UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

MÉCANISMES DÉCISIONNELS PRÉSERVANT LA PERFORMANCE DES SYSTÈMES DISTRIBUÉS ADAPTATIFS

MÉMOIRE DE MAÎTRISE

PRÉSENTÉ PAR

VINCENT TALBOT

JANVIER 2011

UNIVERSITÉ DU QUÉBEC EN OUTAOUAIS

Département d'informatique et d'ingénierie

Mémoire :

MÉCANISMES DÉCISIONNELS PRÉSERVANT LA PERFORMANCE DES SYSTÈMES DISTRIBUÉS ADAPTATIFS

Présenté par Vincent Talbot

Pour l'obtention du maître ès science (M.Sc.)

Évalué par un jury composé de :

Dr. Ilham Benyahia…….…… Directrice de recherche
Dr. Luigi Logrippo…….…….Président du jury
Dr. Andrzej Pelc……..….Membre du jury

Remerciements

Merci à la professeure Ilham Benyahia pour son soutien indéfectible tout au long de ce projet.

Merci aux professeurs Luigi Logrippo et Andrzej Pelc pour leur implication en tant que président et membre du jury.

Merci à mon épouse Lyse.

Table des matières

Listes des figures

Résumé

L'évolution rapide et constante des réseaux étendus (*wide area network* en anglais) a favorisé le déploiement de systèmes distribués que l'on retrouve désormais dans une multitude de domaines civils, industriels et militaires. La gestion de ces systèmes complexes doit garantir une qualité de service (QdS) définie selon divers critères reflétant les exigences de leur domaine d'application. Or, les comportements souvent non stables et non complètement connus de l'environnement de ces systèmes rendent parfois difficile le maintien de leur QdS. L'adaptation d'un système distribué à son environnement est une approche efficace lorsque cet environnement présente une certaine régularité de comportement. Par contre, lorsque l'environnement est sujet à de fortes perturbations, les adaptations peuvent être déstabilisantes et contribuer ainsi à la dégradation de la QdS plutôt qu'à son maintien. Nous développons dans ce projet une méthodologie pour contrer ce problème en intégrant au sein du processus adaptatif un mécanisme décisionnel multi-critères employant une approche par calcul évolutionnaire basée sur "l'algorithme évolutionnaire de Pareto adapté par une pénalité" (PEAP). Cette approche optimise dans un temps borné l'architecture de traitement d'un système distribué en fonction du nouveau comportement de son environnement. La décision de gestion d'un réseau ultimement produite par cette architecture est soumise à des règles de tolérance qui déterminent s'il faut agir sur le système distribué ou simplement laisser passer la perturbation afin de ne pas le déstabiliser davantage. Une étude de cas dans le domaine des transports intelligents nous permet de confirmer les apports du mécanisme décisionnel multi-critères avant la reconfiguration dynamique.

Abstract

The last few decades have seen a phenomenal evolution in network computing which has favored the deployment of a multitude of distributed systems in the civil, industrial and military domains. The real-time management of complex systems such as distributed applications has to cope with unforeseeable events from their environment and incomplete data about these same environments while guaranteeing a Quality of Service (QoS) defined by multiple criteria reflecting distributed system needs. The adaptation of a distributed system to its environment is efficient when this environment is reasonnably stable. However if this environment is subject to intense and frequent perturbations the adaptation may contribute to further destabalization of the system instead of improving the QoS. To enable such applications to adapt to their changing environments we define a methodology for dynamic architecture reconfiguration based on multi-criteria decision making (MCDM) using evolutionary computing (EC). We use in this project the *Pareto Evolutionary Algorithm adapting the Penalty* (PEAP), a category of EC selected to deal with time-consuming online processing required by basic EC such as genetic algorithms. Our methodology optimizes within a real-time deadline the processing architecture of a distributed application in accordance with its new environment. We also address the problem of destabilization which can result from repeated reconfigurations in response to rapid environment changes. A final set of tolerance rules determines wether is is better to apply the new configuration or simply leave the distributed system as is until its environment returns to normal. Our results relating to our road safety case study highlight the benefits of MCDM prior to such reconfiguration.

Chapitre 1 Introduction

1.1 Contexte

L'évolution rapide et constante des réseaux étendus (*wide area networks* en anglais) au cours des dernières décennies a favorisé le déploiement de systèmes distribués que l'on retrouve maintenant dans une multitude de domaines civils, industriels et militaires. Pensons par exemple aux réseaux de téléphonie mobile, de télécommunication sans fil ou de distribution d'électricité, aux systèmes de transport intelligents, ou encore de suivi et de contrôle de missions militaires. Pour certains systèmes distribués tels ceux déployés sur Internet, le niveau de performance du service fourni est généralement garanti selon la politique du "meilleur effort possible". Cependant pour la plupart des systèmes distribués, et en particulier les systèmes distribués en temps réel, cette politique est inadéquate. De tels systèmes doivent plutôt satisfaire des critères de qualité de service reflétant les exigences de leur domaine d'application, telles des contraintes de temps réponse ou de précision des résultats. On désigne par le terme *qualité de service* (QdS) [51] la notion de contraintes imposées par un client (usager ou composant logiciel) sur le comportement des services ou sur la qualité des données qu'un système fournit à une application.

L'un des principaux risques encourus par les systèmes distribués réside dans la dégradation possible de la QdS à cause des comportements inconnus et des perturbations imprévues de leurs environnements. La tâche qui consiste à maintenir la QdS incombe en majeure partie au mécanisme de survivabilité d'un système. La survivabilité des systèmes distribués a fait l'objet de nombreux travaux de recherche qui ont produit des techniques ou des stratégies efficaces de nature statique ou dynamique. Les stratégies statiques telles la redondance des ressources matérielles et la réplication des données sont des stratégies éprouvées mais malheureusement dispendieuses à mettre en œuvre [22] et donnent généralement lieu à un gaspillage important de ressources [51]. Les stratégies d'adaptation dynamique quant à elles sont conçues pour répondre aux fluctuations de l'environnement qui affectent la QdS.

L'adaptation dynamique peut prendre diverses formes telles la reconstruction dynamique des tables de routage [36a] dans le cas des réseaux de télécommunication, la reconfiguration des ressources [11][32] pour les systèmes en temps réel, la reconfiguration des composants de l'architecture ou du cycle de traitement [41], la sélection en ligne d'algorithmes ou de protocoles mieux adaptés au contexte [15a], le déplacement du lieu d'exécution d'un composant logiciel du système ou encore la mobilité du code [8a][23][27], l'équilibre de la charge (*load balancing*) [36a], la renégociation de la bande passante des flux de données [2], etc.. Notre étude s'intéresse pour sa part aux approches adaptatives par la reconfiguration de l'architecture des systèmes à base de composants. Les problèmes découlant de ces reconfigurations seront maintenant introduits.

1.2 Notion de perturbation

Les systèmes distribués interagissant avec un environnement sont sujets à une grande variabilité du niveau d'activité de cet environnement. De tels systèmes opèrent souvent dans des environnements fortement dynamiques subissant des perturbations fréquentes ou continues. Ces perturbations ne sont pas forcément dues à des comportements anormaux ou à des événements indésirables. Par exemple, dans le cas d'un système de téléphonie mobile, les transferts (*hand-offs* en anglais) dûs aux mobiles qui se déplacent de cellule en cellule, rapidement et en grand nombre, causent souvent de telles perturbations [57]. Dans un second exemple de système radar, les perturbations résultent de nombreuses cibles qui apparaissent subitement dans le champ de surveillance. Dans un troisième cas de réseau local sans fil, les perturbations sont provoquées par des facteurs externes comme de l'interférence ou le désalignement des antennes [37]. Ce troisième exemple montre qu'on retrouve parfois au sein d'une même application des perturbations multiples de nature fort différente et qui nécessiteront des approches d'adaptation tout aussi différentes.

1.3 Impact et limites des stratégies d'adaptation dynamique

L'adaptation dynamique peut redresser dans certains cas la QdS d'une application distribuée qui subit des perturbations. Toutefois l'adaptation dynamique peut parfois engendrer des impacts négatifs dont il faut tenir compte. Par exemple, durant l'adaptation, une partie des ressources de traitement est momentanément consacrée au mécanisme d'adaptation et des délais de traitement supplémentaires sont temporairement induits dans le système. Dans le cas de perturbations fréquentes, une activation répétée du mécanisme d'adaptation risque ainsi d'entraîner une dégradation supplémentaire de la QdS. Si le mécanisme d'adaptation réagit à sa propre dégradation, une telle rétroaction du mécanisme d'adaptation sur lui-même provoquera la déstabilisation ou l'oscillation du système [57]. Une approche d'adaptation dynamique doit considérer ces limitations sans quoi la situation que l'on cherche à corriger se détériorera plutôt que de s'améliorer.

1.4 Problème considéré

Le problème considéré dans ce projet de recherche consiste à contrer, au moyen d'un processus d'adaptation dynamique, la dégradation de la QdS provoquée par des perturbations d'intensité et de durée variables. Plus particulièrement, nous voulons prendre en considération le fait qu'une suite de reconfigurations peut elle-même être dans certains cas une source supplémentaire de déstabilisation menant à un accroissement de la dégradation de QdS plutôt qu'au redressement de la QdS. Pour situer ce problème, considérons le modèle d'un système distribué simple tel que présenté à la figure 1.1.

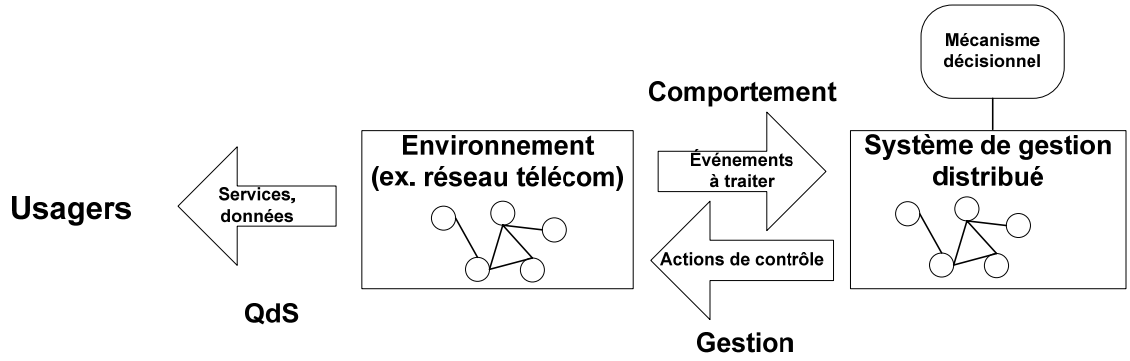

Figure 1.1 Exemple de système distribué

Ce système distribué est en charge de la gestion d'un réseau télécom fournissant des services de téléphonie et de transmission de données à une clientèle d'usagers qui souhaitent bénéficier d'un niveau convenu de service. Le système de gestion reçoit continuellement du réseau télécom des événements à traiter (ex. alarmes de panne et de congestion des circuits, indicateurs d'état, données d'utilisation du réseau, etc.). Le système de gestion agit pour sa part en sens inverse sur le réseau télécom en lui transmettant des actions de contrôle et de reconfiguration (ex. réallocation de ressources, reroutage de circuits).

