pourcentage, c'est-à-dire lorsqu'on a deux parents C_1 et C_2 on va faire le croisement de pourcentage de 75%, le chromosome enfant sera composé de 75% de C_1 et le reste de C_2.

9.3. La mutation uniforme

Apres l'étape du croisement, la prochaine génération est construite à partir de la mutation de tous les chromosomes de l'ancienne génération.

La mutation du chromosome se fera par super-gène à l'aide d'un pourcentage de 25% déterminer qui ne pas doit être dépassé.

10. Les critères d'arrêt

Le critère d'arrêt peut être fixé en imposant un nombre maximal de générations (max gen).On estime alors que l'algorithme a convergé et que l'individu de plus forte performance en une des générations correspond à la solution au problème.

11. conclusion

A travers ce chapitre nous avons présenté tout d'abord le cas d'étude sur lequel nous avons testé notre approche. Nous avons également détaillé la conception des deux parties de conception de l'AG proposé c.à.d. le codage des chromosomes et la fonction de fitness auxquels nous avons accordé beaucoup d'importance afin d'assurer la convergence vers les solutions rapidement. Nous avons également présenté la troisième partie de la conception de l'AG qui consiste à pratiquer l'évolution des générations, cette partie est la moins importante dans la phase de conception car elle peut changer après les tests. Le chapitre suivant sera donc consacré à exposer les tests ainsi que les résultats obtenus de nos expérimentions.

Chapitre VI :

Implémentation de l'application et tests

1. Introduction

Dans le chapitre précédent nous avons présenté notre approche pour la gestion d'évolution d'architectures logicielles, nous allons à présent présenter les étapes d'implémentation de notre approche, on a commençant par d'écrire l'environnement et les outils de travail utilisés pour sa réalisation ensuite nous allons décrire les expérimentations menées ainsi que les résultats obtenus.

2. Environnement et outils de travail

Le choix des outils de travail pour notre travail de fin d'étude n'été pas facile vue la variété des technologies existantes, notre choix était fait par rapport aux nos objectifs.

2.1 L'environnement de développement Eclipse

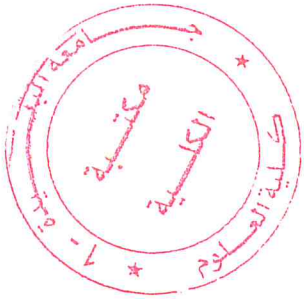
Eclipse[2] est un environnement de développement intégré libre, puissant, extensible, universel et polyvalent, permettant potentiellement de créer des projets de développement mettant en œuvre n'importe quel langage de programmation.

Il est développé par IBM il est gratuit est disponible pour la plupart des systèmes d'exploitation, il est principalement écrit en java, il offre une plateforme ouverte pour le développement d'applications, il est facile à comprendre mais aussi facile à étendre.

Il existe plusieurs versions d'Eclipse on note : Eclipse 2.0, Eclipse 3.0, Eclipse 3.2 Europa, Eclipse 3.5 Indigo, Eclipse 4.2 Juno, Eclipse Mars, Eclipse Oxygen, dans notre cas on a utilisé Eclipse Oxygen.

2.2 JavaFX

JavaFX[3] est une bibliothèque Java utilisée permettant de créer et de fournir des applications de bureau , ainsi que des applications Internet riches. Les applications écrites à l'aide de cette bibliothèque peuvent être exécutées de manière cohérente sur plusieurs plates-formes. Les applications développées à l'aide de JavaFX peuvent s'exécuter sur divers périphériques tels que les ordinateurs de bureau, les téléphones mobiles, les téléviseurs, les tablettes, etc.



[2] https://www.eclipse.org
[3] https://www.tutorialspoint.com/javafx/javafx_overview.htm

Pour développer des applications GUI (*Graphical User Interface*) utilisant le langage de programmation Java, les programmeurs s'appuient sur des bibliothèques telles que Advanced Windowing Toolkit et Swing. Après l'avènement de JavaFX, ces programmeurs Java peuvent désormais développer des applications d'interface graphique avec un contenu riche.

2.3 Scene Builder

JavaFX fournit une application nommée Scene Builder[3]. En intégrant cette application dans les IDE tels que Eclipse et NetBeans, les utilisateurs peuvent accéder à une interface de conception par glisser-déposer, utilisée pour développer des applications FXML (tout comme les applications Swing Drag & Drop et DreamWeaver).

2.4 JFoenix

JFoenix[4] est une bibliothèque Java open source, qui implémente GOOGLE Matériel Design en utilisent des composant Java. Pour commencer à utiliser JFoenix, il suffit de le télécharger depuis GitHub.

3. Algorithmes de simplification (Douglas-Peucker)

L'option Conserver les points critiques (Douglas Peucker[5]) offre un algorithme de simplification de ligne simple et rapide. Elle conserve les points critiques qui décrivent la forme globale d'une ligne et supprime tous les autres points. L'algorithme commence par connecter les extrémités d'une ligne à une ligne tendance. La distance de chaque sommet par rapport à la courbe de tendance est mesurée perpendiculairement. Les sommets plus proches de la ligne que la tolérance sont éliminés. La ligne est ensuite divisée par le sommet le plus éloigné de la ligne tendance, ce qui a pour effet de créer deux lignes de tendance. Les sommets restants sont mesurés par rapport à ces lignes et le processus continu jusqu'à ce que tous les sommets qui se situent dans la plage de tolérance soient éliminés. Cette option repose sur l'algorithme défini par Douglas et Peucker (1973).

[3] https://www.tutorialspoint.com/javafx/javafx_overview.htm
[4] http://www.jfoenix.com/documentation.html
[5] https://pro.arcgis.com/fr/pro-app/tool-reference/cartography/how-simplify-line-works.htm

4. Présentation de l'application

L'application de notre projet ainsi que les tests effectués sont réalisés sous Eclipse Oxygen et JDK-9.0.1-windows10 x64 et processeur intel core i5 6ème génération avec une RAM 4GB DDR3. Au lancement de notre application, l'utilisateur verra s'afficher les interfaces suivantes.

4.1. Interface d'accueil

Lorsque l'application est exécutée, une interface de départ (d'accueil) apparaît à l'écran pour présenter le thème de notre projet. Cette interface permet également de charger le menu principal qui se situe à gauche de notre interface qui permet d'accéder aux points suivants : Serveur ; algorithme génétique ; simulation et quitter pour fermer l'application comme le montre Figure IV.1

Figure VI.1 : Interface d'accueil

4.2. Interface serveur

Cette interface comme le montre la Figure IV.2, permet à l'utilisateur à saisir les informations concernant les serveurs à savoir : le nombre des serveurs total ; nombre des serveurs allumés ; capacité de serveur (RAM), qui vont être utilisés dans les simulations.

