ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAITRISE EN GÉNIE
M.Ing.

PAR
Sébastien GAGNÉ

AUTOMATISATION DE LA RECHERCHE ET DE L'APPLICATION DE TACTIQUES DE PERFORMANCE SUR DES ARCHITECTURES LOGICIELLES

MONTRÉAL, LE 20 DÉCEMBRE 2010

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE

M. Roger Champagne, directeur de mémoire
Département de génie logiciel et des TI à l'École de technologie supérieure

Mme Sylvie Ratté, présidente du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

Mme Ghizlane El Boussaidi, membre du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 14 DÉCEMBRE 2010

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

J’aimerais tout d’abord remercier Roger Champagne, mon directeur de mémoire et mentor lors de cette aventure. Il était une source de conseils, de soutien, et les remarques qu’il m’offrait étaient une aide précieuse. J’aimerais aussi remercier les membres du jury qui ont pris le temps de lire et d’évaluer mon mémoire et ma soutenance. Un merci tout spécial va aussi à Andres Diaz Pace du SEI pour le support sur ArchE qu’il nous a gracieusement offert.

De plus, j’aimerais remercier l’ÉTS et le FQRNT pour le support financier qu’ils m’ont offert avec leur programme de bourses. Ils m’ont permis de me consacrer pleinement au projet sans crainte de problèmes financiers.

En terminant, j’aimerais remercier mes parents, Richard Gagné et Jocelyne Croisetière, pour avoir soutenu mes décisions; ma conjointe Manon Maheux, pour l'aide et la motivation au quotidien; mon frère Alexandre Gagné, pour le soutien spirituel et la motivation à vivre sa vie immédiatement et mon ami, Alexandre Laroche, qui a su me motiver dans les moments plus difficiles.

Merci à vous tous. Il n’y aurait pas eu de maitrise sans vous.

AUTOMATISATION DE LA RECHERCHE ET DE L'APPLICATION DE TACTIQUES DE PERFORMANCE SUR DES ARCHITECTURES LOGICIELLES

Sébastien GAGNÉ

RÉSUMÉ

ArchE est un système expert développé par le Software Engineering Institute conçu pour assister un concepteur lors de l'élaboration d'une architecture logicielle. Actuellement, il contient deux cadres de raisonnement; il est donc en mesure de raisonner à propos de deux attributs de qualité : la modificabilité et la performance.

Le but de ce projet est de développer une version automatisable d'une série de tactiques architecturales de performance et de les intégrer dans le cadre de raisonnement de performance d'ArchE. Cette intégration permet de valider la faisabilité de l'automatisation des tactiques de performance.

Une solution pour l'intégration des tactiques a été développée en tenant compte des points forts et des défis de l'implémentation du cadre de raisonnement de modificabilité. Ensuite, en utilisant la description des tactiques du livre « Software Architecture in Practice » (Bass, Clements et Kazman, 2003), une série de règles ont été développées pour déterminer si, et comment, chaque tactique doit s'appliquer automatiquement sur l'architecture logicielle. Quatre tactiques ont été intégrées : l'augmentation des ressources disponibles, la réduction du temps d'exécution d'une responsabilité, l'augmentation de la période d'un scénario et l'augmentation de la priorité d'une responsabilité.

La validation a permis de déterminer que les tactiques fonctionnent correctement et qu'elles sont suggérées et appliquées automatiquement sur une architecture logicielle. On en conclut que les tactiques de performance peuvent être automatisées dans un système expert. Il faut cependant tenir compte du contexte et des limitations du système expert utilisé, car ils peuvent limiter les capacités d'automatisation des tactiques architecturales de performance. Deux contributions ressortent de ce mémoire : une conception plus simple pour l'intégration des tactiques et une série des règles pour automatiser certaines tactiques de performance.

Les principales recommandations découlant de ce projet sont l'ajout de nouvelles tactiques de performance, la modification de l'analyse du cadre de raisonnement de performance, l'ajout de nouveaux cadres de raisonnement et l'amélioration d'ArchE lui-même.

Mots-clés : ArchE, tactiques architecturales, performance, automatisation, architecture logicielle

AUTOMATION OF THE SELECTION AND APPLICATION OF PERFORMANCE TACTICS ON SOFTWARE ARCHITECTURES

Sébastien GAGNÉ

ABSTRACT

ArchE is an expert system, developed by the Software Engineering Institute, that helps software architects with the elaboration of software architectures. Currently, it includes two reasoning frameworks, and is therefore able to analyze two quality attributes: modifiability and performance.

The goal of this project is to automate the selection and application of performance architectural tactics and to integrate them in ArchE's performance reasoning framework. This integration confirms the feasibility of performance tactic automation.

A design for the tactics' integration was developed by taking into account the strengths and challenges of the modifiability reasoning framework. Using the tactics descriptions included in the book "Software Architecture in Practice" (Bass, Clements et Kazman, 2003), a series of rules were created to determine if and how each tactic must be applied automatically to a given software architecture. Four tactics were integrated: increase available resources, reduce the responsibility's execution time, increase the scenario's period and increase the responsibility's priority.

Validation of the proposed approach shows that the tactics work as expected and they are suggested and applied automatically on a simple, yet non-trivial software architecture. We conclude that support for performance tactics can in fact be automated in an expert system. However, the context and the expert system's limitations must be taken into account because they may limit the automation of some architectural tactics. Two contributions emerge from this project: a simpler design for the integration of tactics and a set of rules to automate a set of performance tactics.

The main recommendations issued at the term of this project are the addition of new performance tactics, the change of the analysis performed by the performance reasoning framework, the addition of new reasoning frameworks and improvements of the ArchE core program.

Keywords: ArchE, architectural tactics, performance, automation, software architecture

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ADD	Attribute-driven design
ArchE	Architectural Expert
CCL	Construction and composition language
ICM	Intermediate constructive model
RF	Reasoning framework
RMA	Rate monotonic analysis
SAP	Software architecture in practice
SEI	Software Engineering Institute
UML	Unified modeling language

INTRODUCTION

Dans notre monde technologique, les logiciels sont omniprésents dans tous les domaines d'activité. Qu'elle soit explicitement définie ou non, tout logiciel a une architecture. Une architecture bien conçue est nécessaire pour la création d'un système répondant aux exigences des clients. Pour relever ce défi adéquatement, le Software Engineering Institute (SEI) de l'université Carnegie Mellon a développé une méthode de conception d'architectures logicielles appelée « Attribute-Driven Design » (ADD). Cette méthode basée sur les attributs de qualité propose une série d'étapes permettant de satisfaire les exigences de qualité de l'architecture. ADD requiert la création et la gestion de plusieurs artéfacts tels que les requis fonctionnels, les contraintes et les requis d'attributs de qualité. ADD, comme plusieurs approches architecturales issues du SEI, utilise les scénarios de qualité pour spécifier les exigences de qualité. De plus, ADD propose d'utiliser les tactiques architecturales, un autre concept courant dans les approches du SEI, pour atteindre ces objectifs de qualité. Les tactiques sont des pistes de solution qui peuvent être explorées afin de modifier l'architecture en fonction d'un attribut de qualité.

Pour aider l'architecte logiciel à appliquer la méthode ADD dans l'élaboration d'une architecture, le SEI a développé un système expert d'assistance à l'élaboration d'architectures logicielles nommé ArchE (« Architectural Expert »).

ArchE ne remplace pas l'architecte, il est un assistant qui facilite l'application des étapes de la méthode ADD. ArchE en implémente les différents aspects, tels que les scénarios de qualité, les fonctionnalités du système, les différentes vues architecturales ainsi que les tactiques. ArchE utilise différents cadres de raisonnement afin d'analyser certains aspects de l'architecture actuellement élaborée. Avec la dernière version d'ArchE (v3), il est maintenant possible, pour les développeurs externes au SEI, d'ajouter de nouveaux cadres de raisonnement et de modifier ceux déjà en place. Actuellement, ArchE contient deux cadres de raisonnement : un analysant la modificabilité de l'architecture (*modifiability*) et l'autre qui analyse la performance. Le cadre de raisonnement de modificabilité est complet, c'est-à-dire

qu'il analyse l'architecture et peut proposer des tactiques, alors que le cadre de raisonnement de performance analyse uniquement l'architecture sans proposer l'application de tactiques qui modifient l'architecture, cette partie étant spécifiée comme optionnelle dans la définition d'un cadre de raisonnement dans ArchE.

L'utilisation des tactiques est une étape indispensable lors de la conception d'une architecture. Cependant la recherche et l'application manuelles de celles-ci peuvent être longues et fastidieuses. ArchE règle une partie de ce problème avec les tactiques de modificabilité; il en est cependant autrement avec les tactiques de performance. Le SEI décrit déjà un certain nombre de tactiques de performance dans différents ouvrages. Cependant, il n'est pas décrit comment leur application pourrait être automatisée, ni si une telle automatisation est possible. Avec les outils nécessaires (un cadre de raisonnement), on pense qu'il serait possible d'intégrer dans ArchE des fonctions aidant l'utilisateur à chercher et appliquer automatiquement des tactiques de performance. L'objectif de ce projet est donc d'analyser les tactiques architecturales liées à la performance afin de concevoir une version automatisable de leur application pour l'implémenter dans un prototype de cadre de raisonnement de performance. Pour y arriver, on débutera avec un prototype capable d'analyser la performance d'une architecture pour ensuite y intégrer des fonctionnalités supplémentaires permettant de suggérer des tactiques de performances basées sur l'analyse déjà disponible. Le résultat visé est un système plus complet, capable d'analyser une architecture et de proposer automatiquement des modifications pour améliorer sa performance.

Ce projet contribue à plusieurs aspects du domaine des tactiques architecturales. Premièrement, il apporte une meilleure compréhension de ce qui est nécessaire pour permettre l'application de tactiques automatisées. Bien que des tactiques de modificabilité automatisées existent déjà, il s'agit d'une première itération. L'ajout des tactiques de performance raffine ce processus et permet d'identifier les aspects importants à considérer lors de l'automatisation de tactiques architecturales dans un cadre de raisonnement. Deuxièmement, le projet permet d'évaluer les possibilités et les limitations concernant

l'automatisation des tactiques de performance dans l'architecture actuelle d'ArchE. Certains obstacles sont difficiles à contourner; les identifier permet de les éviter lors de nouvelles itérations. De plus, il confirme la faisabilité de l'automatisation des tactiques de performance.

Pour définir la portée de ce projet, certaines contraintes ont été établies. Premièrement, nous utilisons le système expert ArchE, car il automatise déjà la méthode ADD avec laquelle nous sommes déjà familiers. De plus, il inclut déjà un cadre de raisonnement qui analyse et applique automatiquement des tactiques de modificabilité : ceci donne une bonne référence avec des exemples qui peuvent servir de base pour les tactiques de performance. Un autre aspect en faveur de l'utilisation d'ArchE est qu'il fournit déjà une analyse de la performance des architectures. Il est important de noter que l'on suppose que cette analyse est valide; les résultats seront donc pris tels quels sans validation supplémentaire. Puisqu'ArchE contient plusieurs cadres de raisonnement, plusieurs tactiques peuvent être proposées pour plusieurs attributs de qualité. Bien qu'en réalité ces tactiques ne le soient pas, nous présumons qu'elles sont toutes indépendantes et n'étudions pas les interactions entre elles; uniquement les tactiques de performances seront prises en considération. Comme dernière contrainte, les tactiques seront indépendantes d'un domaine d'application. Ceci est en partie dû à ArchE qui est lui-même indépendant d'un domaine d'application, mais aussi pour démontrer que l'automatisation des tactiques de performance d'une façon générale n'est pas affectée par le domaine d'application.

Pour atteindre les objectifs de ce projet, il faut définir une certaine méthodologie. Premièrement, une familiarisation avec le domaine s'impose afin de comprendre la théorie sur le système expert utilisé, les cadres de raisonnement, la méthode d'analyse ainsi que les tactiques de performance. Avec les connaissances suffisantes, l'analyse du fonctionnement d'ArchE, de son architecture et de ses cadres de raisonnement peut se faire. Cette analyse apporte les connaissances nécessaires pour effectuer des modifications au cadre de raisonnement de performance pour y ajouter les fondations pour l'intégration des tactiques. Les différentes tactiques sont ensuite analysées et une version automatisée est développée et

intégrée dans le cadre de raisonnement de performance d'ArchE. Pour terminer, la solution proposée est validée afin de déterminer que les modifications répondent bien aux objectifs.
Voici un résumé des cinq étapes de la méthodologie :

1. revue de la littérature sur le domaine (architecture logicielle, méthode ADD, tactiques de performance, cadres de raisonnement, analyse de performance, etc.);
2. analyse détaillée d'ArchE, de son architecture et de ses cadres de raisonnement;
3. modification du cadre de raisonnement de performance pour y inclure les aspects manquants pour les tactiques;
4. analyse et intégration des tactiques dans le cadre de raisonnement de performance;
5. validation de l'ajout des tactiques.

Il est important de noter que, bien qu'elles aient été présentées séquentiellement, ces étapes sont réalisées selon une approche itérative, où une partie de l'analyse est faite, suivie de l'intégration de nouvelles tactiques, pour revenir à l'analyse et à la modification. Cette approche donne une plus grande flexibilité lors du développement et permet d'avoir des résultats rapidement, tout en minimisant le risque de concevoir une solution qui finalement ne s'applique pas.

Ce mémoire est divisé en quatre chapitres. Le premier chapitre présente une revue de la littérature, suivi d'un chapitre expliquant les modifications effectuées au cadre de raisonnement de performance pour permettre l'ajout des tactiques. Le chapitre 3 contient l'information sur l'intégration d'une méthode automatisée des tactiques. La validation individuelle de chaque tactique est également réalisée au chapitre 3 (c.-à-d., validation « unitaire »). Le bon fonctionnement de l'implémentation de l'ensemble des tactiques retenues (c.-à-d., validation « système ») est validé dans le dernier chapitre. Une conclusion termine le tout et est suivie des recommandations.

CHAPITRE 1

REVUE DE LA LITTÉRATURE

Pour commencer ce projet, il faut étudier la littérature. C'est la première étape et elle permet d'avoir les connaissances nécessaires pour accomplir les étapes suivantes de la méthodologie.

Figure 1.1 Méthodologie : revue de la littérature.

Les étapes suivantes utilisent les connaissances acquises afin d'analyser en détail le fonctionnement d'ArchE, de modifier le cadre de raisonnement de performance, d'analyser les tactiques architecturales de performance pour leur automatisation, ainsi que pour valider cette version automatique des tactiques.

1.1 Architecture logicielle

L'architecture d'un logiciel est le résultat de décisions d'affaires et techniques faites à propos de celui-ci. Plusieurs sources vont influencer la conception du logiciel et sa réalisation va dépendre, entre autres, du domaine d'application et des fonctions que le logiciel devra remplir (Bass, Clements et Kazman, 2003). Par exemple, un contrôleur de robot va être influencé par le type d'applications auxquelles ce robot est destiné : est-ce un robot d'exploration autonome où la fiabilité est primordiale ou est-ce un contrôleur automobile où les critères de performance sont plus stricts? Ce sont là des questions qui vont influencer l'architecture du logiciel.

Une architecture logicielle est importante, car elle apporte plusieurs choses. Notamment, elle sert d'outil de communication entre les différents intervenants, elle permet de prendre des décisions de conception tôt dans le développement et elle permet d'avoir une abstraction

compacte du système. De plus, une architecture logicielle peut être réutilisée en partie pour des projets similaires et ainsi économiser du temps avec des problèmes déjà résolus. Une architecture bien définie va faire économiser du temps lors du développement du logiciel en permettant de bien répondre aux besoins des intervenants (Bass, Clements et Kazman, 2003).

Certaines personnes peuvent parler de désavantages au développement d'une architecture en soulevant le coût et le temps nécessaire pour la créer. Cependant, les avantages au cours du projet vont, en général, compenser l'effort supplémentaire requis au début du projet. Développer une architecture n'est pas une tâche simple. Plusieurs intervenants influencent son développement. L'architecte doit donc prendre des décisions à propos des demandes et créer une architecture qui leur répond le mieux possible (Bass, Clements et Kazman, 2003). Bien qu'il soit compliqué de prendre en compte toutes ces décisions, il est préférable de considérer ces décisions au début du projet qu'au milieu de celui-ci où les implications pourraient demander beaucoup plus de travail.

Étant donné que l'architecture logicielle est un domaine relativement jeune, plusieurs définitions et points de vue existent. Ces derniers diffèrent surtout par leur portée : certains décrivent la structure à un haut niveau en utilisant des composantes ou des modèles alors que d'autres vont étudier le système en exécution (Bass, Clements et Kazman, 2003, pp. 23-24). Dans le cadre de ce mémoire, la définition et le point de vue du Software Engineering Institute (SEI) seront utilisés. Elle est décrite dans la section suivante.

1.1.1 L'architecture logicielle selon le SEI

La définition de l'architecture logicielle selon le SEI est : « L'architecture d'un programme ou d'un système informatique est la structure ou l'ensemble des structures du système, incluant les éléments logiciels, les relations entre ces éléments, ainsi que les propriétés visibles de l'extérieure de ces éléments » (Bass, Clements et Kazman, 2003, p. 21). Chacun de ces éléments logiciels peut être vu comme une partie du système qui accomplit une tâche de celui-ci. Il existe plusieurs types de relations entre les éléments (par exemple, des relations

d'utilisation ou des relations d'exécution). Elles définissent les différentes interactions que les éléments peuvent avoir. Les propriétés visibles de l'extérieur fournissent l'information nécessaire pour analyser et comprendre le système. Par exemple, elles peuvent indiquer le temps d'exécution des différents éléments ou la probabilité qu'une modification se propage à travers une relation d'utilisation entre deux éléments.

Pour aider les architectes, le SEI a créé une méthode de conception d'architecture logicielle basée sur l'ensemble de leurs approches architecturales : « Attribute-Driven Design » (ADD). Cette méthode de conception basée sur les attributs de qualité sera expliquée en détail dans la prochaine section, mais avant il faut définir certains termes utilisés pour décrire une architecture selon le SEI et la méthode ADD.

Premièrement, la méthode ADD est une méthode basée sur les attributs de qualités (« Quality attributes »). Un attribut de qualité peut être défini comme une qualité que le système possède. Par exemple, on peut décrire un système en parlant de sa performance, de sa modificabilité, de sa sécurité, de sa disponibilité, etc. Les attributs de qualités vont varier selon le système développé. Un système en temps réel sera évalué selon plusieurs critères de performance, alors qu'il y aura une plus grande attention aux critères de sécurité pour un système bancaire. Ces attributs de qualités sont orthogonaux aux fonctionnalités que doit avoir le système. En effet, il est possible de faire des choix indépendants pour chacun d'eux. Pour un même ensemble de fonctionnalités, un architecte peut décider de mettre un accent sur la performance, alors qu'un autre pourrait privilégier la modificabilité (Bass, Clements et Kazman, 2003).

Pour définir les requis par rapport aux attributs de qualité (« Quality attribute requirements »), on utilise des scénarios d'attributs de qualité (« Quality attribute scenario »). Il est alors possible d'analyser la qualité du système et de la comparer aux requis originaux. Un scénario d'attribut de qualité comprend six parties (Bass, Clements et Kazman, 2003, p. 75) :

1. la source du stimulus : un humain, un autre système, un capteur, etc.;
2. le stimulus : un événement qui doit être géré par le système;
3. l'environnement : l'état du système lors de l'arrivée du stimulus (ex : normal, surcharge);
4. l'artéfact : la partie du système qui reçoit le stimulus (peut être tout le système);
5. la réponse : l'activité que doit accomplir le système pour répondre au stimulus;
6. la mesure de réponse : la réponse doit être mesurable pour que le requis puisse être vérifié.

Par exemple, un scénario traitant de la modificabilité d'un système de positionnement pourrait se lire : « Le concepteur (1) veut changer la communication GPS (2) du système (4) en production (3). Ce changement doit être complété (5) en moins de deux personnes jours (6) ».

Chaque requis de qualité est traduit en un scénario. Lors du développement du système, le concepteur peut vérifier si le système qu'il conçoit répond aux différents scénarios. Pour y arriver, il analyse son système et vérifie que les mesures de réponses sont toutes respectées. Différentes approches existent actuellement, mais l'adéquation entre les scénarios (exigences) de qualité et l'architecture est encore aujourd'hui établie en grande partie par l'intuition et le jugement de l'architecte.

Pour définir les requis fonctionnels, les responsabilités sont utilisées (Wirfs-Brock et McKean, 2003). Une responsabilité correspond à une tâche que doit accomplir le système. Par exemple, la communication GPS et le gestionnaire de position sont deux responsabilités qui peuvent se retrouver dans le système de positionnement. Les responsabilités peuvent être raffinées et divisées afin de détailler le système. Ce raffinement peut se faire de différentes manières. Une de ces méthodes, développée par le SEI, sera examinée dans la prochaine section.

1.1.2 Méthode ADD

Pour structurer et rendre plus systématique la conception d'architectures logicielles, une méthode de conception dirigée par les attributs de qualité (« Attribute-Driven Design » – ADD) a été développée par le SEI pour permettre de satisfaire à la fois les exigences de qualités et les exigences fonctionnelles. ADD est une méthode récursive basée sur la décomposition du système ou d'un élément du système en appliquant des modifications architecturales ou des tactiques et des patrons qui sont justifiés par les requis des attributs de qualité (Wojcik *et al.*, 2006). ADD permet de créer de manière incrémentale l'architecture du système en se basant surtout sur les requis de qualité. La méthode est appliquée jusqu'à ce que tous les requis architecturaux (fonctionnels et de qualités) soient satisfaits.

Comme données d'entrée, la méthode ADD a besoin des requis fonctionnels, des requis de qualité et des contraintes de conception. Il est important de noter qu'il n'est pas opportun de perdre du temps à essayer de bien classifier une exigence parmi ces trois catégories, car elles servent toutes les trois d'entrée à la démarche. Certaines exigences peuvent être difficiles à assigner à une catégorie « fonctionnelle » ou « qualité », par exemple. Le but est de considérer l'ensemble des exigences, non de les classifier selon un modèle strict. Le résultat de la méthode ADD est une conception pour un système comprenant des responsabilités, des propriétés et des relations entre les différents éléments. Pour y arriver, ADD définit huit étapes (Wojcik *et al.*, 2006).

Étape 1 : confirmer qu'il y a suffisamment d'information sur les requis. Cette étape consiste à vérifier les données d'entrées et à s'assurer qu'il y a assez d'informations pour commencer les étapes suivantes d'ADD. Les requis devraient être priorisés par les intervenants et les requis de qualités devraient être définis sous la forme de scénarios d'attribut de qualité.

Étape 2 : choisir un élément du système pour le décomposer. L'élément choisi sera utilisé pour les étapes suivantes. Ce choix peut être basé sur plusieurs facteurs : l'architecture courante, le risque et la difficulté, un critère d'affaires ou un critère organisationnel.

Étape 3 : identifier les motivations architecturales (« architectural drivers »). Dans cette étape, les requis reliés à l'élément choisi sont priorisés une deuxième fois selon leur impact sur l'architecture. Les requis les plus prioritaires (selon les intervenants et l'impact) sont choisis comme point central pour les étapes suivantes.

Étape 4 : choisir une conception qui répond aux motivations architecturales. Cette étape consiste à choisir une manière de décomposer l'élément choisi en une série de sous-éléments ayant des relations entre eux pour qu'elle réponde aux motivations architecturales choisies. Pour effectuer cette décomposition, le concepteur utilisera ses connaissances et des tactiques architecturales pour explorer différentes solutions et choisir la plus appropriée à la situation.

Étape 5 : créer les éléments architecturaux et leur allouer des responsabilités. Cette étape va créer les sous-éléments architecturaux choisis dans l'étape 4. Les responsabilités seront allouées selon le type des nouveaux éléments. Il est possible que de nouvelles responsabilités s'ajoutent selon la décomposition qui a été choisie.

Étape 6 : définir les interfaces des éléments créés. Selon le niveau de conception, l'interface des nouveaux éléments sera définie de différentes manières. L'interface définit ce qui est produit et ce qui est requis par chacun des éléments.

Étape 7 : vérifier et raffiner les requis et les transformer en contraintes pour les nouveaux éléments. Cette étape vérifie que le nouveau système répond toujours aux requis fonctionnels, aux requis de qualité et aux contraintes du problème. Les nouvelles responsabilités des nouveaux éléments sont aussi transformées en requis fonctionnels pour cet élément.

Étape 8 : répéter les étapes 2 à 7 pour le prochain élément du système à décomposer.

Lorsqu'il n'y a plus d'éléments à décomposer, la méthode est terminée. En suivant ces étapes, on a un système qui devient progressivement plus complet jusqu'à ce qu'il soit satisfaisant. La figure suivante résume les étapes à suivre pour la méthode ADD.

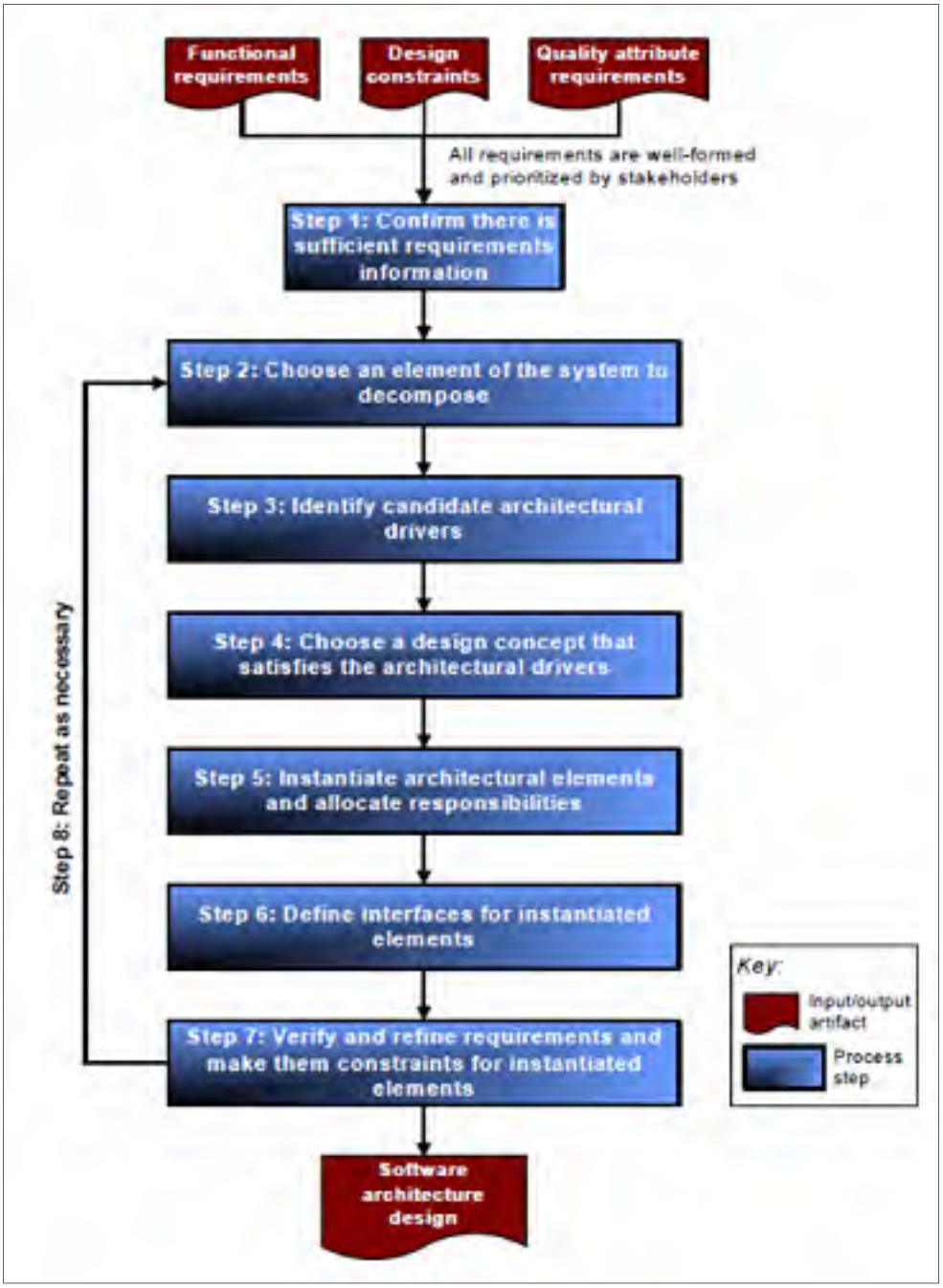

Figure 1.2 Résumé des étapes de la méthode ADD.
Tirée de (Wojcik *et al.*, 2006, p. 5)

1.1.3 Difficultés de la conception des architectures logicielles

Avoir une méthode comme ADD simplifie un peu la tâche pour les architectes, car ils peuvent suivre une ligne directrice, mais cette approche requiert quand même une expérience et des connaissances non triviales. De plus, le raffinement d'un élément peut en affecter d'autres qui ne sont pas dans la portée actuelle des changements (Bachmann, Bass et Klein, 2003b). Un logiciel aidant l'architecte au travers de ces étapes serait avantageux, car il permettrait de contenir toute l'information nécessaire à l'élaboration des architectures logicielles et de montrer tous les impacts d'une modification.