Dans le cas illustré, le réseau télécom est assimilé à l'environnement externe sous supervision. Ce réseau télécom est sujet à divers événements internes (ex. pannes) ou externes (fluctuations de la demande des usagers) qui viendront modifier le comportement qu'il présente au système de gestion.

Le système de gestion est constitué d'un ensemble distribué de nœuds ou d'agents. Grâce à ces composants, le système dispose des ressources (unités de traitement, mémoire, bande passante, senseurs, activateurs, etc.) ou des services (protocoles de communication, de sécurité, d'authentification, etc.) pour superviser et gérer dynamiquement le réseau télécom. Façe à un changement de comportement de l'environnement qui pourrait affecter les services télécoms, le système de gestion pourra dans un premier temps se reconfigurer pour optimiser sa performance selon le nouveau comportement. Dans un exemple simpliste, si l'environnement, suite à des pannes, est sujet à des problèmes de congestion, on pourrait constater une reconfiguration de l'architecture de traitement du

système de gestion optimisant les composants de reroutage adapté à la congestion au détriment d'une autre type de reroutage adapté aux pannes.

1.5 Objectif de recherche

L'objectif de notre projet est de concevoir un mécanisme décisionnel qui guidera le processus d'adaptation d'un système distribué ayant des contraintes de qualité de service et interagissant avec un environnement qui est sujet à des variations. En outre nous recherchons une approche suffisamment générique pour être indépendante d'un domaine d'application.

Ce mémoire est organisé comme suit. Nous présentons dans le chapitre 2 une revue de littérature de thèmes sous-jacents à l'adaptation dynamique tels les techniques de reconfiguration et les techniques de décisions intégrées. Nous discutons dans le chapitre 3 de l'approche décisionnelle multicritères et des techniques de décision applicables dans notre cas. Une méthodologie de développement de notre cadre et une étude de cas pour sa mise en œuvre dans le domaine des transports intelligents sont présentées dans le chapitre 4. Finalement le chapitre 5 présente une conclusion et suggère des travaux futurs.

Chapitre 2 État de l'art

2.1 Introduction

Les systèmes distribués sont sujets aux perturbations de leur environnement qui peut engendrer une dégradation de leur qualité de service (QdS). Afin de contrer cette dégradation, des approches basées sur l'adaptation dynamique ont fait l'objet de plusieurs études qui ont apporté de nouvelles contributions. Plusieurs des concepts de l'adaptation dynamique découlent de la recherche sur la survivabilité des réseaux dans le domaine des télécommunications. Déjà en 1993, le *Working Group on Network Survivability Performance (T1A1.2)* répertoriait un ensemble de techniques et de stratégies de reconfiguration agissant sur les diverses couches d'un réseau [61]. Depuis, la recherche sur l'adaptation dynamique s'est élargie aux systèmes distribués, notamment les systèmes distribués en temps réel, et propose un ensemble diversifié d'approches telles la renégociation des requêtes clients, la réaffectation des tâches parmi les nœuds de traitement ou encore la reconfiguration dynamique de l'architecture du système.

2.2 L'adaptation par la reconfiguration dynamique

Divers travaux de recherche considèrent la reconfiguration dynamique et la gestion des ressources à des fins d'optimisation de la performance comme étant une solution aux conséquences des changements de l'environnement affectant la QdS des systèmes en temps réel. Il a été toutefois démontré que la performance des composants de gestion [3] et des composants de communication [4] changent selon les patrons d'événements de l'environnement. De même, dans les expériences de [8], la performance d'heuristiques et de métaheuristiques de routage dynamique diffère selon la stochasticité des requêtes de traitement. La performance d'un système distribué varie donc selon le comportement de son environnement,. La corrélation des événements fait que chaque changement de comportement d'environnement diffère des autres changements. Dans ces conditions, le retour à un comportement stable ne se fera pas nécessairement dans un délai prescrit et il s'avère donc difficile de garantir le maintien de la QdS. Pour toutes ces raisons, la reconfiguration d'un système n'est pas toujours désirable et il faut donc envisager d'avoir

recours à un processus décisionnel avant de procéder ou non à la reconfiguration dynamique d'un système distribué [5]. Nous examinons dans les sections suivantes les principes et les techniques d'adaptation qu'il faut considérer pour supporter un tel processus.

2.2.1 Aperçu sur les approches adaptatives

Pour Hiltunen et Schlichting [25], "un système adaptatif est un système qui modifie son comportement en fonction des changements de son environnement". Selon Kramer et Magee[31], "la reconfiguration dynamique est la modification ou l'extension d'un système distribué durant son exécution". La reconfiguration se fait en ajoutant, en remplaçant ou en retirant un ou plusieurs composants qui contribuent au traitement effectué par le système. Le déplacement d'un composant (ex. agent mobile) vers un autre nœud de traitement [2] ou la réaffectation de tâches parmi les processeurs d'une architecture parallèle [49] en sont des variantes.

Porcarelli et al. [46] distinguent deux classes de reconfiguration : la première agit au niveau des composants individuels d'un système et change les paramètres d'utilisation de ces composants; la seconde agit au niveau de l'architecture d'un système et modifie sa topologie en remplaçant ou en relocalisant les composants. Sullivan et al. [53] positionnent la reconfigurabilité d'un système distribué selon trois dimensions : 1) selon la localisation des composants logiciels, 2) selon la conception d'implantation des composants (ex. paramétrage) et 3) selon les interconnexions entre les composants. La reconfiguration basée sur les capacités d'interconnexion dynamique fait appel à la flexibilité de plates-formes intergicielles telles Corba, DCOM ou RMI.

L'adaptation dynamique opère selon un algorithme général qui comporte les étapes suivantes: surveillance du système, détection d'une condition d'adaptation, notification du mécanisme décisionnel, identification du contexte (état du système), sélection d'une alternative de reconfiguration, mise en œuvre de l'alternative et retour éventuel à la normale [61]. Au sein de cet algorithme adaptatif de haut niveau, la reconfiguration dynamique pose plusieurs problèmes intéressants tels les coûts associés à la

reconfiguration et leur impact sur la QdS, la préservation de la cohérence de l'état du système suite à la reconfiguration, le blocage partiel ou complet des traitements pendant la reconfiguration, et le bornage de la durée de traitement des processus associés à la reconfiguration dynamique. La section suivante passe en revue ces problèmes.

2.2.2 Impacts de l'adaptation dynamique

Une contrainte importante de la reconfiguration dynamique est de préserver la cohérence de l'état global du système reconfiguré avant et après la reconfiguration. Ceci donne lieu à des problèmes de suspension de traitement et de transfert d'état qui engendrent des temps de reconfiguration non négligeables. Le temps d'exécution est critique dans un contexte d'application en temps réel : il doit être à la fois borné et prévisible [50]. Ce temps d'exécution est un paramètre qu'il faut donc contrôler afin de minimiser l'impact de la reconfiguration sur la QdS du système.

Plusieurs chercheurs s'intéressent à la performance des mécanismes d'adaptation et à leur impact sur la QdS du système adaptatif. Hillman et Warren [24] mesurent les coûts associés à la reconfiguration dynamique en termes de délai pour effectuer une reconfiguration, et de degré de perturbation[1] que la reconfiguration impose sur les propriétés non fonctionnelles du système (ex. disponibilité, temps-réponse, débit). Leur approche emploie un banc d'essai pour tester et mesurer ces coûts par simulation durant la phase de conception. Le mécanisme que l'on désire tester doit être adapté selon une interface bien définie qui permet son intégration au banc d'essai. L'approche est fondée sur la capacité d'introspection structurelle et comportementale de certaines architectures logicielles et de leurs composants. Cette capacité permet aux composants et à l'architecture elle-même d'être informés dynamiquement sur la configuration et les activités du système. Des composants nommés *intercepteurs* s'interfacent aux connecteurs des composants pour épier la communication entre les composants distribués et pour effectuer une surveillance et une comptabilité des coûts de perturbation. L'approche proposée facilite l'analyse et la sélection de mécanismes d'adaptation. En

[1] Perturbation fonctionnelle du traitement, due à la reconfiguration par le système, à ne pas confondre avec les perturbations de l'environnement.

travaillant au niveau des connecteurs, elle limite aussi l'impact des intercepteurs sur la performance des composants. Toutefois l'approche n'aborde pas le problème de production, d'évaluation et de sélection des alternatives de reconfiguration qui est sujet à une explosion combinatoire lorsque le nombre de composants augmente. Or, un mécanisme d'adaptation ne pourra être efficace si le mécanisme de sélection de la nouvelle architecture dont il dépend est lui-même inefficace.

Rosu et al. [48] évaluent l'impact d'un mécanisme d'adaptation dynamique sur la performance d'applications en temps réel distribuées et réactives. Leur modèle considère explicitement la latence du mécanisme d'adaptation lorsque le processus de restauration de la QdS est en cours. Ils définissent un ensemble de métriques (*metrics* en anglais) de performance tels le temps de réaction, le temps de récupération et la laxité de la performance. Leur modèle précise pour chaque alternative de configuration les coûts de configuration associés à leur mise en place. Ces coûts sont principalement liés au transfert d'état (démarrage et arrêt) ainsi qu'à l'initialisation des composants. L'évaluation de la performance globale du mécanisme d'adaptation se fait en fonction de critères temporels tels la latence de traitement de bout en bout et les temps de traitement relatifs de chaque composant de traitement. L'approche favorise l'obtention d'une performance satisfaisante (*satisfiability* en anglais) relativement aux échéances de temps réel plutôt que l'atteinte d'une performance optimale relativement à la capacité de traitement du système. Cette stratégie réduit les coûts d'adaptation et améliore la réactivité de l'adaptation. L'approche proposée ne traite pas explicitement le cas des adaptations multiples et rapprochées qui peuvent potentiellement se chevaucher. Il faudrait envisager certains raffinements du modèle, par exemple l'ajout de métriques concernant les états de reconfiguration, si l'on veut considérer son utilisation dans un contexte de fortes perturbations.