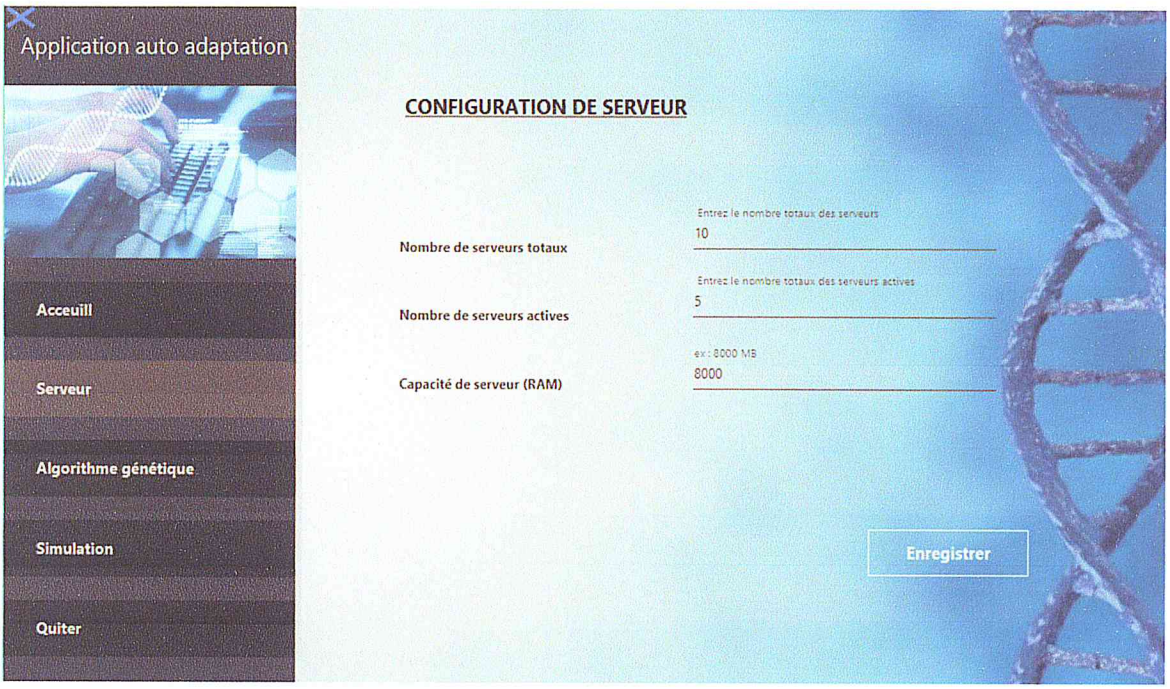


Figure IV.2 : Interface serveur

4.3. Interface algorithme génétique

Afin d'examiner le comportement de notre algorithme, et examiner la convergence de la fonction de fitness, nous devons passer par l'étape de la configuration pour choisir les paramètres de simulation de l'AG. Notre application permet la configuration à partir de l'interface d'AG présenté dans la Figure IV.3.

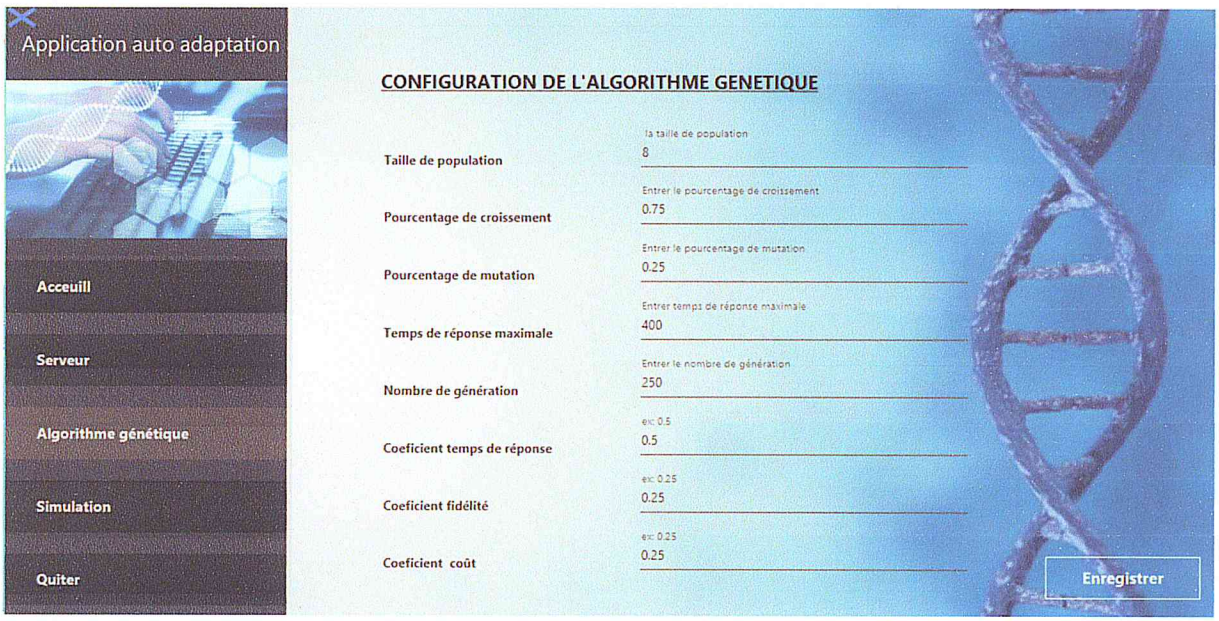


Figure IV.3 : Interface algorithme génétique

4.4. Interface simulation

Notre approche doit passer par l'étape de configuration pour choisir les paramètres de simulation, notre application permet de configurer les paramètres de simulation comme le temps de début de simulation ; la granularité et le temps de fin comme il est illustré dans la Figure IV.3. Après avoir rempli tous les champs, en cliquent sur les boutons « Sans adaptation » ou « Avec algorithme génétique » ou « Avec ME » (méthode exacte) qui se situe en bas de notre interface afin de lancer la simulation sans adaptation ou avec algorithme génétique ou simulation avec méthode exacte, puis une autres interface « courbes graphique « s'affichera pour chaque simulation. Lorsque toutes les simulations sont terminées, en cliquent sur le bouton « Statistiques » pour comparer entre les simulations précédentes sous forme de graphes de et un diagramme à band.

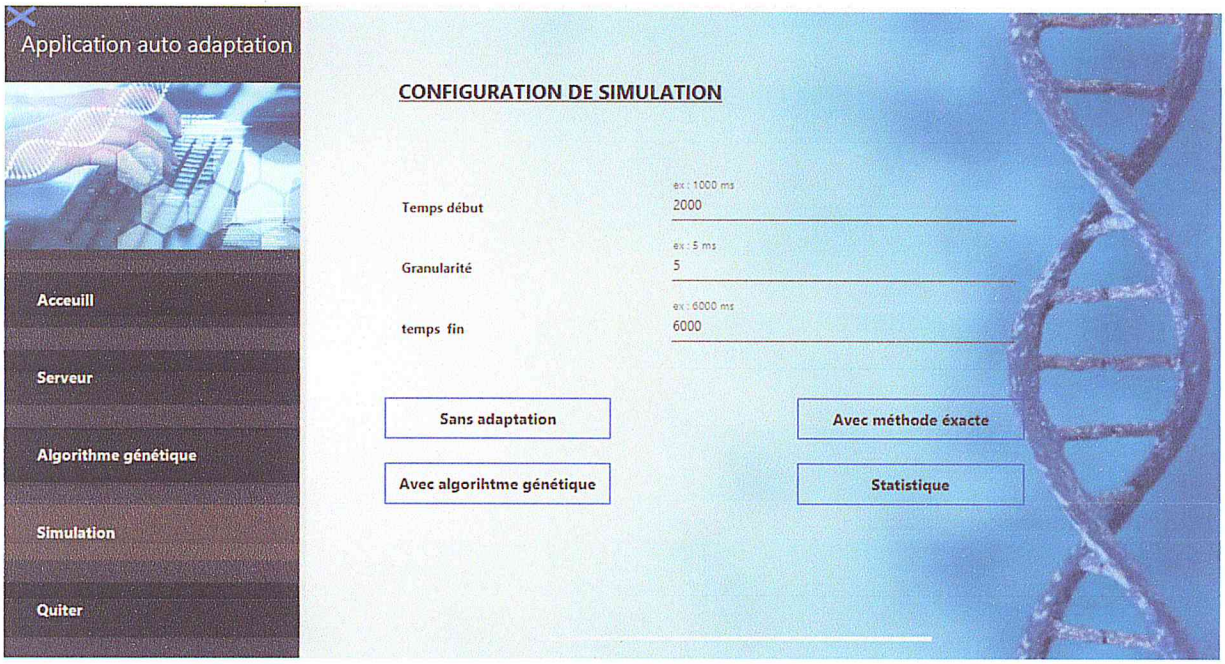


Figure IV.4 : Interface simulation

4.5. Interface des courbes graphiques

Après chaque simulation, notre application permet de générer des graphes depuis quatre interfaces des courbes graphiques à l'aide de la bibliothèque JLineChart et JBarChart, ces courbes permettent de voir les caractéristiques et les résultats de chaque simulation, en représentant la courbe de temps de réponse moyen; fidélité moyenne ; charge moyenne en fonction de temps d'exécution et aussi le diagramme à bande seulement dans la simulation avec algorithme génétique en cliquant sur l'une des courbes comme il est montré dans la Figure IV.5.