Il existe un grand nombre d'attributs de qualités tels que la performance, la fiabilité, la sécurité, la modificabilité, la disponibilité, etc. Même avec l'aide d'une méthode structurée comme ADD, être expert dans tous ces attributs de qualité est difficile pour une seule personne. De plus, le concepteur doit gérer un grand nombre de contraintes de toutes sortes, ce qui rend la tâche extrêmement compliquée sans l'aide d'outils spécialisés (Bachmann, Bass et Klein, 2003b, p. 31). Pour ces raisons, une approche assistée par logiciel est une avenue intéressante.

En plus d'un grand nombre de considérations, il faut aussi gérer les liens entre elles. En effet, une modification affectant positivement un attribut de qualité peut avoir des répercussions négatives sur un ou plusieurs autres attributs de qualité de l'architecture. Améliorer la modificabilité peut avoir des impacts négatifs sur la performance et vice-versa. Ceci rend encore plus complexe la conception d'une architecture, car il faut gérer tous ces liens pour arriver à un résultat adéquat pour tous les attributs de qualité désirés. Un logiciel capable de montrer et de gérer tous ces liens serait d'une grande aide.

En plus de la quantité d'information qu'il faut gérer pour concevoir une architecture, il faut aussi gérer les changements qui vont y survenir. En effet, le changement d'un seul composant

peut être compliqué à intégrer et peut avoir un impact important sur l'architecture du système au complet (Sharma et Trivedi, 2007).

Un logiciel gérant la conception d'architectures logicielles serait en mesure de faciliter ces changements, car il permettrait de visualiser rapidement les impacts du changement, leur ampleur et suggérer des modifications possibles pour parvenir à une solution satisfaisante. De plus, un tel logiciel permettrait aux concepteurs d'essayer plusieurs changements rapidement en considérant tout de même les aspects importants de l'architecture et ainsi de faire un meilleur choix pour la solution retenue.

Automatiser la création d'une architecture logicielle est une tâche ardue (Bachmann, Bass et Klein, 2003b). Il y a beaucoup d'inconnues lors de la création d'un système expert de conception d'architecture. Il y a beaucoup d'aspects à considérer et le logiciel doit permettre d'en gérer un maximum. De plus, il n'existe pas de solution unique qui va répondre aux besoins de tout le monde dans toutes les situations. Ceci implique que le logiciel qui assiste l'architecte doit être flexible pour prendre en compte tous les aspects de la situation et de pouvoir s'adapter selon la situation.

Cette conjoncture est propice au développement d'un système expert, car il aiderait les architectes à gérer la grande quantité d'informations, les liens entre tous les attributs, ainsi que les changements reliés à la création d'une architecture. C'est pourquoi le SEI a développé un système expert pour la création d'architectures logicielles : ArchE (« Architectural Expert »).

1.2 ArchE

Tel qu'indiqué précédemment, ArchE s'inspire de la méthode ADD, elle aussi développée par le SEI. Son objectif est d'aider l'architecte principalement aux étapes 4[1] et 5[2] de la

[1] Choisir une conception qui répond aux motivations architecturales

méthode ADD (Bachmann *et al.*, 2005; Wojcik *et al.*, 2006). Ces étapes demandent une grande quantité d'information pour choisir et appliquer des modifications à l'architecture. En effet, l'architecte doit choisir, parmi un grand nombre de tactiques, les tactiques « optimales » à appliquer sur l'architecture pour atteindre l'objectif de qualité visé. Il doit aussi appliquer ces tactiques sur l'architecture afin de créer une nouvelle architecture et démarrer la nouvelle itération de la méthode (Wojcik *et al.*, 2006). Avec sa grande capacité de calcul, ArchE peut analyser l'architecture actuelle, vérifier si les scénarios de qualité sont respectés, essayer automatiquement différentes tactiques pertinentes et retourner les meilleures suggestions à l'utilisateur, tout cela beaucoup plus rapidement qu'un être humain pourrait le faire par lui-même. En plus des suggestions de modifications, l'utilisateur a accès aux résultats de l'analyse, ce qui réduit considérablement sa tâche. Les suggestions de modifications de l'architecture sont aussi accompagnées des résultats de l'analyse, comme si elles avaient été appliquées. Ces informations permettent à l'utilisateur de voir immédiatement les résultats qu'il pourrait obtenir avant même d'effectuer une modification à l'architecture. Pour terminer, l'architecte qui utilise ArchE n'a pas besoin de connaître au préalable comment appliquer toutes les modifications suggérées, car toutes les étapes à suivre sont déjà contenues dans ArchE (Bachmann *et al.*, 2005).

La figure suivante résume les intrants considérés (scénarios et responsabilités), les extrants produits (analyse, suggestions de tactiques et conception) et les modules utilisés (cadres de raisonnement) par ArchE. Chacune des composantes sera examinée dans cette section.

---

[2] Créer les éléments architecturaux et leur allouer des responsabilités

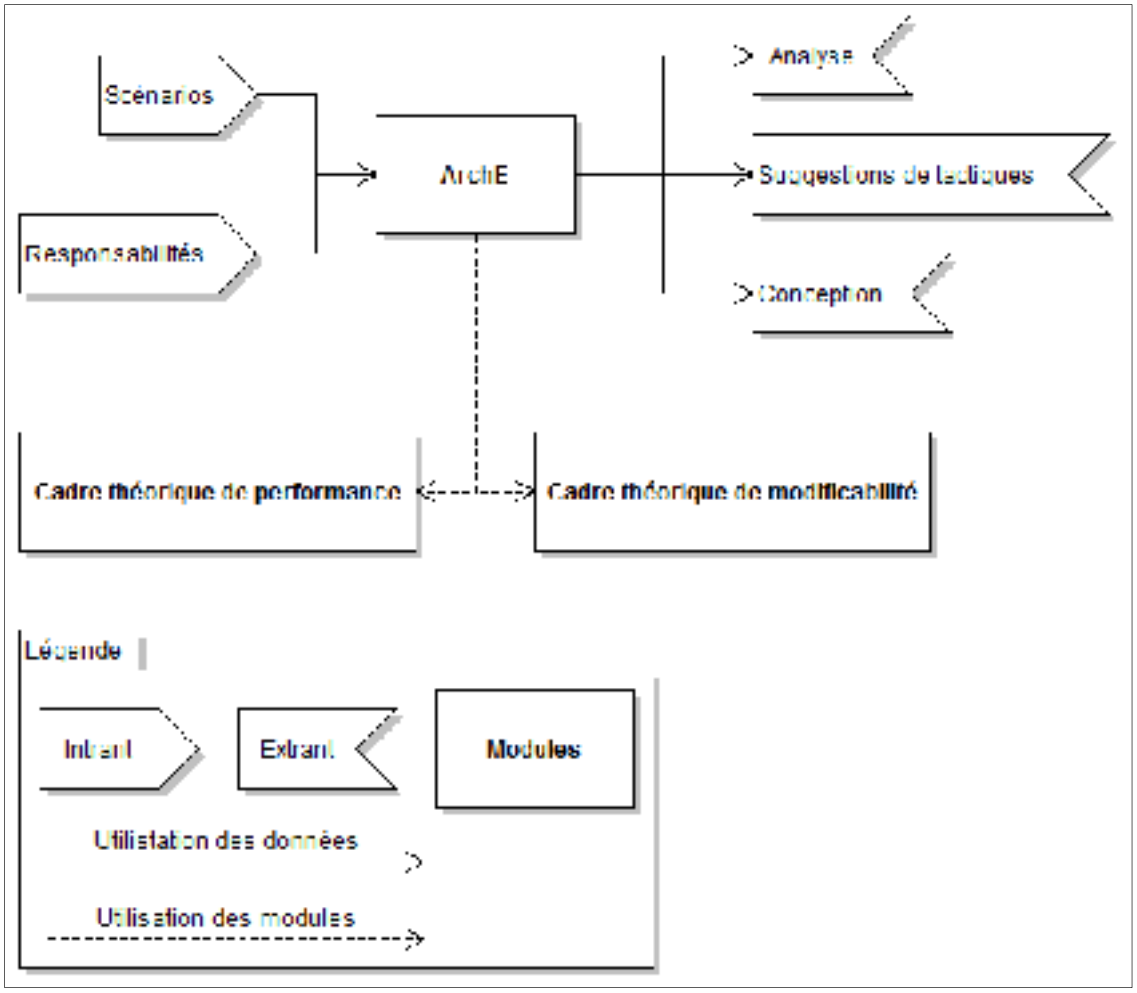

Figure 1.3 Résumé des composantes d'ArchE.

Pour analyser l'architecture du système, ArchE encapsule l'information nécessaire pour traiter un attribut de qualité dans un cadre de raisonnement (« reasoning framework »). ArchE est composé de plusieurs cadres de raisonnement indépendants, chacun analysant son propre aspect de l'architecture. Chaque cadre de raisonnement gère uniquement un seul attribut de qualité. Le rôle d'ArchE est de gérer ces cadres de raisonnement et leurs interrelations afin d'arriver à une conception valide (Bachmann, Bass et Klein, 2003b; Bachmann *et al.*, 2005). Chaque cadre de raisonnement va uniquement analyser les scénarios qui concernent son attribut de qualité. Si l'analyse indique qu'un scénario ne respecte pas son critère, le cadre de raisonnement pourra suggérer d'appliquer des tactiques architecturales (voir section 1.5). Au moment de démarrer ce projet, il y a deux cadres de raisonnement dans

ArchE : un pour la performance et un pour la modificabilité. Les cadres de raisonnement sont analysés plus en détail dans une section ultérieure.

Pour déterminer les requis concernant les attributs de qualité, ArchE utilise les scénarios d'attribut de qualité fournis par l'utilisateur (Bachmann, Bass et Klein, 2003b; Bass *et al.*, 2005). Comme indiqué à la section 1.1.1, les scénarios contiennent la source du stimulus, le stimulus, l'environnement, l'artéfact affecté, la réponse et la mesure de la réponse. De plus, ces scénarios contiennent l'attribut de qualité auquel il est rattaché. Celui-ci indique à ArchE le cadre de raisonnement qu'il doit utiliser pour vérifier le scénario en question. Les scénarios vont lui permettre de déterminer les informations manquantes requises pour l'analyse (Bass *et al.*, 2005). Par exemple, pour un scénario de performance, le stimulus deviendra la période de ce scénario et la mesure de la réponse donnera le temps de réponse maximal que celui-ci doit respecter. Pour certains types de scénarios, ArchE aura besoin d'informations supplémentaires pour faire son analyse. Pour ce faire, ces paramètres supplémentaires doivent être définis par l'utilisateur (Bachmann, Bass et Klein, 2003a).

Pour déterminer les fonctionnalités que doit accomplir le système, ArchE utilise une structure de données basée sur les responsabilités. Les responsabilités sont rattachées à une fonctionnalité du système. À la base, les responsabilités n'ont aucune information, outre leur nom, qui leur est attachée. En fonction de l'analyse effectuée, celle-ci peut requérir des informations supplémentaires. Pour y parvenir, les cadres de raisonnement peuvent définir de nouveaux paramètres pour avoir plus d'information sur les responsabilités et ainsi pouvoir effectuer leur analyse. Par exemple, une analyse de performance pourrait avoir besoin du temps d'exécution et de la priorité de chacune des responsabilités.

Le fait qu'ArchE se base uniquement sur des scénarios d'attributs de qualité et des responsabilités est un avantage, car il reste indépendant des technologies ou du matériel utilisé pour le produit final. Il utilise uniquement des informations sur les exigences fonctionnelles et de qualité pour créer un modèle de l'architecture, il n'est donc pas limité à un seul type d'architectures.

Lorsqu'une conception valide est construite, ArchE peut passer à la dernière étape du processus : la création de l'architecture. Lorsqu'ArchE analyse les scénarios et qu'il construit sa conception, les responsabilités sont uniquement des concepts reliés entre eux. Pour créer l'architecture finale, ArchE doit assigner les responsabilités à différents éléments architecturaux (lesquels peuvent être des modules, des processus, etc). Cette attribution se fait selon ce qui est dicté par les différents cadres de raisonnement : une responsabilité peut être assignée à un ou plusieurs modules ou un module peut avoir plusieurs responsabilités (Bachmann, Bass et Klein, 2003b). C'est ensuite le travail de l'architecte de traduire cette architecture indépendante du domaine en modules réels prêts à être développés par l'équipe de développement.

Malgré ses différentes avancées, ArchE reste relativement peu développé. Actuellement, il contient deux cadres de raisonnement; il tient compte uniquement de la performance et de la modificabilité. De plus, les cadres de raisonnement ne tiennent compte que d'une partie des connaissances leur étant associées (Bachmann, Bass et Klein, 2003b). Cependant, la plus récente version d'ArchE (v3) permet d'y ajouter des cadres de raisonnement sous la forme de plugiciel (« plugin »). Cette nouvelle fonctionnalité simplifie l'ajout et la modification de cadre de raisonnement, car ils sont maintenant accessibles. Cette facilité peut motiver un plus grand nombre de personnes à développer des modules pour ArchE et ainsi le rendre plus complet (Lee et Bass, 2005). Bien qu'il soit maintenant plus simple de développer et de modifier des cadres de raisonnement, ce n'est tout de même pas une tâche triviale. Pour y arriver, les chercheurs ont besoin de beaucoup de connaissances sur le domaine choisi.

L'utilisation d'ArchE n'est pas la seule technique pour obtenir un système qui répond aux exigences de qualité. Par exemple, le NFR Framework (Chung, Nixon et Yu, 2000) propose une technique basée sur les requis non-fonctionnels (c.-à-d., les requis de qualité). La différence clé entre ces deux approches est que le NFR Framework examine les requis de qualité pour faire des suggestions architecturales, alors qu'ArchE examine les responsabilités qui composent l'architecture pour les modifier et ainsi affecter les attributs de qualité.

Les promesses faites par ArchE sont intéressantes pour les architectes. Il s'intègre à la méthode ADD déjà en utilisation, il utilise les scénarios d'attribut de qualité pour obtenir les requis de qualité, il utilise les responsabilités pour définir les requis fonctionnels, il analyse les données à l'aide de cadres de raisonnement et il construit une architecture à partir d'une conception valide qu'il génère.

1.3 Cadres de raisonnement

Comme énoncé précédemment, les cadres de raisonnement sont la base d'ArchE. Chaque cadre de raisonnement est basé sur une théorie analytique spécifique utilisée pour mesurer un seul attribut de qualité. Pour effectuer ses calculs, le cadre de raisonnement peut avoir besoin d'informations supplémentaires qui pourraient ne pas être incluses dans les scénarios d'attributs de qualité; c'est la responsabilité du cadre de raisonnement de demander ces informations à l'utilisateur et de les sauvegarder. Un cadre de raisonnement est composé de six éléments : la description du problème, la théorie analytique, les contraintes d'analyse, la représentation du modèle, l'interprétation du modèle et la procédure d'évaluation (Bass *et al.*, 2005).

Comme la liste des éléments le montre, il n'y a pas de modification architecturale dans le cadre de raisonnement; il ne fait que mesurer un attribut de qualité. Pour cette raison, les cadres de raisonnement doivent être utilisés conjointement avec une approche pour contrôler les attributs de qualité (Bass *et al.*, 2005). Cette approche est basée sur les tactiques. C'est avec elles que l'architecte logiciel va modifier l'architecture. Ces modifications vont influencer les attributs de qualité dans le but de satisfaire les exigences. Bien qu'en théorie les tactiques ne font pas partie du cadre de raisonnement, elles y sont tout de même incluses dans ArchE, car elles ont besoin des informations recueillies par celui-ci afin de faire un choix en tenant compte de la situation actuelle (Lee et Bass, 2005).

Lors de son exécution, ArchE va demander séparément à chacun des cadres de raisonnement d'analyser le système et de proposer des tactiques. Les cadres de raisonnement retournent le résultat de l'analyse et, si nécessaire, une série de suggestions de tactiques architecturales. Si des tactiques sont proposées, ArchE va demander aux cadres de raisonnement de les appliquer sur une copie de l'architecture pour déterminer l'impact qu'aurait chacune d'elles sur l'architecture actuelle. ArchE présente ensuite tous ses résultats (analyse et proposition de tactiques) à l'utilisateur. Ce dernier doit choisir la prochaine étape : appliquer une des tactiques suggérées ou modifier manuellement le système. L'approche utilisée par ArchE est de type tableau noir (« blackboard », un style architectural) où tous les cadres de raisonnement peuvent lire et modifier les informations contenues sur le tableau (l'architecture dans ce cas) et où l'utilisateur détermine les critères d'acceptation du système avec les scénarios d'attributs de qualité.

Par exemple, le cadre de raisonnement de modificabilité détermine qu'une modification à la communication GPS du système se ferait en trois personnes jours. Ce résultat dépasse la mesure de réponse qui est de deux personnes jours. Le cadre de raisonnement propose, en se basant sur ses règles internes, une tactique : l'ajout d'un intermédiaire entre le module et le gestionnaire. ArchE va demander au cadre d'ajouter automatiquement cet intermédiaire et le cadre détermine que cette tactique donne un effort de modification d'une personne jour. La tactique d'ajout d'un intermédiaire est ensuite proposée à l'utilisateur.

Bien que les cadres de raisonnement soient indépendants les uns des autres, ils ont quand même des interactions puisqu'ils modifient la même architecture. En effet, un cadre de raisonnement *A* peut avoir quatre types de relations avec un autre cadre de raisonnement *B* (Bachmann *et al.*, 2005) :

- *A* peut affecter les responsabilités qui ont des propriétés dans *B*;
- *A* peut générer un nouvel attribut de qualité pris en charge par *B*;
- *A* peut modifier une variable de l'attribut de qualité de *B*;
- *A* et *B* peuvent raisonner à propos du même attribut de qualité.

Tous ces cas doivent être gérés par ArchE et l'utilisateur; ce dernier va prioriser certains attributs de qualité plus que d'autres et devra faire des choix pour sa conception. De plus, lors de la création d'un cadre de raisonnement, il faut garder ces interactions en tête. Le but de ce projet n'est pas de préciser les différents liens entre les cadres de raisonnement, mais un gestionnaire de compromis entre les différents attributs de qualité serait une aide intéressante pour les architectes.

Dans le cadre de ce projet, il n'est pas prévu de modifier la théorie contenue dans les cadres de raisonnement. Les nouvelles tactiques vont se baser sur les connaissances déjà disponibles dans les cadres de raisonnement.

En bref, chaque cadre de raisonnement analyse l'architecture courante en considérant un seul attribut de qualité pour déterminer si chaque scénario de qualité lié à cet attribut est satisfait ou non. Pour faire ses calculs, il se base sur les scénarios fournis par l'utilisateur. Pour maximiser son utilité, un cadre de raisonnement doit être utilisé avec des tactiques architecturales pour permettre la modification des architectures. Bien qu'ils soient techniquement indépendants, les cadres de raisonnement ont tout de même des relations entre eux, car ils travaillent avec le même jeu de scénarios et responsabilités, donc sur la même architecture. Dans la prochaine section, nous examinerons les tactiques plus en détail.

1.4 Cadre de raisonnement de performance

Cette section a pour but de décrire plus spécifiquement le cadre de raisonnement de performance qui est utilisé dans ArchE v3.0. Le choix des tactiques architecturales (décrites à la section 1.5) est affecté par le type d'analyse de performance utilisé. Bien comprendre la méthode d'analyse permet de faire un choix éclairé lors de la sélection des tactiques applicables à l'architecture.

Dans cette section, nous décrivons le type d'analyse de performance effectuée, les différentes étapes nécessaires pour effectuer l'analyse, ainsi qu'un résumé de la théorie utilisée pour obtenir les résultats de l'analyse.

1.4.1 Type d'analyse

Plusieurs types d'analyse de performance peuvent être effectués sur un même système. Dans le cadre de raisonnement de performance d'ArchE, l'analyse effectuée est de type « pire temps de réponse possible » (« worst-case latency »). Le cadre de raisonnement va vérifier que le délai maximal pour compléter l'exécution d'un scénario ne dépasse pas l'échéance définie dans la mesure de réponse de celui-ci. Tous les scénarios sont périodiques avec une période constante qui est définie dans le stimulus.

Une analyse de ce type a besoin d'informations supplémentaires sur l'architecture. Premièrement, il faut connaitre les liens entre les différentes responsabilités (quelles responsabilités sont sollicitées pour chaque scénario et dans quel ordre). Ces relations sont baptisées « réactions » dans le cadre de raisonnement de performance d'ArchE. Un lien « réaction » signifie que lorsque la responsabilité parente de la relation s'exécute, la responsabilité enfant s'exécutera ensuite. Les responsabilités liées par des liens de réactions vont donc former une séquence d'exécution qui sera exécutée chaque fois que la première responsabilité dans la chaine sera exécutée. L'analyse a également besoin de deux informations sur chacune des responsabilités : son temps d'exécution et sa priorité. Avec ces informations, il sera possible d'analyser l'ordonnancement à priorité fixe avec préemption qui est utilisé par le cadre de raisonnement.

Pour effectuer l'analyse de la performance, le cadre de raisonnement utilise un outil externe nommé Lambda-WBA (Moreno et Hansen, 2009). Cet outil encapsule une partie de la théorie RMA (« Rate Monotonic Analysis ») (Klein *et al.*, 1993). Pour effectuer l'analyse RMA, Lambda-WBA utilise à son tour un autre outil : MAST (Gonzalez Harbour *et al.*, 2001). C'est ce dernier qui va effectuer l'analyse avec la théorie RMA sur les paramètres

d'entrées. Le rôle de Lambda-WBA est de rendre plus simple l'utilisation des fonctionnalités de MAST. Ces outils, et les traductions nécessaires à leur utilisation, sont décrits dans les prochaines sections. Voici une figure résumant les différents outils nécessaires pour effectuer l'analyse.

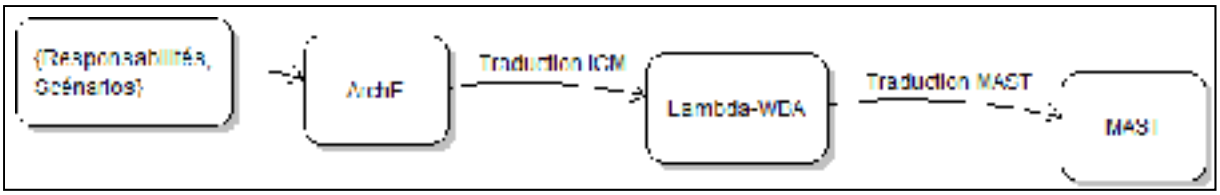

Figure 1.4 Outils utilisés pour l'analyse de performance.

1.4.2 Traduction ArchE vers Lambda-WBA

La première étape dans l'analyse de performance de l'architecture consiste à traduire la représentation ArchE du système (responsabilités, scénarios, liens, paramètres) dans le format ICM (« intermediate constructive model ») qu'utilise Lambda-WBA (Moreno et Hansen, 2009; Moreno et Merson, 2008). Lambda-WBA fait partie d'une série d'outils plus généraux nommés Lambda-*, qui eux font partis de l'initiative « Predictable Assembly from Certifiable Code (PACC) »[3] du SEI. Lambda-WBA analyse un système selon le pire cas avec blocage et asynchronisme (« Worst-case, Blocking, and Asynchrony »).

Pour faire la traduction, un module dans le cadre de raisonnement a été développé par le SEI. Ce module manipule les différents objets Java fournis par Lambda-WBA afin de reproduire sémantiquement l'architecture d'ArchE dans le format ICM. Voici une liste des différentes étapes suivies pour la traduction :

1. chaque responsabilité d'ArchE correspond à une composante (« component ») ICM où le temps d'exécution et la priorité de la responsabilité sont contenus dans le port d'entrée de la composante;

[3] http://www.sei.cmu.edu/predictability/tools/starterkit/index.cfm

2. chaque lien « réaction » entre les responsabilités est traduit en un lien entre le port de sortie (« source pin ») de la composante parent et le port d'entrée (« sink pin ») de la composante enfant;
3. chaque scénario est traduit en un service ICM où la valeur du stimulus (période associée à l'arrivée d'un nouveau stimulus) correspond à la période du service;
4. chacun des liens entre un scénario et une responsabilité est traduit en un lien entre le port de sortie du service (« service source pin ») et le port d'entrée de la responsabilité.

Voici un tableau résumant la traduction des termes :

Tableau 1.1 Correspondance entre les concepts d'ArchE et ICM (Lambda-WBA)

ArchE	ICM
Responsabilité	Composante
Scénario	Service
Lien « réaction »	Lien entre un port de sortie d'une responsabilité et port d'entrée d'une autre responsabilité
Lien scenario-responsabilité	Lien entre un port de sortie de service et un port d'entrée d'une responsabilité

Lorsque la traduction au format ICM est terminée, la cadre de raisonnement de performance appelle la fonction d'interprétation de Lambda-WBA avec cette traduction et démarre ainsi l'analyse du système.

1.4.3 Traduction Lambda-WBA vers MAST

Bien que le cadre de raisonnement utilise Lambda-WBA pour effectuer l'analyse de performance, ce dernier sert en fait uniquement à encapsuler un autre outil pour appliquer la théorie nécessaire. Cet autre outil est MAST[4] (Gonzalez Harbour *et al.*, 2001). Pour utiliser MAST, les données d'entrées doivent être écrites dans un fichier avec un format spécifique.

[4] http://mast.unican.es/

Le rôle de Lambda-WBA est donc de traduire la représentation ICM de l'architecture en un fichier d'entrées valide pour MAST. Cette traduction est effectuée automatiquement par Lambda-WBA, il n'y a donc aucune action ou modification possible de la part de l'utilisateur.

1.4.4 Analyse RMA

Pour effectuer l'analyse de la performance du système, la théorie RMA est utilisée afin de déterminer le pire temps de réponse pour un scénario lorsque les priorités varient à l'intérieur de celui-ci. Ce cas correspond à la Technique 6 du livre de référence RMA (Klein *et al.*, 1993, pp. 4.42-4.53; Moreno et Hansen, 2009, p. 21). Il serait judicieux, pour ceux qui désirent bien comprendre la théorie RMA utilisée, d'étudier la référence initiale. Autrement, uniquement les formules nécessaires seront présentées lors de l'intégration des tactiques.

Cette technique s'utilise sur des systèmes avec un seul processeur où des événements (scénarios dans ArchE) déclenchent une série de tâches (responsabilités dans ArchE) à exécuter avant une échéance. Les échéances sont définies dans la mesure de réponse dans les scénarios d'ArchE. Chacune des tâches peut avoir une priorité et un temps d'exécution différents. L'analyse RMA retourne le pire temps de latence possible pour un scénario.

Avec ce cadre de raisonnement de performance, l'utilisateur connait le temps de réponse des scénarios de son système et sait si celui-ci respecte ou non les exigences de performance établies. Si le système ne répond pas à ses besoins (c.-à-d., au moins un des scénarios de performance n'est pas satisfait), une possibilité pour remédier à la situation est d'appliquer des tactiques architecturales.

1.5 Tactiques architecturales

Une tactique architecturale est « une manière de satisfaire une exigence de qualité d'un attribut en manipulant certains aspects du modèle d'attribut de qualité par des décisions architecturales » (Bachmann, Bass et Klein, 2003a, p. 47). Pour y arriver, chaque tactique

doit avoir un minimum d'information. Premièrement, il faut qu'elle ait un lien vers un ou plusieurs cadres de raisonnement pour savoir lorsqu'il faut l'utiliser. Deuxièmement, il faut que la tactique ait des informations sur ses relations avec un ou plusieurs paramètres et une certaine mesure de l'attribut pour déterminer comment agir sur ces paramètres. Finalement, la tactique doit contenir une explication pour faire le lien entre la tactique et le fragment de conception architectural (« design fragment ») (Bachmann, Bass et Klein, 2003a).

Concrètement, une tactique est une modification de l'architecture et son utilisation a des implications. Habituellement, l'application d'une tactique modifie l'architecture, soit en altérant sa structure, en modifiant une de ses propriétés ou en créant de nouvelles responsabilités qui lui sont nécessaires[5] (Bachmann, Bass et Klein, 2003b; Bachmann *et al.*, 2005; Lee et Bass, 2005). En plus de modifier ou d'ajouter de nouveaux éléments, il faut prendre en compte le fait qu'une tactique peut modifier, positivement ou négativement, plus d'un attribut de qualité lors de son application (Bachmann, Bass et Klein, 2003a). Par exemple, l'ajout d'un intermédiaire pour localiser les changements ajoute du traitement supplémentaire ce qui réduira la performance du système. Toutes ces informations doivent être accessibles à l'utilisateur pour lui permettre de faire un choix de tactiques éclairé.