Un des impacts possibles de l'adaptation dynamique est l'induction dans le système d'une phase d'instabilité de la QdS suite à l'adaptation. Cette instabilité se présente sous forme d'une oscillation de la QdS durant la phase de recouvrement. Toh et Tsai [57] discutent un tel cas tiré du domaine des réseaux WATM (*Wireless Asynchronous Transfer Mode* en anglais). Les auteurs démontrent que les techniques de la théorie du

contrôle sont viables et efficaces pour maîtriser ce type d'impact. Leur approche est fondée sur une conception statique de boucles de contrôles fermées qui gèrent le mécanisme d'adaptation. Cette technique favorise la stabilité du système, c'est à dire sa capacité à ramener ses paramètres de QdS à un état stable de performance. D'autres auteurs ont aussi considéré les techniques de la théorie du contrôle pour leur approche adaptative [33][34][35]. Les approches de ces travaux sont toutefois basées sur l'adaptation par la renégociation des ressources allouées pour le traitement. Leur applicabilité à l'adaptation par la reconfiguration dynamique d'architectures à base de composants est l'objet de travaux futurs.

Un autre domaine d'étude de la reconfiguration dynamique concerne les applications embarquées. Dan ce contexte, Schneider et al. [50] étudient la reconfiguration dynamique de services. Leurs travaux considèrent l'impact de la reconfiguration selon deux délais : le délai de reconfiguration pour remplacer complètement un service par un autre et le délai de temps mort (*blackout* en anglais) pour effectuer le transfert d'état qui nécessite un blocage du traitement. Considérant que ces deux délais sont inversement proportionnels, l'objectif poursuivi est de trouver un compromis entre les deux. Il s'agit d'un exemple de choix multi-critères que nous abordons également dans notre projet.

Kramer et Magee sont parmi les premiers à s'être intéressés aux problèmes liés à la reconfiguration dynamique, en particulier à celle concernant la perturbation fonctionnelle d'un système [31]. Ils proposent d'en diminuer l'impact par un algorithme qui réduit le nombre de composants impliqués dans la reconfiguration, ce qui diminue le temps global de reconfiguration. Leur approche, conçue à l'origine pour des architectures transactionnelles de type client-serveur, est reprise et raffinée par Wermelinger [60], Rasche et Polze [47] ainsi que Schneider et al. [50] de manière à l'appliquer à un contexte de systèmes en temps réel distribués.

2.3 Approches décisionnelles de la reconfiguration dynamique

Les approches décisionnelles liées à la reconfiguration dynamique sont variées. [29] et [38] recensent dans une liste partielle les systèmes à bases de règles, les méthodes

fondées sur la théorie du contrôle et les méthodes associées à l'intelligence artificielle telles les planificateurs temporels, les analyseurs de tendance, les systèmes experts ou encore les ensembles flous. Plusieurs approches développées reposent sur des modèles statistiques [39][49][62]. [29] note de plus une tendance récente à développer des approches hybrides combinant des approches décisionnelles existantes qui se complémentent, par exemple un modèle statistique couplé à un modèle de la théorie du contrôle.

Les approches décrites ci-haut touchent deux problèmes fondamentaux pour le présent projet soit 1) le problème de modélisation de la structure décisionnelle et 2) le problème de la prise de décision. La premier problème concerne les structures de représentation d'alternatives de reconfiguration dynamique. La notion d'alternative fait ici référence aux multiples possibilités de combinaisons architecturales de composants de traitement qu'un système distribué peut adopter en cours d'exécution relativement à la dynamique de son contexte de fonctionnement. Cette notion est étroitement liée à la structure de représentation d'une architecture distribuée, des composants de traitement et des interconnexions qui relient ces composants [53] pour former une architecture particulière. La modélisation des alternatives doit contenir les informations nécessaires au second problème qui concerne le traitement à effectuer, c'est à dire dans notre cas la prise de décision. Ce traitement consiste essentiellement à faire des choix : choix de la meilleure alternative de reconfiguration dans l'espace d'alternatives propres au contexte courant et choix du moment opportun pour reconfigurer.

2.3.1 Modélisation de la structure de décision

Porcarelli et al. [46] proposent d'optimiser la viabilité d'un système distribué en réaction à la détection d'un ou plusieurs événements critiques pour la survivabilité du système (ex. pannes de composants). Les auteurs modélisent les configurations alternatives à l'aide des réseaux de Pétri stochastiques pour générer des sous-modèles représentant les composants de base du système distribué. Ces sous-modèles sont sauvegardés dans un répertoire et au moment de la reconfiguration, le mécanisme décisionnel consulte le répertoire pour composer dynamiquement des alternatives selon des règles et des

spécifications propres au système. Une fonction de surveillance fournit les entrées dynamiques nécessaires pour instancier l'état des alternatives. Le mécanisme considère l'état du système, l'état des composants et l'état de l'environnement. Dans cette approche, le choix du modèle de représentation du système et des composants découle en partie de l'interdépendance de fiabilité entre les composants. Lorsque l'état de fiabilité d'un composant est influencé ou lié de manière probabiliste à celui des autres composants, la modélisation doit donc représenter toutes les combinaisons composant-état. L'espace de combinaisons est d'autant plus grand qu'il doit aussi considérer des entrées d'état dynamiques provenant de l'environnement. L'approche paraît applicable à première vue à une modélisation des dépendances de performance entre les composants d'une architecture logicielle.

Les travaux de Console et al. [15] concernent le diagnostic des fautes d'un système en temps réel. L'approche propose un type d'arbre décisionnel intégrant une dimension de temps. On associe aux nœuds de l'arbre une étiquette temporelle qui constitue une condition supplémentaire pour la prise de décision du nœud. Dans une situation où l'information disponible est incertaine ou incomplète l'approche suggère de retarder la décision jusqu'à ce que le contexte se précise et que des événements supplémentaires se produisent ou non à l'intérieur d'un délai prescrit. De tels arbres sont bien adaptés pour prendre des décisions relativement à des contextes incertains qui se précisent avec le temps.

Pomerol et Brezillon étudient la modélisation de connaissances pour la décision et la gestion de résolution de problèmes [45]. Les auteurs posent l'hypothèse que la connaissance du contexte est plus utile pour la prise de décision que les probabilités d'occurrence d'événements. La surveillance d'entrées dynamiques permet de déterminer le contexte réel du système. Toutefois, comme en [15], l'incorporation de ces variables dans un arbre décisionnel peut susciter une explosion combinatoire de l'espace d'alternatives. Pour contrer ce problème, les auteurs présentent une structure appelée "graphe contextuel" s'inspirant des arbres de décision mais qui résout le problème de l'explosion combinatoire affectant ces derniers. Un graphe contextuel est constitué de différents chemins représentant chacun une alternative de résolution. L'approche modifie

la structure traditionnelle d'un arbre décisionnel grâce aux concepts de nœud contextuel, de macro-action et de branchement temporel. Un nœud contextuel sélectionne uniquement une branche selon le contexte dynamique qui prévaut : aucune sélection n'est faite sur la base de probabilités. Une macro-action encapsule des sous-séquences répétées d'actions et permet de condenser le modèle. Plusieurs branches peuvent mener à une macro-action commune, ce qui transforme effectivement l'arbre en graphe. Le concept de branchement temporel incorpore à la structure du graphe contextuel la notion de parallélisme de certaines sous-décisions indépendantes. L'approche nécessite d'énumérer exhaustivement chaque contexte d'exécution possible. Ce prétraitement peut être complexe et la génération d'un graphe contextuel exige une connaissance approfondie du domaine applicatif.

Castaldi et al. [12] étudient le problème de sélection d'une configuration pour un système à base de composants garantissant une qualité de service. Le but est d'améliorer la QdS du système lorsque la demande des clients change subitement et que l'on détecte une violation de la QdS garantie. La reconfiguration débute par la génération dynamique des alternatives. L'approche de génération proposée nécessite trois classes d'entrées : un modèle de référence, des variables de performance et des politiques de reconfiguration. Le modèle de référence décrit la topologie courante de l'application. Les variables de performance fournissent les données dynamiques sur l'état du système et de son environnement de déploiement. Les politiques de reconfiguration spécifient les contraintes à respecter pour les configurations valides. Plusieurs modèles de performance stochastiques sont générés conformément aux politiques en vigueur et en respectant la topologie du modèle de référence. Ces modèles sont ensuite initialisés dynamiquement en y insérant les valeurs détectées des variables de performance. Finalement la performance de chaque modèle est comparée à celle des autres modèles et le modèle le plus performant est choisi. L'approche se limite toutefois à des scénarios de reconfiguration qui ne modifient pas la fonctionnalité du système.

La prise de décision utilisée dans [62] concerne la réallocation dynamique des ressources afin de rééquilibrer la charge et d'optimiser dynamiquement la gestion des ressources. Elle est conçue dans le but de trouver le meilleur compromis entre la rapidité de la

décision et la qualité de celle-ci. L'obtention à intervalle régulier des événements de détection et de la valeur de latence permet de construire les deux modèles probabilistes. Le premier, essentiellement réactif, est un modèle probabiliste bayésien qui réagit à court terme aux évènements de détection. Le second modèle, essentiellement prédictif, est un modèle probabiliste markovien qui opère sur des intervalles de temps plus longs (plusieurs fois la durée inter événement) et surveille les transitions d'états de la latence du système, afin de prédire la dégradation globale de la QdS du système. Ce second modèle agit comme chien de garde du premier modèle décisionnel en détectant des situations de réallocation que le premier modèle ne peut détecter à court terme.

Le problème de modélisation décisionnelle que nous décrivons cherche à représenter l'information pertinente à la prise de décision d'une manière efficiente (c.a.d. compacte, ouverte, flexible), tout en permettant l'inclusion d'entrées dynamiques pour obtenir le modèle le plus complet possible. Les contraintes du traitement décisionnel pour leur part sont de nature différente et nous regardons dans la section 2.3.2 comment le mécanisme décisionnel traite le modèle décisionnel afin de produire une décision correcte relativement à la QdS tout en satisfaisant les contraintes de temps réel.

2.3.2 Traitements décisionnels

Dans l'approche de Castaldi et al. [12], le traitement décisionnel implique un solutionneur (*solver* en anglais) analytique qui évalue les modèles stochastiques décrits en 2.3.1 et identifie en ligne l'alternative offrant la meilleure possibilité de gain de performance. Les coûts de mise en place des alternatives sont pris en considération dans le pointage de chaque modèle. La reconfiguration est activée par la détection simple d'une violation de la QdS et une fois engagé, le cycle de reconfiguration doit se compléter. La période entre les événements de violations de la QdS doit être supérieure à la période du cycle de reconfiguration. Cette limitation de l'approche est incompatible avec un contexte de perturbations intenses de l'environnement.