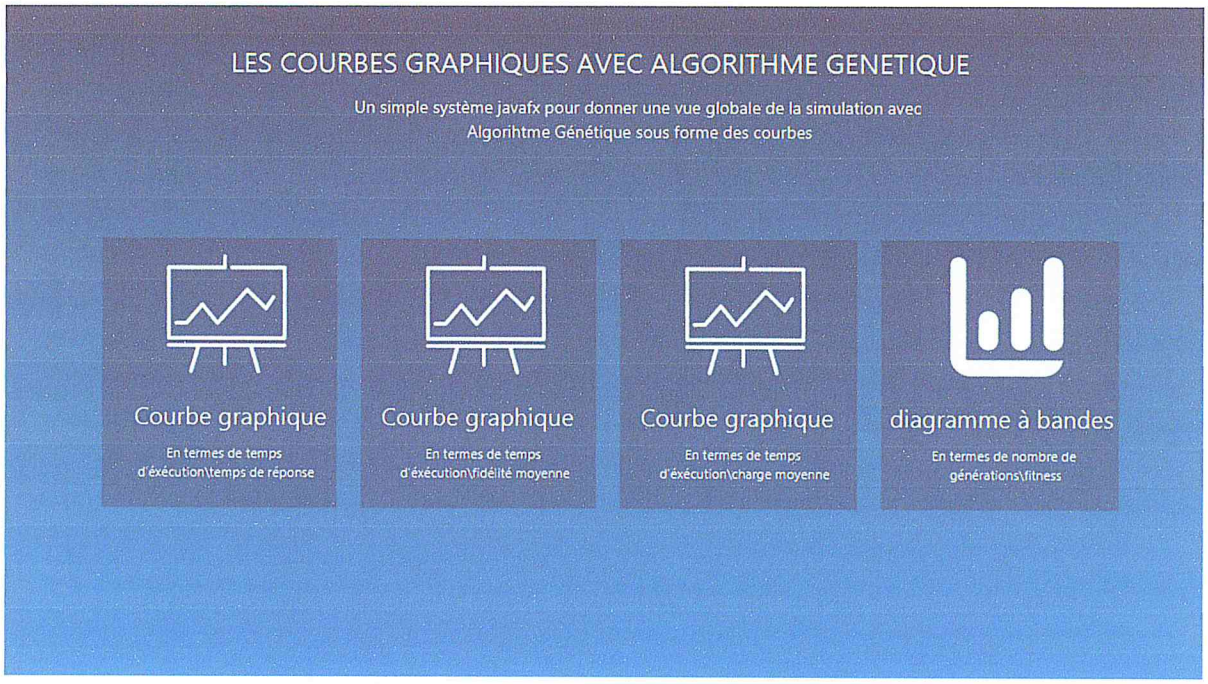


Figure IV.5 : Interface des courbes graphiques

4.6. Interface des courbes statistiques

Cette interface comme indiqué dans la Figure IV.6 ci-dessous, permet de générer les courbes de temps d'exécution en fonction de nombre de serveurs et le cout en fonction temps d'exécution et un diagramme à bande de fitness en fonction nombre de génération, ces courbes nous permettent de comparer et d'observer la différence entre les simulations.

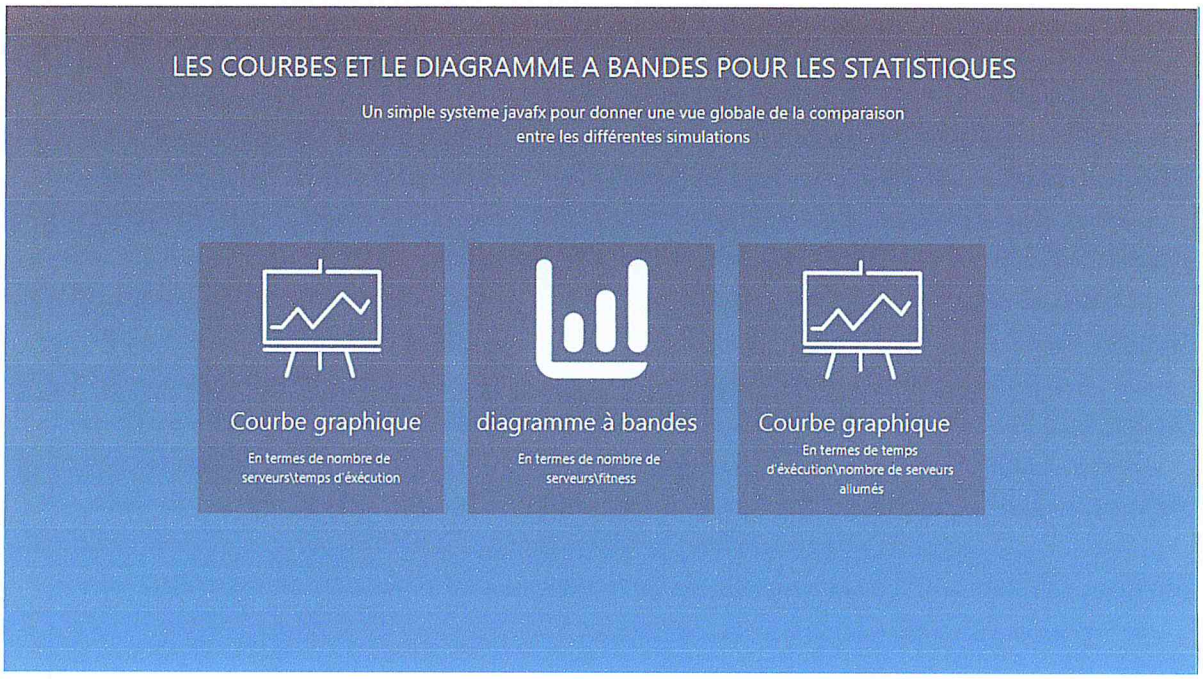


Figure IV.6 : Interface des courbes statistiques

5. Expérimentions et résultats

Cette section présente les résultats des essais expérimentaux que nous avons effectués sur le cas d'étude Znn.com, en appliquant trois approches différentes qui sont : approche sans adaptation ; approche basée sur l'algorithme génétique ; approche basée sur une méthode exacte. Les simulations sont effectuées sur trois approches pour pouvoir les comparés selon les critères suivants :

- ✓ Temps de réponse
- ✓ Fidélité
- ✓ charge

Ces tables illustrent les paramètres que nous avons utilisés dans les trois simulations :

- Pour les serveurs :

Tableau IV.1 : Paramètres de serveurs

Paramètre	*Valeur*
Nombre de serveurs total	10
Nombre de serveurs allumés	5
Capacité (RAM)	8000 MB

- Pour l'algorithme génétique :

Tableau IV.2 : Paramètres de l'algorithme génétique

Paramètre	*Valeur*
Taille population	8
Pourcentage de croisement	0.75
Pourcentage de mutation	0.25
Temps de réponse maximal	400
Nombre de génération	250

- Pour la simulation :

Tableau IV.3 : Paramètres de la simulation

Paramètre	*Valeur (ms)*
Temps de début	2000
Granularité	10
Temps de fin	4000

Cette section démontre les résultats d'un premier essai de la simulation dans un tracé en temps réel sans adaptation, sachant que nous avons trois graphs représentent évolution de temps de réponse, fidélité moyenne, et de la charge moyenne en fonction de temps d'exécution successivement pour la même expérience.