Il est important de noter qu'une tactique ne pourra pas modifier n'importe quel paramètre. En effet, certains vont être fixés (ex : le nombre de processeurs) ou bornés (ex : le temps de réponse d'un composant acheté) selon les spécifications de l'utilisateur. Ces paramètres sont entrés manuellement lors de la saisie des scénarios. Le reste des paramètres associés aux responsabilités est considéré comme libre et ArchE pourra les faire varier afin d'obtenir une conception valide. Il est aussi important de noter que les paramètres fixes pourraient faire en sorte qu'il n'existe aucune solution possible pouvant répondre à toutes les exigences de qualité. Si tel est le cas, il faut relaxer un des paramètres fixes. Les cadres de raisonnement en combinaison avec les tactiques seront en mesure de proposer certaines modifications aux

[5] Par exemple, l'utilisation d'une tactique d'ordonnancement avec priorité fixe demande d'ajouter un ordonnanceur pour gérer les différents fils d'exécution.

paramètres fixes (typiquement ceux des scénarios) ayant pour effet de se rapprocher un peu plus du but. L'utilisateur devra choisir quelles modifications intégrer, car elles ne seront pas toutes réalisables (Bachmann, Bass et Klein, 2003a; Bachmann *et al.*, 2005).

Dans un logiciel comme ArchE, l'utilisation de tactiques dans un cadre de raisonnement lui permet d'explorer l'espace des solutions possibles pour le problème en cours (Bachmann *et al.*, 2005). Cette exploration peut être réalisée manuellement en explorant les tactiques potentielles une à une. Cependant, en plus de prendre du temps, seul un expert dans le domaine sera en mesure de bien analyser la situation. C'est pour cette raison qu'ArchE a été développé : il va permettre aux non-experts d'avoir accès à toutes ces connaissances. De plus, ceci implique que plus il y a de tactiques disponibles, plus l'espace de solution à explorer est grand, ce qui implique que la solution « optimale » pour la situation actuelle a plus de chance d'y être, tout en étant potentiellement plus difficile à trouver.

Pour permettre l'application des tactiques sur l'architecture, chacune d'elles doit être associée à un fragment de conception (Bachmann, Bass et Klein, 2003a). Un fragment de conception est un exemple montrant comment la tactique doit être appliquée, ainsi que les règles qui doivent être suivies pour y arriver. Par exemple, l'utilisation d'une tactique d'ordonnancement à priorité fixe doit déterminer quelles responsabilités doivent avoir une priorité plus élevée que d'autres. Un fragment de conception ressemblerait à quelque chose comme :

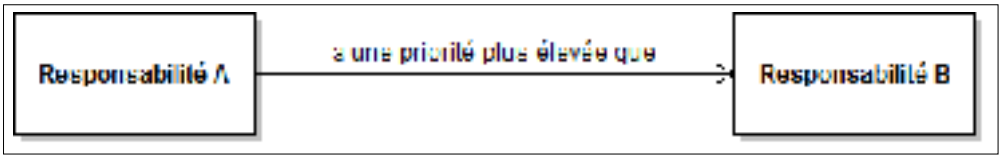

Figure 1.5 Fragment de conception générique.

Ce fragment est générique, car il est contenu dans la définition de la tactique. Cependant, il faudra l'instancier dans la conception de l'application en cours. Le fragment spécifique sera contenu dans la conception finale lorsque la tactique aura été appliquée au contexte :

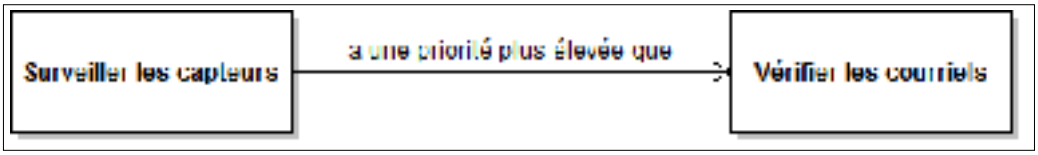

Figure 1.6 Fragment de conception spécifique.

En fonction des tactiques, le fragment de conception va varier. L'utilisation ne sera pas toujours identique; il faut donc qu'il y ait des règles d'application qui y soient associées. Par exemple, il faut utiliser un ordonnanceur lorsque la tactique « utiliser un ordonnancement à priorité fixe » est utilisée. Le fragment de conception est très important, car c'est ce qui va guider l'utilisateur dans son choix et lors de l'application de la tactique.

Pour sélectionner une tactique, il faut évidemment se baser sur des règles. Une grande partie des règles est basée sur les paramètres fixes. En effet, si une tactique doit modifier un paramètre fixe du système actuel, elle est rejetée. Une partie de l'analyse des règles sert donc à déterminer dans quel ordre les tactiques vont vérifier si leur paramètre est libre ou non. Plus spécifiquement, les règles doivent inclure (Bachmann, Bass et Klein, 2003a) :

- Des questions d'identification et de filtrage : pour déterminer les tactiques potentielles.
- Des questions d'optimisation : des heuristiques permettant d'utiliser des raccourcis lors de la sélection des tactiques.
- Des questions de relaxation : pour permettre de relâcher certaines contraintes trop restrictives.
- Des questions à poser à l'utilisateur : pour obtenir des informations additionnelles pour l'application de la tactique.

Par exemple, la tactique d'ajout d'un intermédiaire sera proposée si la probabilité qu'une modification se propage est plus grande que 75 %. Pour bien appliquer cette tactique, l'utilisateur devra donner le coût d'une modification de l'intermédiaire. La tactique va conserver la même probabilité que le changement se propage entre la source du changement et l'intermédiaire, mais va automatiquement réduire la probabilité de la propagation de 50 %,

par exemple, entre l'intermédiaire et la responsabilité originalement affectée par le changement.

Les règles d'application automatique de la tactique sont la partie la plus importante du projet, car c'est ce qui va aider les architectes à créer de meilleures architectures. Si les règles sont invalides ou mal intégrées, les suggestions faites par ArchE seront invalides ou non optimales.

1.6 Conclusion

En conclusion, une tactique sert à satisfaire une exigence de qualité par des modifications architecturales. L'utilité première des tactiques est d'explorer l'espace de solution à l'aide des cadres de raisonnement pour proposer des modifications intéressantes à l'utilisateur. Pour appliquer une tactique, il faut utiliser le fragment de conception et les règles qui lui sont associés. Les tactiques sont incluses dans un cadre de raisonnement pour lui permettre de faire des modifications sur l'architecture. Les cadres de raisonnement sont la base d'ArchE et lui permettent de raisonner à propos de plusieurs attributs de qualité. ArchE va servir aux architectes pour leur donner accès aux connaissances expertes en création d'architecture en utilisant la méthode ADD. Maintenant que la théorie a été examinée, l'analyse détaillée des cadres de raisonnement livrés avec ArchE suit.

CHAPITRE 2

ANALYSE ET MODIFICATION DES CADRES DE RAISONNEMENT

Le cadre de raisonnement de performance livré avec ArchE est uniquement capable d'effectuer l'analyse de l'architecture. Pour y ajouter des tactiques, il doit être modifié afin d'y ajouter la fondation sur laquelle les tactiques vont s'appuyer pour qu'elles puissent être proposées à l'utilisateur. Pour nous aider, le cadre de raisonnement de modificabilité est analysé, car il contient déjà un jeu de tactiques proposé à l'utilisateur. De plus, certaines fonctionnalités nécessaires aux tactiques de performance sont manquantes; elles sont aussi examinées dans ce chapitre.

Figure 2.1 Méthodologie : Analyse et modification d'ArchE.

Dans ce chapitre, les étapes 2 et 3 de la méthodologie sont exécutées, c'est-à-dire l'analyse des cadres de raisonnement contenus dans ArchE ainsi que la modification du cadre de raisonnement de performance pour y ajouter les fonctionnalités nécessaires au support des tactiques. L'analyse sert à identifier les forces et les défis des cadres de raisonnement afin de pouvoir intégrer de manière optimale les tactiques de performance. Les changements et ajouts effectués sont décrits en détail dans les sections suivant l'analyse.

2.1 Description d'un cadre de raisonnement dans ArchE

Dans la version 3.0 d'ArchE, les cadres de raisonnement sont externes au programme principal. Ils sont faits en code Java et sont intégrés comme plugiciel (« plugin ») dans ArchE. Pour communiquer avec les cadres de raisonnement, ArchE utilise le programme XmlBlaster qui se charge de transmettre les messages entre les différents intervenants. Afin

que la communication soit possible, ArchE fournit une interface que les cadres de raisonnement doivent implémenter.

Dans les prochaines sous-sections, les différents aspects d'un cadre de raisonnement sont décrits, notamment les fonctionnalités de base d'un cadre de raisonnement, les détails de l'interface utilisée pour communiquer, les différents aspects de la configuration d'un cadre de raisonnement et une description de l'utilisation d'un cadre de raisonnement par ArchE.

2.1.1 Fonctionnalités de base d'un cadre de raisonnement dans ArchE

Les cadres de raisonnement d'ArchE supportent cinq fonctionnalités de base (Diaz-Pace *et al.*, 2008, p. 11) :

- la description du cadre de raisonnement (par son manifeste, voir section 2.1.3);
- l'analyse;
- la suggestion de tactiques;
- l'application de tactiques;
- la description de tactiques.

Il n'est pas nécessaire qu'un cadre de raisonnement implémente toutes les fonctionnalités pour être intégré dans ArchE. Minimalement, il doit fournir un manifeste pour se décrire. En général, le cadre de raisonnement implémente aussi une forme d'analyse sur le système. Pour modifier l'architecture du système, le cadre de raisonnement doit implémenter, en plus de la description et l'analyse, la suggestion, l'application et la description de tactiques.

Par exemple, le cadre de raisonnement de modificabilité fourni avec ArchE implémente toutes les fonctionnalités. Il peut donc analyser le système et suggérer des améliorations si c'est nécessaire. Par contre, le cadre de raisonnement de performance implémente uniquement la description et l'analyse. Il fournit donc des informations sur la performance du système (c.-à-d., si les scénarios de performance sont satisfaits avec l'architecture actuelle),

mais ne peut rien proposer pour l'améliorer, advenant le cas où au moins un scénario de performance n'est pas satisfait.

2.1.2 Interface des cadres de raisonnement

Afin de rendre possible l'ajout de cadres de raisonnement externes à ArchE, une interface commune doit être respectée. Cette interface est fournie avec ArchE. Elle a pour but de faciliter l'ajout de cadres de raisonnement externes en servant de base pour les fonctionnalités qu'ils vont fournir. Cette section examine brièvement les différentes parties de l'interface.

Plusieurs parties de l'interface sont déjà implémentées afin de faire abstraction des détails pour la communication avec XmlBlaster et pour l'exécution des commandes d'ArchE. ArchE communique avec cette partie de l'interface déjà implémentée qui communique ensuite avec les parties implémentées par le développeur. Avec cette interface, le développeur voulant ajouter un nouveau cadre de raisonnement a uniquement à implémenter les connaissances théoriques de l'attribut de qualité pour répondre aux différentes requêtes d'ArchE.

Plusieurs paquetages (« package ») Java sont contenus dans l'interface. Cependant, un seul est intéressant du point de vue d'un développeur voulant créer un nouveau cadre de raisonnement : le paquetage « reasoningframeworks ». Ce paquetage contient plusieurs classes, certaines sont déjà implémentées alors que d'autres ont besoin d'être modifiées.

Certaines classes servent de conteneur d'information utile à la communication avec ArchE. Elles sont déjà implémentées en totalité et peuvent être utilisées telles quelles. Le cœur du cadre de raisonnement se trouve dans la classe « ArchEReasoningFramework ». Bien que cette classe abstraite soit en partie implémentée (pour fournir différents services de base), il faut en hériter et ajouter les différents services d'un cadre de raisonnement que l'on veut fournir.

Les méthodes à implémenter dans la classe « ArchEReasoningFramework » sont :

- *checkRFDependencies* : pour valider l'architecture actuelle et faire des ajustements à celle-ci (par exemple, si un autre cadre de raisonnement l'a modifiée, on peut définir ou ajuster des paramètres);
- *analyzeAndSuggest* : pour analyser le système et proposer des tactiques si un scénario n'est pas satisfait; est utilisée quand ArchE évalue l'architecture;
- *analyze* : pour analyser uniquement le système par rapport à l'attribut de qualité; est utilisée lorsqu'ArchE veut uniquement obtenir le résultat de l'application automatique d'une tactique sur une architecture temporaire;
- *applySuggestedTactic* : pour appliquer une tactique de manière automatisée sans information supplémentaire fournie par l'utilisateur;
- *applyTacticByUserQuestion* : pour appliquer une tactique de manière automatisée avec des données fournies par l'utilisateur (après avoir été affichée à celui-ci);
- *describeTactic* : pour décrire la tactique afin qu'ArchE puisse l'afficher à l'utilisateur;
- *describeOtherThanTactic* : pour décrire des messages autres que les tactiques (ex. : des erreurs).

Puisqu'il n'est pas nécessaire d'implémenter tous les services d'un cadre de raisonnement, il n'est pas nécessaire d'implémenter toutes ces méthodes. Il faut cependant noter qu'un cadre de raisonnement implémentant *analyze* doit aussi implémenter *analyzeAndSuggest* en ne retournant aucune tactique, car les deux méthodes sont appelées à différents moments lors du processus d'ArchE.

Le tableau suivant fait le lien entre les services offerts par le cadre de raisonnement et les méthodes de l'interface qui réalisent ce service.

Tableau 2.1 Correspondance entre les services offerts par un cadre de raisonnement et les méthodes de l'interface d'ArchE

Service	Méthode
Description du cadre de raisonnement	Voir manifeste section 2.1.3
Analyse	*checkRFDependencies, describeOtherThanTactic, analyzeAndSuggest , analyze*
Suggestions de tactiques	*analyzeAndSuggest*
Application de tactiques	*applyTacticByUserQuestion*, *applySuggestedTactic*
Description de tactiques	*describeTactic*

Cette section examine l'interface fournie par ArchE de manière statique. Les deux prochaines sections décrivent comment un cadre de raisonnement est utilisé par ArchE.

2.1.3 Configurations et paramètres des cadres de raisonnement

Un cadre de raisonnement possède deux fichiers de configuration : le manifeste et les questions. Cette section décrit ces deux fichiers.

2.1.3.1 Description du manifeste

La seule partie obligatoire d'un cadre de raisonnement est sa description par un manifeste; c'est grâce à lui que le cadre de raisonnement s'annonce à ArchE. Le manifeste est la manière pour le cadre de raisonnement d'énoncer la structure des informations pour les types de données dont il va avoir besoin. Le manifeste décrit ces types, mais ne donne aucune information quant à leur utilisation. ArchE utilise ces informations pour déduire les fenêtres et les types requis par le cadre de raisonnement pour fonctionner adéquatement.

Le manifeste est un fichier XML. La racine de ce fichier (balise <*rf*>) décrit le cadre de raisonnement. Cette balise contient quatre parties : la description des types de scénario (balise <*scenarioTypes*>), la description de la structure des responsabilités (balise <*responsibilityStructure*>), la description du modèle (balise <*model*>) et la description de la

vue (balise <*view*>). Le reste de cette section se concentrera sur les aspects généraux du manifeste. Pour avoir plus d'informations techniques sur le format du manifeste, voir (Diaz-Pace, Kim et Bianco, 2008).

La description des scénarios permet de définir un nouveau type de scénario pour l'attribut de qualité analysé par le cadre de raisonnement. Les six parties du scénario y sont définies. Le manifeste indique le nom, le type, la valeur par défaut, ainsi que leur correspondance dans la base de données de chacune de ces parties.

Le manifeste spécifie aussi la structure des responsabilités nécessaires pour le cadre de raisonnement. Cette structure définit deux choses : les paramètres associés aux responsabilités et les relations possibles entre les responsabilités. L'ajout de paramètres aux responsabilités sert à obtenir plus d'information sur le système. Chaque cadre de raisonnement peut spécifier les paramètres dont il a besoin et les gérer comme il le veut à l'interne. Il en est de même pour les relations entre les responsabilités. Plusieurs types de relations pour différentes fonctions peuvent être définis. De plus, il est aussi possible de définir des paramètres s'appliquant à la relation elle-même (par exemple, la probabilité qu'une modification se propage sur un lien d'association entre deux responsabilités).

La description du modèle et de la vue sont des paramètres gérés à l'interne par le cadre de raisonnement. Ces spécifications servent à indiquer quelles informations spécifiques au cadre de raisonnement doivent être sauvegardées dans la base de données d'ArchE.

2.1.3.2 Description du fichier de questions

En plus du manifeste, ArchE lit et traite un fichier contenant les questions. Les questions sont utilisées pour toutes les communications entre le cadre de raisonnement et l'utilisateur, notamment pour l'avertir s'il y a un problème lors de l'analyse et pour lui proposer d'appliquer une tactique.

Contrairement au manifeste, le fichier de questions est un fichier de texte simple. Chaque question est représentée par un identifiant unique. Pour chacun des identifiants, plusieurs paramètres doivent être configurés. Les paramètres sont définis en utilisant la forme « identifiant.paramètre = valeur ». Chaque ligne du fichier peut contenir un paramètre ou un commentaire.

Les paramètres configurables d'une question sont :

- le type : détermine la présentation et le type de réponse à la question (case à cocher, oui/non, …);
- la catégorie : affichée comme titre de la question;
- le but : affiché comme sous-titre de la question;
- la question : affichée comme texte de la question;
- les étiquettes : paramètre variable en nombre (ou optionnel, selon le type de question) qui est utilisé pour demander la réponse à la question.

Une fonctionnalité importante dans le texte de la question (qui est statique) est la possibilité d'y ajouter des étiquettes remplaçables dynamiquement. Ces étiquettes ont la forme <*n*> et seront remplacées par le $n^{ième}$ paramètre. Ces paramètres sont définis par le cadre de raisonnement dans la question retournée à ArchE ce qui permet de personnaliser une question selon la situation.

Lors du démarrage, ArchE lit le fichier et combine toutes les informations pour chaque identifiant. Lorsque le cadre de raisonnement décrit quelque chose et qu'il veut utiliser une des questions, il utilise l'identifiant de celle-ci et ArchE affiche les informations correspondantes.

2.1.4 Utilisation d'un cadre de raisonnement par ArchE

Les spécifications de l'interface des cadres de raisonnement ont été décrites dans les sections précédentes. Cette section examine le fonctionnement d'ArchE avec cette interface. On y

explique les différentes étapes d'une exécution typique d'un cadre de raisonnement du point de vue d'ArchE. L'implémentation plus spécifique des cadres de raisonnement existants est examinée dans les prochaines sections où nous verrons comment ils répondent aux différentes requêtes d'ArchE.

Le diagramme de séquence suivant montre une exécution normale d'ArchE dans deux cas d'utilisation : lorsqu'un utilisateur modifie manuellement l'architecture (ex. : changer un temps d'exécution) et lorsqu'un utilisateur décide d'appliquer automatiquement une tactique (ex. : ajouter un intermédiaire pour réduire l'impact d'un changement). Il faut noter que ce diagramme montre un seul cadre de raisonnement. Dans le cas où il y a plusieurs cadres de raisonnement, ArchE répète les mêmes étapes pour chacun d'eux.

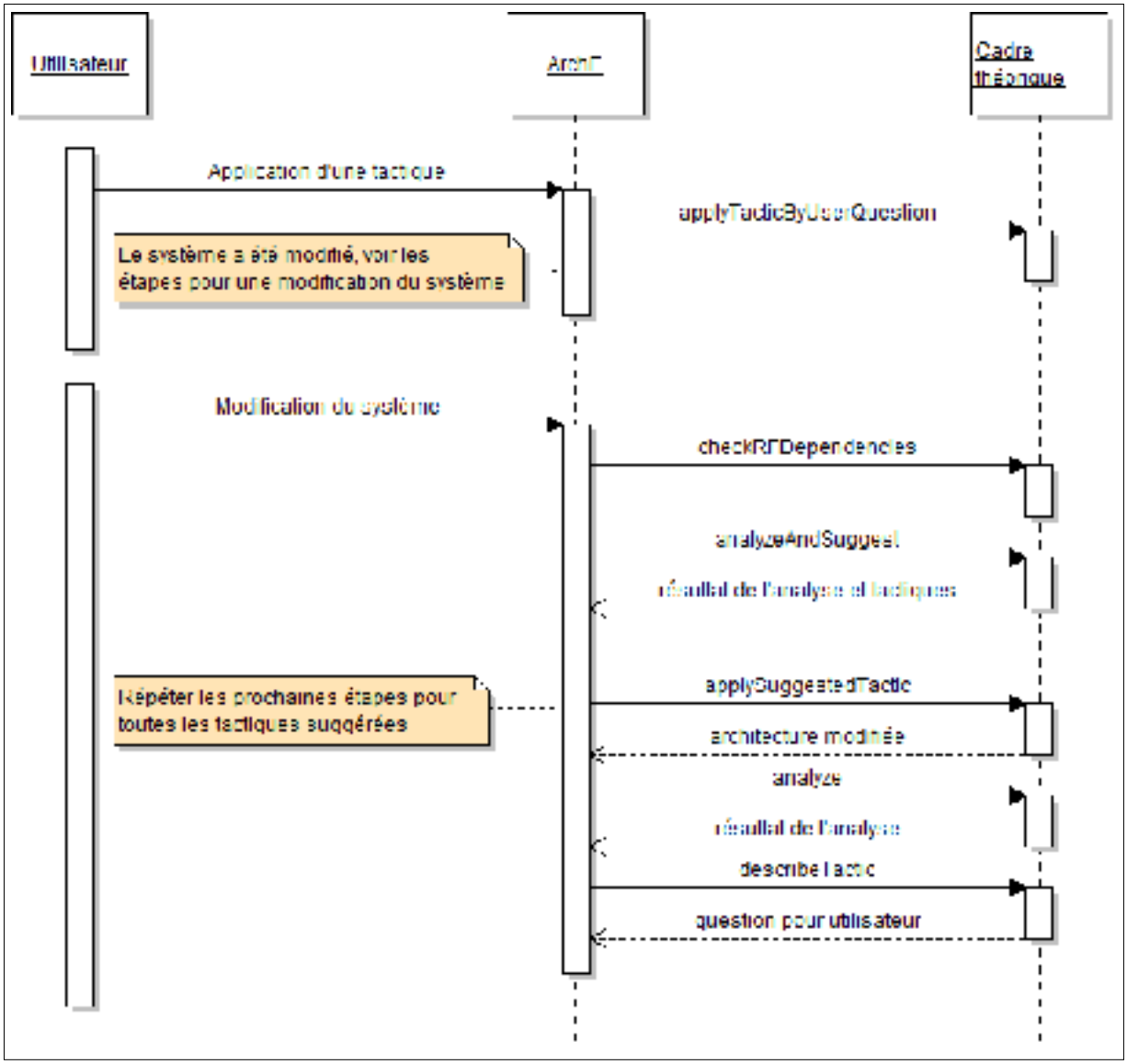

Figure 2.2 Diagramme de séquence des étapes suivies par ArchE avec un cadre de raisonnement.

L'exécution de ces étapes peut donner deux résultats. Premièrement, si l'architecture répond aux exigences de tous les scénarios définis, ArchE affiche les résultats et aucune tactique ne devrait être retournée par les cadres de raisonnement. Le deuxième résultat se produit lorsque l'architecture ne répond pas aux exigences : les tactiques sont suggérées et examinées et ArchE affiche les questions retournées par le cadre de raisonnement à l'utilisateur. C'est avec ces questions qu'il va pouvoir décider d'appliquer (ou non) une tactique.

ArchE analyse chacune des tactiques proposées pour plusieurs raisons. Premièrement, cela lui permet de déterminer si la tactique est applicable sur l'architecture. Si elle a été suggérée et que la méthode d'application ne parvient pas à l'appliquer correctement, ArchE va rejeter la suggestion. L'analyse des suggestions sert aussi à déterminer l'impact que chacune des tactiques va avoir sur l'architecture. Avec ces informations, ArchE va trier les tactiques et afficher l'impact potentiel qu'elles auraient sur le système à l'utilisateur.

De plus, ArchE élimine les doublons dans les suggestions. Deux suggestions sont identiques lorsqu'il s'agit de la même tactique qui s'applique avec les mêmes paramètres. Si des doublons sont trouvés, ArchE va rejeter le double et garder une seule suggestion.

2.2 Implémentation actuelle du cadre de raisonnement de performance

Tel que livré avec ArchE 3.0, le cadre de raisonnement de performance effectue uniquement l'analyse sur le système. Aucune tactique n'est proposée si le système ne répond pas aux exigences de performance définies. Cette section examine le fonctionnement du cadre de raisonnement de performance afin de déterminer ce qui existe et ce qui manque pour l'intégration de tactiques de performance.

Le diagramme de séquence suivant montre les différentes étapes effectuées lorsqu'une modification au système est effectuée et qu'il est analysé.

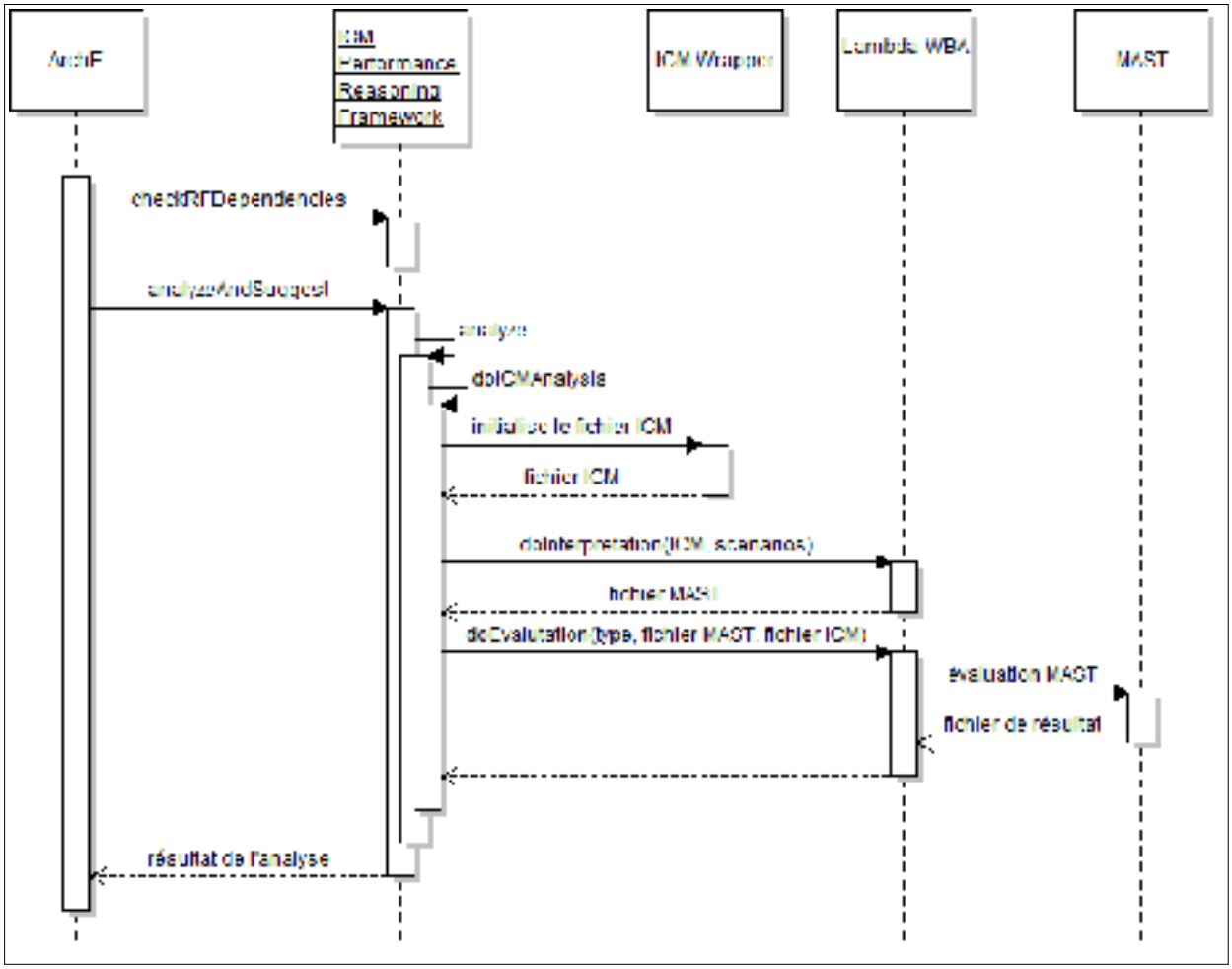

Figure 2.3 Diagramme de séquences du cadre de raisonnement de performance original.

Les étapes de communication entre Lambda-WBA et MAST sont inaccessibles dans le code du cadre de raisonnement. Elles ont été déduites par les commentaires laissés dans le code par ses développeurs ainsi qu'avec des tests effectués directement avec l'outil MAST.

La méthode *checkRFDependencies* va vérifier que le paramètre du temps d'exécution existe. Si le paramètre n'existe pas, il est créé avec une valeur par défaut[6]. Cette méthode supprime aussi les liens « réaction » orphelins (lorsqu'une des deux responsabilités est supprimée).

[6] Une valeur par défaut ajoutée automatiquement sera affichée en vert dans l'interface d'ArchE afin d'en informer l'utilisateur. Il aurait été possible de retourner une erreur à ArchE. Cependant, la définition automatique des paramètres permet à l'analyse des autres cadres de raisonnement de continuer normalement.

Le rôle de l'objet *ICMWrapper* est de produire un fichier ICM pour l'analyse. Pour y arriver, il examine l'architecture du système et traduit les différents objets dans leur correspondance ICM. Pour plus de détails concernant la correspondance entre les différents objets et concepts ICM, veuillez vous référer à la section 1.4.2.