Dans l'approche de Porcarelli et al. [46], le traitement décisionnel procède à l'évaluation analytique des alternatives et sélectionne la meilleure alternative selon une fonction de

rétribution qui considère d'une part les gains de fiabilité réalisables et d'autre part les coûts à encourir tels le temps de prise de décision et les coûts de reconfiguration. Un des enjeux de l'approche est la rapidité avec laquelle la modélisation des alternatives (sous formes de réseaux de Petri) et la prise de décision dynamique peuvent être complétées. Cette approche emploie en ligne un outil automatisé nommé DEEM (*DEpendability Evaluation of Multiple-phased systems* en anglais) pour l'évaluation des réseaux de Pétri stochastiques, et un processus de Markov pour la solution analytique. Ces outils se prêtent bien aux problèmes décisionnels en temps réel car ils produisent des décisions dans un temps prévisible et des solutions qui sont précises.

L'algorithme présenté en [15] est capable de synthétiser un arbre temporel à partir d'une base d'observations fournissant des données concrètes de comportement et des actions résultantes. La technique des auteurs s'inspire des schémas de compilation. L'algorithme de génération de l'arbre cherche à détecter des patrons de comportement et à associer ces patrons à des actions. Il faut au préalable modéliser le système qui intègre des variables de performance et des modes d'utilisation, puis établir la relation entre domaines de valeur de variables et modes d'opération. Le modèle sera utilisé pour simuler le comportement du système et compiler l'arbre décisionnel qui servira par la suite de modèle comparatif pour prédire à l'aide des observations dynamiques le comportement futur du système.

Dans le système d'assistance à la prise de décision présenté dans [45], les contraintes de traitement sont dictées par la précision des résultats plutôt que par la rapidité du traitement, tel que cela est le cas dans les approches déjà mentionnées. La logique temporelle qui contrôle le cheminement dans le graphe contextuel recherche l'atteinte d'un niveau de certitude suffisant pour reconnaître un contexte spécifique et proposer les actions associées à ce contexte. Tout au long de l'exécution du système, la reconnaissance du contexte nécessite l'obtention d'entrées dynamiques qui seront obtenues par l'entremise d'une fonction de surveillance. L'identification du contexte se fait de manière déterministe et non pas probabiliste.

La prise de décision en [62] se déroule en deux étapes. Le mécanisme décisionnel analyse d'abord les données sur l'état du système puis détermine s'il y a eu dégradation ou s'il y a tendance à la dégradation de la QdS à moyen terme. Dans le cas d'une dégradation, le mécanisme décisionnel évalue les coûts de réallocation des ressources et détermine si un gain de performance peut être réalisé. Les décisions sont prises rapidement sur la base d'informations dynamiques obtenues et compilées en mode continu. L'approche bayésienne accorde une importance centrale à la diminution du temps de réaction décisionnelle qu'elle équilibre avec la recherche de qualité de la décision produite. L'approche markovienne permet de palier l'incertitude du comportement d'un environnement partiellement connu en partie grâce à sa capacité de compiler en continu des probabilités de transition d'états.

2.3.3 Analyse des approches décisionnelles existantes

Dans un contexte de systèmes en temps réel, tous les aspects du mécanisme d'adaptation doivent être à la fois rapides et bornés dans le temps. Ceci inclut les mécanismes de décision et reconfiguration. Une exception admissible concerne les systèmes en temps réel souples pour lesquels un dépassement d'échéance n'est pas catastrophique. Les approches probabilistes basées sur les réseaux bayésiens, les chaînes de Markov [62], les réseaux de Pétri stochastiques [46], ou les modèles de performance stochastiques [12] sont de nature analytique et peuvent satisfaire aisément ces deux exigences temporelles. Les arbres décisionnels, lorsqu'ils sont compacts, peuvent compléter leur décision dans un temps court et borné. Cette durée est une fonction de la profondeur de l'arborescence qui elle-même est liée aux nombres de variables conditionnelles statiques et dynamiques. Le nombre d'entrées impliquées dans une décision devra donc être restreint afin de maintenir le temps décisionnel très court. Pour leur part, les arbres décisionnels temporels [15] utilisent le temps pour différencier des patrons contextuels qui autrement seraient identiques. Comme dans ce type d'arbre le traitement décisionnel est un cheminement continu, parallèle à l'exécution du système, la notion de temps de décision ne s'applique pas. Il en va de même pour les graphes contextuels [45] qui contiennent des

branchements temporels. Les graphes contextuels sans branchement temporel sont compacts et peuvent produire une décision rapidement.

De manière générale, chacune des approches des sections en 2.3 offre un traitement performant et adéquat selon les critères présentés. La difficulté réside plutôt dans la génération des modèles. La qualité des modèles dépend de la disponibilité et de la qualité des données historiques disponibles utilisées pour générer les alternatives. Ces exigences ne sont pas satisfaites aisément. En outre, dans le cas d'un système opérant dans un environnement incertain, il nous faut considérer que les alternatives de configuration et les conditions d'exécution ne sont pas pleinement connues au moment de l'exécution. Porcarelli et al. soulignent ce problème dans [46], de même que Pomerol dans [44] lorsqu'il mentionne les *alternatives pleinement extensionnées* c'est à dire des alternatives tenant compte de toutes les possibilités dynamiques d'états du système et de l'environnement. Pour les systèmes qui nous intéressent, l'espace des alternatives pleinement extensionnées est très vaste et peut difficilement être anticipé exhaustivement hors ligne.

L'approche de génération et d'initialisation en ligne en [12] améliore la qualité des modèles. Comme les politiques de reconfiguration sont en nombre fini, les temps de génération et d'initialisation demeurent finis. Toutefois cette manière de procéder en ligne alourdit le temps de reconfiguration de manière significative et compromet l'applicabilité de l'approche au domaine du temps réel.

Dans le cas des graphes contextuels, c'est plutôt l'incertitude du contexte qui pose la difficulté puisque par définition, cette approche recherche des contextes complètement connus. Toutefois, en assouplissant cette exigence, on peut envisager un graphe contextuel partiellement incertain confinant l'incertitude à une seule variable d'état de l'environnement liée au degré de perturbation. Cette adaptation définirait en quelques sorte un graphe contextuel "flou" et elle est bien adaptée à notre problème d'incertitude. Considérant que toutes les autres entrées décisionnelles respectent le critère de certitude, ce compromis fournirait une technique d'atténuation de l'incertitude et pourrait améliorer la qualité de la décision.

Plusieurs approches [12][15][[44][45][46] proposent d'intégrer dynamiquement des entrées fournissant l'état du système ou de son environnement de déploiement pour améliorer la qualité du modèle et de la décision. Toutefois peu d'approches intègrent aussi l'état interne du mécanisme de reconfiguration comme entrée dans leur modèle décisionnel. Face à un problème de perturbations fréquentes, la prise en considération de l'état interne du mécanisme de reconfiguration pourrait contribuer à raffiner la ventilation des coûts de reconfiguration, le choix du moment de reconfiguration et par conséquent améliorer le traitement décisionnel.

D'autre part, une fréquence élevée de perturbations peut provoquer un problème de chevauchement de reconfigurations. En général, le temps requis pour atteindre une décision de reconfiguration et la mettre en œuvre doit être beaucoup plus court que le temps moyen d'occurrence d'événements de reconfiguration [46]. Les approches de reconfiguration sont fréquemment contraintes à un seul événement déclencheur dans l'intervalle entre deux reconfigurations [39][46][62]. Cette contrainte est difficile à respecter dans le contexte de notre projet. La prise en considération de l'état du mécanisme de reconfiguration dans le processus décisionnel pourrait toutefois nous être utile pour mitiger la décision de reconfigurer dans ces cas.

Dans plusieurs des approches proposées, la prise de décision n'est guère plus que la simple consultation d'une base de règles statiques dont les règles sont vérifiées systématiquement lorsque le système détecte des variations dans la performance du système ou dans le comportement de l'environnement. Pour notre projet, la prise de décision est plutôt entrevue comme un processus dynamique relativement complexe qui évalue en continu la pertinence de reconfigurer ou non le système et qui doit sélectionner le moment venu la meilleure alternative pour l'architecture de traitement.

2.3.4 Optimisation multi-critères des alternatives de reconfiguration

Les approches de reconfiguration basées sur des modèles hiérarchiques ont souvent recours à une logique temporelle ou de premier ordre afin de générer ou de synthétiser

des solutions. De telles approches sont toutefois mal adaptées à la gestion d'espace de solutions très larges ou encore à une catégorie de solutions visant l'optimisation d'objectifs conflictuels. Le domaine des problèmes d'optimisation multi-critères (MOP) ainsi que sa branche spécialisée des algorithmes évolutifs se basent tout particulièrement sur l'optimisation d'objectifs conflictuels. Il existe de nombreux algorithmes évolutifs dans le domaine de l'optimisation sans contrainte tel par exemple le *Strength Pareto Evolutionary Algorithm* (SPEA) [63] qui exploite la force relative de solutions (appelées individus) appartenant à un espace commun de solutions (appelé population). L'optimisation multi-critères est cependant parfois sujette à des contraintes de performance (qualitative ou temporelle) et à des règles sémantiques (par exemple la validité d'un assemblage de composants). [56] a récemment décrit une technique hybride qui combine les fronts de Pareto (*Pareto-optimal sets* en anglais) à des fonctions de pénalité. Les fonctions de pénalité donnent au processus de reproduction génétique la possibilité d'inclure un certains nombres d'individus imparfaits (sémantiquement invalides) dans une proportion équivalente au degré de pénalité. De tels individus seraient normalement rejetés bien qu'ils possèdent certaines qualités. Leur inclusion peut contribuer au maintien de la diversité génétique des nouvelles générations. Cet avantage peut s'avérer utile dans le cadre de notre projet.

2.4 Modélisation et gestion des alternatives de reconfiguration

2.4.1 Reconfiguration dynamique et architecture à base de composants

Les systèmes ayant des capacités de reconfiguration dynamique sont souvent implantés dans des environnements à base de composants qui se prêtent bien aux techniques présentées dans les sections précédentes et qui facilitent la modélisation, la gestion et la restructuration de leur architecture interne en nécessitant un minimum voire même aucune intervention humaine. Des environnements modernes tels OpenRec [59] et Fractal [9][10] ont recours à des modèles de composants capables d'introspection, d'extensibilité et de composition. Récemment, ces deux environnements ont été dotés de leur propre langage formel de spécification : Alloy dans le cas d'OpenRec et Focal pour ce qui est de Fractal. Alloy et Focal sont des méta-langages employés pour modéliser la configuration d'un système, pour vérifier automatiquement la rectitude sémantique de cette configuration et pour prouver qu'elle satisfait les contraintes fonctionnelles. De telles méthodologies n'arrivent toutefois pas à démontrer si les contraintes de performance temporelle sont satisfaites ou non.