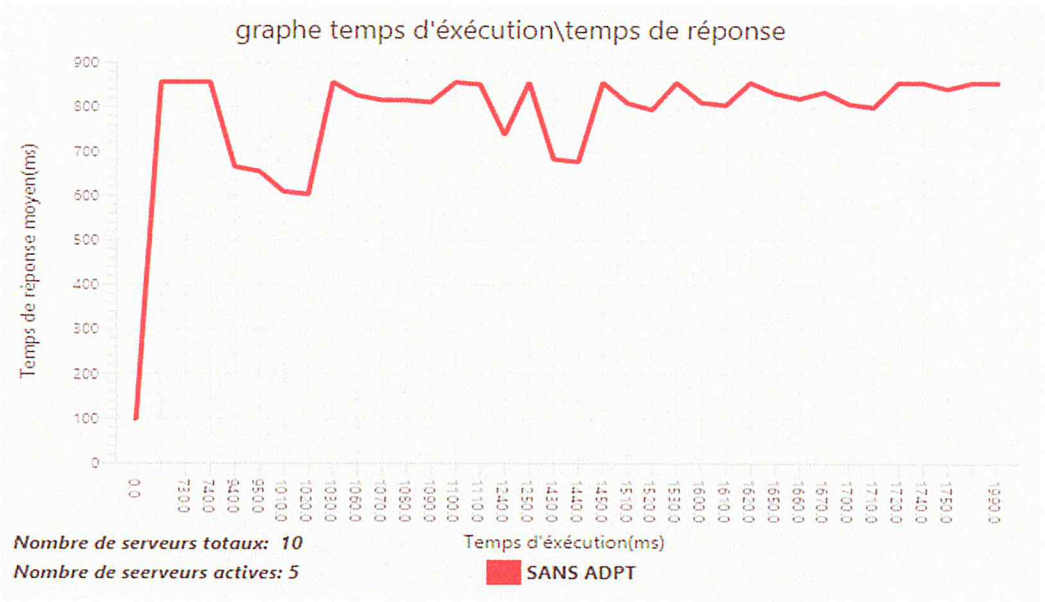


Figure IV.7 : Temps de réponse en fonction de temps d'exécution (sans adaptation)

Ce tracé en temps de réponse comme indiqué sur la Figure IV.7 montre clairement que sans adaptation il y'a eu un très grand dépassement par rapport le temps de réponse qui normalement ne doit pas dépasser le seuil maximal de (400 ms).

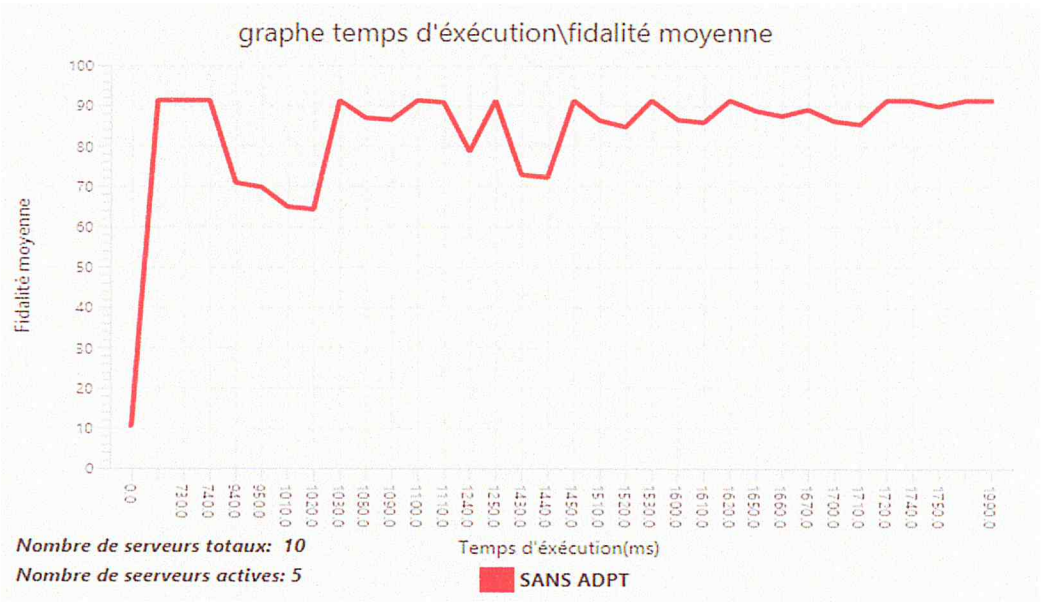


Figure IV.8 : Fidélité moyenne en fonction de temps d'exécution (sans adaptation)

La figure IV.8 représente la fidélité moyenne en fonction du temps. Il peut être remarqué le rendement relativement fiable qui ne répond pas aux besoins nécessaires système étudié

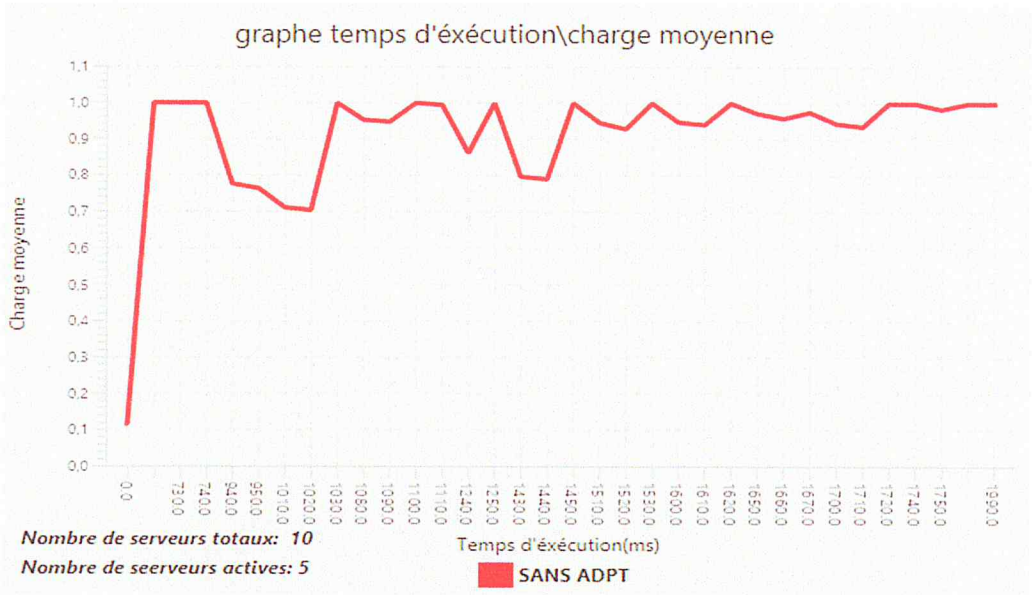


Figure IV.9 : Charge moyenne en fonction de temps d'exécution (sans adaptation)

Comme nous pouvons le constater à partir de ce troisième graphe la valeur de pourcentage de la charge moyenne est presque maximale (100%) toute au long du temps d'exécution .Ce qu'il s'agit de l'incapacité du système à avoir plus de requêtes d'utilisateurs.