La méthode *doInterpretation* traduit automatiquement le fichier ICM en fichier MAST, alors que la méthode *doEvalutation* utilise le fichier MAST produit pour analyser la performance du système et retourne les résultats dans un nouveau fichier. Celui-ci est lu par le cadre de raisonnement et les résultats sont retournés à ArchE.

Puisqu'aucune tactique n'est retournée, les autres méthodes de l'interface du cadre de raisonnement ne sont pas utilisées. ArchE ignore si le cadre de raisonnement a implémenté ou non des tactiques, il utilise uniquement les données qui lui sont retournées (dans ce cas, seulement des résultats d'analyse). Tout ce qui importe à ArchE est que le cadre de raisonnement implémente complètement l'interface, même si certaines méthodes peuvent n'avoir aucun contenu et qu'elles ne sont jamais utilisées.

2.2.1 Parties manquantes au cadre de raisonnement de performance pour l'application des tactiques

Avec l'analyse du cadre de raisonnement de performance, il est possible de déduire certains aspects théoriques manquant afin que l'ajout de la suggestion et de l'application automatique des tactiques de performance puisse être réalisé. Cette section examine les points manquants par rapport à l'interface des cadres de raisonnement.

La première étape pour intégrer des tactiques dans le cadre de raisonnement de performance est d'implémenter la partie « suggestion » de la méthode *analyzeAndSuggest*. Cette partie sert à retourner des suggestions de tactique à ArchE pour qu'il puisse les analyser et les proposer à l'utilisateur.

Plusieurs nouvelles méthodes doivent être implémentées : *applySuggestedTactic*, *applyTacticByUserQuestion* et *describeTactic*. La description de la tactique permet d'afficher une question à l'utilisateur lui demandant d'entrer les informations supplémentaires nécessaires pour la tactique et de confirmer son choix. Il est important de noter que les deux méthodes d'application appliquent la tactique de la même manière sur le système. Cependant, *applySuggestedTactic* utilise, lorsque nécessaire, des valeurs par défaut afin d'avoir une estimation de l'effet de la tactique et *applyTacticByUserQuestion* utilise les informations fournies par l'utilisateur une fois qu'il a répondu aux questions transmises par ArchE.

Une dernière partie du cadre de raisonnement doit être modifiée. Il s'agit de l'ajout d'un nouveau paramètre pour les responsabilités : sa priorité. Actuellement, la priorité est assignée séquentiellement (selon l'ordre d'apparition des objets dans les différentes structures de données contenant l'information sur l'architecture) lors de la création du fichier ICM utilisé pour l'analyse. Cet ordre n'est pas contrôlable et peut varier à chaque exécution de l'analyse. Il n'est donc pas possible de reproduire facilement un même système entre les différentes exécutions. De plus, si un objet se trouve deux fois dans une des structures de données, il aura deux priorités différentes (par exemple, une responsabilité faisant partie de deux liens).

Puisque l'analyse se base sur la priorité pour déduire quelle responsabilité s'exécutera à tout moment, il faut que l'utilisateur puisse la déterminer. De plus, certaines tactiques pourraient utiliser ce paramètre afin de modifier l'architecture. Il faut donc y avoir accès facilement. En plus de modifier les fichiers de configuration pour ajouter ce paramètre, il faut modifier la méthode *checkRFDependencies* afin qu'elle vérifie la présence et la validité de la priorité et modifier l'analyse afin qu'elle utilise le paramètre plutôt qu'une attribution séquentielle.

Cette section a décrit le cadre de raisonnement de performance dans son état initial ainsi que les aspects théoriques manquant pour ajouter des tactiques. Afin d'avoir une base concrète sur le fonctionnement de la suggestion et de l'application automatique des tactiques, la prochaine section examine un autre cadre de raisonnement ayant cette capacité.

2.3 Implémentation actuelle du cadre de raisonnement de modificabilité

En plus du cadre de raisonnement de performance, ArchE 3.0 est livré avec un cadre de raisonnement de modificabilité. Ce dernier est intéressant, car il possède la capacité de suggérer et d'appliquer automatiquement des tactiques de modificabilité. Même si les tactiques sont différentes, il est tout de même possible d'en étudier le fonctionnement. Cette section examine donc le processus utilisé par le RF de modificabilité pour suggérer et appliquer automatiquement des tactiques. Il est ainsi possible d'en sortir les forces et les défis pour ainsi avoir une base intéressante pour intégrer les tactiques dans le cadre de raisonnement de performance.

Le diagramme de séquence suivant résume l'implémentation du cadre de raisonnement de modificabilité. Puisque ce qui nous intéresse est le processus d'utilisation des tactiques, les détails de l'implémentation de l'analyse de modificabilité ont été omis afin d'alléger le diagramme.

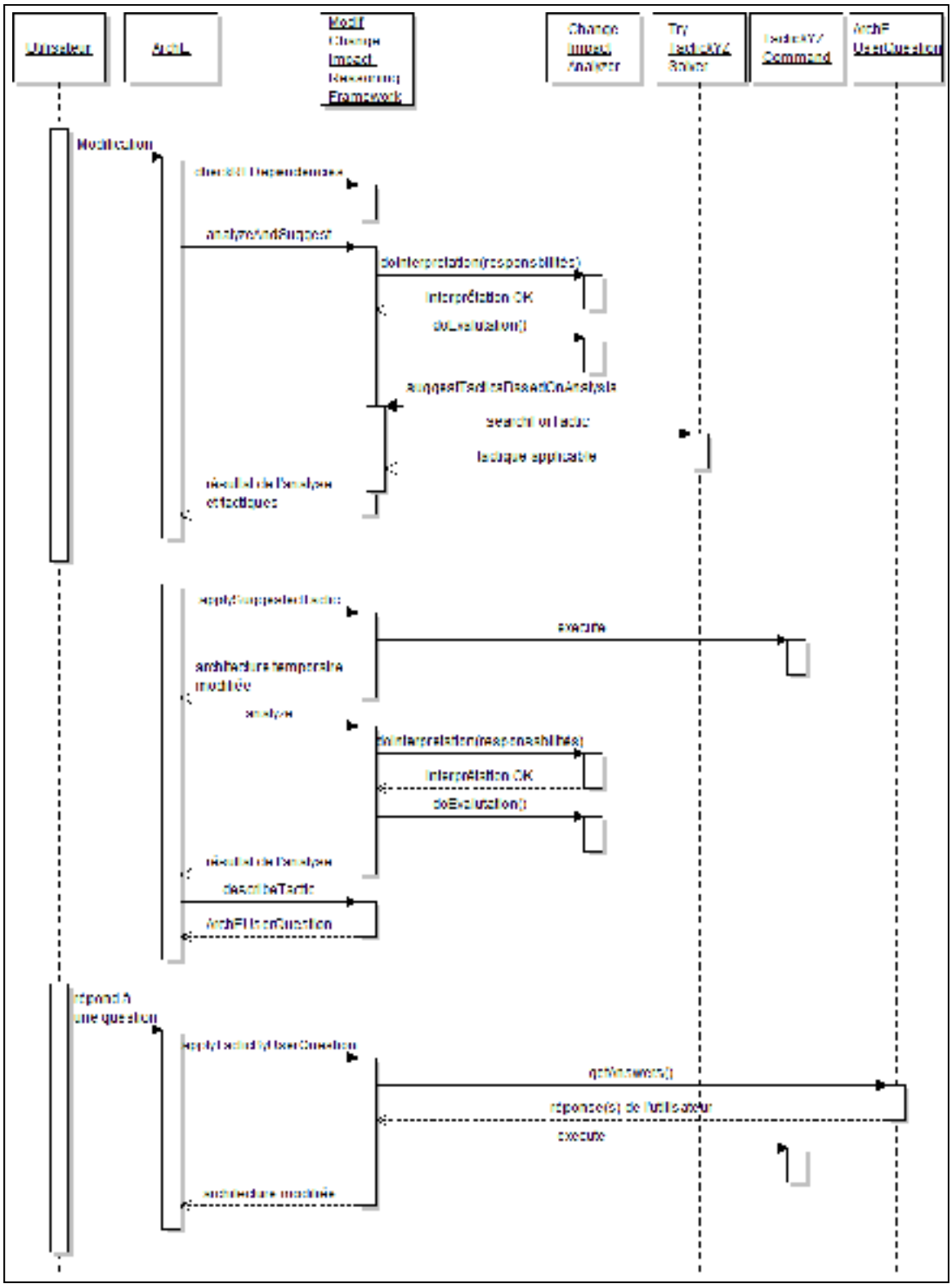

Figure 2.4 Diagramme de séquence du cadre de raisonnement de modificabilité, avec les tactiques.

La méthode *checkRFDependencies* n'effectue aucun traitement spécifique pour la gestion des tactiques. Tout comme la méthode dans le cadre de raisonnement de performance, elle vérifie les paramètres et les liens contenus dans l'architecture.

Les méthodes *doInterpretation* et *doEvalutation* sont utilisées pour effectuer l'analyse. L'interprétation initialise l'analyseur, alors que l'évaluation va effectuer et retourner le résultat de l'analyse. Si les résultats de l'analyse font en sorte qu'un scénario n'est pas satisfait, la méthode *suggestTacticsBasedOnAnalysis* est appelée. Cette méthode interne du cadre de raisonnement utilise une série de conditions où chacune utilise un résolveur différent. Chacun de ces résolveurs est une classe séparée du cadre de raisonnement et n'a qu'une seule fonction : vérifier si une tactique s'applique au système. Si elle s'applique, une suggestion de tactique sera retournée à ArchE. Il peut y avoir zéro, une ou plusieurs suggestions de tactiques retournées à ArchE.

Les méthodes *applySuggestedTactic* et *applyTacticByUserQuestion* sont pratiquement identiques dans leur forme et leur fonction. La seule différence vient du fait que *applyTacticByUserQuestion* va chercher les réponses de l'utilisateur dans la question, alors que *applySuggestedTactic* va utiliser des valeurs par défaut inscrites dans le cadre de raisonnement. Ces deux méthodes vont utiliser des objets internes (privés) au cadre de raisonnement en utilisant le patron « commande ». La commande est initialisée avec les paramètres de la tactique ainsi qu'avec les informations fournies par l'utilisateur (*applyTaticByUserQuestion*) ou avec les informations par défaut (*applySuggestedTactic*). Dans les deux cas, la commande s'exécutera de la même manière en utilisant des données provenant des différentes sources.

Pour terminer, la méthode *describeTactic* retourne une question décrivant la tactique. ArchE appelle cette méthode pour chacune des tactiques proposées par le cadre de raisonnement. Pour retourner la bonne question, une séquence de conditions vérifie de quel type de tactique il s'agit et retourne la question appropriée. Les questions contiennent plusieurs informations. Il y a le type de question dont il s'agit (pour obtenir le texte contenu dans le fichier de

questions), la tactique à laquelle la question réfère (pour l'application par l'utilisateur) et des paramètres définis par le concepteur du cadre de raisonnement (soit pour remplacer des variables dans le texte de la question, soit pour l'application de la tactique).

2.3.1 Problèmes de l'implémentation actuelle

En analysant la gestion des tactiques du cadre de raisonnement de modificabilité, plusieurs points problématiques ont été repérés. Cette section examine ces points afin d'y remédier ou de les éviter afin de développer une meilleure stratégie pour gérer les tactiques dans le cadre de raisonnement de performance.

L'aspect le plus problématique vient du couplage très élevé entre l'exécution du cadre de raisonnement et les tactiques. En effet, la classe correspondant au cadre de raisonnement devrait s'occuper uniquement de l'exécution des commandes provenant d'ArchE et reléguer les tâches d'analyse connexe à d'autres modules. Cette approche est déjà suivie pour l'analyse (avec la classe *ChangeImpactAnalyser*). Par contre, le code servant à gérer les tactiques se retrouve en majorité dans la classe du cadre de raisonnement.

En effet, la suggestion des tactiques se fait dans une méthode privée du cadre de raisonnement. Même si cette méthode utilise des résolveurs externes au cadre de raisonnement, il reste tout de même une longue série de conditions vérifiant chacun des résolveurs. Il y a un point positif avec la séparation des résolveurs pour déterminer si une tactique s'applique au système : l'utilisation d'une interface commune. Cette interface pourrait faire abstraction des détails de l'implémentation des résolveurs, mais ce n'est pas le cas dans cette implémentation. En effet, le cadre de raisonnement doit créer lui-même chacun des résolveurs implémentant cette interface. Ceci fait en sorte qu'il a un lien direct avec chacun d'eux, alors qu'il ne devrait pas les connaître. Il faudrait isoler les résolveurs correctement (ex. : dans chacune des tactiques).

Contrairement à la suggestion des tactiques, les deux méthodes d'application vont utiliser des classes internes (privées) au cadre de raisonnement pour effectuer leurs opérations. Cependant, le reste est similaire. Il y a une série de conditions, exécutées dans le cadre de raisonnement, vérifiant chaque type de tactique pour déterminer la bonne commande à exécuter. De plus, le code des deux méthodes d'application est pratiquement une copie conforme. Cette duplication pose un défi de maintenance du code, car une modification faite dans une des méthodes doit absolument être faite dans l'autre, sinon il pourrait y avoir des conflits.

La description des tactiques est aussi problématique. Premièrement, il s'agit encore d'une série de conditions vérifiant chacune le type de la tactique décrite. De plus, la description est elle aussi faite dans le cadre de raisonnement. Extraire cette information et la conserver dans un module de tactiques externe permettrait d'augmenter la cohésion du cadre de raisonnement.

Tous ces points, en plus de réduire la cohésion du cadre de raisonnement en incluant d'autres tâches, font que des modifications aux tactiques vont aussi toucher au cadre de raisonnement. La nouvelle implémentation devrait mieux isoler les tactiques du cadre de raisonnement. Cela permettrait de faciliter l'ajout et la modification de tactiques.

En résumé, l'implémentation de chaque tactique devrait être mieux encapsulée. Cette encapsulation permettrait d'extraire toutes les informations concernant les tactiques (recherche, application et description) du cadre de raisonnement et de les contenir dans des objets représentants des tactiques.

2.4 Implémentation proposée des tactiques dans le cadre de raisonnement de performance

L'analyse des cadres de raisonnement effectuée dans les deux dernières sections permet de développer une solution pour l'intégration des tactiques dans le cadre de raisonnement de performance. Cette section décrit l'implémentation choisie et comment celle-ci règle les

problèmes énoncés précédemment. La description se limite à l'aspect technique de l'implémentation et ne touche pas à l'implémentation des tactiques de performance. Le développement d'une solution automatisée pour les tactiques de performance dans ArchE est décrit dans le troisième chapitre.

Le diagramme de classes suivant résume les différentes interactions entre les composantes actuelles du cadre de raisonnement et des nouvelles composantes ajoutées pour les tactiques. Il ne s'agit pas d'un diagramme UML strict, car il y a des composantes de type « méthode » qui correspondent aux méthodes de l'interface d'ArchE. Ces méthodes sont incluses afin de faciliter la compréhension de l'utilisation des classes puisqu'elles contiennent l'appel que chacune de ces méthodes doit faire. De plus, deux exemples de tactiques de performance sont inclus, sans plus de détails, afin de montrer l'utilisation des objets.

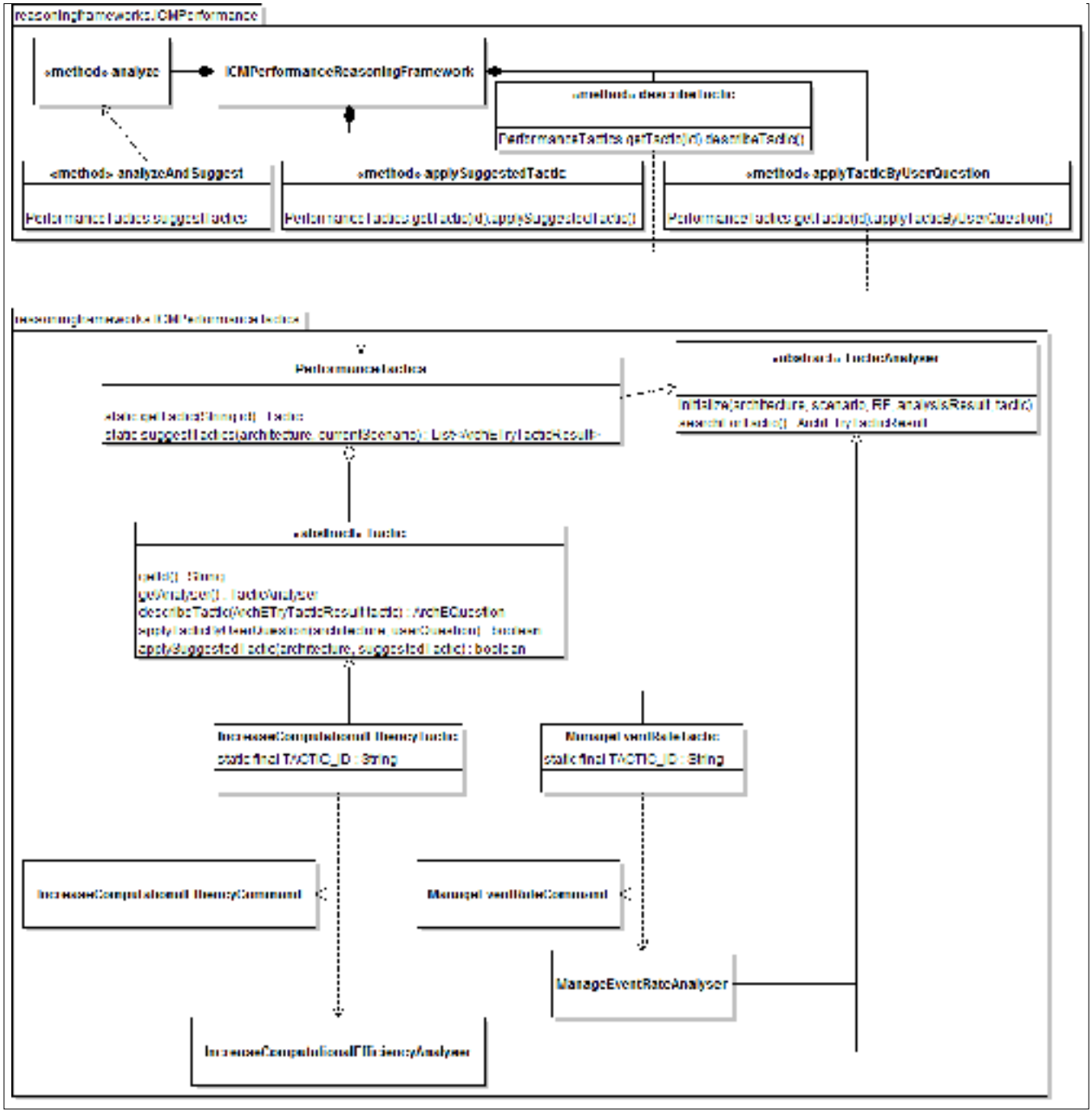

Figure 2.5 Diagramme de classes de l'implémentation des tactiques dans le cadre de raisonnement de performance.

L'objet statique *PerformanceTactics* est le point d'entrée pour toutes les tâches reliées aux tactiques. Les méthodes du cadre de raisonnement utilisent la méthode *getTactic* avec l'identifiant d'une tactique pour obtenir l'objet *Tactic* voulu. Grâce à l'héritage et au

polymorphisme, le cadre de raisonnement utilise uniquement l'objet générique *Tactic* et ne verra jamais l'implémentation spécifique (ex : *ManageEventRateTactic*). De plus, la recherche de la tactique selon l'identifiant n'a besoin d'être faite qu'à un seul endroit plutôt que dans chacune des méthodes du cadre de raisonnement. Conséquemment, l'ajout d'une nouvelle tactique a un impact à un seul endroit. Puisqu'il contient toutes les tactiques, *PerformanceTactics* a aussi la tâche de déterminer les tactiques à suggérer. Pour ce faire, le cadre de raisonnement peut appeler la méthode *suggestTactics*. Cette nouvelle méthode correspond à la méthode *suggestTacticsBasedOnAnalysis* du cadre de raisonnement de modificabilité. Elle vérifie, à l'aide des analyseurs fournis par les tactiques, si celles-ci ont des suggestions à proposer. S'il y en a, la méthode les rassemble et les retourne au cadre de raisonnement.

La classe *Tactic* est la classe de base pour toutes les tactiques à intégrer. Chacune de ces tactiques doit être instanciée dans l'objet statique *PerformanceTactics*. La tactique contient les paramètres utiles pour son analyse (par exemple, l'identifiant et les valeurs par défaut). La classe fournit aussi des méthodes pour accéder aux autres objets liés :

- la méthode *getSolver* retourne l'analyseur correspondant à la tactique;
- chaque méthode d'application utilise l'objet *command* de la tactique en le configurant selon la situation;
- la méthode *describeTactic* retourne une description de la tactique sous la forme d'une question pour l'utilisateur.

Les analyseurs correspondent aux résolveurs dans l'implémentation du cadre de raisonnement de modificabilité. Ils sont initialisés et retournés par la tactique correspondante afin d'être utilisés par *PerformanceTactics* pour déterminer les suggestions de tactiques possibles. Contrairement au cadre de raisonnement de modificabilité, ces analyseurs sont connus uniquement par la tactique correspondante; une interface commune est ensuite utilisée pour toutes les opérations.

Bien qu'aucune interface ne soit définie pour ce type, il est suggéré de créer un objet de type « commande » pour chacune des tactiques. Cet objet est responsable de modifier l'architecture selon les paramètres définis par la tactique. Les commandes ne sont pas définies spécifiquement (par exemple, avec une interface) afin d'être facilement adaptables à toutes les situations. Puisqu'elles touchent uniquement à une seule tactique, les différentes implémentations sont très localisées. Bien entendu, si l'application de la tactique est peu compliquée, le code pourrait être contenu directement dans les méthodes puisqu'elles sont utilisées pour une seule tactique. Ces objets « commandes » correspondent aux objets commandes du cadre de raisonnement de modificabilité, mis à part qu'ils sont gérés par leur tactique plutôt que d'êtres internes au cadre de raisonnement.

Puisqu'il possède toutes les informations de la tactique, il est naturel que l'objet *Tactic* soit utilisé pour sa propre description. De cette manière, chacun des objets *Tactic* décrit ce qu'il connait, ce qui permet d'enlever des connaissances et des responsabilités du cadre de raisonnement.

En conclusion, l'implémentation satisfait les exigences pour l'implémentation des tactiques tout en évitant ou en réglant les problèmes rencontrés dans l'implémentation des tactiques de modificabilité. Toutes les informations concernant les tactiques sont contenues dans un module séparé du cadre de raisonnement, les interfaces sont correctement utilisées afin de minimiser les liens avec les implémentations et le cadre de raisonnement de performance n'a jamais accès aux détails des tactiques. Chacune des tactiques connait uniquement son analyseur, son objet commande et sa description; elle n'a aucune information sur l'existence des autres tactiques. Il y a uniquement la classe *PerformanceTactics* qui connait toutes les tactiques disponibles. Comparée à celle du cadre de raisonnement de modificabilité, cette implémentation réduit le couplage du cadre de raisonnement avec les tactiques et augmente la cohésion des tactiques.

2.5 Autres modifications nécessaires au cadre de raisonnement

En plus de l'ajout des classes, deux autres modifications sont requises : l'ajout du paramètre de priorité pour les responsabilités et l'ajout des questions pour les tactiques.

2.5.1 Ajout du paramètre de priorité

Pour ajouter un nouveau paramètre de priorité pour les responsabilités, trois choses doivent être modifiées dans le cadre de raisonnement : le manifeste, la vérification de l'architecture et l'analyse.

Pour rendre disponible le paramètre de priorité pour les responsabilités dans le cadre de raisonnement de performance, il faut d'abord l'ajouter dans le fichier de configuration. La priorité est définie comme un entier avec une valeur par défaut de -1. Elle est ensuite associée aux responsabilités.

En plus de l'ajout au fichier de configuration, le nouveau paramètre doit être validé dans la méthode *checkRFDependencies*. Cette méthode est nécessaire afin de valider l'existence et la valeur du paramètre. La priorité est valide si elle a une valeur entre 1 et 399 inclusivement. La valeur par défaut de -1 donnera donc une erreur et forcera l'utilisateur à définir sa propre valeur. Les contraintes pour la valeur des priorités (c.-à-d., 1 à 399) sont définies par l'outil MAST. Si un autre type d'analyse était effectué, le type du paramètre et les valeurs valides pourraient être différents.

L'analyse doit aussi être modifiée pour utiliser la valeur de la priorité de la responsabilité plutôt qu'une valeur séquentielle. L'interface d'ArchE rend disponibles ces informations par diverses méthodes. En utilisant la valeur contenue dans la responsabilité, on s'assure que chaque responsabilité a une priorité clairement définie et qu'elle est la même dans tous les liens où la responsabilité se trouve.

2.5.2 Ajout des questions

Afin d'afficher les tactiques à l'utilisateur, il faut ajouter deux choses dans le fichier de questions : les erreurs et les tactiques. Premièrement, de nouvelles erreurs doivent être définies, notamment lorsque le paramètre de priorité d'une responsabilité n'existe pas ou que la valeur est invalide. Ensuite, une question doit être définie pour chacune des tactiques. Cette question contiendra le type de réponse requise (par exemple, une réponse textuelle ou la sélection d'une option), le titre, le sous-titre et le texte à afficher à l'utilisateur pour décrire la tactique. L'utilisation de paramètres peut être utile afin de donner des informations spécifiques à l'utilisateur concernant l'application de la tactique sur le système actuel.

2.6 Conclusion

Dans ce chapitre, les cadres de raisonnement ont été analysés dans le contexte d'ArchE. L'utilisation générale des cadres de raisonnement par ArchE a été décrite en premier, pour ensuite analyser spécifiquement les deux cadres de raisonnement livrés avec celui-ci. Les parties manquantes du cadre de raisonnement de performance ont été décrites et les forces et les faiblesses de l'implémentation des tactiques du cadre de raisonnement de modificabilité ont été analysées.

Des modifications ont aussi été proposées pour l'implémentation des tactiques dans le cadre de raisonnement de performance. Ces modifications sont majoritairement reliées à la séparation des tâches entre le cadre de raisonnement et les tactiques de performance. Cette nouvelle implémentation permet d'avoir une plus grande cohésion des composantes tout en réduisant le couplage entre les fonctionnalités. Des modifications supplémentaires ont été proposées, notamment concernant l'ajout d'un nouveau paramètre ainsi que des nouvelles questions pour communiquer avec l'utilisateur.

CHAPITRE 3

TACTIQUES AJOUTÉES

Avec les modifications effectuées au cadre de raisonnement, il est maintenant possible d'y ajouter les tactiques de performance. Pour faire cet ajout, les tactiques proposées dans le livre (Bass, Clements et Kazman, 2003, p. 111) seront analysées pour développer une méthode automatisée qu'il serait possible d'intégrer dans le cadre de raisonnement de performance d'ArchE. La Figure 3.1 présente un résumé de cette liste de tactiques telles que présentés dans le livre. Les tactiques seront développées en détail plus loin dans cette section.

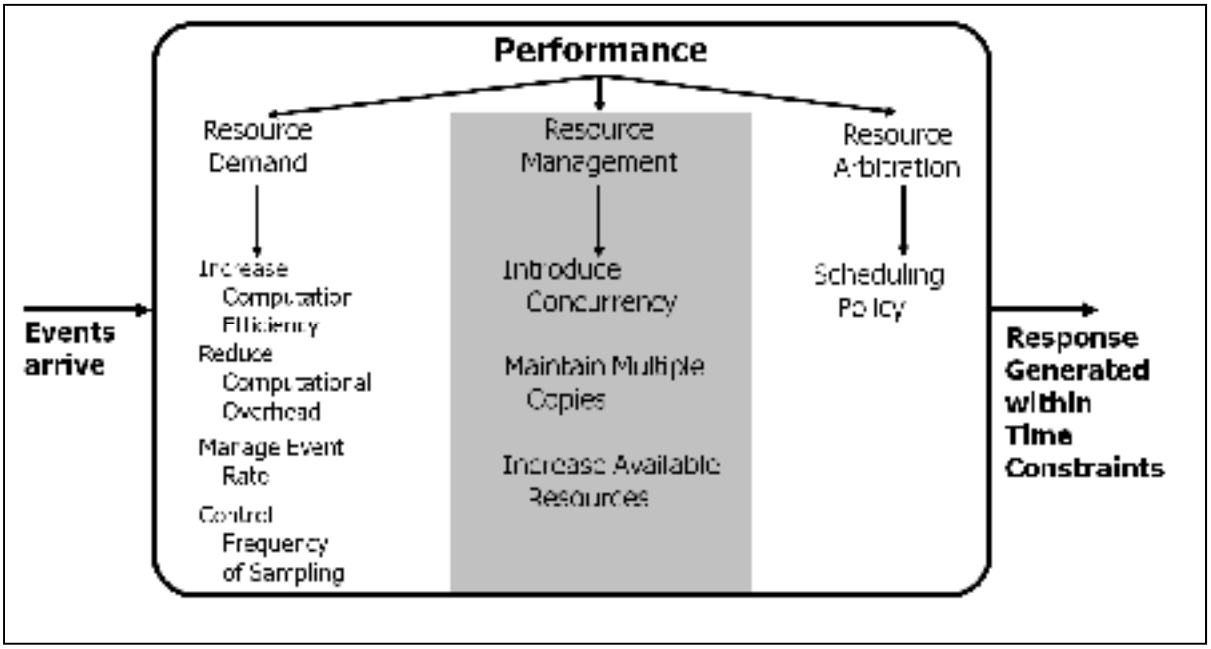

Figure 3.1 Résumé des tactiques de performance.
Tirée de (Bass, Clements et Kazman, 2003, p. 116)

Après analyse de ces tactiques, le développement d'une méthode automatisée et son implémentation dans le cadre de raisonnement sont effectués. Il faut aussi valider que l'implémentation de chacune des tactiques soit réalisée correctement dans ArchE. Ce chapitre se concentre donc sur la quatrième et une partie de la cinquième étape de la méthodologie :

l'ajout des tactiques et la validation individuelle des tactiques. Le prochain et dernier chapitre a comme objectif de compléter la validation et de résumer les résultats obtenus.