La performance d'une architecture peut être démontrée dès l'étape de la conception en ayant recours à une méthodologie de spécifications de temps réel telle UML-RT [21]. Toutefois la vérification formelle de la performance d'un système s'adaptant dynamiquement aux conditions incertaines de son environnement n'est pas démontrée. Les méthodologies basées sur la simulation sont plus appropriées à cette fin.

La simulation permet de tester la performance des configurations alternatives envisagées pour un système auto-adaptatif. Un environnement tel J-Sim [58] est un exemple d'environnement de simulation approprié pour les systèmes ayant des contraintes de temps réel et qui peuvent intégrer des composants matériels. J-Sim est conçu autour d'une architecture de composants autonomes (ACA) selon un modèle qui s'inspire du domaine des circuits intégrés. Il implante le concept de contrat entre composants de manière à isoler leur implantation interne de leur comportement externe. Cette

caractéristique rend la plate-forme J-Sim apte à la mise en œuvre de plusieurs des techniques exposées dans les sections précédentes. Les propriétés compositionnelles de J-Sim se prêtent bien à la spécification, à l'assemblage dynamique de la configuration d'un système, puis à sa simulation contrôlée et mesurable par des processus externes. Notons que cet environnement est retenu dans le cadre de notre étude.

2.4.2 Gestion des alternatives de reconfiguration

Pour faire face aux changements de son environnement un système adaptatif doit chercher, découvrir ou construire des configurations alternatives puis sélectionner la configuration la plus appropriée au contexte d'exécution courant. L'espace des configurations possibles peut être prédéterminé au moment de l'adaptation. Par exemple, en [16] les solutions alternatives pour un domaine d'application donné sont captés et sauvegardées dans un dépôt propre à ce domaine. Des travaux plus récents cherchent plutôt à gérer dynamiquement l'espace des solutions. Les auteurs de [55] décrivent un système pouvant effectuer une reconfiguration en générant dynamiquement des plans réactifs (c.a.d. des reconfigurations) en fonction d'objectifs exprimés selon une logique temporelle. L'approche emploie un modèle conceptuel à trois couches, soit une couche de gestion des objectifs, une couche de gestion des changements et enfin une couche de composants. Des approches similaires existent dans le domaine de la Radio Cognitive (*Cognitive Radio* en anglais) qui utilise une gestion hiérarchique pour découvrir et pour composer dynamiquement de nouvelles architectures logicielles et matérielles pour des systèmes radio cognitifs [19]. De même [14] décrit un banc d'essai dans le domaine de la radio générique pour la prise de décision autonome en regard d'objectifs opérationnels multiples, possiblement conflictuels sous un contexte variable dans le temps.

Chapitre 3 Décisions basées sur l'approche multi-critères

3.1 Introduction

Le mécanisme décisionnel d'un système adaptatif a pour principal objectif d'optimiser la configuration de son architecture de traitement selon de multiples critères de sélection appliqués à un ensemble d'alternatives de configuration. Les variables décisionnelles à considérer (type d'environnement, intensité des perturbations, charge interne des composants, etc.) peuvent être nombreuses, de même que la taille de la librairie de composants servant à configurer les diverses alternatives. La conception du mécanisme décisionnel multi-critères devra en outre tenir compte du fait que plusieurs des variables impliquées sont dynamiques, ce qui contre-indique un mécanisme décisionnel exploitant uniquement des règles statiques. Idéalement, le mécanisme sera plutôt conçu de manière à combiner aux entrées statiques les entrées obtenues dynamiquement au moment de l'exécution afin de baser la décision finale sur un contexte le plus complet possible. La taille de la librairie combinée à celle du vecteur décisionnel influenceront fortement la performance du mécanisme.

On recherche une approche décisionnelle adaptée aux situations où : 1) l'espace de solutions considéré est sujet à une explosion combinatoire; 2) les critères de sélection travaillent possiblement en opposition (ex. coût versus qualité); 3) la pondération relative des critères ne peut être aisément normalisée. Nous présentons dans les sections qui suivent diverses considérations techniques qui nous permettrons de finaliser un choix sur une approche décisionnelle convenant au projet.

3.2 Discussion sur les approches existantes

3.2.1 Notion de dominance et front de Pareto

Les problèmes d'optimisation multi-critères n'ont pas à priori de solution (d'alternative optimale) unique mais plutôt un ensemble de solutions dites non-dominées. La notion de dominance entre les solutions est définie comme suit :

> Soit un problème d'optimisation qui consiste à minimiser la valeur de chaque critère d'un vecteur de critères Y : on dira que Y *domine* un autre vecteur Y* si la valeur de chaque paramètre de Y n'est pas supérieure au paramètre correspondant de Y* et si il existe au moins un paramètre pour lequel la valeur de Y est inférieure à la valeur correspondante de Y*.

Considérons un problème d'optimisation consistant à minimiser concurremment deux critères de durée et de qualité de traitement et dont les valeurs sont obtenues par les fonctions respectives f1 et f2. Une solution générée par l'algorithme évolutionnaire pour le vecteur 'a' correspond au tuple de valeurs {(f1(a), f2(a)} et est représentée par un point dans un espace de solutions défini par les axes f1(a) et f2(a) (fig. 3.1). L'espace des solutions est partitionné en trois sous-espaces. Les solutions optimales non-dominées délimitent la frontière entre les sous-espaces des solutions faisables et non faisables (invalides). Dans un espace à deux dimensions, cette frontière correspond à une courbe dénommée *Front de Pareto* qui définit une relation d'ordre partiel adéquate pour des paramètres conflictuels et indépendante de toute normalisation des paramètres. Les solutions optimales sont plus ou moins dispersées le long de ce front entre la limite supérieure de f1 et celle de f2. Les solutions situées au dessus de ce front correspondent aux solutions faisables mais toutefois dominées sur tous leurs critères. Ces solutions sont faisables sans être optimales. Les solutions situées au dessous du front de Pareto sont non faisables. Toutefois, selon le type d'algorithme évolutionnaire choisi, on pourra retenir pour des fins de diversité de la reproduction un certain nombre d'individus non faisables situés à proximité du front sous celui-ci.

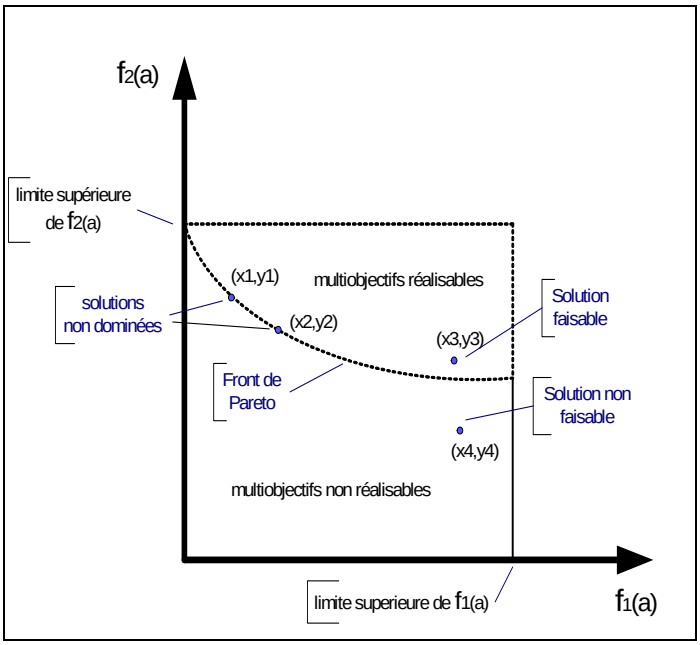

Figure 3.1 Front de Pareto

3.2.2 Méthodes d'optimisation classiques

Diverses méthodes d'optimisation multi-critères permettent la recherche systématique de solutions optimales dans un espace de solutions. [52] souligne que les méthodes dites classiques telles l'agrégation, la mesure de Chebyshev ou la méthode des e-contraintes sont faciles à mettre en œuvre mais comportent toutefois des inconvénients importants tels la lourdeur des calculs (à cause de l'explosion combinatoire), l'obligation de pondérer à priori les variables décisionnelles ou les contraintes (ce qui limite leur applicabilité à des contextes dynamiques) ou encore l'incapacité d'accéder aux solutions qui se trouveraient dans des portions concaves d'un front de Pareto (notamment dans le cas de l'agrégation). Les méthodes évolutionnaires d'optimisation multi-objectifs cherchent à contourner ces inconvénients et surtout à disperser la recherche dans tout l'espace.

On notera que dans le cas des méthodes classiques, la qualité de l'ensemble de solutions obtenues sera tributaire de la qualité des pondérations relatives définies sur l'espace des variables de décision. Dans le cas des méthodes évolutionnaires que nous examinerons sous peu, c'est

plutôt la qualité de la technique de représentation des solutions qui influencera le plus fortement la qualité des solutions obtenues.

3.2.3 Méthodes d'optimisation évolutionnaires

Plusieurs méthodes d'optimisation multi-critères sont basées sur des algorithmes évolutionnaires exploitant la notion de dominance et de front de Pareto. L'algorithme *Strength Pareto Evolutionary Algorithm* (SPEA2) proposé par [63] et l'algorithme *Pareto Evolutionary Algorithm with Penalty* (PEAP) proposé par [56] en sont deux exemples.

L'algorithme *Strength Pareto Evolutionary Algorithm* (SPEA2)

SPEA2 est un algorithme évolutionnaire ayant recours à une définition du degré d'adaptation (*fitness* en anglais) basée sur la force de domination relative d'un individu sur les autres individus. La manière de calculer le degré d'adaptation selon la force de domination relative (*Strength Pareto* en anglais) donne son nom à l'algorithme. L'algorithme emploie un mécanisme d'archivage externe qui permet de préserver de génération en génération une élite d'individus qui se régénère tout en demeurant de taille fixe. La taille de cette élite est maintenue fixe de manière à éliminer les problèmes de ballonnement (*bloathing* en anglais), c.a.d. un accroissement démesuré de la taille de l'élite engendrant une consommation excessive de mémoire interne qui alourdit le traitement. Ce contrôle de la taille est effectué par l'entremise d'un processus d'amalgamation que nous décrirons sous peu.

Les travaux de [63] portant sur le problème classique du sac à dos (*knapsack* en anglais) et ceux de [26] offrent une bonne démonstration de la mise en œuvre de cet algorithme.