Dans cette deuxième section, les résultats expérimentaux d'un deuxième essai sont présentés pour montrer la validité et la performance de notre approche basée sur l'AG, les figures suivantes présentes les résultats obtenus pour les trois tests effectués comme dans la première approche.

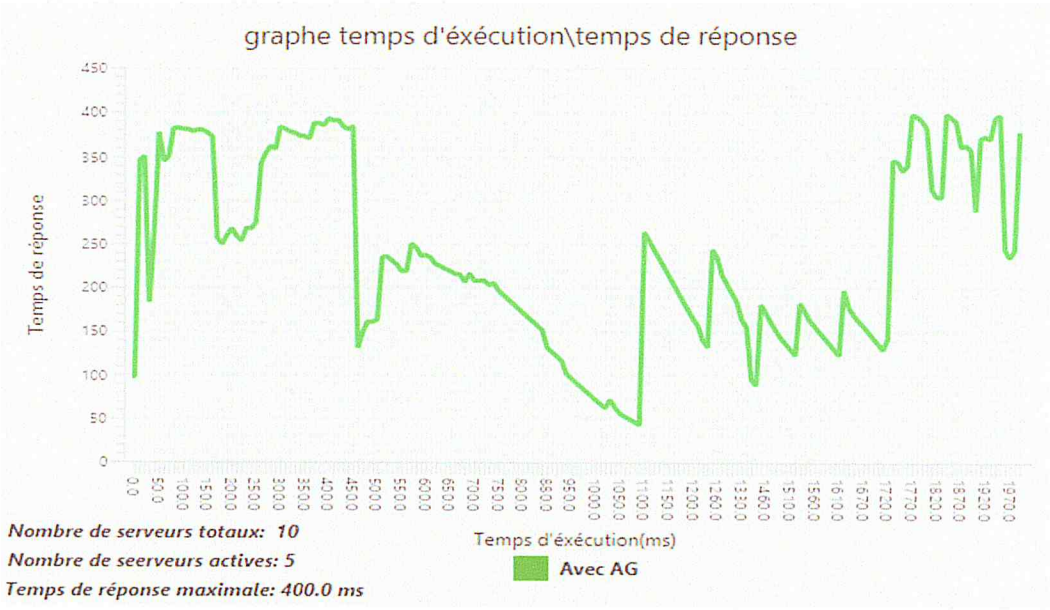


Figure IV.10 : Temps de réponse en fonction de temps d'exécution (avec AG)

Comme nous le remarquons dans ce tracé de la Figure IV.10 Le résultat obtenu est satisfaisant où le temps de réponse est raisonnable et ne dépasse pas le seuil max ce qui répond parfaitement aux exigences du système. Nous pouvons conclure que notre approche a bien aidé le système Znn.com pour s'adapter aux grand nombre de requêtes d'utilisateurs.

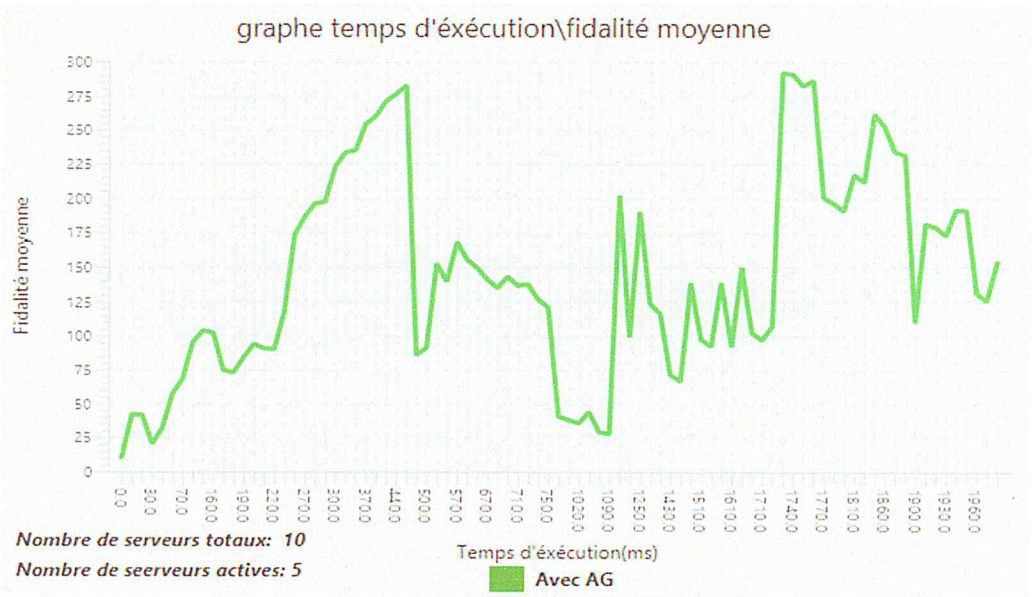


Figure IV.11 : Fidélité moyenne en fonction de temps d'exécution (avec AG)

La figure ci-dessus présente la a fidélité moyenne, selon les résultats obtenus on remarque un rendement de fidélité, mais le système fait des oscillations légères avec le temps pour satisfaire le plus grand du demandes d'utilisateurs (c.à.d que les serveurs change leur qualité de contenu).

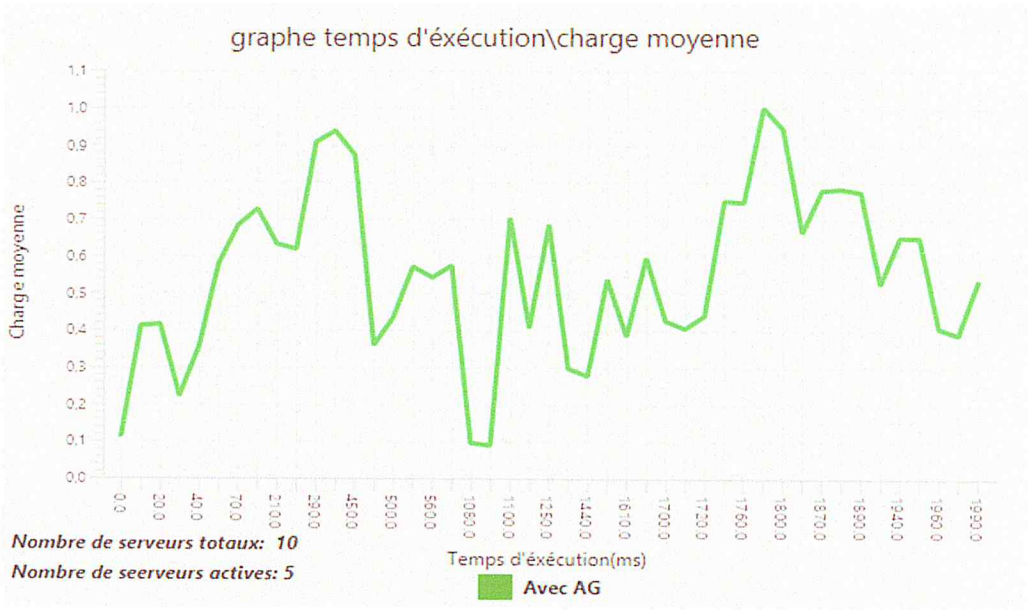


Figure IV.12 : Charge moyenne en fonction de temps d'exécution (avec AG)

Dans ce troisième graphe nous pouvons constater que la charge moyenne prend des valeurs réduites avec un pourcentage moyen, loin de l'extrême (100%).Ce qui indique qu'avec notre approche proposée, le système sera plus capable d'avoir plus de requêtes d'utilisateurs.