Figure 3.2 Méthodologie : Ajout des tactiques et une partie de leur validation.

La partie de la validation effectuée dans ce chapitre utilisera des tests individuels. Ces tests se concentrent sur une seule tactique à la fois et ont pour but de valider que la tactique soit suggérée automatiquement, qu'elle soit suggérée uniquement lorsque c'est nécessaire et que l'application automatique par l'utilisateur se fasse correctement.

Notons que, dans certains des exemples présentés, il pourrait y avoir plus d'une tactique proposée par ArchE. Dans le cadre des tests individuels, ces suggestions supplémentaires seront ignorées pour uniquement examiner les suggestions de la tactique testée. Cette précision est importante, car certaines tactiques sont proposées systématiquement lorsque l'architecture ne respecte pas les contraintes des scénarios. Il est donc impossible, sans modifier le code source du cadre de raisonnement, de développer tous les tests pour qu'une seule tactique soit proposée. Cette combinaison de proposition et d'application des tactiques est examinée dans le prochain chapitre.

Il est important de rappeler que, lorsque tous les scénarios sont satisfaits, la recherche de tactiques n'est pas effectuée. Le cas où la tactique n'est pas suggérée lorsque les scénarios sont satisfaits n'est pas examiné puisqu'il sera toujours valide pour toutes les tactiques.

Dans le contexte d'ArchE, les tactiques sont toutes indépendantes. Lorsque l'utilisateur applique une des tactiques proposées, ArchE redémarre l'évaluation avec la nouvelle architecture. Il est donc possible que l'application d'une tactique modifie l'architecture d'une manière où certaines tactiques précédemment proposées ne le soient plus. Chaque tactique

est donc évaluée par elle-même et on ne considère pas les interactions possibles, bonnes ou mauvaises, qu'elles pourraient avoir entre elles.

De plus, il est important de noter que les scénarios non satisfaits sont analysés indépendamment des autres. Il se peut donc qu'une tactique proposée pour un scénario puisse nuire à un autre scénario. Ces interactions possibles entre les scénarios ne sont pas prises en compte dans les tactiques, ce sera à l'utilisateur de déterminer si le compromis proposé par la tactique est intéressant ou non.

Chaque tactique doit inclure une logique d'application automatique et une logique utilisant des informations supplémentaires pouvant être fournies par l'utilisateur. Cet aspect est très important, car pour effectuer sa recherche, ArchE doit pouvoir appliquer automatiquement les tactiques proposées par le cadre de raisonnement afin de déterminer l'effet potentiel de la tactique. Cette application automatique utilisera, si nécessaire, des valeurs par défaut afin d'obtenir une estimation du résultat. La partie utilisant les informations fournies par l'utilisateur sert à l'application finale de la tactique. Cette application doit tenir compte des paramètres de l'utilisateur qui est le seul à connaitre les possibilités et les limites de son architecture.

Bien qu'il ne s'agisse pas d'une liste exhaustive des tactiques de performance, il a été décidé que les efforts soient concentrés sur les tactiques proposées par ces auteurs. Ce projet se concentre sur la faisabilité de la recherche et l'application automatique des tactiques de performance et non sur l'implémentation exhaustive de toutes les tactiques existantes, ni sur la définition de nouvelles tactiques. Trois catégories de tactiques sont décrites : la demande en ressources (« resource demand »), la gestion des ressources (« resource management ») et l'arbitrage des ressources (« resource arbitration »).

La demande en ressources est caractérisée par différents flux d'événements. Ces flux ont comme caractéristiques le temps entre chaque événement ainsi que la quantité de ressources consommé par un événement. Les tactiques dans cette catégorie modifient ces

caractéristiques, soit en réduisant le nombre d'événements traités, soit en réduisant la quantité de ressources nécessaire pour traiter un événement. S'il est impossible de modifier la demande en ressources, le deuxième groupe de tactiques peut modifier la gestion de celles-ci. Les modifications proposées servent à traiter plus efficacement et rapidement les événements entrant dans le système. Le troisième groupe se concentre sur l'arbitrage des ressources quand celles-ci sont demandées par plusieurs responsabilités en même temps. Des tactiques peuvent aider à choisir la bonne stratégie pour gérer plus efficacement les événements traités.

Voici un tableau énumérant les tactiques de la Figure 3.1 avec leur contrepartie dans ArchE :

Tableau 3.1 Concordance entre les tactiques de la Figure 3.1 et celles dans ArchE

Tactique de la figure	**Tactique dans ArchE**
Demande en ressources (« Resource Demand »)	
Increase Computation Efficiency	Réduire le temps d'exécution d'une responsabilité
Reduce Computational Overhead	---
Manage Event Rate	Augmenter la période d'un scénario
Control Frequency of Sampling	Augmenter la période d'un scénario
Gestion des ressources (« Resource Management »)	
Introduce Concurrency	---
Maintain Multiple Copies	---
Increase Available Resources	Augmenter les ressources disponibles
Arbitrage des ressources (« Resource Arbitration »)	
Scheduling Policy	Augmenter la priorité d'une responsabilité

Compte tenu de certaines limitations, certaines tactiques n'ont pas de correspondance dans ArchE; ces tactiques sont examinées en détail dans une section ultérieure qui décrit spécifiquement pourquoi elles n'ont pas pu être intégrées dans le cadre de raisonnement. Les sections suivantes se concentreront chacune sur l'analyse et l'intégration d'une tactique.

3.1 Augmenter les ressources disponibles

3.1.1 Description

Lorsqu'un système ne répond pas à ses exigences de performance, une tactique conceptuellement simple est d'augmenter les ressources du système pour qu'il soit plus performant : « des processeurs plus rapides, des processeurs additionnels, de la mémoire additionnelle et un réseau plus rapide ont tous un potentiel de réduire la latence maximale » (Bass, Clements et Kazman, 2003, p. 114).

L'augmentation des ressources (ex. : un processeur ou de la mémoire plus rapide) va diminuer le temps d'exécution des responsabilités, car le nouveau matériel va permettre d'effectuer plus de travail dans le même intervalle de temps. Cette diminution du temps d'exécution des responsabilités aura donc pour effet de réduire la latence finale des scénarios.

Bien que cette tactique puisse être théoriquement appliquée dans toutes les conditions où le temps d'exécution des responsabilités est trop long, elle ne l'est cependant peut-être pas en réalité. En effet, plusieurs aspects peuvent rendre l'augmentation des ressources impossible ou moins intéressante. Parmi ceux-ci, le coût peut limiter le type de ressources à notre disposition et l'aspect technique où des ressources plus performantes n'existent peut-être pas ou ne peuvent pas être facilement intégrées dans l'architecture existante. Puisque l'application de la tactique se veut générale, ce sera à l'utilisateur de déterminer quel genre d'accélération du système est possible dans son cas spécifique.

3.1.2 Analyse

Puisque nous n'avons aucun détail spécifique sur la nature de l'architecture matérielle, nous allons présumer qu'une augmentation des capacités de calcul va influer également et sur l'ensemble des responsabilités.

Bien que cette approche soit un peu naïve, il n'est pas possible de connaître le type de ressources pouvant être augmentées, ni de savoir à quel degré ce changement affectera chacune des responsabilités. Il est presque certain qu'une augmentation de 50 % de la vitesse du processeur n'apportera pas une amélioration de la performance de 50 %. Pour qu'il en soit ainsi, il faudrait que le système soit basé uniquement sur une seule ressource, ce qui en pratique est rarement le cas; le processeur, la mémoire vive, le disque dur, le réseau, etc. sont toutes des ressources influant indépendamment sur la performance d'un système.

L'utilisateur est responsable de vérifier et d'ajuster le temps d'exécution des responsabilités ne bénéficiant pas totalement de l'augmentation des ressources qu'il a décidé d'effectuer.

Afin de déterminer le nouveau temps d'exécution, le calcul suivant est effectué pour déterminer le nouveau temps d'exécution de chacune des responsabilités :

$$\boxed{C' = \frac{C}{1 + A}} \tag{3.1}$$

Où :

- C = ancien temps d'exécution (secondes)
- C' = nouveau temps d'exécution (secondes)
- A = pourcentage (ramené à 1) d'augmentation des ressources

Cette formule part du principe qu'une responsabilité prend un certain nombre d'opérations pour se compléter. Augmenter les ressources ne changera pas ce nombre puisque la responsabilité n'est pas modifiée. Si une responsabilité prend C secondes pour s'exécuter avec des ressources qui permettent de faire R opérations par secondes, la responsabilité prendra donc $C * R$ opérations pour se compléter. Puisque ce nombre ne change pas et qu'il est possible de faire une plus grande quantité d'opérations par secondes (dû à l'augmentation des ressources), le nouveau temps d'exécution sera plus petit. En résumé, voici les étapes suivies pour obtenir l'équation ci-dessus :

$$C * R = C' * R' = C' * R * (1 + A) \tag{3.2}$$

$$C' = \frac{C * R}{R * (1 + A)} = \frac{C}{1 + A} \tag{3.3}$$

Où :

- R = ressources disponibles (opérations/secondes)
- R' = nouvelles ressources disponibles, calculées avec $R' = R * (1 + A)$

3.1.3 Application de la tactique

Puisque cette tactique s'applique à toutes les responsabilités en même temps et qu'aucun état des responsabilités n'est requis, aucune règle de recherche élaborée n'est nécessaire : si au moins un scénario n'est pas satisfait, cette tactique est proposée à l'utilisateur.

3.1.3.1 Interactions

Dans cette section sur les interactions, ArchE réfère au cœur du programme contenant la logique d'exécution et l'interface graphique. Celui-ci utilise les cadres de raisonnement externes (RF pour « Reasoning Framework ») développés par d'autres chercheurs. Ces cadres de raisonnement servent à effectuer différentes analyses sur une architecture. Dans ce cas, il s'agit du cadre de raisonnement de performance. L'utilisateur réfère à une personne qui utilise ArchE pour l'assister dans la création d'un système.

Tableau 3.2 Interactions entre les acteurs pour la tactique « Augmenter les ressources disponibles »

#	Acteur	Action
1.1	ArchE	Analyse l'architecture actuelle : au moins un scénario de performance n'est pas satisfait
1.2	ArchE	Demande au cadre de raisonnement de suggérer des tactiques.
2	RF	Retourne automatiquement que la tactique peut s'appliquer.
3	ArchE	Demande d'appliquer automatiquement la tactique sur une copie de l'architecture.
4	RF	Modifie le temps d'exécution de toutes les responsabilités en utilisant un pourcentage d'augmentation par défaut (ex : 30 %).
5.1	ArchE	Analyse la copie de l'architecture modifiée.
5.2	ArchE	Affiche les tactiques et les résultats à l'utilisateur.
6	Utilisateur	Examine la question posée, inscrit le pourcentage d'augmentation voulu et confirme son choix.
7	ArchE	Transmet les informations entrées par l'utilisateur à la tactique.
8	RF	Modifie le temps d'exécution de toutes les responsabilités en utilisant le pourcentage fourni par l'utilisateur.

3.1.4 Exemples et validation de la tactique

Le résultat de l'application de la tactique d'augmentation des ressources est une réduction du temps d'exécution de toutes les responsabilités du système. Cette réduction est calculée à partir du pourcentage d'augmentation des ressources fourni par l'utilisateur. Cette tactique est toujours proposée à l'utilisateur lorsqu'au moins un scénario n'est pas satisfait.

Les tests pour cette tactique sont séparés en deux sections. Premièrement, le bon fonctionnement de l'étape de suggestion de la tactique sera validé, pour ensuite vérifier si l'application de la tactique et les responsabilités sont modifiées correctement.

3.1.4.1 Suggestion de tactique

Pour que la tactique soit suggérée, il doit y avoir au moins un scénario qui ne soit pas satisfait. Ce premier exemple illustre ce cas :

Tableau 3.3 Exemple avec un scénario non satisfait

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	60	30	R1	10	10	40	Non
			R2	10	20		
S2	60	30	R3	10	30	20	Oui
			R4	10	40		

Dans cet exemple, le scénario *S1* n'est pas satisfait; puisque la pire latence (40) dépasse l'échéance (30), la tactique d'augmenter les ressources disponibles est proposée. Cette suggestion reste identique même si plus d'un scénario n'est pas satisfait. La tactique est proposée une seule fois plutôt qu'une fois par scénario non satisfait, car elle est identique dans tous les cas (même tactique pour toutes les responsabilités avec la même ressource). Cette situation est illustrée avec ce deuxième exemple :

Tableau 3.4 Exemple avec plus d'un scénario non satisfait

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	60	30	R1	10	10	55	Non
			R2	10	20		
S2	60	30	R3	10	30	35	Non
			R4	10	40		
S3	60	30	R5	15	50	15	Oui

3.1.4.2 Application de la tactique

En choisissant la suggestion de la tactique « Augmenter les ressources disponibles » pour le premier exemple (Tableau 3.3) et en spécifiant 35 % d'augmentation des ressources, on obtient ce nouveau système :

Tableau 3.5 Résultat de l'application de la tactique sur l'exemple du Tableau 3.3

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	60	30	R1	7.407	10	29.63	Oui
			R2	7.407	20		
S2	60	30	R3	7.407	30	14.81	Oui
			R4	7.407	40		

La tactique a bien réduit le temps d'exécution de toutes les responsabilités, incluant celles qui étaient dans les autres scénarios. De plus, l'application de la tactique permet d'obtenir un système où tous les scénarios sont satisfaits.

Les résultats obtenus correspondent bien aux résultats attendus selon la formule (3.1) :

$$C' = \frac{C}{1+A} = \frac{10}{1+0.35} = 7.407 \tag{3.4}$$

3.2 Réduire le temps d'exécution d'une responsabilité

3.2.1 Description

Une partie du traitement de l'exécution d'un scénario peut être d'appliquer un ou plusieurs algorithmes. Améliorer ces algorithmes dans les parties critiques du programme va améliorer le temps de réponse (Bass, Clements et Kazman, 2003, p. 113). La tactique « Increase computational efficiency » (améliorer l'efficacité des calculs) propose de réduire le temps d'exécution d'une responsabilité critique de l'architecture. Cette responsabilité correspond à celle qui a le plus grand potentiel pour réduire le pire temps de latence d'un scénario. Il faut donc choisir judicieusement sur quelle responsabilité appliquer la tactique. Pour un petit système, il est assez simple d'y aller par essai et erreur, mais cette technique est plus difficile à appliquer intuitivement dans un système de plus grande envergure, d'où l'intérêt d'un outil qui nous indique les endroits probables où un impact non négligeable est envisageable

Bien que la traduction littérale du nom de la tactique soit « améliorer l'efficacité des calculs », le nom choisi est plus simple, plus général et plus clair quant au but de la tactique. En effet, la réduction du temps d'exécution d'une responsabilité peut être réalisée de plusieurs manières, par exemple une ressource peut être échangée pour une autre, des données peuvent être gardées en mémoire plutôt que d'être recalculées, des calculs redondants peuvent être éliminés, des boucles d'exécution peuvent être mieux gérées, un algorithme peut être optimisé ou être échangé pour un autre plus performant, etc.

3.2.2 Analyse

Le premier point à noter en ce qui concerne l'analyse de cette tactique est l'indépendance du domaine d'ArchE. Ceci a pour conséquence qu'il est impossible pour le cadre de raisonnement d'identifier les responsabilités qui peuvent ou non être modifiées et qu'il est impossible de déterminer l'ampleur exacte des réductions possibles sur les différentes responsabilités. Pour cette raison, toutes les responsabilités seront considérées dans l'analyse. De plus, l'analyse utilisera un pourcentage par défaut de réduction du temps d'exécution qui s'appliquera également à toutes les responsabilités. Le pourcentage par défaut ne peut être modifié, car ArchE ne possède pas une interface où des paramètres pourraient être ajustés. Cependant, cette valeur par défaut est utilisée uniquement par l'analyse afin de déterminer la responsabilité ayant le plus d'impact. Même si le pourcentage était modifié, le résultat de l'analyse serait le même, car toutes les responsabilités sont affectées également par celui-ci.

Un pourcentage est utilisé, car il s'applique de manière uniforme sur toutes les responsabilités, contrairement à une valeur absolue de réduction, qui aurait moins d'impact sur une responsabilité avec un temps d'exécution long que sur une ayant un temps d'exécution court. Cette réduction par défaut est utilisée uniquement pour l'analyse afin de choisir une responsabilité, le résultat sera affiché à l'utilisateur, mais il sera de son devoir de déterminer le nouveau temps d'exécution réduit de cette responsabilité.

Un scénario est composé de responsabilités. Le temps d'exécution de ces responsabilités a un effet direct sur le temps d'exécution maximal du scénario. De plus, les autres responsabilités ne se trouvant pas dans le scénario à l'étude peuvent indirectement avoir un effet sur la latence de celui-ci. En effet, si une de ces responsabilités à une priorité plus élevée qu'une autre se trouvant dans le scénario à l'étude, elle pourrait s'exécuter en priorité et voir son temps d'exécution ajouté à l'échéance du scénario à l'étude. Ceci implique qu'il ne faut pas uniquement considérer les responsabilités du scénario à l'étude, mais toutes les responsabilités du système.

Peu importe les détails de l'analyse de performance, en utilisant directement la méthode d'analyse fournie par le cadre de raisonnement, il est possible de connaître exactement l'amélioration obtenue avec la réduction du temps d'exécution d'une responsabilité. Aucun heuristique de décision n'a besoin d'être élaboré, on base les décisions sur des résultats fiables, en l'occurrence une latence calculée dans le pire des cas avec un cadre de raisonnement éprouvé.

Pour terminer l'analyse, il est important de noter que le cadre de raisonnement de performance connait les modifications possibles, mais pas si elles sont réalisables. Ce sera le rôle de l'utilisateur de déterminer si la réduction du temps d'exécution proposée pour la responsabilité est une option viable pour le système, ainsi que de déterminer quelle réduction est réellement possible et raisonnable, compte tenu du contexte.

3.2.3 Choix des responsabilités

Pour déterminer la meilleure option pour l'utilisateur, il a été décidé d'examiner individuellement toutes les options possibles et de retourner celle qui donne le meilleur temps de latence pour le scénario. Pour y arriver, la tactique réduit le temps d'exécution d'une seule responsabilité, demande au cadre de raisonnement d'analyser cette architecture modifiée et récupère le résultat. Ces étapes sont répétées pour chacune des responsabilités, ce qui permet à la tactique d'obtenir les résultats de toutes les modifications possibles.

Avec ces résultats, plusieurs options sont possibles. Premièrement, si une ou plusieurs modifications permettent d'avoir un système qui répond aux exigences du scénario, elles seront toutes retournées à l'utilisateur, car elles vont toutes permettre à l'utilisateur d'atteindre son objectif. Si cette situation ne se produit pas, cela veut dire qu'aucune modification d'une responsabilité ne va permettre d'obtenir un système satisfaisant. Dans ce cas, la tactique retourne uniquement la responsabilité qui donnera la meilleure réduction du délai maximal pour le scénario. Même si cette tactique ne répond pas totalement aux besoins de l'utilisateur, elle lui permettra de voir quelle responsabilité aura le plus d'impact sur le scénario et de pouvoir s'approcher plus près du but.

3.2.4 Application de la tactique

Lors de l'application automatique par ArchE de la tactique proposée sur la responsabilité, un pourcentage par défaut est utilisé afin d'obtenir un résultat qui sera utilisé par ArchE. L'application est simple : réduire le temps d'exécution de la responsabilité proposée du pourcentage par défaut.

En ce qui concerne l'application de la tactique avec l'aide de l'utilisateur, elle se fait en deux étapes. Premièrement, la tactique est proposée à l'utilisateur en affichant la responsabilité et le temps d'exécution déterminé lors de l'analyse de la tactique. Ensuite, l'utilisateur entre le nouveau temps d'exécution réduit de la responsabilité. Cette nouvelle valeur remplace le temps d'exécution de la responsabilité. La valeur proposée peut servir de base à l'utilisateur pour déterminer l'effort de réduction nécessaire pour obtenir un système qui répond à ses exigences.

3.2.4.1 Interactions

Tableau 3.6 Interactions entre les acteurs pour la tactique « Réduire le temps d'exécution d'une responsabilité »

#	Acteur	Action
1.1	ArchE	Analyse l'architecture actuelle : au moins un scénario de performance n'est pas satisfait.
1.2	ArchE	Demande au cadre de raisonnement de suggérer des tactiques.
2.1	RF	Pour chacune des responsabilités :
2.1.1	RF	Réduit le temps d'exécution de la responsabilité du pourcentage par défaut ;
2.1.2	RF	Analyse l'architecture modifiée et conserve le résultat.
2.1.3	RF	Retourne le temps d'exécution de la responsabilité à sa valeur originale.
2.2	RF	Si un ou plusieurs résultats donnent un système valide, retourne une suggestion de tactique pour chacun de ces résultats.
2.3	RF	Sinon, retourne une tactique pour l'amélioration la plus grande trouvée.
3	ArchE	Demande d'appliquer automatiquement la tactique sur une copie de l'architecture.
4	RF	Réduit le temps d'exécution de la responsabilité proposée du pourcentage par défaut.
5.1	ArchE	Analyse la copie de l'architecture modifiée.
5.2	ArchE	Affiche les tactiques et les résultats à l'utilisateur.
6	Utilisateur	Examine la question posée, la faisabilité de la réduction et donne le nouveau temps d'exécution voulu pour la responsabilité proposée.
7	ArchE	Transmets les informations entrées par l'utilisateur à la tactique.
8	RF	Change la valeur du temps d'exécution de la responsabilité proposée par la valeur entrée par l'utilisateur.

3.2.5 Exemples et validation de la tactique

Cette tactique propose de réduire le temps d'exécution d'une responsabilité clé de l'architecture. Cette responsabilité est celle qui, en réduisant son temps d'exécution, produit la plus grande réduction du temps de latence maximal du scénario à l'étude. Cette responsabilité peut être contenue dans le scénario à l'étude ou dans un autre scénario. Il faut noter que lors de la recherche pour l'application de cette tactique, le cadre de raisonnement utilise un pourcentage de réduction du temps d'exécution qui a été fixé à 15 %.

Dans tous les cas où il y a au moins un scénario qui n'est pas satisfait, cette tactique sera toujours proposée au moins une fois. En effet, pour qu'un scénario de performance ne soit pas satisfait, il doit contenir au moins une responsabilité, donc il en existe toujours une sur laquelle la tactique pourrait théoriquement être appliquée.

3.2.5.1 Cas de base

Le cas le plus simple de l'application de cette tactique est un scénario qui contient une seule responsabilité qui a un temps d'exécution plus élevé que le délai maximal du scénario.

Tableau 3.7 Exemple d'un scénario avec une responsabilité

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	50	R1	55	10	55	Non

La tactique est proposée à l'utilisateur pour la responsabilité *R1*. La tactique a essayé avec un nouveau temps d'exécution de 46.75ms (réduction de 15 %). Le système répondrait aux exigences du scénario, alors la tactique est proposée correctement. Ensuite, en se basant sur la proposition de la tactique l'utilisateur examine la responsabilité et détermine que le nouveau temps d'exécution de cette responsabilité pourrait être réduit à 45ms. Il entre cette nouvelle valeur dans la proposition de tactique et l'application donnera le système suivant :

Tableau 3.8 Résultat de l'application de la tactique sur l'exemple du Tableau 3.7

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	50	R1	45	10	45	Oui

3.2.5.2 Responsabilité partagée

Dans certains cas, la responsabilité clé ne se présente pas de façon évidente. En effet, on pourrait être porté à penser aux responsabilités ayant le plus grand temps d'exécution, mais dans certains cas il pourrait en être autrement si une responsabilité avec un court temps d'exécution est utilisée souvent, ou se retrouve sur un « chemin critique » en termes de l'impact du temps d'exécution de cette responsabilité sur la pire latence d'un ou plusieurs scénarios.

Tableau 3.9 Exemple avec une responsabilité partagée

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	75	R1	25	20	80	Non
			R2	10	10		
S2	100	100	R3	25	30	80	Oui
			R2	10	10		
S3	100	100	R2	10	10	80	Oui

La tactique propose de réduire le temps d'exécution de *R2* à 8.5ms, même si une réduction de 15 % du temps d'exécution de *R1* et *R3* est plus grande (3.75 ms pour *R1* et *R3*, contre 1.50 ms pour *R2*). Cette proposition provient du fait que la responsabilité R2 est utilisée dans trois scénarios, donc cette plus petite réduction est multipliée par trois et donne une réduction de la latence de 4.5ms pour le scénario *S1*.

Tableau 3.10 Application de la tactique sur l'exemple du Tableau 3.9

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	75	R1	25.0	20	75.5	Non
			R2	8.5	10		
S2	100	100	R3	25.0	30	75.5	Oui
			R2	8.5	10		
S3	100	100	R2	8.5	10	75.5	Oui

3.2.5.3 Scénario avec une courte période

Un autre cas où une responsabilité avec un court temps d'exécution peut avoir un plus grand effet qu'une autre ayant un plus grand temps d'exécution est lorsqu'elle fait partie d'un scénario ayant une courte période (la valeur du stimulus).

Tableau 3.11 Exemple avec un scénario avec une courte période

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	10	10	R1	7	20	7	Oui
S2	100	50	R2	25	10	88	Non

Dans cet exemple, la tactique est proposée pour la responsabilité *R1*, car une réduction de 15 % de son temps d'exécution (5.95 ms) réduit la latence de *S1* à 66.65ms, alors qu'une réduction de 15 % du temps d'exécution de *R2* (21.25 ms) réduit uniquement la latence de *S1* à 77.25ms.

Ceci est dû au fait que l'exécution de la responsabilité R2 sera interrompue plusieurs fois par la responsabilité *R1*, car elle est contenue dans un scénario ayant une période très courte comparée à *S2*. Le temps d'exécution de la responsabilité *R1* est donc additionné plusieurs fois à la latence du scénario *S2*. La réduction du temps d'exécution sera donc multipliée par toutes ces exécutions.

En terminant, cet exemple permet aussi de montrer que les suggestions pour cette tactique ne se limitent pas uniquement aux responsabilités contenues dans le scénario à l'étude; toutes les responsabilités sont examinées.

3.2.5.4 Système résultant satisfait les exigences

Si une ou plusieurs suggestions d'application de la tactique de réduction du temps d'exécution d'une des responsabilités permettent d'obtenir un système qui répond aux exigences, celles-ci seront toutes proposées à l'utilisateur.

Tableau 3.12 Exemple avec plusieurs suggestions pour obtenir un système valide

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	50	R1	11	10	53	Non
			R2	20	10		
			R3	22	10		

Dans cet exemple, les suggestions proposent de réduire de 15 % *R2* (17 ms) ou *R3* (18.7 ms). Dans les deux cas, l'application de la tactique permet d'obtenir un système qui répond aux exigences du scénario *S1* avec une échéance de 50 ms en modifiant *R2* ou de 49.7 ms en modifiant *R3*.

3.2.5.5 Système résultant ne satisfait pas les exigences

Lorsqu'aucune suggestion d'application de la tactique de réduction du temps d'exécution d'une responsabilité ne permet d'obtenir un système qui répond aux exigences, la suggestion ayant le plus grand potentiel de réduction est retournée à l'utilisateur.

Tableau 3.13 Exemple avec une suggestion pour un système qui ne satisfait pas les exigences

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	50	R1	15	10	57	Non
			R2	20	10		
			R3	22	10		

Dans cet exemple, aucune suggestion (avec la réduction par défaut de 15 %) ne permet d'obtenir de système répondant aux exigences de *S1*. La cadre de raisonnement suggère donc de réduire le temps d'exécution de *R3*, car cette responsabilité réduit au maximum la latence de *S1* (c.-à-d. 53.7 ms, comparées à 54.75 ms avec *R1* et 54 ms avec *R2*).

Dans le cas où plusieurs suggestions permettent d'obtenir le même meilleur résultat pour la pire latence du scénario, elles seront toutes retournées à l'utilisateur.