L'algorithme *Pareto Evolutionary Algorithm with Penalty* (PEAP)

PEAP est un algorithme évolutionnaire inspiré de l'algorithme SPEA (un prédécesseur de SPEA2) et ayant recours à une définition similaire du degré d'adaptation (c.a.d. employant la notion de force relative de domination) à laquelle on ajoute une valeur de pénalisation relative au degré d'invalidité de l'individu. L'approche permet ainsi de considérer des individus invalides à

proximité sous le front de Pareto lors de l'étape de reproduction. L'algorithme PEAP considère que de tels individus peuvent contribuer à la qualité de la recherche dans l'espace de solutions en favorisant sa dispersion. En contrepartie, l'algorithme diminue la probabilité que ces individus invalides participent au sous-groupe de reproduction en introduisant dans le calcul du degré d'adaptation une pénalité qui augmente rapidement en fonction du nombre de contraintes non satisfaites.

[56] propose un exemple d'utilisation de l'algorithme PEAP pour solutionner un problème d'optimisation multi-objectifs de réseau optique WDM. La représentation chromosomique choisie dans ce cas est une représentation linéaire de bits. Dans l'approche proposée par l'auteur, le front de Pareto est réduit par amalgamation à un ensemble restreint à quelques solutions, puis une décision Markovienne applique une politique prédéterminée de reconfiguration pour effectuer le choix final parmi cet ensemble restreint.

3.3 Discussion sur les techniques d'algorithmique évolutionnaire

3.3.1 La représentation des solutions évolutionnaires

Dans le cas des approches évolutionnaires, le choix de la méthode de représentation (d'encodage) des solutions dans un algorithme évolutionnaire constitue l'une des principales difficultés à surmonter. En effet, les caractéristiques intrinsèques d'un problème d'optimisation doivent être maintenues lors de la transformation d'une solution (d'un individu) vers sa représentation génétique. Cette transformation doit de plus mener à la réduction de l'ampleur des calculs nécessaires pour parvenir à une solution (un optimum) comparativement aux calculs qui seraient normalement effectués pour traiter le problème sous sa représentation originelle [1]. Les types de représentation classiques que l'on retrouve dans les algorithmes évolutionnaires (chromosome linéaire binaire, arborescence, etc.) demeurent couramment employés de nos jours. Chaque type possède des avantages et des inconvénients selon la nature de problème auquel on l'applique.

Plusieurs chercheurs examinent cependant des méthodes visant à mieux exploiter ces représentations classiques, telles la conception d'opérateurs spécifiques au problème considéré [18][30][40][43], ou encore l'intégration de paramètres de domination à même la représentation des individus [17]. Quelle que soit la représentation, il est souvent difficile de concevoir des opérateurs génétiques qui garantiront la production d'une nouvelle génération d'individus tous sémantiquement valides, c'est-à-dire respectant l'ensemble des contraintes sous-jacentes aux critères [13]. Idéalement, l'obtention d'une garantie de validité sémantique des individus issus de la reproduction par croisement et par mutation ne doit pas se faire au détriment de la performance générale de l'algorithme évolutionnaire.

Nous examinons maintenant trois techniques complémentaires permettant d'améliorer la performance des algorithmes évolutionnaires.

3.3.2 L'élitisme

L'élitisme améliore la performance de la convergence vers une solution optimale en ayant recours à la rétention et à la mémorisation explicite des meilleurs individus d'une génération à l'autre [7]. Ceci facilite leur réutilisation dans les cycles de reproduction futurs.

L'élitisme exerce un contrôle sur la taille de l'élite à chaque cycle de reproduction afin d'éviter l'augmentation excessive du temps de traitement d'une génération à l'autre. Par exemple, l'approche de [63] maintient la taille de l'archive constante tout en conservant systématiquement les solutions limites; l'approche de [56] amalgame répétitivement la taille de l'élite jusqu'à ce que cette taille tombe en deça d'une limite prédéterminée.

L'élitisme pourra prendre la forme d'une mémorisation interne et éphémère, et dans ce cas l'algorithme évolutionnaire débutera avec une élite vide à la première génération. L'élitisme pourra d'autre part recourir à une mémorisation externe et durable (par exemple dans un fichier ou une librairie) et dans ce cas l'algorithme évolutionnaire pourra initialiser l'élite de première génération à partir de l'archive.

L'élitisme et la capacité d'archivage d'une élite, ou encore de remplacement d'une élite sont des caractéristiques utiles pour s'adapter à un changement qualitatif du comportement de l'environnement tel que signalé par le module de surveillance (détection exogène du changement). On suppose ici que l'on peut établir par expérimentation une relation entre une élite archivée et une loi de distribution caractérisant une perturbation. Lorsque le module de surveillance détecte une perturbation et identifie la loi de distribution y correspondant, l'élite en vigueur est remplacée par une élite archivée correspondant à cette nouvelle loi.

3.3.3 L'amalgamation

L'amalgamation (*clustering* en anglais) est une technique complémentaire à celle de l'élitisme [56][63]. Il s'agit d'un processus d'épuration de la population employé pour contrer un accroissement excessif de la taille de la mémoire associée à l'élite. La bonne gestion des ressources de traitement est nécessaire pour maintenir la performance de l'algorithme évolutionnaire qui autrement se voit affectée par un alourdissement du traitement.

L'amalgamation exploite la densité des sous-groupes d'individus le long et à proximité du front de Pareto. Elle retient un nombre limité d'individus pour chaque sous-groupe tout en maintenant une dispersion maximale sur tout le front de Pareto. L'amalgamation peut ainsi maintenir la dispersion de l'élite sur le front de Pareto et en ce sens elle favorise la qualité de la recherche d'une solution optimale dans l'espace des alternatives.

3.3.4 La pénalisation

Une manière d'assurer la validité des individus engendrés lors de l'étape de reproduction serait d'avoir recours à des opérateurs spécialisés de croissement et de mutation ne pouvant pas à priori générer des individus invalides. Une autre manière serait d'invoquer à posteriori des opérateurs spécialisés pouvant rectifier un individu mal formé. Toutefois le recours à de telles approches engendre généralement un accroissement important de la complexité et du temps de traitement de la phase de reproduction. D'autre part, l'élimination à priori des individus invalides peut interférer avec le processus exploratoire en réduisant la dispersion et nuire ainsi à la convergence vers une solution globalement optimale.

L'approche de pénalisation que nous examinerons maintenant prône plutôt la tolérance d'individus invalides au sein de la population. Dans ce cas, on affaiblit le degré d'adaptation des individus invalides en introduisant une pénalité qui sera fonction du degré d'invalidité sémantique de l'individu. L'idée maîtresse de l'approche avec pénalité est de permettre aux solutions invalides à proximité de la zone de validité de contribuer à la dispersion de la recherche exploratoire lors du processus de reproduction. L'approche avec pénalité définit une zone dite de quasi-faisabilité (*near feasability* en anglais) dans laquelle on peut considérer que les individus s'y trouvant possèdent des caractéristiques utiles à la reproduction. [13] formule un problème d'optimisation sujet à des contraintes par l'équation 1 :

$$\min z(x) \mid x \in A \wedge x \in B \qquad \text{(éq. 1)}$$

où :

x	un vecteur de variables décisionnelles (ex. coût, qualité, délai)
$z(x)$	une fonction à minimiser sur le vecteur x
A	un ensemble de contraintes sémantiques faciles à satisfaire
B	un ensemble de contraintes sémantiques difficiles à satisfaire

Comme il est plus difficile de concevoir des opérateurs génétiques pouvant satisfaire à la fois les contraintes de A et de B, on relaxera les contraintes de B en les impliquant dans une fonction de pénalité externe. On obtient ainsi l'équation 2.

$$\min z(x) + p(d(x, B)) \mid x \in A \qquad \text{(éq. 2)}$$

où :

$p(y)$	est une fonction de pénalité tel que $p(0) = 0$
$d(x, B)$	est une fonction donnant une mesure de la distance entre le vecteur x et la région définie par les solutions satisfaisant les contraintes de B

[13] proposent trois classes de fonctions de pénalité applicables à l'optimisation par algorithme génétique :

A) Pénalisation booléenne dite d*e mise à mort*

Dans ce cas, la pénalisation d'un individu est soit nulle soit infinie selon que l'individu est valide ou invalide.

$$\forall x \in B, \quad p(x) = 0$$
$$\forall x \notin B, \quad p(x) = \infty$$

Cette approche a pour principal avantage d'être efficace et pour principal défaut de ne permettre aucune zone de quasi-faisabilité, ce qui sera acceptable si le processus de recherche dans l'espace de solution est naturellement bien dispersé.

B) Pénalisation selon le nombre de violations de contraintes

Dans ce cas, la pénalisation d'un individu est proportionnelle au nombre de contraintes violées par cet individu sans égard au degré ou à l'importance relative des violations constatées :

$$c_i \in C(B),\ p(x,B) = \sum_{i=0}^{|C|} c_i(x) \qquad \text{(éq. 3)}$$

où :

C est l'ensemble des contraintes c associées à l'ensemble B

p(*x*,B) est une fonction définie sur l'ensemble des entiers naturels

$c_i(x)$ fonction binaire retournant 0 ou 1 selon que la contrainte i appliquée sur *x* est satisfaite (0) ou non (1)

Cette approche a le mérite d'introduire une zone de quasi-faisabilité sans toutefois permettre de relativiser l'importance des violations les unes par rapport aux autres. La pénalisation d'un individu est proportionnelle au degré de sévérité cumulatif des contraintes violées par cet individu sans égard au degré ou à l'importance relative des violations constatées.

C) Pénalisation selon une mesure de distance par rapport à la zone de faisabilité

Dans ce cas, on définit une graduation de la sévérité des violations de manière à raffiner le dosage de la contribution de la zone de quasi-faisabilité. On raffine l'équation 3 de manière à obtenir une fonction de pénalisation produisant une valeur réelle qui sera associée à une zone de tolérance définie dans la région des solutions invalides selon l'équation 4 :

$$\min z(x) + p(d(x,B)) \mid x \in A \qquad \text{(éq. 4)}$$

où :

- p(y) est une fonction de pénalité tel que p(0) = 0
- d(*x*,B) est une fonction donnant une mesure de la distance entre le vecteur x et la région définie par les solutions satisfaisant les contraintes de l'ensemble B

3.3.5 Conclusion sur les techniques d'algorithmique évolutionnaire

Les algorithmes SPEA2 [63] et PEAP [56] sont de bons candidats pour notre projet du point de vue de leur performance et de leur applicabilité à un mécanisme décisionnel multicritères. Dans les deux cas, la représentation choisie par leurs auteurs est un chromosome de bits. La recherche récente privilégie toutefois une représentation la plus proche possible du problème à modéliser. Comme le domaine applicatif de notre projet repose sur une architecture de traitement à base de composants, une représentation par graphe sera plus adaptée et sera mise en œuvre plus naturellement. Sachant qu'il existe des algorithmes de recouvrement permettant une conversion performante d'une structure en graphe vers une structure arborescente, nous opterons pour une représentation génétique sous forme d'expression arborescente. Nous prévoyons que les opérateurs de croisement et de mutation des individus produiront un certain nombre d'individus invalides. A cet effet, l'algorithme PEAP avec sa fonction de pénalité est à priori mieux adapté pour gérer cet inconvénient que l'algorithme SPEA2 avec sa fonction de force relative des individus.