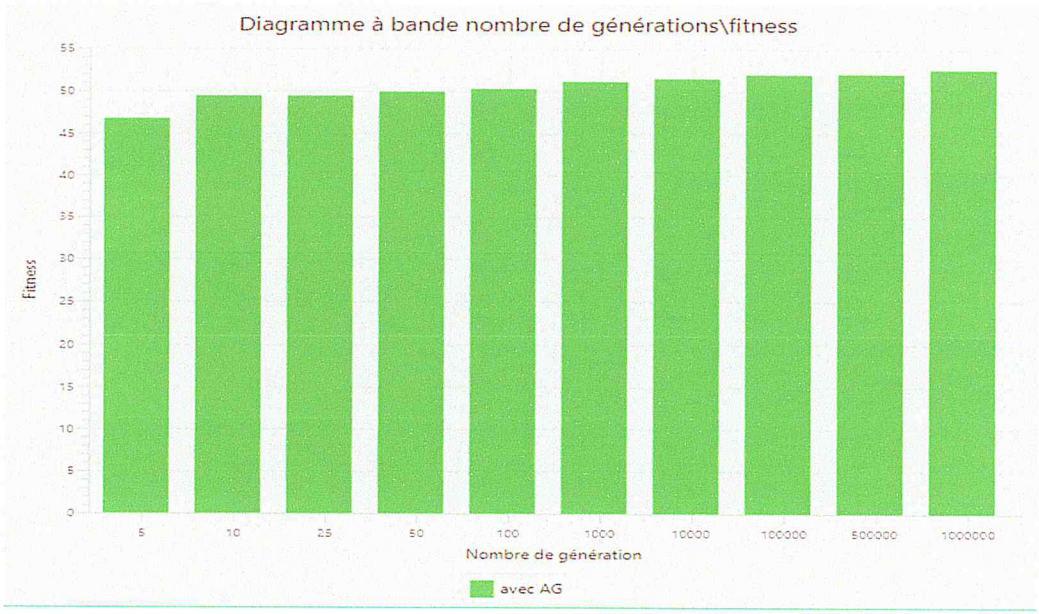


Figure IV.13 : Fitness en fonction de nombre de génération (avec AG)

Le Figure IV.13 présente les résultats de dix tests effectués en augmentant le nombre de génération pour trouver que la solution converge vers les 1000 générations pour un nombre de serveur = 100.

Dans cette dernière section, les résultats expérimentaux sont présentés pour montrer la performance de notre approche basée AG comparé à une approche basée sur ME (méthode exacte).

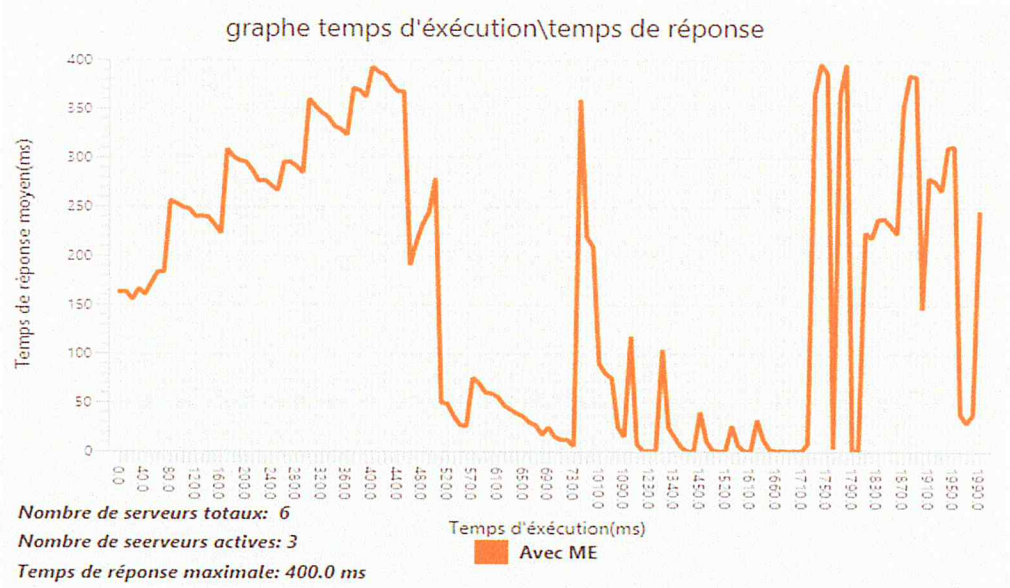


Figure IV.14 : Temps de réponse en fonction de temps d'exécution (avec ME)

Comme indiqué sur les Figures IV.14 IV.15 et IV.16. Le résultat obtenu est très satisfaisant et optimal par rapport aux autres expériences présidentes vu que l'approche vise à choisir la solution optimale pour adapter le système.

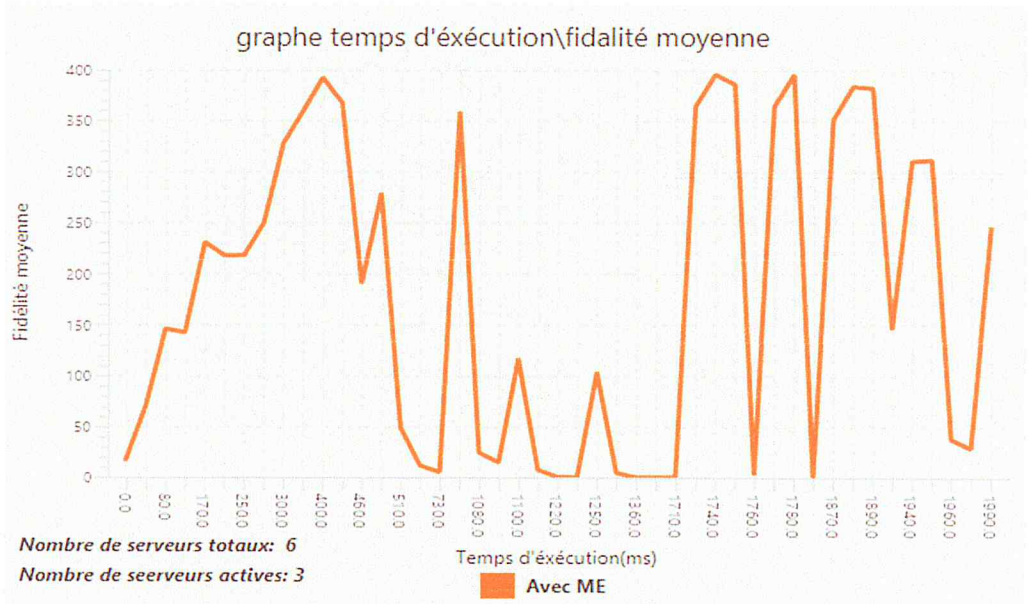


Figure IV.15 : Fidélité moyenne en fonction de temps d'exécution (avec ME)