Tableau 3.14 Exemple avec plus d'une suggestion pour un système qui ne satisfait pas les exigences

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	50	R1	15	10	55	Non
			R2	20	10		
			R3	20	10		

Dans cet exemple, puisque les responsabilités *R2* et *R3* permettent toutes les deux la même amélioration de la latence (52 ms), une suggestion est retournée pour chacune d'elles en proposant de réduire leur temps d'exécution à 17 ms.

Dans l'exemple précédent, les deux responsabilités utilisées ont le même temps d'exécution dans le même scénario, mais un résultat similaire est possible en combinant deux responsabilités avec des temps d'exécution différents dans deux scénarios différents, comme démontré dans ce dernier exemple :

Tableau 3.15 Exemple avec plus d'une suggestion avec des temps d'exécution différents

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	50	R1	15	10	55	Non
			R2	20	10		
S2	40	40	R3	10	20	10	Oui

Dans cet exemple, une suggestion est proposée pour *R2* (avec 17 ms) et une autre pour *R3* (avec 8.5 ms). Il y a deux suggestions, car la période de *S2* est plus courte que la pire latence de *S1* (40 ms < 55 ms) et *R3* a une priorité plus élevée que *R1* et *R2*. Ceci veut dire que le scénario *S2* s'exécutera deux fois pendant l'exécution de *S1*. Le temps d'exécution de *R3* sera compté deux fois (donc 20 ms) dans la latence de *S1*, tout comme *R2*.

3.3 Augmenter la période d'un scénario

3.3.1 Description

La période est un des aspects importants d'un scénario de performance. Elle dicte la fréquence à laquelle un scénario sera exécuté. Si la période d'un scénario est augmentée, le nombre d'événements traités par unité de temps est réduit, ce qui a pour conséquence que le code correspondant à son exécution va s'exécuter moins souvent et donc cela laissera plus de temps aux autres scénarios pour s'exécuter. En bref, augmenter la période revient à relaxer un aspect d'un des scénarios.

Pour y arriver, deux tactiques sont proposées (Bass, Clements et Kazman, 2003, p. 113) :

- Gérer le taux d'arrivée des événements (« Manage event rate ») : si l'on réduit la fréquence d'arrivée des événements dans le système, la demande peut être diminuée.
- Contrôler la fréquence d'échantillonnage (« Control frequency of sampling ») : si la fréquence d'arrivée est hors de notre contrôle, les événements pourraient être mis en file d'attente, puis être lus à une fréquence plus basse. Ceci réduirait la demande, mais il pourrait en résulter des pertes de données.

Ces tactiques ne sont que des exemples de possibilité pour le développeur. Cependant, puisqu'ArchE ne connaît pas le domaine de l'application étudiée, il lui est impossible de dire exactement quelle tactique utiliser. Dans le cadre de cette tactique, si ArchE détermine qu'une augmentation de la période pourrait être bénéfique, il proposera la tactique à l'utilisateur en lui indiquant où elle serait utile, mais ce sera à ce dernier de déterminer comment il l'intègrera dans son architecture.

3.3.2 Analyse

Augmenter la période d'un scénario (la valeur du stimulus) peut réduire le nombre d'événements traités et permettre de réduire la latence. Augmenter la période d'un scénario où celle-ci est plus courte que la latence maximale du scénario à l'étude permettra de laisser plus de temps pour l'exécution de ce dernier et donc de possiblement réduire sa latence.

La Figure 3.3 montre un chronogramme où le scénario *S1* a toujours priorité sur *S2*. *S1* a une période de 30 ms et un temps d'exécution total de 20 ms et *S2* a une période de 150 ms et un temps d'exécution total de 50 ms.

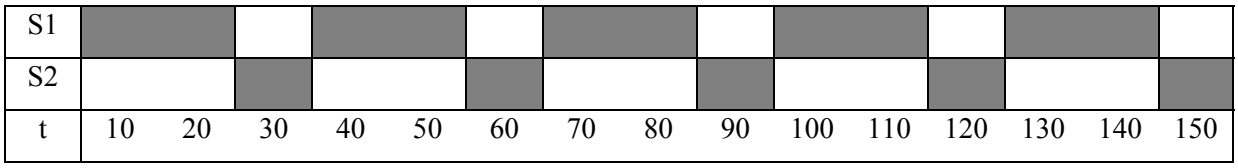

Figure 3.3 Chronogramme avec une période de 30 ms pour *S1*.

Dans cet exemple, la latence maximale de S2 serait de 150 ms, car ce scénario ne peut s'exécuter que durant 10 ms sur chaque tranche de 30 ms. Si la période de *S1* était augmentée à 40 ms, il y a maintenant 20 ms de temps libre pour l'exécution de *S2* et sa latence diminue à 110 ms, comme le montre ce nouveau chronogramme.

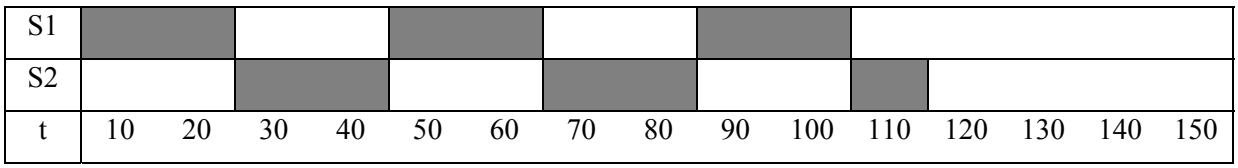

Figure 3.4 Chronogramme avec une période de 40 ms pour S2.

Si la période de *S1* est plus longue que la latence de *S2* (ex. : 150 ms), il y a qu'une seule exécution du scénario et aucun changement ne se produirait en augmentant plus la période. Notons que cette nouvelle période de *S1* montre la latence minimale de *S2* possible avec cette tactique.

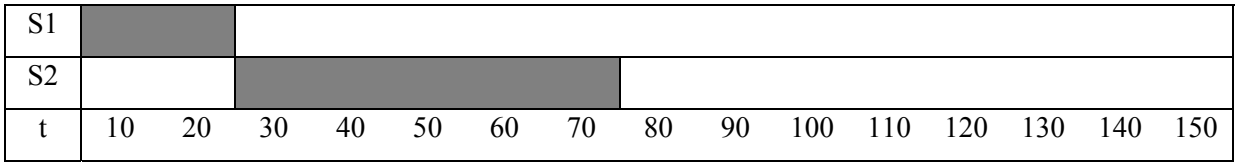

Figure 3.5 Chronogramme avec une période plus longue que la latence.

Dans cet exemple, S1 avait toujours priorité sur S2, cependant l'analyse RMA utilise une classification plus fine des scénarios qui sera décrite à la section suivante.

3.3.3 Définition des types de scénario

Pour effectuer l'analyse RMA, l'outil MAST classifie les scénarios en différents types selon la priorité des responsabilités qu'il contient. Voici un exemple présentant la définition des types pour une série de scénarios. Les types sont définis pour le scénario 3.

Tableau 3.16 Exemple de scénarios dans des groupes

Scénario	**Priorité responsabilité 1**	**Priorité responsabilité 2**	**Type**
1	50	40	H
2	30	10	HL
3	**20**	**20**	**n/d**
4	10	40	LH
5	10	10	L

Quatre types existent :

- Type *H* : La priorité de toutes les responsabilités qui composent le scénario comparé est plus élevée que la priorité de toutes les responsabilités qui composent le scénario à l'étude. Autrement dit, si le scénario comparé est sollicité (i.e. son stimulus se manifeste) pendant que le scénario à l'étude s'exécutait, le scénario à l'étude est bloqué et le scénario comparé s'exécute complètement, avant que le scénario à l'étude puisse continuer à s'exécuter.

- Type *HL* : La priorité de la première responsabilité du scénario comparé est plus élevée que la priorité de toutes les responsabilités qui composent le scénario à l'étude, mais au moins une responsabilité subséquente du scénario comparé a une priorité moins élevée qu'au moins une des responsabilités du scénario à l'étude. Autrement dit, si le scénario comparé est sollicité (i.e. son stimulus se manifeste) pendant que le scénario à l'étude s'exécutait, le scénario à l'étude est bloqué et le scénario comparé commence son exécution, mais éventuellement il sera bloqué à son tour par le scénario à l'étude, qui pourra alors continuer son exécution.
- Type *LH* : La priorité de la première responsabilité du scénario comparé est moins élevée qu'au moins une des responsabilités qui composent le scénario à l'étude, mais au moins une responsabilité subséquente du scénario comparé a une priorité plus élevée que toutes les responsabilités du scénario à l'étude. Autrement dit, si le scénario comparé est sollicité (i.e. son stimulus se manifeste) pendant que le scénario à l'étude s'exécutait, le scénario à l'étude n'est pas bloqué immédiatement, mais le sera éventuellement, par le scénario comparé.
- Type *L* : La priorité de toutes les responsabilités qui composent le scénario comparé est plus basse que la priorité de toutes les responsabilités qui composent le scénario à l'étude. Autrement dit, le scénario comparé ne peut en aucun cas bloquer le scénario à l'étude.

Ces types ont chacun un impact différent sur le temps de latence maximal du scénario à l'étude. Cet impact sera examiné à la prochaine section. Pour toute l'information sur la classification des scénarios, veuillez vous référer à la référence originale (Klein *et al.*, 1993).

3.3.4 Types de scénarios qui ont une influence

Il y a deux types de scénarios dont la période peut avoir une influence sur sa latence maximale : la période du scénario à l'étude et la période des scénarios dans le groupe *H*. Ces types ont tous les deux étés déduits des formules de la Technique 6 de RMA (Klein *et al.*, 1993). Ces dernières seront détaillées dans les deux prochaines sections.

3.3.4.1 Période du scénario à l'étude

La première période qui sera examinée est celle du scénario à l'étude. En effet, cette période est la première que l'on rencontre avec cette technique. Les formules utilisées dans ce cas servent à déterminer le nombre nécessaire d'exécutions du scénario pour obtenir le pire cas possible.

$$a_{n+1} = S + \left\lceil \frac{a_n}{T_i} \right\rceil C_i + \sum_{k \in H(1)} \left\lceil \frac{a_n}{T_k} \right\rceil C_k \qquad N = \frac{a_{n+1}}{T_i} \tag{3.5}$$

Cette étape de l'analyse RMA (Klein *et al.*, 1993, pp. 4-45) sert à déterminer pendant combien de temps l'exécution du scénario est bloquée avant même de pouvoir exécuter sa première responsabilité. La valeur de a_{n+1}, donnera un temps de latence maximal plus pessimiste qu'il ne l'est en réalité et la valeur de N donnera le nombre d'exécutions maximales du scénario qu'il faudra vérifier. En général, plus N est grand, plus le délai est grand, car la dernière exécution devra tenir compte du temps d'exécution de toutes les exécutions précédentes.

En étudiant la formule, nous pouvons voir que si T_i (la période du scénario étudié) augmente, la valeur de N et de a_{n+1} peut diminuer. Le but est de faire diminuer la valeur de N assez pour qu'elle prenne une valeur inférieure à celle actuelle et donc avoir une exécution du scénario en moins. En utilisant l'exemple « Example_Varying_Priorities » livré avec MAST v1.3.7.8 [7], voici un exemple d'effets possibles de l'augmentation de la période d'un scénario (*S5* dans ce cas).

[7] http://mast.unican.es/

Tableau 3.17 Résultats de l'exemple avant le changement de la période du scénario *S5*

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	40	40	R1_1	1	10	28	Oui
			R1_2	5	7		
S2	100	100	R2_1	10	4	98	Oui
			R2_2	5	8		
			R2_3	5	4		
S3	50	50	R3_1	8	5	47	Oui
			R3_2	12	8		
S4	200	200	R4_1	10	9	195	Oui
			R4_2	20	2		
			R4_3	3	3		
S5	**300**	**300**	R5_1	2	3	**313**	**Non**
			R5_2	12	1		
			R5_3	10	6		

Tableau 3.18 Résultats de l'exemple après le changement de la période du scénario *S5*

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	40	40	Voir Tableau 3.17			28	Oui
S2	100	100				98	Oui
S3	50	50				47	Oui
S4	200	200				195	Oui
S5	**400**	**300**				**223**	**Oui**

Dans cet exemple, la période de *S5* a été modifiée, ce qui a pour effet de réduire sa pire latence. Ceci montre que la période d'un scénario peut influer sur son propre temps de latence maximal.

3.3.4.2 Période des scénarios dans le groupe *H*

En ce qui concerne la période des scénarios dans le groupe *H*, elle se retrouve dans une autre formule dans l'analyse RMA (Klein *et al.*, 1993, pp. 4-46).

$$a_{n+1} = S + \sum_{k \in H} \left\lceil \frac{a_n}{T_k} \right\rceil C_k \tag{3.6}$$

Cette formule est utilisée à plusieurs reprises afin de déterminer le temps de complétion des responsabilités du scénario. Nous pouvons voir que si une ou plusieurs périodes (T_k) des scénarios qui se trouvent dans le groupe H augmentent, la sommation pourrait diminuer et donc le temps d'exécution final (a_{n+1}) serait plus court.

Il faut noter que les périodes de tous les scénarios se trouvant dans le groupe H ont un impact sur le temps d'exécution final, donc toute augmentation de la période d'un scénario dans ce groupe pourrait avoir un effet bénéfique. De plus, il est intéressant de noter que l'augmentation de la période d'un scénario qui a un plus grand temps d'exécution (C_k) aura un plus gros impact sur la sommation. Cependant, puisqu'il n'est pas toujours possible d'augmenter la période du scénario ayant le temps d'exécution le plus long, la proposition de tactique n'y sera pas limitée.

En utilisant le même exemple que précédemment, mais en modifiant une autre période, nous illustrerons les concepts énoncés dans cette sous-section. Voici les résultats de l'analyse avec MAST :

Tableau 3.19 Résultats de l'exemple avant le changement de la période du scénario *S1*

Scénario	**Période (ms)**	**Échéance**	**Responsabilité**	**Temps exécution (ms)**	**Priorité**	**Pire latence (ms)**	**Échéance Respectée**
S1	**35**	40	Voir Tableau 3.17			28	Oui
S2	100	100				98	Oui
S3	50	50				47	Oui
S4	200	200				**273**	**Non**
S5	400	400				**414**	**Non**

Tableau 3.20 Résultats de l'exemple après le changement de la période du scénario *S1*

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	**40**	40	Voir Tableau 3.17			28	Oui
S2	100	100				98	Oui
S3	50	50				47	Oui
S4	200	200				**195**	**Oui**
S5	400	300				**223**	**Oui**

Puisque *S1* possède des priorités plus élevées, celui-ci se retrouve dans le groupe *H* des scénarios *S4* et *S5*. C'est pour cette raison que leurs temps de réponse sont affectés par la modification de la période de *S1*, supportant ainsi l'hypothèse précédemment énoncée. Il est important de noter qu'une augmentation de 5 ms de la période de S1 a permis de réduire les pires latences de *S4* et *S5* de 78 et 181 ms, respectivement. De telles réductions sont difficilement atteignables si l'on essaie d'altérer intuitivement les paramètres des scénarios et des responsabilités.

En terminant, il est important de noter que, bien que les exemples démontrés affectent uniquement une partie des scénarios, il n'en est pas toujours ainsi. Il se pourrait qu'une modification de la période réduise le temps d'exécution du scénario à l'étude ainsi que celle des autres. De plus, certaines modifications de la période n'auront aucun effet sur les temps de réponse. L'utilisation d'ArchE permettra d'essayer rapidement ces différentes options afin de déterminer celle qui aura un meilleur effet sur l'architecture développée.

3.3.5 Estimation de la nouvelle période

Pour tous les scénarios de performance pouvant avoir un impact sur le scénario à l'étude, une nouvelle valeur pour la période devra être proposée.

Dans toutes les formules où la période se retrouve, celle-ci divise toujours le temps de réponse (a_n) et ensuite la partie entière par excès (plafond, ou arrondissement à l'entier immédiatement supérieur) est prise. L'objectif de la tactique est d'obtenir une valeur

résultante plus petite. L'augmentation de la période doit donc être assez grande pour que la partie entière soit plus petite. Puisque les calculs sont effectués dans l'outil externe MAST, il est impossible de retracer toutes les itérations faites pour arriver à la valeur finale de a_n. Il faudra donc se contenter de la valeur finale pour effectuer l'estimation suivante :

$$\boxed{T' = \frac{a_n}{\left(\left\lceil \frac{a_n}{T} \right\rceil - 1\right)}} \tag{3.7}$$

Pour déterminer cette estimation, il faut diminuer la valeur du terme $\left\lceil \frac{a_n}{T} \right\rceil$ dans les équations en utilisant uniquement une nouvelle période *T'*. Puisque la partie entière est prise, il faut que la valeur descende d'au moins un pour qu'un changement se produise avec la nouvelle période. Ce concept est illustré par la formule suivante et l'estimation est trouvée en y isolant le terme *T'* :

$$\frac{a_n}{T'} = \left\lceil \frac{a_n}{T} \right\rceil - 1 \tag{3.8}$$

Pour valider l'estimation, on peut l'insérer dans la formule (3.6) de l'analyse. On peut ainsi voir que la formule revient à son état original sauf que la valeur de C_k (pour le scénario modifié) sera additionnée une fois de moins à la sommation.

$$a_{n+1} = S + \sum_{k \in H} \left\lceil \frac{a_n}{T'} \right\rceil C_k = S + \sum_{k \in H} \left\lceil \frac{a_n}{\frac{a_n}{\left(\left\lceil \frac{a_n}{T_k} \right\rceil - 1\right)}} \right\rceil C_k = S + \sum_{k \in H} \left\lceil \left\lceil \frac{a_n}{T_k} \right\rceil - 1 \right\rceil C_k \tag{3.9}$$

En utilisant le deuxième exemple illustré, l'estimation aurait donné une période de 37.6 ms pour *S1*, donnant ainsi un temps de réponse de 247 ms pour *S4* et de 406 ms pour *S5*. Cette estimation améliore la situation, mais le temps de réponse est encore trop long. Dans cette situation, l'utilisateur pourrait réappliquer la tactique, ce qui donnerait une période de 41.4 ms pour *S1* et un système qui répond aux exigences. Bien que la valeur finale soit plus

élevée que celle utilisée dans l'exemple (c.-à-d. 40), nous sommes arrivées à un résultat positif grâce à cette estimation.

Dans certains cas, il sera impossible de diminuer cette valeur, par exemple lorsque a_n est plus petit ou égal à T (il y aurait une division par zéro). Dans cette situation, le scénario en question devra être écarté des possibilités.

3.3.6 Déterminer les scénarios d'intérêts

Afin de déterminer les scénarios à considérer lors de la recherche de la tactique, les étapes suivantes doivent être suivies :

1. ajouter les scénarios du groupe H dans le groupe d'intérêt :
   a) Déterminer la plus basse priorité dans le scénario à l'étude;
   b) Déterminer la plus basse priorité dans les autres scénarios;
   c) Pour tous les scénarios où leur plus basse priorité est égale ou plus élevée que celle du scénario à l'étude, l'ajouter dans le groupe d'intérêt;
2. ajouter le scénario à l'étude dans la liste des scénarios d'intérêts;
3. retirer les scénarios où il est impossible d'estimer une nouvelle période (c.-à-d., si $T \geq a_n$).

Pour chacun de ces scénarios, une tactique séparée est proposée à l'utilisateur. Si aucun scénario n'est intéressant, la tactique ne retourne rien et ArchE n'aura pas d'alternatives à examiner. Chacune des tactiques proposera à l'utilisateur une nouvelle période ayant de bonnes chances d'améliorer le temps de réponse.

3.3.7 Application de la tactique

Dans l'esprit d'ArchE et des tactiques, une tactique devrait modifier uniquement les spécifications de l'architecture (c.-à-d., les responsabilités). Les scénarios, faisant partie des spécifications des exigences de qualité, sont donc hors limite pour être modifiés par les tactiques.

Cependant, la tactique reste valide, car l'utilisateur devra modifier son architecture afin d'augmenter la période. L'application automatique n'est pas possible, car ArchE n'a pas de connaissances spécifiques sur le domaine et sur les détails de l'implémentation de l'architecture.

3.3.7.1 Interactions

Tableau 3.21 Interactions entre les acteurs pour la tactique « Augmenter la période d'un scénario »

#	Acteur	Action
1.1	ArchE	Analyse l'architecture actuelle : au moins un scénario de performance n'est pas satisfait.
1.2	ArchE	Demande au cadre de raisonnement de suggérer des tactiques.
2.1	RF	Détermine les scénarios d'intérêts.
2.2	RF	Estime la période pour chacun des scénarios.
2.3	RF	Retourne une tactique pour chacun des scénarios.
3	ArchE	Demande d'appliquer automatiquement la tactique sur une copie de l'architecture.
4	Tactique	Aucune modification effectuée.
5.1	ArchE	Analyse la copie de l'architecture modifiée.
5.2	ArchE	Affiche les tactiques et les résultats à l'utilisateur.
6.1	Utilisateur	Examine les propositions de périodes à modifier.
6.2	Utilisateur	Modifie manuellement la période d'un scénario avec la valeur proposée par la tactique.

3.3.8 Exemples et validation de la tactique

Cette tactique augmente la période d'un scénario (la valeur du stimulus) afin de réduire le nombre d'événements traités et permettre de réduire la latence. Il y a deux types de scénarios dont leur période peut avoir une influence sur la latence maximale du scénario à l'étude : la période du scénario à l'étude et la période des scénarios dans le groupe *H*. Pour que la tactique soit proposée, la période du scénario doit être plus courte que la latence maximale du scénario à l'étude.

Il faut noter qu'il n'est pas possible pour le cadre de raisonnement de modifier les scénarios. Pour cette raison, les suggestions pour cette tactique devront être appliquées manuellement par l'utilisateur pour voir les effets.

3.3.8.1 Exemple « Example_Varying_Priorities »

Pour les tests de cette tactique, l'exemple « Example_Varying_Priorities » livré avec MAST v1.3.7.8 [8] sera utilisé. Dans la définition originale de l'exemple, le système répond à toutes les exigences des scénarios. Il sera modifié afin de démontrer les différents aspects de cette tactique. Les modifications touchent uniquement les scénarios, les responsabilités (incluant leur temps d'exécution et leur priorité) resteront inchangées.

Tableau 3.22 Définition originale de l'exemple « Example_Varying_Priorities »

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	40	40	R1_1	1	10	28	Oui
			R1_2	5	7		
S2	100	100	R2_1	10	4	98	Oui
			R2_2	5	8		
			R2_3	5	4		
S3	50	50	R3_1	8	5	47	Oui
			R3_2	12	8		
S4	200	200	R4_1	10	9	195	Oui
			R4_2	20	2		
			R4_3	3	3		
S5	400	400	R5_1	2	3	223	Oui
			R5_2	12	1		
			R5_3	10	6		

Cet exemple a été choisi, car le taux d'utilisation du processeur est très élevé avec une utilisation de 99.218 %. Ceci implique qu'il y a très peu de temps processeur libre et que l'exemple est très sensible à toute variation de ses paramètres. De plus, l'exemple contient

[8] http://mast.unican.es/

plusieurs scénarios qui ont beaucoup d'influence les uns sur les autres. Ces aspects font en sorte que l'exemple est facilement utilisable pour démontrer les validations de cette section. Pour les tests, le scénario *S5* sera modifié.

Tableau 3.23 Modification de l'exemple « Example_Varying_Priorities » pour les tests

Scénario	**Période (ms)**	**Échéance**	**Responsabilité**	**Temps exécution (ms)**	**Priorité**	**Pire latence (ms)**	**Échéance Respectée**
S1	40	40	Voir Tableau **3.22**			28	Oui
S2	100	100				98	Oui
S3	50	50				47	Oui
S4	200	200				195	Oui
S5	**300**	**300**				**313**	**Non**

Avec ce système, la tactique d'augmentation de la période est proposée pour tous les scénarios. Elle est proposée pour le scénario *S5*, car sa latence maximale (313 ms) dépasse sa période de 300 ms (dans cette situation, l'utilisateur pourrait déterminer que le problème vient de l'échéance trop courte, mais la tactique pourrait aussi aider la situation). La tactique est aussi proposée pour tous les autres scénarios, car *S5* contient la responsabilité avec la plus basse priorité (*R5_2* avec une priorité de 1), donc tous les autres scénarios font partie du groupe *H* du scénario *S5*. Différentes applications de la tactique seront examinées à partir de ce même exemple.

La première suggestion examinée sera celle pour le scénario à l'étude. Cette suggestion est faite, car le temps de latence maximal dépasse la période du scénario. La période proposée pour ce scénario est de 313 ms ce qui correspond à la valeur attendue par l'estimation établie à la formule (3.7).

$$T' = \frac{a_n}{\left(\left\lceil \frac{a_n}{T} \right\rceil - 1\right)} = \frac{313}{\left(\left\lceil \frac{313}{300} \right\rceil - 1\right)} = 313\ ms \tag{3.10}$$

En modifiant manuellement la période de *S5* à 313 ms, il y a une amélioration de la latence maximale de *S5* permettant d'obtenir un système qui répond aux exigences.

Tableau 3.24 Résultat de la modification de la période de *S5*

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	40	40	Voir Tableau **3.22**			28	Oui
S2	100	100				98	Oui
S3	50	50				47	Oui
S4	200	200				195	Oui
S5	**313**	300				300	**Oui**

En reprenant le système original qui ne répond pas aux exigences, on examinera les autres suggestions pour cette tactique. Tous les autres scénarios (*S1*, *S2*, *S3* et *S4*) font partie du groupe *H* du scénario *S5* et ont une période plus courte que la latence maximale de *S5*. Il y a donc une suggestion pour chacun d'eux.

Une des suggestions propose d'augmenter la période de *S1* à 44.714 ms. Cette suggestion correspond à l'estimation et permet d'obtenir un système qui répond aux exigences.

$$T' = \frac{a_n}{\left(\left\lceil \frac{a_n}{T} \right\rceil - 1\right)} = \frac{313}{\left(\left\lceil \frac{313}{40} \right\rceil - 1\right)} = 44.714\ ms \tag{3.11}$$

Tableau 3.25 Résultat de la modification de la période de *S1*

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	**44.714**	40	Voir Tableau **3.22**			28	Oui
S2	100	100				98	Oui
S3	50	50				47	Oui
S4	200	200				195	Oui
S5	300	300				**217**	**Oui**

Examinons la suggestion faite pour le scénario *S2*. Pour ce scénario, une période 104.333 ms est proposée. Appliquer cette suggestion laisse la latence du scénario *S5* inchangée. La tactique est alors proposée à nouveau et suggère une période de 156.5 ms pour le scénario *S2*. Appliquer cette deuxième suggestion permet d'obtenir un système qui répond aux exigences.

Tableau 3.26 Résultat de la deuxième modification de la période de *S2*

Scénario	**Période (ms)**	**Échéance**	**Responsabilité**	**Temps exécution (ms)**	**Priorité**	**Pire latence (ms)**	**Échéance Respectée**
S1	40	40	Voir Tableau **3.22**			28	Oui
S2	**156.5**	100				98	Oui
S3	50	50				47	Oui
S4	200	200				195	Oui
S5	300	300				**223**	**Oui**

Avec ces résultats, on peut en conclure que la suggestion est faite correctement, mais qu'il est possible qu'elle ne soit pas suffisante pour répondre totalement aux exigences du système. De plus, toutes les suggestions n'ont pas le même effet sur la latence maximale, certaines peuvent en avoir plus ou moins que d'autres selon la situation (par exemple, modifier *S1* a donné un meilleur résultat que la modification de *S5*). De plus, il a été vérifié que la suggestion proposée correspond bien à l'estimation définie.

Les tests ont été effectués avec les valeurs suggérées. Cependant, l'utilisateur devra s'assurer que cette valeur soit réalisable dans son système. De plus, l'utilisateur peut utiliser des valeurs plus petites ou plus grandes que celles proposées par la tactique et pourra vérifier l'effet que celles-ci auraient grâce à l'analyse d'ArchE.

3.3.8.2 Cas où la tactique n'est pas proposée

Contrairement à certaines tactiques précédentes, notamment l'augmentation des ressources disponibles, l'augmentation de la période d'un scénario n'est pas toujours proposée. Cette section examinera donc divers cas afin de vérifier que la tactique n'est pas proposée de

manière erronée. Dans cet exemple, puisque la période du scénario à l'étude est plus grande que la latence maximale, la tactique n'est pas proposée.

Tableau 3.27 Période du scénario à l'étude trop longue

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	50	40	R1	45	10	45	Non

Si un scénario dans le groupe *H* (*S2* dans l'exemple qui suit) a une période plus longue que la latence maximale, la tactique ne sera pas proposée.

Tableau 3.28 Période du scénario dans le groupe *H* trop longue

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	50	40	R1	45	10	50	Non
S2	60	60	R2	5	20	5	Oui

La tactique n'est pas proposée pour les scénarios dans les groupes autres que *H*, même si leur période est plus courte que la latence maximale obtenue pour *S1*. Dans les exemples suivants, *S2* est dans le groupe *L*, *S3* est dans le groupe *LH* et *S4* est dans le groupe *HL*. La tactique n'est proposée dans aucun de ces cas.