Relativement au concept d'élitisme, l'élitisme par mémorisation externe et durable est compatible avec la création d'une librairie d'élites qui seront associées à des environnements connus. En effet les environnements connus sont basés sur des données historiques et ne sont

donc pas sujets à changer fortement. Ainsi une élite déjà établie pour un environnement connu pourra servir d'amorce pour la première population de l'algorithme évolutionnaire lors du calcul d'une solution de configuration optimale pour une légère variation de cet environnement. De même, s'il y a mise à jour des composants de traitement de la librairie de composants d'un système adaptatif, une telle mise à jour sera susceptible d'affecter chacune des solutions optimales pré-calculées pour le système adaptatif. Dans un tel cas il faudra donc recalculer chacune des solutions de configuration. Or, pour chaque environnement, on réduira grandement le temps de recalcul en amorçant la population initiale avec l'élite qui lui aura été associée auparavant.

3.4 Algorithme PEAP adapté au mécanisme décisionnel

3.4.1 Conception de la fonction de pénalisation pour PEAP

L'algorithme PEAP exploite un mécanisme de pénalisation que nous devrons ajuster en fonction des contraintes propres aux configurations de notre problème. Ces configurations sont des assemblages ordonnés de composants. Les principales contraintes à contrôler seront les contraintes de redondance (présence multiple d'un même composant), les contraintes d'ordonnancement (règles de précédence parmi les composants) et dans certains cas les contraintes de connectivité (graphes avec boucles, fortement connexes ou avec sous-graphes partiels). Nous choisissons d'employer une méthode de pénalisation proportionnelle au nombre de contraintes violées. Pour ce faire, nous définissons trois contraintes :

- Cr : contrainte interdisant la redondance d'un composant
- Cp : contrainte établissant un ordre de précédence des composants
- Cd : contrainte définissant le degré maximal de connectivité d'un composant

De plus, à chaque contrainte nous associerons des coefficients de pondération qui seront fonction de la sévérité relative d'une contrainte en rapport aux autres contraintes (par exemple un composant multiple serait moins pénalisant qu'un composant excédant son degré maximal de connexion). Partant de l'équation 3 (section 3.3.4), le calcul de la fonction de pénalité sera fourni par l'équation 5.

Soit :

- L un ensemble (une librairie) de composants de dimension |L|
- G l'ensemble des composants *x* présents dans la configuration courante et représentée sous forme de graphe
- *x* un composant de la configuration courante
- C l'ensemble des contraintes = {Cr, Cp, Cd} définies sur G
- $c_i(x)$ une fonction binaire retournant 0 ou 1 selon que la contrainte i appliquée sur *x* est satisfaite ou non
- p(*x*,C) une fonction de pénalité définie sur l'ensemble des entiers naturels

$$c_i \in C,\ x_j \in G,\quad p(x) = \sum_{i=0}^{|C|} \sum_{j=0}^{|G|} c_i(x_j) \qquad \text{(éq. 5)}$$

Les opérateurs de croisement et de mutation sont susceptibles d'engendrer un grand nombre de solutions ayant des composants redondants ou mal ordonnés. Pour cette raison une approche multi-objectifs exploitant la technique de pénalisation nous apparaît comme un premier choix naturel et nous avons donc retenu l'algorithme PEAP que nous décrirons plus en détail dans la section qui suit. Le calcul d'une pénalité pourrait être fondé sur diverses caractéristiques du graphe (d'une solution), de manière à pénaliser les solutions incompatibles avec les chaînes de traitement en raison d'une trop forte connexité, d'une grande redondance de chemins, etc..

3.4.2 Présentation de l'algorithme PEAP adapté au mécanisme décisionnel

Soit :

- P : population d'individus
- D : taille de la population où D = |P|
- E : élite de taille D'
- D' : taille de l'élite où D' = |P| / 2
- S : archive secondaire de taille D

α: configuration courante de l'architecture de traitement

T : nb maximum de générations

t : génération courante $t \mid t \geq 0 \wedge t < T$

P_t : population à la génération t

$FP(P_t, C_i, Q_i)$: front de Pareto de la population P_t (c.a.d. individus non-dominés)

ω : sous-ensemble des individus non dominés

ω' : sous-ensemble des individus dominés

F_1 : fonction objective de coût de reconfiguration

F_2 : fonction objective de qualité de traitement

C_i : coût de reconfiguration de l'individu i

Q_i : qualité de la configuration de l'individu i

FB_i : degré brut d'adaptation d'un individu

F_e : degré d'adaptation d'un individu de l'élite

F_p : degré d'adaptation d'un individu de la population dominée

Pe(i) : pénalité telle que définie par l'équation 4 sur l'individu i

L'algorithme PEAP comportera les étapes suivantes :

Étape 1. Initialisation

1.1 Initialisation du contexte de départ

L'architecture de traitement transmet les paramètres de départ déterminant le contexte dans lequel la décision de reconfiguration sera prise. Ces paramètres sont 1) la configuration courante de l'architecture dynamique de traitement, 2) le type d'environnement auquel l'architecture est couramment soumise (stable ou instable), 3) l'intensité estimée de la perturbation de l'environnement (voir note) ayant déclenché le processus de reconfiguration et 4) la durée estimée de la perturbation.

Note : le domaine de valeurs possibles pour l'intensité des perturbations subies par l'environnement est lui-même partitionné lors de la phase de calibration en trois sous domaines égaux de valeurs *sigma* dénommés *petit sigma*, *moyen sigma* et *grand sigma*.

1.2 Chargement de l'élite initiale $\mathbf{E}_0$

Une élite de taille **D**/2 et correspondant au contexte d'exécution courant (c.a.d à une loi de distribution caractérisant l'environnement perturbé) est chargée en mémoire.

Note : à chaque contexte identifié lors de la phase de calibration du système correspond une élite d'individus obtenue au préalable par simulation lors de cette phase de calibration. Chaque élite est archivée dans un fichier distinct et l'ensemble de ces fichiers constitue une librairie d'élites disponibles pour l'algorithme évolutionnaire.

1.3 Création d'une population initiale $\mathbf{P}_0$

Une population de taille **D,** initialement vide, est créée. La configuration courante de l'architecture de traitement est encodée sous sa représentation génétique afin de créer le premier individu de cette population. Les individus composant l'élite de l'étape précédente sont ensuite chargés dans la population. Enfin les individus nécessaires pour compléter la population sont générés aléatoirement. Une telle approche de chargement contribue à donner au processus évolutif de l'algorithme évolutionnaire une certaine avance dès le départ tout en assurant une bonne dispersion dans l'espace de recherche.

Étape 2. Évaluation des individus

2.1 Les individus de la génération courante (population et élite) sont évalués en leur appliquant les fonctions objectives $\mathbf{F_1}$ (coût de reconfiguration relativement à la configuration courante α) et de $\mathbf{F_2}$ (qualité de traitement).

$$\forall p \in \mathbf{P_t}, \qquad C_p = \mathbf{F_1}(p, \alpha), \; Q_p = \mathbf{F_2}(p)$$

2.2 Les individus de la génération courante sont pénalisés par la fonction de pénalisation qui vérifie le nombre de contraintes violées par chaque individu.

$\forall p \in P_t$, Pe = Pénalisation(p)

2.3 La force relative S de chaque individu par rapport aux autres individus est calculée.

$\forall e \in \mathbf{E_t}$, $p \in \mathbf{P_t}$, Se = (nb d'individus p dominé par e) / (D+1)

$\forall p \in \mathbf{P_t}$, $i \in \mathbf{P_t}$, Sp = (nb d'individus i dominé par p) / (D+1)

2.4 Le degré brut d'adaptation est assigné à chaque individu de l'élite et de la population.

FBe = Se + Pe(e)

FBp = Sp + Pe(p)

Étape 3. Mise à jour de l'élite E_t

3.1 Le sous-ensemble des individus non dominés est recherché dans la population

$\omega \leftarrow FP(P_t\, C_{i,}\, Q_i)$

3.2 On ajoute ces individus à l'élite

$\mathbf{E_t} \leftarrow \mathbf{E_t} \cup \omega$

3.3 On élimine les individus nouvellement dominés parmi l'élite

$\omega \leftarrow FP(\mathbf{E_t}\, C_{i,}\, Q_i)$

$\mathbf{E_t} \leftarrow \omega$

3.4 On réduit la taille de l'élite à sa dimension maximale D' par la méthode d'amalgamation

Si ($|E_t| > D'$) alors { $S \leftarrow$ amalgamation(E_t, D'); $\mathbf{E_t} \leftarrow S$}

Étape 4. Vérification du critère d'arrêt

La méthode de terminaison de l'algorithme évolutionnaire sera basée sur la satisfaction de l'un de critères d'arrêt suivants :

- Critère d'arrêt #1 : Un nombre minimal de générations Gmin est atteint et l'élite est devenue complètement stable depuis Gmin/10 générations.

- Critère d'arrêt #2 : Le nombre maximal de générations Gmax est atteint.

Si l'un des deux critères est satisfait, procéder à l'étape 7; sinon retour à l'étape 2.

Étape 5. Sélection par tournoi des individus pour la reproduction

5.1 Un groupe de reproduction de taille vide est initialement créé.

5.2 Les individus de l'élite sont sélectionnés deux à deux de manière aléatoire et leur degré brut d'adaptation respectif **FB** est comparé. L'individu possédant le meilleur degré brut d'adaptation est assigné au sous-groupe de reproduction. Cette sélection se poursuit jusqu'à ce que le sous-groupe de reproduction soit complet.

Étape 6. Reproduction

6.1 On procède à la reproduction de la prochaine génération en appliquant sur les individus de la population courante des opérateurs de croissement et de mutation selon une méthode de programmation génétique, développée par Koza [30a], qui représente les individus sous forme d'expressions arborescentes. Suite à cette étape, retour à l'étape 2.

Étape 7. Sélection des solutions

PEAP collecte les solutions non-dominées dispersées le long du front de Pareto et produit un ensemble qui constitue la sortie finale de l'algorithme.