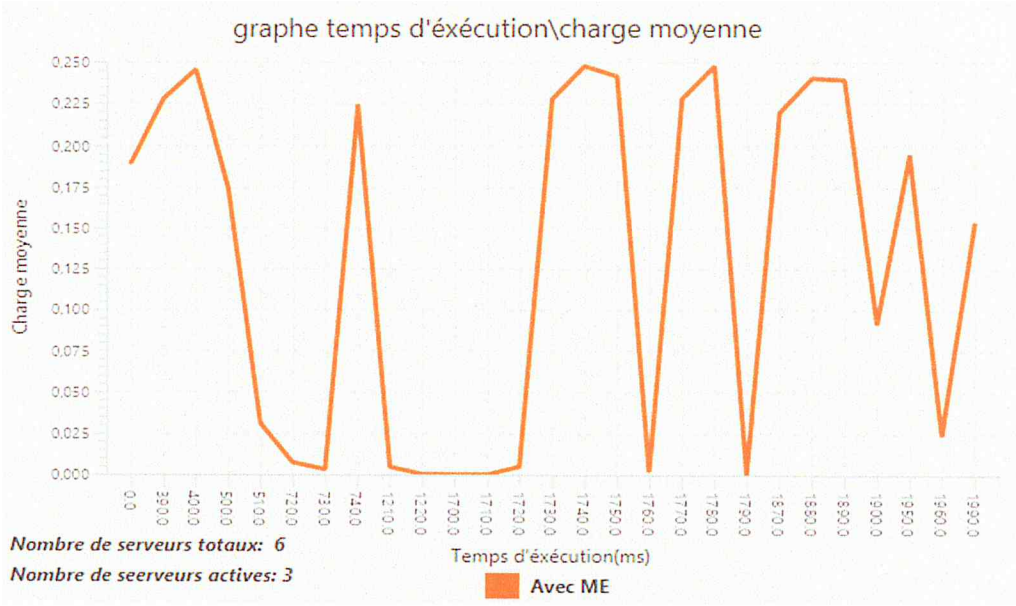


Figure IV.16 : Charge moyenne en fonction de temps d'exécution (avec ME)

6. Etude comparative des performances de différentes approches

Dans cette section, Nous allons comparer les performances des différentes approches utilisées.

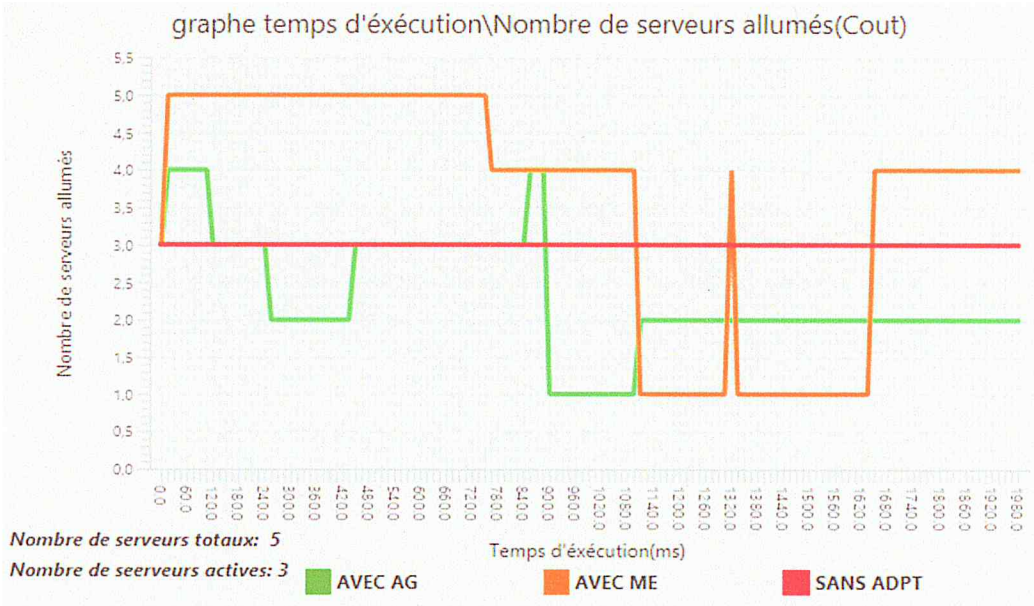


Figure IV.17 : Nombre de serveurs allumés (cout) en fonction de temps d'exécution

La figure IV.17 montre une comparaison du nombre de serveurs allumés (cout) en fonction du temps d'exécution des trois approches (AG, ME, sans ADPT. Nous constatons que notre approche basée sur l'AG obtient des résultats différents de ceux de l'approche avec ME mais satisfait les besoins de l'architecture.

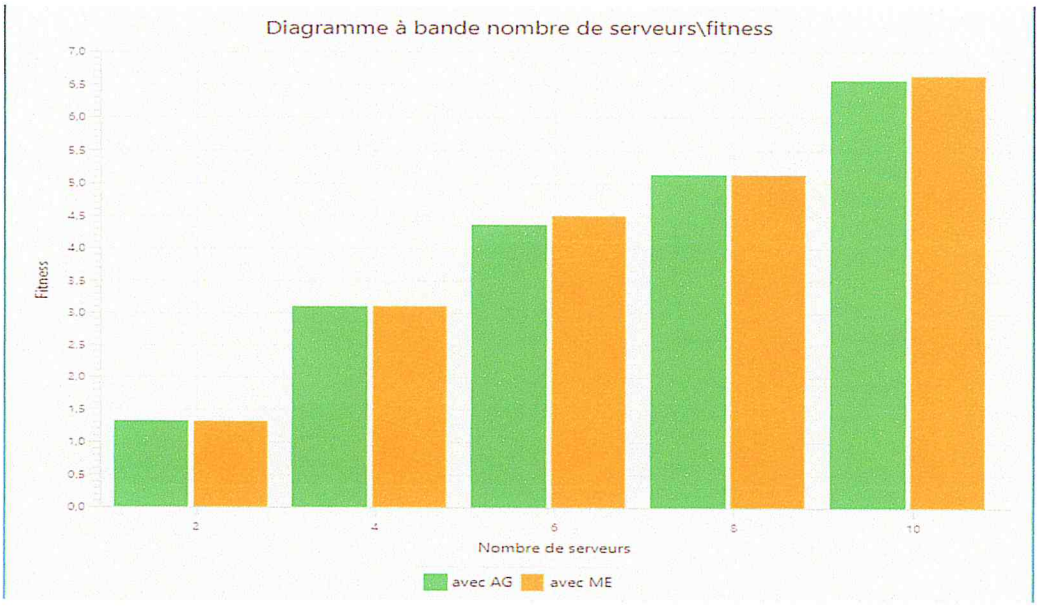


Figure IV.18 : Fitness en fonction de nombre de serveurs

Ce diagramme comme indiqué sur la Figure IV.18 présente un aperçu comparatif entre les valeurs de la fonction de fitness des deux approches d'adaptation appliquées sur notre système AG, ME). Les résultats obtenus après plusieurs tests montrent que les deux approches donne des valeurs de fitness approchées ce qui montre que notre approche converge aussi bien que la ME vers la solution optimale.

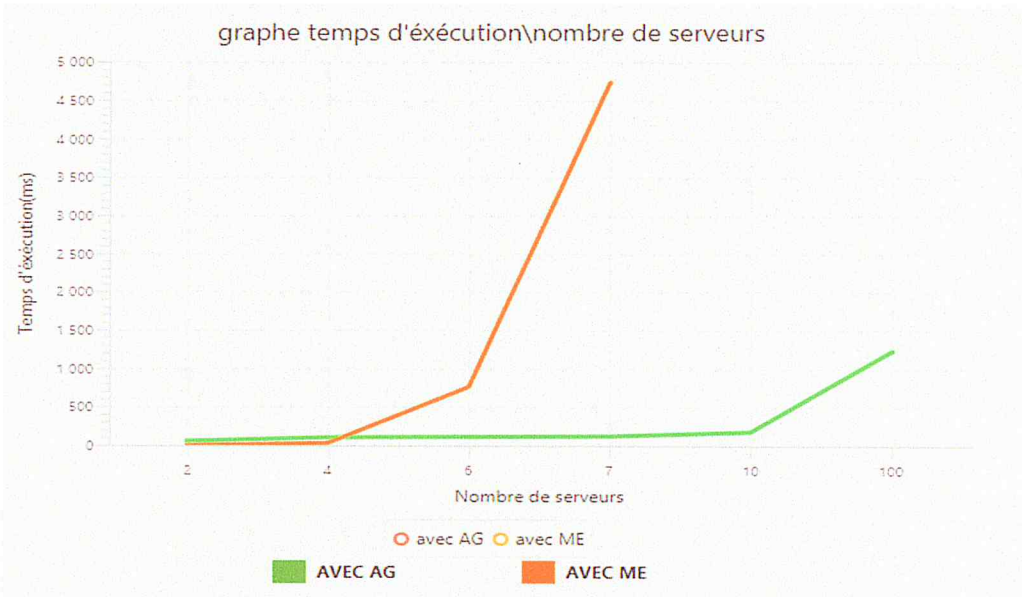


Figure IV.19 : Temps d'exécution en fonction de nombre de serveurs

La figure IV.19 montre une comparaison entre les performances des différents algorithmes qui ont été successivement appliqués sur notre architecture logicielle (AG, ME). Ce teste étudie la scalabilité des deux approches, en mesurant le temps d'exécution nécessaire pour chaque approche pour trouver le bon plan d'adaptation tout en variant le nombre de serveurs.