Tableau 3.29 Exemple avec un scénario dans le groupe *L*

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	150	20	R1	50	50	50	Non
S2	20	100	R2	10	10	60	Oui

Tableau 3.30 Exemple avec un scénario dans le groupe *LH*

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	150	20	R1	50	50	60	Non
S3	50	100	R2	10	10	70	Oui
			R3	10	100		

Tableau 3.31 Exemple avec un scénario dans le groupe *HL*

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	150	20	R1	50	50	60	Non
S4	50	100	R4	10	100	70	Oui
			R5	10	10		

Ces exemples démontrent que, même si une partie des conditions sont réunies pour suggérer la tactique, elle ne le sera pas si elles ne sont pas toutes respectées. La tactique est donc suggérée uniquement lorsqu'elle est potentiellement pertinente.

3.4 Augmenter la priorité d'une responsabilité

3.4.1 Description

Dès qu'il y a plus d'une responsabilité dans un même système, celles-ci entrent en compétition pour les différentes ressources (par exemple, le processeur). Pour gérer les demandes concurrentes des responsabilités, les ordonnanceurs ont été développés; ils déterminent à tout moment quelle tâche s'exécute sur le processeur. Plusieurs stratégies d'ordonnancement existent et donneront des résultats différents selon la situation. Le choix de cette stratégie fait l'objet d'une tactique de performance : l'arbitrage des ressources (Bass, Clements et Kazman, 2003, p. 114).

Dans l'analyse effectuée par le cadre de raisonnement de performance d'ArchE, il n'y a qu'une stratégie d'ordonnancement disponible : l'ordonnancement à priorités statiques fixées à l'avance. Bien que cette stratégie ne puisse être changée, ses paramètres peuvent l'être. Cette tactique examinera la priorité des responsabilités afin de déterminer si le scénario à l'étude pourrait bénéficier d'une augmentation de celle-ci pour les responsabilités qui le composent.

Modifier la priorité des responsabilités va modifier la dynamique de l'exécution des scénarios dans ArchE. Une responsabilité avec une priorité plus élevée va avoir préséance sur celles avec une priorité plus basse; son exécution ne sera donc pas bloquée par ces dernières. Un scénario où toutes ses responsabilités sont constamment bloquées va terminer son exécution avec un délai plus grand que s'il avait été seul à s'exécuter. L'augmentation de la priorité des responsabilités d'un tel scénario permettrait à celui-ci de s'exécuter en étant bloqué moins fréquemment et donc de terminer son exécution plus rapidement.

3.4.2 Analyse

Afin de déterminer la pire latence possible, l'analyse divise les scénarios en plusieurs groupes. La formation de ces groupes (voir section 3.3.3) est basée sur la priorité des différentes responsabilités qui composent un scénario. Dans le cadre de cette tactique, la priorité minimale (*PMin*) du scénario à l'étude sera utilisée pour la formation des groupes. Ceci simplifie grandement l'analyse et donnera des résultats plus prudents. Plusieurs de ces groupes ont une influence sur le temps de réponse final. Ils seront examinés séparément dans les prochains paragraphes.

Le premier groupe analysé sera le groupe *H*, car il s'agit du groupe ayant le plus d'influence sur la latence du scénario à l'étude. En effet, toutes les responsabilités contenues dans un scénario de ce groupe contribuent au temps d'exécution final, car leur priorité est toujours plus élevée que la priorité minimale du scénario à l'étude (*PMin*). Ces responsabilités vont donc avoir préséance sur le scénario à l'étude lorsqu'elles vont s'exécuter. Pour qu'un

scénario dans le groupe *H* ne fasse plus partie de ce groupe, il faut que *PMin* passe au-dessus de la priorité d'une des responsabilités de ce scénario. Selon la configuration de la priorité des responsabilités de ce scénario, il se retrouvera soit dans le groupe *HL*, *LH* ou *L*. Peu importe dans quel groupe il se retrouve, cette modification de groupe devrait avoir un effet positif, car la responsabilité qui se retrouve sous *PMin* ne sera plus prise en considération dans le temps d'exécution final.

En ce qui concerne les groupes *HL* et *LH*, l'augmentation de *PMin* au-dessus de la priorité d'une de leurs responsabilités va avoir pour effet de raccourcir le segment prioritaire utilisé dans l'analyse. Dans le cas d'un scénario de type *HL*, ce changement devrait avoir un effet positif, car tous les segments sont comptés. Il en est autrement pour le groupe *LH* où il n'y a que le plus long segment qui est utilisé. Dans la situation où le segment *LH* d'un scénario est raccourci et qu'il n'est pas le plus long, le changement n'aura aucun impact. Dans tous les autres cas, le temps de réponse devrait diminuer pour le scénario à l'étude.

Pour terminer, le groupe *L* n'a aucune influence sur la latence finale, car les scénarios qui s'y trouvent ne peuvent pas bloquer le scénario à l'étude. Les scénarios qui s'y trouvent contiennent tous des responsabilités ayant une priorité plus basse que *PMin*; il est donc inutile de considérer ces responsabilités.

Le but de cette tactique est donc d'augmenter la valeur de *PMin* au-dessus d'une responsabilité d'un autre scénario avec une priorité plus élevée. La priorité plus élevée de la responsabilité fera en sorte que le scénario qui la contient va se retrouver dans un scénario d'un groupe autre que *L* et qu'une amélioration du temps de réponse sera possible. Il faut cependant s'assurer que la responsabilité avec une priorité plus élevée ne soit pas utilisée dans le scénario à l'étude, car en augmentant la priorité, la responsabilité qui avait la priorité plus élevée deviendrait la nouvelle responsabilité correspondant à *PMin* dans le scénario à l'étude et celui-ci serait encore bloqué par les mêmes scénarios.

Dans le cas où une responsabilité avec une priorité plus élevée ferait partie de plusieurs scénarios, le changement sera plus important, car en augmentant *PMin*, plusieurs exécutions de la responsabilité (une ou plusieurs exécutions pour chacun des scénarios) ne bloqueront plus le scénario à l'étude. Si la responsabilité correspondant à *PMin* est utilisée dans des scénarios autres que celui à l'étude, il sera impossible d'éliminer ceux-ci complètement de l'analyse. La tactique pourra cependant réduire au minimum le nombre de responsabilités qui bloquent l'exécution du scénario à l'étude.

En terminant, il faut noter que plusieurs augmentations de la priorité pourraient être nécessaires pour voir une amélioration de la latence (par exemple, si la responsabilité qui ne bloque plus le scénario à l'étude faisait partie d'un scénario dans un groupe *LH* non utilisé).

3.4.3 Déterminer la priorité plus élevée

Dans le contexte de l'automatisation de cette tactique, il ne serait pas souhaitable d'augmenter la priorité *PMin* au dessus de toutes les autres priorités, ce serait un changement trop drastique. Il faut conserver le sens original du programme et augmenter la priorité de façon progressive. C'est pour cette raison que la responsabilité choisie sera celle ayant la prochaine valeur de priorité plus élevée que *PMin*.

Plus précisément, le groupe de responsabilités potentielles sera formé des responsabilités qui ont une priorité plus élevée ou égale à *PMin*, qui ne sont pas dans le scénario à l'étude et qui font partie d'au moins un scénario. Dans ce groupe, la responsabilité ayant la priorité la plus basse sera choisie. On notera la valeur de cette priorité *PSup* (priorité supérieure). Si le groupe est vide, la tactique ne peut pas s'appliquer. De plus, si *PSup* a la même valeur que la priorité maximale du système (c.-à-d. 399) la tactique ne pourra pas s'appliquer, car on ne peut pas dépasser cette valeur maximale. Si *PMin* et *PSup* ont la même valeur, la tactique s'applique quand même, car il faut que *PMin* soit strictement plus grand que *PSup*. L'objectif de cette tactique est donc d'augmenter la valeur de *PMin* au-dessus de *PSup*.

3.4.4 Responsabilités affectées

Dans certains cas, *PMin* et *PSup* correspondent à deux responsabilités ayant des valeurs consécutives, c'est-à-dire qu'il n'y a aucune autre responsabilité avec une priorité entre *PMin* et *PSup*. Cependant, dans d'autres situations il n'en sera pas ainsi. Cette section examine les différents scénarios et les responsabilités qui seront affectés par l'augmentation de la priorité.

Premièrement, si plusieurs responsabilités dans le scénario à l'étude correspondent à *PMin*, la priorité de toutes ces responsabilités doit être augmentée. Si plusieurs responsabilités ont la valeur *PSup* comme priorité, l'application de la tactique n'est pas modifiée, car ces responsabilités ne sont pas touchées par le changement.

Un deuxième cas où plusieurs responsabilités devront être modifiées est celui où il y a une ou plusieurs responsabilités du scénario à l'étude entre *PMin* et *PSup* (inclusivement). Dans cette situation, la priorité de toutes les responsabilités entre *PMin* et *PSup* devra être augmentée. Si ces responsabilités n'étaient pas incluses, il n'y aurait pas de changement, car la responsabilité *PSup* va toujours bloquer celles qui se trouvent encore entre *PMin* et *PSup*.

Enfin, si d'autres responsabilités ne faisant partie d'aucun scénario se retrouvent avec une priorité entre *PMin* et *PSup*, elles seront ignorées par cette tactique, car elles ne font pas partie de l'analyse de performance du système.

En résumé, il faut augmenter la priorité des responsabilités du scénario à l'étude qui ont une priorité *P* de telle sorte que *PMin* <= *P* <= *PSup*.

3.4.5 Application de la tactique

L'application de la tactique de façon automatique ou à la demande de l'utilisateur se fait de la même manière : la priorité de toutes les responsabilités affectées est modifiée à la valeur *PSup* + 1. Lorsque la suggestion est proposée à l'utilisateur, celui-ci n'a qu'à confirmer qu'il

veut appliquer cette tactique. Dans la question affichée, l'utilisateur peut voir les responsabilités affectées ainsi que la nouvelle valeur de la priorité.

En modifiant la priorité à la valeur *PSup* + 1, on s'assure que toutes les responsabilités modifiées ne sont plus bloquées par l'exécution de la responsabilité correspondant à *PSup*. Il est possible que cette nouvelle valeur (*PSup* + 1) existe déjà dans une autre responsabilité, mais cela n'a aucun impact, car les doublons de valeur de priorité sont admis et une nouvelle application de l'augmentation de la priorité fera quand même passer la valeur de la priorité au-dessus d'une valeur identique.

3.4.5.1 Interactions

Tableau 3.32 Interactions entre les acteurs pour la tactique « Augmenter la priorité d'une responsabilité »

#	Acteur	Action
1.1	ArchE	Analyse l'architecture actuelle : au moins un scénario de performance n'est pas satisfait.
1.2	ArchE	Demande au cadre de raisonnement de suggérer des tactiques.
2.1	RF	Examine toutes les responsabilités
2.2	RF	Sélectionne la responsabilité qui a la priorité la plus basse parmi les responsabilités ayant une priorité plus haute que *PMin*, qui ne font pas partie du scénario à l'étude et qui font partie d'au moins un scénario.
2.3	RF	Retourne une tactique pour cette responsabilité (si elle existe).
3	ArchE	Demande d'appliquer automatiquement la tactique sur une copie de l'architecture.
4	Tactique	Augmente la priorité des responsabilités affectées à la valeur *PSup* + 1.
5.1	ArchE	Analyse la copie de l'architecture modifiée.
5.2	ArchE	Affiche les tactiques et les résultats à l'utilisateur.
6	Utilisateur	Examine la question posée et donne son accord pour l'augmentation automatique de la priorité.
7	ArchE	Transmet la décision de l'utilisateur à la tactique.
8	RF	Augmente la priorité des responsabilités affectées à la valeur *PSup* + 1.

3.4.6 Exemples et validation de la tactique

En résumé, l'application de la tactique d'augmentation de la priorité va modifier la valeur de la priorité de la responsabilité ayant la plus basse priorité dans le scénario à l'étude afin qu'elle ait une valeur plus élevée que la priorité la plus proche d'une responsabilité hors du scénario à l'étude. La tactique s'applique tout le temps sauf s'il n'existe pas de responsabilités ayant une priorité plus élevée ou si cette priorité correspond à la valeur maximale possible.

Les tests qui suivent sont divisés en trois sections : les tests où il n'y a qu'une seule responsabilité à modifier, les tests où plusieurs responsabilités doivent être modifiées et les situations où la tactique ne devrait pas être proposée à l'utilisateur.

3.4.6.1 Une seule responsabilité à augmenter

Les cas les plus simples commencent la validation : une seule responsabilité doit être modifiée pour obtenir un changement dans la pire latence. Le cas le plus simple est lorsqu'il y a deux scénarios avec chacun une responsabilité.

Tableau 3.33 Exemple avec deux scénarios avec chacun une responsabilité

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	25	R1	10	10	30	Non
S2	100	30	R2	20	20	20	Oui

Dans cet exemple, la tactique est suggérée pour la responsabilité *R1* du scénario *S1*. La proposition est juste, car *S2* fait partie du groupe *H* de *S1* et il est possible d'augmenter la priorité de *R1* (10) au dessus de celle de *R2* (20). Pour appliquer la tactique, l'utilisateur confirme son choix et l'augmentation de la priorité de la responsabilité se fait automatiquement. La priorité de la responsabilité *R1* est augmentée à 21 (20 + 1) et le système obtenu est valide.

Tableau 3.34 Résultat de l'augmentation de la priorité sur l'exemple du Tableau 3.32

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	25	R1	10	**21**	**10**	**Oui**
S2	100	30	R2	20	20	30	Oui

Une extrapolation du dernier exemple pourrait être un système où la responsabilité *R2* est divisée en deux parties avec des priorités différentes. Dans cet exemple, l'augmentation de la priorité de *R1* au dessus de celle de *R2_1* est suffisante pour obtenir un système valide. La responsabilité R2_2 conserve sa priorité plus élevée.

Tableau 3.35 Exemple où la responsabilité avec une priorité haute est séparée en deux

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	25	R1	10	10	30	Non
S2	100	30	R2_1	10	20	20	Oui
			R2_2	10	30		

La tactique est proposée pour *R1* du scénario *S1*. Après l'application de la tactique, la priorité de *R1* est à 21. La responsabilité se retrouve entre les priorités de *R2_1* (qui ne bloque plus *R1*) et *R2_2* (qui bloque encore l'exécution de *R1*), mais le système répond tout de même aux exigences.

Tableau 3.36 Résultat de l'augmentation de la priorité sur l'exemple du Tableau 3.35

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	25	R1	10	**21**	**20**	**Oui**
S2	100	30	R2_1	10	20	30	Oui
			R2_2	10	30		

Lorsque plusieurs responsabilités ont la même priorité, l'analyse considère que les responsabilités des autres scénarios avec une priorité identique sont plus élevées afin d'obtenir le pire cas possible.

Tableau 3.37 Exemple avec des priorités identiques

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	25	R1	10	10	30	Non
S2	100	30	R2	20	10	30	Oui

Dans ce cas, l'analyse présume que chacune des responsabilités va bloquer l'autre lors de leur exécution. La tactique est proposée pour *R1* du scénario *S1*, car il est possible d'augmenter la priorité de *R1* pour faire en sorte qu'elle ne soit plus bloquée par *R2*. En appliquant la tactique, *R1* obtient une priorité plus élevée que *R2* et le système est valide.

Tableau 3.38 Résultat de l'augmentation de la priorité sur l'exemple du Tableau 3.36

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	25	R1	10	**11**	**10**	**Oui**
S2	100	30	R2	20	10	30	Oui

3.4.6.2 Plusieurs responsabilités dans un scénario

Lorsqu'il y a plusieurs responsabilités dans un scénario, l'analyse examine uniquement la responsabilité avec la plus basse priorité du scénario (*PMin*). Dans l'exemple suivant, augmenter la priorité de *R1* au-dessus la priorité de *R3* n'aura aucun effet puisque *R3* va encore bloquer l'exécution de *R2*.

Tableau 3.39 Exemple avec plusieurs responsabilités dans plusieurs scénarios

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	30	R1	10	30	40	Non
			R2	10	10		
S2	100	35	R3	10	40	30	Oui
			R4	10	20		

La tactique suggère d'augmenter la priorité de *R2* à 21 pour être au dessus de celle de *R4* (20). Appliquer cette suggestion permet d'obtenir un changement positif pour la latence du scénario à l'étude, contrairement à l'augmentation de la priorité de *R1* qui ne fait aucun changement.

Tableau 3.40 Résultat de la première application de la tactique sur l'exemple du Tableau 3.39

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	30	R1	10	30	**30**	**Oui**
			R2	10	**21**		
S2	100	35	R3	10	40	**40**	**Non**
			R4	10	20		

Un effet secondaire de cette tactique est qu'elle peut nuire aux autres scénarios. En effet, la dernière application de cette tactique réduit la latence maximale du scénario *S1*, mais elle a aussi augmenté celle du scénario *S2* qui est passé d'un délai maximal de 30 ms à 40 ms. Dans un système normal, l'utilisateur devra évaluer ces changements, faire des compromis et décider si l'augmentation de la priorité est avantageuse. Dans certains cas, le délai pourrait augmenter, mais tout de même rester valide.

Appliquer la tactique d'augmentation de la priorité sur R4 ne donnera pas un résultat intéressant, car la situation reviendrait à celle que l'on avait avant. L'exemple est donc dans une impasse en ce qui concerne la tactique d'augmentation de la priorité.

Afin de résoudre cette impasse et continuer avec les tests, l'échéance des scénarios sera modifiée : celle de *S2* sera de 40 ms (pour devenir valide) et celle de *S1* sera de 20 ms (pour redevenir invalide). Dans cette situation, le scénario *S1* est invalide et il est de nouveau possible d'augmenter la priorité de ses responsabilités puisqu'il y a *R3* avec une priorité plus élevée.

La suggestion propose donc d'augmenter la priorité des responsabilités *R1* et *R2* à une valeur de 41 ms. La tactique doit augmenter la priorité des deux responsabilités, car *R3* bloquerait encore la responsabilité avec la priorité inchangée et il n'y aurait pas d'amélioration de la latence. L'application de la tactique donne un système valide.

Tableau 3.41 Résultat des modifications et de la deuxième application de la tactique

Scénario	**Période (ms)**	**Échéance**	**Responsabilité**	**Temps exécution (ms)**	**Priorité**	**Pire latence (ms)**	**Échéance Respectée**
S1	100	**20**	R1	10	**41**	**20**	**Oui**
			R2	10	**41**		
S2	100	**40**	R3	10	40	40	Oui
			R4	10	20		

3.4.6.3 Situations où la tactique ne doit pas être proposée

Dans cette section, les cas où la tactique ne devrait pas être proposée sont examinés. Premièrement, le cas où il n'existe pas de responsabilités avec une priorité plus élevée que *PMin* est examiné. Dans ce cas, il est impossible d'appliquer la tactique, car toutes les responsabilités du scénario sont déjà les plus élevées du système. Le raisonnement est le même si le système est composé d'un seul scénario.

Tableau 3.42 Exemple où aucune autre responsabilité n'a de priorité plus haute

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	20	R1	10	20	30	Non
			R2	10	15		
S2	100	50	R3	10	10	30	Oui

L'outil MAST fixe une valeur maximale pour la priorité pour les responsabilités de 399. Il est donc impossible d'augmenter la priorité d'une responsabilité au dessus de la priorité d'une autre responsabilité, si cette dernière est égale à 399.

Tableau 3.43 Exemple où il y aurait dépassement de la valeur maximale pour la priorité

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
S1	100	15	R1	10	10	20	Non
S2	100	30	R2	10	399	10	Oui

3.5 Tactiques non implémentées

Telles que vues précédemment, toutes les tactiques contenues dans la Figure 3.1 ne sont pas intégrées à ArchE. En effet, ArchE reste général dans la conception de l'architecture et n'a aucune idée du domaine d'application du système qu'il analyse. Dans cette section, les limitations qui ont empêché l'intégration de certaines tactiques nommées précédemment seront expliquées.

3.5.1 Réduire les calculs supplémentaires lors du traitement d'une opération (« Reduce Computational Overhead »)

La tactique de réduction des calculs supplémentaires lors du traitement d'une opération indique qu'il faut réduire les ressources consommées en trop (« overhead ») lors du

traitement d'un événement comprenant une série d'actions. Pour y arriver, le développeur peut soit retirer des intermédiaires ou réduire les coûts de communication entre les actions.

Cette tactique n'a pas été implémentée, car ArchE n'ajoute pas un coût pour les communications entre chacune des responsabilités. Il est donc impossible de réduire ce coût avec des communications plus rapides.

Le retrait d'intermédiaire pourrait être une option dans le cas où le temps d'exécution total de la nouvelle responsabilité sans intermédiaire serait moindre que l'addition des temps d'exécution de l'intermédiaire et de la responsabilité. Le retrait des liens entre les responsabilités n'aura aucun effet puisqu'il n'y a pas de coût qui y est associé.

Il aurait pu être possible de combiner un intermédiaire avec la responsabilité qu'il masque et de donner à cette nouvelle responsabilité un temps total d'exécution moindre. Cependant, il n'est pas simple de séparer logiquement les intermédiaires des responsabilités contenus dans ArchE, car ils ont la même signification pour ce dernier.

3.5.2 Introduire le parallélisme (« Introduce Concurrency »)

L'exécution parallèle des responsabilités avec plus d'un processeur est une tactique intéressante permettant d'exécuter les tâches plus rapidement. Bien qu'il soit possible dans ArchE d'associer les responsabilités à différents processus qui pourraient s'exécuter en parallèle, l'algorithme pour déterminer le pire cas de latence possible ne supporte pas les systèmes avec l'exécution concurrente d'actions.

Une estimation du gain de l'ajout d'un ou plusieurs processeurs serait possible, mais en pratique il sera impossible d'analyser le système par la suite si l'utilisateur décide d'appliquer la tactique.

3.5.3 Conserver plusieurs copies (« Maintain Multiple Copies »)

Cette tactique propose de conserver une copie des données ou des calculs utilisés fréquemment ou couteux à obtenir. Cette tactique n'a pas été intégrée, car nous n'avons aucun détail sur le type de système conçu ni le type d'implémentation qu'il en résulterait. Il est donc impossible de déterminer les calculs ou les données qui pourraient être mis en cache.

Pour avoir une estimation relativement fiable, il faudrait connaître quelles responsabilités utilisent les données ou les calculs communs, quel est le coût pour obtenir à nouveau ou recalculer cette information, ainsi que le travail supplémentaire nécessaire (ex. : la synchronisation) pour conserver les copies mises en cache. Ces informations n'étant pas disponibles dans ArchE, il n'est donc pas possible de diriger adéquatement l'utilisateur pour l'application de cette tactique.

Bien que ces dernières tactiques auraient tout de même pu, jusqu'à une certaine limite, être intégrées dans ArchE, elles ont été mises de côté, car elles n'offraient que des solutions partielles ou non optimales au problème de l'utilisateur.

3.6 Conclusion

Dans ce chapitre, nous avons vu en détail la description, l'analyse et le développement d'une méthode automatisée pour quatre tactiques dans le but de les intégrer dans le cadre de raisonnement de performance livré avec ArchE. Les tactiques implémentées sont : l'augmentation des ressources disponibles, la réduction du temps d'exécution d'une responsabilité, l'augmentation de la période d'un scénario et l'augmentation de la priorité d'une responsabilité. Chaque tactique possède sa propre analyse et ses propres règles afin de déterminer les paramètres de l'application de la tactique sur le système.

Les tests individuels effectués dans les dernières sections ont vérifié que les tactiques étaient proposées et appliquées correctement selon la situation. Ils ont vérifié que les valeurs obtenues étaient justes et que les responsabilités touchées étaient les bonnes. De plus, les

tests ont vérifié que la suggestion et l'application (avec et sans l'aide de l'utilisateur) des tactiques étaient effectuées automatiquement sans problèmes.

En plus des tests pour démontrer l'application de la tactique, il a été vérifié que, dans les cas pertinents, les tactiques n'étaient pas proposées lorsqu'elles ne devaient pas l'être. Aucune fausse proposition n'a été détectée.

Ces tests ont vérifié chacune des tactiques dans un contexte individuel, c'est-à-dire que les interactions qu'elles pourraient avoir entre elles, ainsi que les suggestions multiples de tactiques, étaient ignorées. Cette limitation avait pour but de concentrer les tests uniquement sur le bon fonctionnement d'une seule tactique à la fois. Puisqu'il a été déterminé que leur fonctionnement individuel était correct, l'utilisation des tactiques en groupe pourra être vérifiée correctement.

CHAPITRE 4

VALIDATION ET RÉSULTATS

Figure 4.1 Méthodologie : Validation de l'ajout des tactiques.

Pour valider le bon fonctionnement des tactiques implémentées dans le chapitre précédent, il faut effectuer certains tests. Ce chapitre se concentre donc sur cette dernière étape de la méthodologie. L'objectif est de vérifier si les tactiques sont suggérées et appliquées automatiquement et correctement.

Pour valider le prototype développé, deux techniques sont utilisées : les tests individuels et les tests de système. Les premiers, qui sont décrits dans le chapitre précédent, servent à valider le bon fonctionnement de chacune des tactiques individuellement, alors que les tests système du présent chapitre valident le fonctionnement collectif des tactiques.

Avant d'examiner les différents tests, il est important de se rappeler certains faits :

- le système est monoprocesseur;
- une stratégie d'ordonnancement à priorités fixes avec préemption est utilisée pour déterminer l'ordre d'exécution des responsabilités;
- tous les temps affichés sont en ms;
- l'analyse de performance est effectuée par l'outil externe MAST qui applique la théorie RMA;
- un système est défini comme étant invalide lorsqu'il s'y trouve un scénario dont l'échéance (la mesure de réponse) est plus courte que son pire cas de latence possible;
- la recherche de suggestions de tactiques est effectuée uniquement si le système est invalide.

4.1 Tests individuels

Dans le but de faciliter la compréhension des tactiques ajoutées dans ArchE, les tests individuels sont décrits avec la description de chacune des tactiques dans le chapitre précédent. Cette section récapitule les résultats obtenus lors de ces tests.

Dans l'ensemble, les tests individuels démontrent que les tactiques sont implémentées correctement : elles sont suggérées automatiquement, elles sont suggérées uniquement lorsqu'il est théoriquement possible de les appliquer et leur application automatique avec la réponse de l'utilisateur se fait selon les spécifications. De plus, les tests vérifient que les valeurs obtenues sont justes et que les responsabilités touchées sont les bonnes.

Lors des tests individuels, les tactiques proposées autres que celle à l'étude étaient ignorées. La prochaine section examinera donc, dans un but de validation, l'utilisation de plusieurs tactiques sur un même problème.

4.2 Tests de système

Les tests individuels sont intéressants pour vérifier le fonctionnement d'une tactique. Cependant, s'il est impossible d'utiliser plusieurs tactiques de performance sur un même système, l'utilité de cette fonctionnalité devient un peu plus limitée. Dans cette section, nous décrivons des tests de portée « système », où plusieurs tactiques de performance sont utilisées séquentiellement pour résoudre un problème.

Pour y arriver, un exemple existant est utilisé. Il est saisi dans ArchE et des tactiques de performances sont suggérées et appliquées afin d'obtenir un système qui répond aux exigences définies dans les scénarios de qualité.

4.2.1 Description du contrôleur de robot original

Le contrôleur de robot utilisé pour effectuer les tests de système est tiré de (Moreno et Hansen, 2009). Bien que l'exemple soit défini en langage CCL, la traduction peut se faire directement puisque ICM (le format utilisé par ArchE pour l'analyse de la performance) est basé sur la terminologie CCL. Les services deviennent des scénarios, les composantes deviennent des responsabilités et les liens entre les composantes deviennent des liens de réactions.

Cet exemple a été choisi, car il a été suggéré par le responsable d'ArchE au SEI, il a été utilisé comme démonstration pour le cadre de raisonnement utilisé (Lambda-WBA), il représente un système facile à comprendre et il est assez substantiel, avec plusieurs paramètres qui peuvent interagir entre eux, sans être trop complexe.