Les résultats obtenus après plusieurs tests montrent que le temps d'exécution de l'approche basée sur ME accroit d'une manière exponentielle ce qui nous a empêché de faire un test de plus de 7 serveurs, alors que dans notre approche basée sur l'AG donne de très bon résultat est réduit le temps d'exécution d'une manière comparée à la ME. Le temps nécessaire pour notre approche pour trouver un plan d'adaptation pour un nombre de serveurs égal à 100 ne dépasse pas une seconde. Nous pouvons conclure que notre approche s'apprête bien pour les systèmes à grandes échelles et offre une bonne solution au problème de la scalabilité.

7. Conclusion

Le présent du chapitre constitue la dernière étape pour présenter notre projet dans ce mémoire, il a été dédié à la présentation de l'application développée ainsi que l'environnement et les outils de travail utilisés. Une grande partie du chapitre a été consacrée aux déférents tests effectués ainsi que les résultats obtenus qui montre bien que notre approche basée sur AG s'accommode très bien à l'autoadaptation pour les systèmes à grande échelles sans nuire à leur bon fonctionnement en réduisant clairement le temps d'exécution nécessaire pour le choix d'une stratégie d'adaptation.

Conclusion générale

Dans ce mémoire nous avons présenté l'ensemble des travaux réalisés dans le cadre de notre projet de fin d'étude. Nous avons tout d'abord présenté l'état de l'art auquel s'inscrit notre problématique à savoir l'architecture logicielle et mettant l'accent sur les architectures logicielles dynamiques. Par la suite, nous avons présenté brièvement quelques métaheurstiques en se basant principalement sur AG que nous avons utilisé dans notre approche. Nous avons présenté notre approche d'autoadaptation de l'architecture logicielle basée sur AG qui vise à choisir le meilleur plan d'adaptation de l'architecture en moindre coût, et cela grâce à la modélisation que nous avons présenté dans le chapitre III. Nous avons développé une application java pour tester notre approche et les résultats obtenus sont très satisfaisante (en terme de temps d'exécution) pour l'autoadaptation des systèmes à grande échelles en remédiant au problème de la scalabilité sans nuire à la performance de ces systèmes.

Nous projetons de poursuivre le développement du processus d'automatisation de notre approche d'adaptation des architectures logicielles des systèmes en utilisant d'autres algorithmes méta-heuristiques, comme l'algorithme PSO (Particle swarm optimization). Aussi, nous avons comme perspective de généraliser notre application à d'autres problèmes réels, autre que le problème Znn.com.

Bibliographie

[1] **Clements P.** *Documenting Software Architectures*, Views and Beyond. Addison-wesly Publishing Reading. Massachusetts, USA, 2003.

[2] **Francisco Jose Moomena.** *Modélisation des architectures logicielles dynamiques, Application à la gestion de la qualité de service des applications à base de services Web*, thèse du Doctorat, l'Institut National Polytechnique de Toulouse France, 2007.

[3] **Mazeiar S, Ladan T.** *Self-adaptive software: Landscape and research challenges*, Journal, University of Waterloo, Waterloo, Canada, May 2009.

[4] **Oussalah M, Sadou N, Tamazilt D**. *A generic Model for managing software architecture evolution.* ICS'05 Proceedings of the 9th WSEAS International Conference on Systems Article No. 35, 2005.

[5] **Costa-Soria C.** *Dynamic evolution and reconfiguration of software architectures through aspects*, PhD thesis in Computer Science, Polyticnical University of Valancia, 2011.

[6] **Ben messaoud Z, Haibaoui F.** *Editeur graphique pour l'aide à la spécification des architectures logicielle dynamique*, mémoire de master en informatique, Université Saad Dahlab de Blida, 2016.

[7] **Wermelinger, Michel.** *Specification of software architecture reconfiguration*, University nova de Lisbon, Juin 1999.

[8] **Guessoum D, Bennouar D.** *Dynamic Software Architecture*, an Overview, Université Saad Dahlab de Blida, Université Bouira.

[9] **Clarisse D, El-Ghazali T.** *Optimisation Multi-critère : approche par métaheuristiques*, Laboratoire d'Informatique Fondamentale de Lille, Université de Lille 1, France.

[10] **Belkhiter Khalid.** *Sélection des services Web à base d'algorithmes génétiques multi-objectifs*, mémoire de master, Université Abou Bakr Belkaid Tlemcen, page 55, 2012.

[11] **FRIDJAT Zineddine, TAMMA Mohammed Elhadi.** *Application des algorithmes génétiques à l'optimisation de la production énergie active dans réseau électrique*. Mémoire de master académique, Université d'EL-Oued, 2014.

[12] **AHMED CHAMSEDDINE, BEN ABDALLAH.** *Optimisation multi-objectif évolutionnaire*, Mémoire de master académique, école polytechnique de Tunisie, 2005.

[13] **M FATAH Amir.** *Etude du fonctionnement des centres de production dans un système de marché libre de l'énergie électrique*, Diplôme de Magistère en électrotechnique, Faculté des sciences, Université de Batna, promotion 2011/2012.

[14] **Mr SOULI YASSINE.** *Evolution dynamique des systèmes logiciels par approche phylogénétique*, Magister en Informatique, Ecole Doctorale de l'Est GL & IA Pole Annaba ,2010.

[15] **Mitchell M.** *An Introduction to Genetic Algorithms*, A Bradford Book The MIT Press, Cambridge, Massachusetts, London, England, Page 128, 1999.

[16] **Tebib H, Titi Z.** *Modélisation par les réseaux de neurones optimisés par les algorithmes génétiques*, département d'électronique, Université L'ARBI BEN M'HIDI Oum El Bouaghi.

[17] **Outi Räihä, Kai Koskimies, Erkki Mäkinen.** *Genetic Synthesis of Software Architecture*, Tampere University of Technology, Finland, 2008.

[18] **Sahraoui H.A, Godin R, Miceli T.** *Can metrics help bridging the gap between the improvement of OO design quality and its automation*, In: Proc. ICSM '00, page 154-162, 2000.

[19] **Chidamber S.R, Kemerer C.F.** *A metrics suite for object oriented design*, IEEE Transactions on Software Engineering 20, page 476-492, (1994).

[20] **Aurora Ramirez, José Raúl Romero, Sebastián Ventura.** *An approach for the evolutionary discovery of software architectures*, University of Córdoba, 14071, Spain, 2014.

[21] **Francoa JM, Correiaa F, et al.** *Improving self-adaptation planning through software architecture-based stochastic modeling, The Journal of Systems and Software,* Center for Informatics and Systems of University of Coimbra, Portugal, July2015.