La figure suivante résume les liens entre les responsabilités et les scénarios :

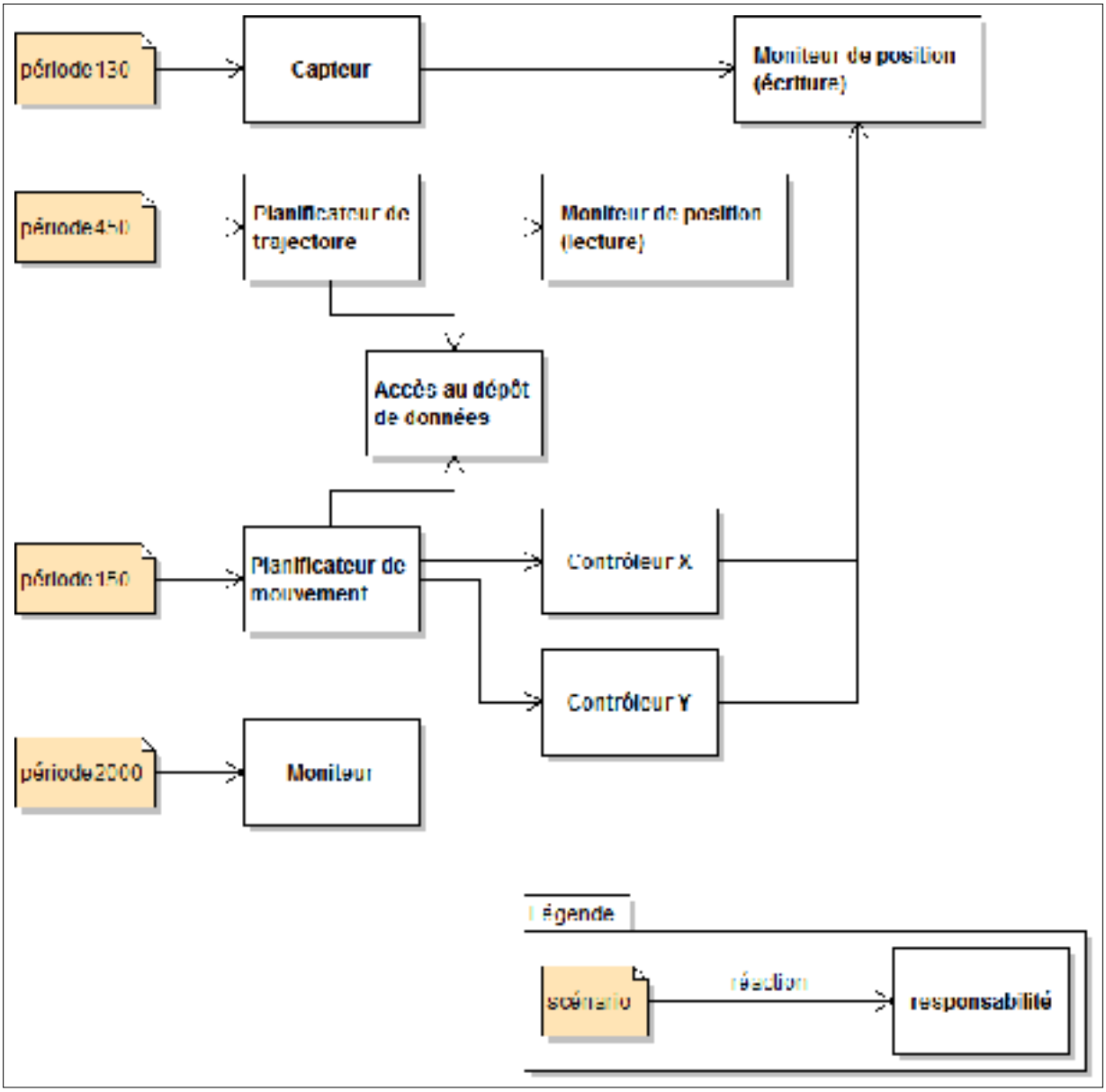

Figure 4.2 Diagramme résumant les composantes du contrôleur de robot.

Pour simplifier la lecture des tableaux de description du système, des codes sont assignés aux responsabilités. Le tableau suivant définit les codes utilisés ainsi que les paramètres associés aux responsabilités. Dans la définition originale du problème, trois temps d'exécution (minimum, moyen, maximum) sont donnés pour chacune des responsabilités. Dans ce test, le temps d'exécution maximal est utilisé pour obtenir le pire cas possible.

Tableau 4.1 Responsabilités codifiées avec leurs paramètres pour le contrôleur de robot

Code	Nom	Priorité	Temps d'exécution (ms)
R1	Accès au dépôt de données	18	20.8
R2	Capteur	10	5.6
R3	Contrôleur X	20	13.5
R4	Contrôleur Y	20	13.5
R5	Moniteur	2	0.5
R6	Moniteur de position (écriture)	12	10.8
R7	Moniteur de position (lecture)	14	3.2
R8	Planificateur de mouvement	16	21.0
R9	Planificateur de trajectoire	4	90.5

Le tableau suivant résume l'état initial du contrôleur de robot dans ArchE. Il faut noter que l'échéance pour certains scénarios n'est pas spécifiée. Elle sera définie lorsque l'exemple sera ajusté pour les tests.

Tableau 4.2 Définition originale du contrôleur de robot

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
période130	130	Aucune	R2	5.6	10	110.0	N/D
			R6	10.8	12		
période450	450	450	R9	90.5	4	337.4	Oui
			R7	3.2	14		
			R1	20.8	18		
période150	150	150	R8	21.0	16	93.6	Oui
			R1	20.8	18		
			R3	13.5	20		
			R6	10.8	12		
			R4	13.5	20		
			R6	10.8	12		
période2000	2000	Aucune	R5	0.5	2	373.0	N/D

En se basant sur la priorité des responsabilités qui les composent, on peut déduire que les scénarios ayant une plus grande priorité sont *période130* et *période150*, alors que *période450* et *période2000* ont une priorité plus faible.

4.2.2 Modifications du système

Afin d'avoir un système propice à l'application de plusieurs tactiques de performance, certains paramètres doivent être modifiés et définis dans l'exemple du contrôleur de robot. Cette section définit le système initial dans ArchE qui sera utilisé pour les tests système. Ce système est basé sur les spécifications du contrôleur de robot, mais avec certains paramètres modifiés.

Tableau 4.3 Modification de l'exemple pour les tests système

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
période130	130	**130**	R2	5.6	**3**	500.1	Non
			R6	**15.0**	12		
période450	450	450	R9	90.5	4	333.0	Oui
			R7	**15.0**	14		
			R1	**22.0**	18		
période150	**140**	150	R8	21.0	16	115.0	Oui
			R1	**22.0**	18		
			R3	13.5	20		
			R6	**15.0**	12		
			R4	13.5	20		
			R6	**15.0**	12		
période2000	2000	**100**	R5	0.5	2	10776.3	Non

Avec ce système qui ne répond plus aux exigences de qualité des scénarios, il est possible d'utiliser les tactiques proposées par le cadre de raisonnement de performance d'ArchE pour modifier le système pour qu'il réponde aux exigences.

4.2.3 Définition du contrôleur de robot dans ArchE

Cette section examine la définition du contrôleur de robot dans ArchE. La figure suivante résume le test de système dans son état initial. Les sections numérotées seront décrites à la suite de la figure.

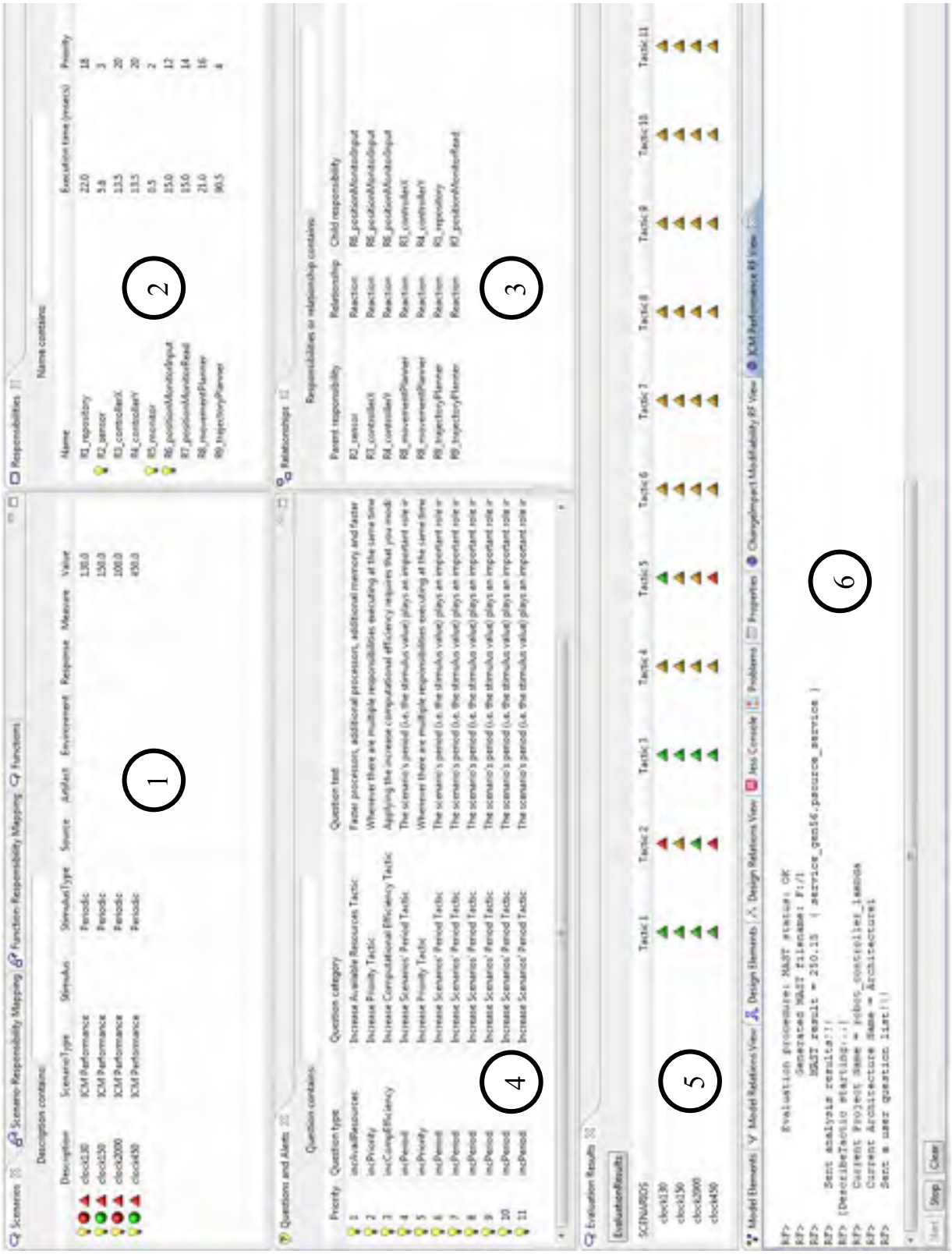

Figure 4.3 Définition du contrôleur de robot dans ArchE.

L'architecture du contrôleur de robot est définie dans les trois premières sections :

- la section 1 contient les spécifications des quatre scénarios;
- la section 2 contient la liste des responsabilités ainsi que les paramètres qui leur sont rattachés (le temps d'exécution et la priorité);
- la section 3 contient les liens entre les responsabilités (les liens « réactions »).

La définition initiale du contrôleur de robot fait en sorte que onze tactiques sont proposées. On peut voir ces tactiques à la section 4. Il est intéressant de noter que les quatre tactiques intégrées sont proposées initialement. C'est dans cette section que l'utilisateur pourra examiner chacune des tactiques en détail, choisir celle qu'il veut appliquer et fournir les informations supplémentaires pour la tactique (si nécessaire).

La section 5 indique, à l'aide d'un code de couleur, l'impact qu'a chacune des tactiques sur les scénarios lorsqu'elles sont appliquées automatiquement avec les valeurs par défaut :

- triangle vert : réduction du temps de latence maximal;
- triangle jaune : aucun impact sur le temps de latence maximal;
- triangle rouge : augmentation du temps de latence maximal.

La section 6 contient l'interface du cadre théorique de performance. Il contient uniquement des boutons pour démarrer et arrêter celui-ci en plus d'une fenêtre pour afficher le journal de bord de l'exécution.

4.2.4 Utilisation d'ArchE pour résoudre le problème

Cette section montre l'utilisation successive de plusieurs tactiques pour arriver à un système qui répond à toutes les exigences fixées dans les scénarios de qualité. Il s'agit ici d'une seule séquence d'utilisation, un système valide pourrait être obtenu avec une séquence d'étapes différentes. Le but de cette section est d'évaluer l'utilisation de plusieurs tactiques sur un même problème, et non d'examiner toutes les possibilités d'utilisation des tactiques sur un même problème.

La première tactique appliquée est celle qui suggère l'augmentation des ressources. Dans ce cas, on utilise une augmentation du taux de calculs effectués par unité de temps de 10 %. Cette amélioration réduit le temps d'exécution de toutes les responsabilités du système. La pire latence de tous les scénarios est réduite. Même s'il y a des réductions importantes (ex. : la pire latence de *période2000*), ce n'est pas suffisant pour que l'échéance de tous les scénarios soit respectée.

Tableau 4.4 Résultat de l'application de la tactique « augmenter les ressources disponibles » avec une valeur de 10%

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
période130	130	130	R2	**5.0**	3	**276.4**	Non
			R6	**13.6**	12		
période450	450	450	R9	**82.2**	4	**302.7**	Oui
			R7	**13.6**	14		
			R1	**20.0**	18		
période150	140	150	R8	**19.0**	16	**104.5**	Oui
			R1	**20.0**	18		
			R3	**12.2**	20		
			R6	**13.6**	12		
			R4	**12.2**	20		
			R6	**13.6**	12		
période2000	2000	100	R5	**0.4**	2	**385.2**	Non

ArchE propose d'augmenter la période des scénarios *période130* et *période150*. En examinant les suggestions, on décide d'augmenter la période du scénario *période150* à 160 ms. La tactique proposait une valeur de 192.6 ms, mais il y a tout de même une amélioration de la latence pour le scénario *période2000* avec la valeur réduite.

Tableau 4.5 Résultat de l'application de la tactique « augmenter la période d'un scénario » sur le scénario *période150* avec une valeur de 160 ms

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
période130	130	130	R2	5.0	3	276.4	Non
			R6	13.6	12		
période450	450	450	R9	82.2	4	302.7	Oui
			R7	13.6	14		
			R1	20.0	18		
période150	**160**	150	R8	19.0	16	104.5	Oui
			R1	20.0	18		
			R3	12.2	20		
			R6	13.6	12		
			R4	12.2	20		
			R6	13.6	12		
période2000	2000	100	R5	0.4	2	**314.3**	Non

Le cadre de raisonnement suggère d'améliorer l'efficacité des calculs de R6. Les responsabilités R6 et R7 sont liées entre elles, car elles effectuent chacune une tâche du moniteur de position. On détermine que le temps d'exécution de R6 peut être réduit à 10.5 ms et celui de R7 à 7.0 ms. Cette modification réduit la latence de tous les scénarios.

Tableau 4.6 Résultat de l'application de la tactique « réduire le temps d'exécution d'une responsabilité » sur les responsabilités R6 (10.5 ms) et R7 (7.0 ms)

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
période130	130	130	R2	5.0	3	**254.1**	Non
			R6	**10.5**	12		
période450	450	450	R9	82.2	4	**215.7**	Oui
			R7	**7.0**	14		
			R1	20.0	18		
période150	160	150	R8	19.0	16	**91.6**	Oui
			R1	20.0	18		
			R3	12.2	20		
			R6	**10.5**	12		
			R4	12.2	20		
			R6	**10.5**	12		
période2000	2000	100	R5	0.4	2	**285.7**	Non

Le dernier type de tactique proposé par ArchE est l'augmentation de la priorité d'une responsabilité. Dans ce cas, ArchE propose d'augmenter la priorité de R2 à 5, ce qui l'élèverait au-dessus de la priorité de R9. Cette modification réduit la latence de *période130* et fait en sorte que ce scénario respecte son échéance. La latence de *période450* augmente, mais reste sous son échéance. Il reste uniquement le scénario *période2000* qui ne respecte pas son échéance.

Tableau 4.7 Résultat de l'application de la tactique « augmenter la priorité d'une responsabilité » sur la responsabilité R2

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
période130	130	130	R2	5.0	**5**	**107.2**	**Oui**
			R6	10.5	12		
période450	450	450	R9	82.2	4	**301.0**	Oui
			R7	7.0	14		
			R1	20.0	18		
période150	160	150	R8	19.0	16	91.6	Oui
			R1	20.0	18		
			R3	12.2	20		
			R6	10.5	12		
			R4	12.2	20		
			R6	10.5	12		
période2000	2000	100	R5	0.4	2	285.7	Non

ArchE propose aussi d'augmenter la priorité de R5. La première application de la tactique propose d'augmenter la priorité à 5 et permet de réduire la latence de *période2000* à 107.6 ms. La tactique est à nouveau proposée pour R5 avec une priorité de 6. Cette application réduit la latence à 92.0 ms et fait en sorte que *période2000* respecte son échéance. Ces applications de la tactique ont nui légèrement à la latence de période130 et de période450, mais ces scénarios respectent toujours leur échéance respective. Le système répond maintenant à toutes les exigences définies dans les scénarios.

Tableau 4.8 Résultat de l'application de la tactique « augmenter la priorité d'une responsabilité » sur la responsabilité R5

Scénario	Période (ms)	Échéance	Responsabilité	Temps exécution (ms)	Priorité	Pire latence (ms)	Échéance Respectée
période130	130	130	R2	5.0	5	**107.6**	Oui
			R6	10.5	12		
période450	450	450	R9	82.2	4	**301.5**	Oui
			R7	7.0	14		
			R1	20.0	18		
période150	160	150	R8	19.0	16	91.6	Oui
			R1	20.0	18		
			R3	12.2	20		
			R6	10.5	12		
			R4	12.2	20		
			R6	10.5	12		
période2000	2000	100	R5	0.4	**6**	**92.0**	**Oui**

4.2.5 Résultats

Telle que décrite à la section précédente, une série d'applications successives de tactiques proposées automatiquement par ArchE nous permet d'obtenir un système qui répond à toutes les exigences de performance définies dans les scénarios d'attributs de qualité. Cette série de tactiques utilise les quatre tactiques de performances intégrées dans ce projet. Ceci montre qu'elles peuvent être utilisées conjointement pour atteindre les objectifs d'un utilisateur.

Bien que le système obtenu ne soit pas identique à l'exemple initial, il répond tout de même aux exigences initiales. Des contraintes supplémentaires auraient pu limiter les modifications possibles (par exemple, une période fixe qui ne peut pas être modifiée), mais elles n'étaient pas spécifiées dans la définition originale.

La série de tactiques utilisée n'est qu'un exemple. Il aurait été possible d'obtenir un système qui répond aux exigences en utilisant les tactiques dans un ordre différent ou avec des paramètres différents. Le nombre et les types des tactiques utilisées pourraient aussi être

différents (par exemple, utiliser uniquement la tactique d'augmentation des ressources avec 130 % d'augmentation donne un système valide).

Un aspect intéressant qui peut être observé est que certaines décisions ne sont pas intuitives pour un utilisateur. Par exemple, la réduction du temps d'exécution de la responsabilité la plus utilisée (R6) a eu un effet beaucoup plus petit que l'augmentation de la priorité de R2 pour le scénario *période130* (réduction de 22.3 ms contre 146.9 ms). De plus, certaines petites modifications peuvent avoir un impact énorme, par exemple, l'augmentation des ressources de 10 % a réduit le temps de latence maximal de *période2000* de 10776.3 ms à 385.2 ms.

4.3 Conclusion

Dans ce chapitre, la validation de portée « système » de l'implémentation des tactiques a été décrite. Les tests de système ont permis de vérifier que l'utilisation conjointe de plusieurs tactiques est possible pour résoudre les problèmes de performance d'un système. En conclusion, les tests ont démontré que la suggestion et l'application automatique des tactiques de performances intégrées dans ArchE fonctionnent correctement à tous les niveaux.

CONCLUSION

L’objectif de ce projet est de déterminer la faisabilité de l’automatisation des tactiques architecturales de performance, car la recherche et l’application manuelles de celles-ci peuvent être longues et fastidieuses. Avant ce projet, les tactiques de performances étaient décrites, mais il n’y avait pas de détails sur la possibilité de les automatiser. L'hypothèse de départ était qu'avec l’aide du système expert ArchE, il est possible d’intégrer des fonctionnalités pour l’automatisation des tactiques de performance.

Pour atteindre notre objectif, nous avons commencé par analyser les cadres de raisonnement inclus avec ArchE. Cette analyse a permis de déterminer les points forts et les défis des tactiques existantes dans le cadre de raisonnement de modificabilité et de développer une solution qui répond à ces problèmes pour l’intégration des tactiques architecturales dans le cadre de raisonnement de performance.

Une série de tactiques de performance tirée du livre « Software Architecture in Practice » (Bass, Clements et Kazman, 2003) a été analysée afin d’en développer une méthode automatisée. Quatre tactiques ont été intégrées : l’augmentation des ressources disponibles, la réduction du temps d’exécution d’une responsabilité, l’augmentation de la période d’un scénario et l’augmentation de la priorité d’une responsabilité. Chacune des tactiques possède une série de règles qui déterminent si et comment la tactique doit être appliquée sur l’architecture logicielle. Certaines tactiques décrites dans le livre n’ont pas pu être implémentées à cause de certaines limitations d’ArchE ou parce qu’elles offraient une solution partielle ou non optimale au problème rencontré.

Deux types de tests ont été effectués sur les tactiques de performance ajoutées : des tests individuels ainsi que des tests de système. Les tests individuels examinaient une seule tactique à la fois et ont montré que les tactiques sont suggérées automatiquement, qu’elles sont suggérées uniquement lorsqu’il est théoriquement possible de les appliquer, que leur application automatique avec la réponse de l’utilisateur se fait selon les spécifications, que

les valeurs obtenues sont justes et que les responsabilités touchées sont les bonnes. Les tests de système ont montré que plusieurs tactiques peuvent être utilisées conjointement pour résoudre les problèmes de performance d'une architecture logicielle.

Les résultats obtenus démontrent que l'automatisation des tactiques de performance est possible en utilisant un système expert. Il faut cependant tenir compte du contexte et des limitations du système expert utilisé, car ils peuvent limiter les capacités d'automatisation des tactiques architecturales de performance.

Bien que les tactiques de performance et des tactiques automatisées existent déjà, nous estimons faire une contribution originale à deux endroits. Premièrement, nous apportons une conception plus simple pour l'intégration de l'automatisation des tactiques. Celle-ci pourrait servir de base pour l'automatisation de tactiques dans d'autres cadres de raisonnement et d'autres systèmes experts. Deuxièmement, les règles pour l'automatisation de l'application des tactiques de performance n'existaient pas et ont dû être développées. L'intégration dans le système expert a permis de valider que ces règles fonctionnent correctement sur plusieurs exemples.

RECOMMANDATIONS

Les résultats obtenus dans ce projet sont intéressants, mais ce n'est que le début en ce qui concerne l'automatisation des tactiques architecturales. En effet, plusieurs aspects peuvent être approfondis ou améliorés pour faire avancer ce domaine.

Premièrement, puisque les tactiques implémentées servaient à vérifier la faisabilité de l'automatisation, leur nombre est limité et leur portée reste générale. Il serait intéressant d'en augmenter le nombre et d'automatiser des tactiques spécifiques à certains domaines. Il pourrait y avoir plusieurs séries des tactiques de performance disponibles pour différents domaines. Puisque le parallélisme est un domaine très populaire depuis l'omniprésence des processeurs à plusieurs cœurs, l'automatisation des tactiques de performance touchant au parallélisme aiderait beaucoup d'architectes à tirer le plein potentiel de leur système.

Plusieurs points pourraient être améliorés au niveau du cadre de raisonnement de performance inclus avec ArchE. Notamment, une validation de la présence de boucles dans les liens « réaction » permettrait d'éviter plusieurs problèmes lors de l'analyse et de la recherche de tactiques. De plus, il pourrait être intéressant de remplacer l'analyse effectuée. En effet, l'utilisation d'un outil (Lambda-WBA) qui en utilise un autre (MAST) complique la tâche de compréhension et limite certaines fonctionnalités. Deux options pourraient être poursuivies : utiliser directement l'outil MAST en utilisant des fichiers XML ou codifier la Technique 6 de RMA (Klein *et al.*, 1993) directement dans le cadre de raisonnement de performance, plutôt que d'utiliser MAST. La première solution (utilisation directe de MAST) nous semble plus intéressante, car MAST offre des fonctionnalités très avancées, par exemple, la détermination automatique des priorités de l'ensemble des responsabilités pour obtenir une solution optimale. De plus, l'utilisation directe de MAST permettrait de traiter des tâches plus générales qui surviennent de manière apériodique ou sporadique.

Évidemment, il serait intéressant de développer de nouveaux cadres de raisonnement complets pour qu'ArchE réponde aux besoins de plus d'utilisateurs. Le cadre de

raisonnement de performance utilise un type d'analyse portant sur un système général. Il serait possible de développer d'autres cadres de raisonnement avec des techniques d'analyse différentes pour les différents domaines. Il serait intéressant de développer un cadre de raisonnement analysant la disponibilité d'un système, car il s'agit d'un aspect important pour bien des systèmes. Par exemple, un cadre de ce type pourrait vérifier le temps de réparation d'un système, vérifier si des composantes de sécurité sont présentes, calculer la probabilité de défaillance du système ou effectuer une combinaison de ces analyses.

Pour terminer, ArchE pourrait lui-même être amélioré de diverses façons. Lors des tests, il s'est avéré évident qu'il y avait un problème de performance lors de l'analyse des suggestions des tactiques. Par exemple, lors des tests de système, l'analyse du système se faisait en environ trois secondes, alors que l'analyse de toutes les suggestions retournées prenait près de 30 secondes. De plus, ces analyses sont refaites à chaque modification, alors il devient très long de faire plusieurs petits ajustements. Une refonte du cœur d'ArchE pourrait s'avérer nécessaire afin d'améliorer sa performance, possiblement en lui ajoutant un certain degré de parallélisme dans l'analyse des résultats.

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

Bachmann, Felix, Len Bass et Mark Klein. 2003a. *Deriving Architectural Tactics: A Step Toward Methodical Architectural Design*. CMU/SEI-2003-TR-004. Pittsburgh: Carnegie Mellon University, 68 p. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr004.html>.

Bachmann, Felix, Len Bass et Mark Klein. 2003b. *Preliminary Design of ArchE: A Software Architecture Design Assistant*. En Ligne. CMU/SEI-2003-TR-021. Pittsburgh, PA: Carnegie Mellon University, 66 p. <http://www.sei.cmu.edu/publications/documents/03.reports/03tr021.html>. Consulté le 2008-11-03.

Bachmann, Felix, Len Bass, Mark Klein et C. Shelton. 2005. « Designing software architectures to achieve quality attribute requirements ». *Software, IEE Proceedings,* vol. 152, n^{o} 4, p. 153-165.

Bass, Len, Paul Clements et Rick Kazman. 2003. *Software Architecture in Practice*, Second Edition. Coll. « SEI Series in Software Engineering ». Addison-Wesley, 560 p.

Bass, Len, James Ivers, Mark Klein et Paulo Merson. 2005. *Reasoning Frameworks*. En Ligne. CMU/SEI-2005-TR-007. Pittsburg, PA: Carnegie Mellon University, 40 p. <http://www.sei.cmu.edu/publications/documents/05.reports/05tr007.html>. Consulté le 2008-11-03.

Chung, L, BA Nixon et E Yu. 2000. *Non-functional requirements in software engineering*. Kluwer Academic.

Diaz-Pace, Andres, Hyunwoo Kim, Len Bass, Phil Bianco et Felix Bachmann. 2008. « Integrating Quality-Attribute Reasoning Frameworks in the ArchE Design Assistant ». In *Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures*. Karlsruhe, Germany: Springer-Verlag.

Diaz-Pace, Andres, Hyunwoo Kim et Philip Bianco. 2008. *The ArchE Reasoning Framework Interface Developer's Guide*. Coll. « Software Architecture Technology Initiative ». Carnegie Mellon University, 88 p.

Gonzalez Harbour, M., J. J. Gutierrez Garcia, J. C. Palencia Gutierrez et J. M. Drake Moyano. 2001. « MAST: Modeling and analysis suite for real time applications ». In., p. 125-134. Coll. « Proceedings 13th Euromicro Conference on Real-Time Systems ». Los Alamitos, CA, USA: IEEE Comput. Soc. <http://dx.doi.org/10.1109/EMRTS.2001.934015>.

Klein, Mark, Thomas Ralya, Bill Pollak, Ray Obenza et Michael Gonzalez Harbour. 1993. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Massachusetts: Springer, 712 p.

Lee, Jinhee, et Len Bass. 2005. *Elements of a Usability Reasoning Framework*. CMU/SEI-2005-TN-030 Pittsburg, PA: Carnegie Mellon University, 68 p. <http://www.sei.cmu.edu/publications/documents/05.reports/05tn030.html>. Consulté le 2008-11-03.

Moreno, Gabriel A., et Jeffrey Hansen. 2009. *Overview of the Lambda-* Performance Reasoning Frameworks*. CMU/SEI-2008-TR-020. Pittsburgh: Software Engineering Institute, 66 p. <http://www.sei.cmu.edu/library/abstracts/reports/08tr020.cfm>. Consulté le 2009-09-02.

Moreno, Gabriel A., et Paulo Merson. 2008. « Model-Driven Performance Analysis ». In *Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures*. Karlsruhe, Germany: Springer-Verlag. <http://www.sei.cmu.edu/pacc/moreno-QoSA08-s.pdf>.

Sharma, Vibhu Saujanya, et Kishor S. Trivedi. 2007. « Quantifying software performance, reliability and security: An architecture-based approach ». *Journal of Systems and Software,* vol. 80, n° 4, p. 493-509.

Wirfs-Brock, Rebecca, et Alan McKean. 2003. *Object Design: Roles, Responsibilities, and Collaborations*. Boston (MA): Addison-Wesley, 416 p.

Wojcik, Rob, Felix Bachmann, Len Bass, Paul Clements, Paulo Merson, Robert Nord et Bill Wood. 2006. *Attribute-Driven Design (ADD), Version 2.0*. CMU/SEI-2006-TR-023. Pittsburg: Software Engineering Institute, 55 p. <http://www.sei.cmu.edu/publications/documents/06.reports/06tr023.html>.