Chapitre 4 Implantation et expérimentation de l'approche

4.1 Méthodologie décisionnelle en deux phases

Le mécanisme décisionnel multicritères réagit aux changements de comportement de l'environnement et décide 1) si la reconfiguration de l'architecture de traitement est nécessaire d'une part; et 2) comment l'architecture devrait être reconfigurée de manière à optimiser la fonction objective ayant des critères conflictuels définissant la performance du système adaptatif (par exemple qualité du traitement et rapidité de traitement).

Une décision multicritères nécessitant l'exécution d'un algorithme évolutionnaire ne pourra généralement pas satisfaire des contraintes d'exécution d'une application en temps réel à cause de la lourdeur de traitement de ce type d'algorithme. Conséquemment, notre méthodologie décisionnelle adopte une approche en deux phases complémentaires qui se déroulent à des moments disjoints: la première phase traite hors ligne les éléments statiques de la décision. Cette phase a lieu une seule fois. La seconde phase, celle de la prise de décision, se répétera à chaque fois qu'une perturbation sera détectée et incorporera des éléments dynamiques lors de cette prise de décision.

La phase statique de calibration inclut les prétraitements décisionnels qui sont lourds, qui exigent un grand temps de traitement et qui ne peuvent être bornés dans le temps. Ces traitements sont effectués sur un banc d'essai utilisant des données historiques sur le comportement de l'environnement applicatif et sur le comportement des perturbations qui l'affectent. Ces données sont traitées, classifiées puis organisées afin de supporter la simulation de scénarios d'exécution. Chaque scénario exécuté déclenche une optimisation multicritères générant un front de Pareto qui sera classifié dans une librairie (librairie de Pareto) contenant les autres fronts générés par l'ensemble des essais. La simulation produit aussi des données de performance qui peuvent être analysées par un expert. Celui-ci en déduira des règles (des politiques) à archiver dans une base décisionnelle (fig. 4.1). Toutes ces sorties seront finalement déployées dans le système adaptatif.

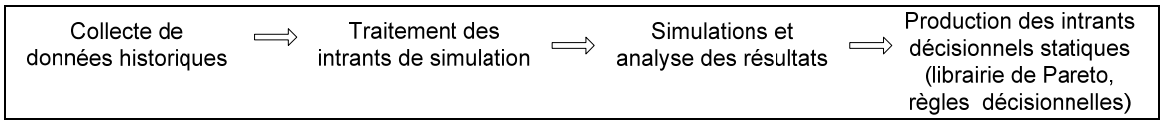

Figure 4.1 Activités de la phase statique de calibration

La phase dynamique de décision est déclenchée par la détection d'une perturbation au moment de l'exécution du système adaptatif (fig. 4.2) A ce moment, les sorties statiques de la première phase sont combinées aux entrées dynamiques caractérisant la perturbation (i.e. stabilité, durée et intensité de la perturbation). Le temps d'exécution de la phase dynamique est à la fois rapide et borné.

En combinant les entrées statiques et dynamiques, le mécanisme décisionnel construit un contexte décisionnel pleinement extensionné (c.à.d. complet) tel que le préconise [45]. L'extension dynamique du contexte compense en partie l'incertitude liée au comportement de l'environnement et contribue à la qualité de la décision. Le détail de ces deux phases sera maintenant décrit ans les prochaines sections.

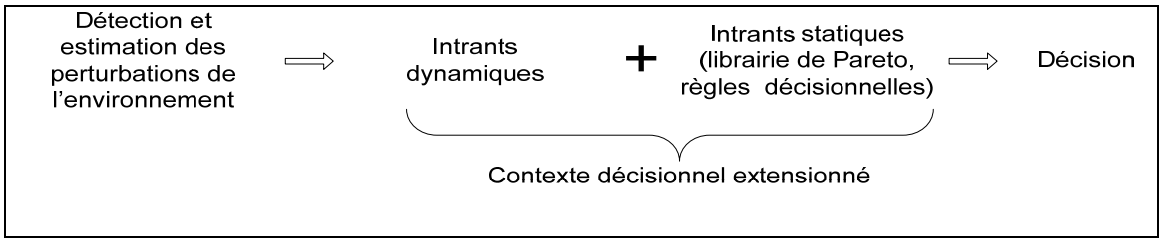

Figure 4.2 Phase dynamique de prise de décision

4.2 Phase statique : optimisation multicritères et calibration

4.2.1 Architecture du banc d'essai

La phase statique est réalisée sur un banc d'essai dont l'activité centrale est l'optimisation multicritères. Le banc d'essai supporte la simulation de divers scénarios de comportement et de perturbation de l'environnement. Il permet d'observer et de mesurer la performance des configurations produites par l'algorithme évolutionnaire. Quatre sous-systèmes (*Environnement*, *Traitement*, *Décision*, *Simulation*) collaborent à l'optimisation multicritères de la décision tel

qu'illustré à la figure 4.3. Les quatre sous-systèmes seront brièvement décrits dans les sections qui suivent.

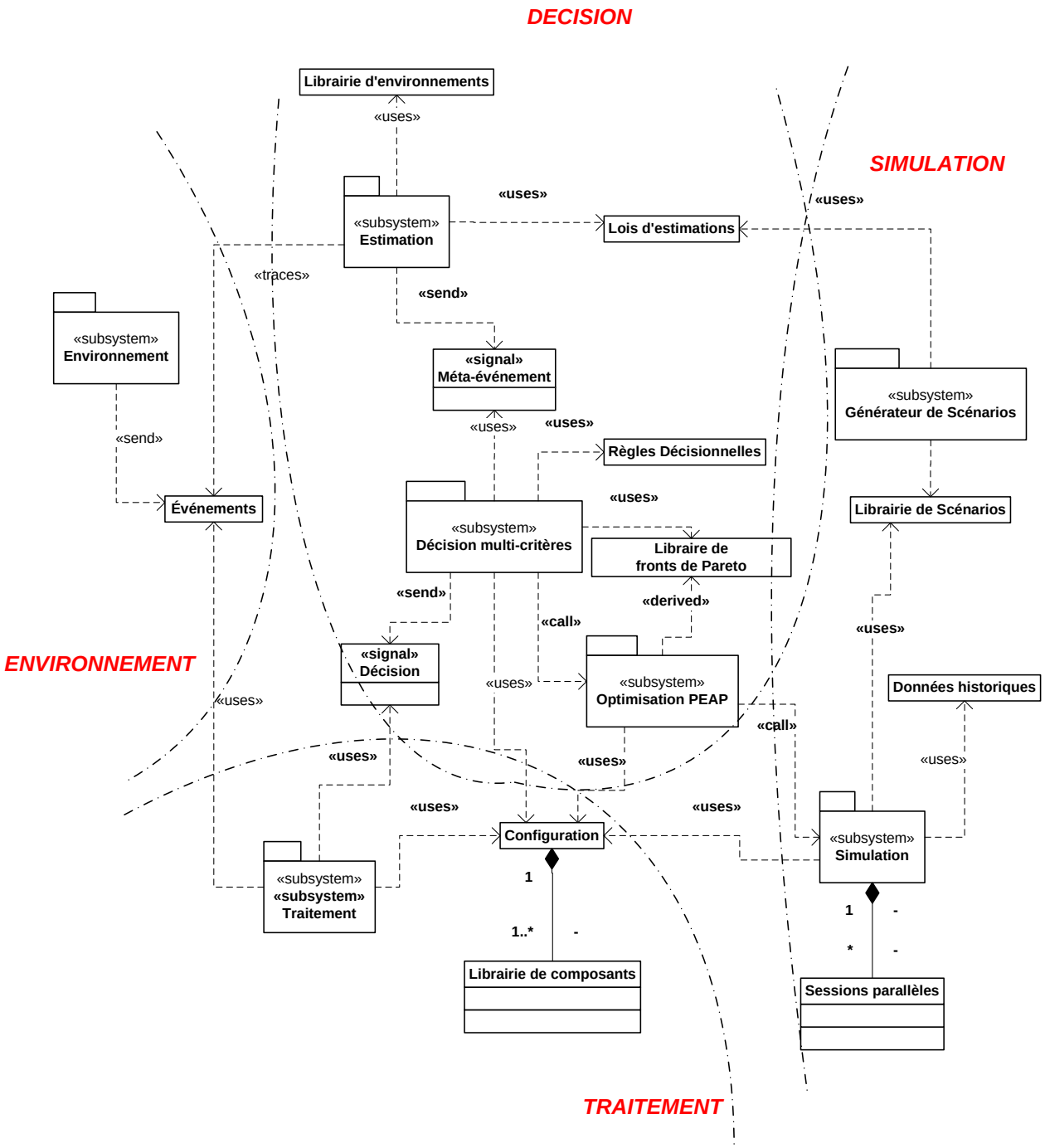

Figure 4.3 Diagramme de collaboration du banc d'essai

4.2.2 Sous-système *Environnement*

L'environnement externe est modélisé par un sous-système qui transmet en continu des événements qui seront traités par le système adaptatif. Le comportement de cet environnement est incertain et sera parfois affecté par des perturbations imprévisibles mais dont la stabilité, la durée et l'intensité peuvent toutefois être estimées.

Ce sous-système collabore d'une part avec le sous-système *Traitement* auquel il fournit les événements à traiter et d'autre part avec le sous-système *Estimation* auquel il fournit des données de fréquence de réception qui seront analysées pour détecter les changements de comportement de l'environnement et en estimer les caractéristiques.

Le comportement de l'environnement est reproduit en utilisant des données historiques ou en générant un flux synthétique d'événements selon des lois de distributions assimilables aux comportements connus de l'environnement et des perturbations.

4.2.3 Sous-système *Traitement*

Le sous-système de traitement encapsule la composition et l'exécution d'une architecture dynamique reconfigurable composée de multiples composants de traitement. Cette architecture effectue le traitement utile sur les événements qu'elle reçoit de l'environnement externe. Le sous-système collabore avec le sous-système *Décision* qui lui transmet des messages de changements de configuration de l'architecture de traitement distribué.

L'architecture de traitement sera construite à partir de composants spécialisés archivés dans une librairie applicative. Cette librairie est partagée avec le sous-système de *Simulation* qui l'emploie pour simuler l'architecture de traitement. Les sous-systèmes *Traitement* et *Simulation* sont fonctionnellement équivalents et interchangeables du point de vue du modèle décisionnel qui demeure inchangé, que l'on soit dans la phase de calibration (hors ligne) ou dans la phase de décision (en ligne).

Le module de traitement encapsule de plus la fonctionnalité relative au processus de reconfiguration dynamique de l'architecture. Lors d'une reconfiguration, le module génère les instructions nécessaires pour la composition de la nouvelle configuration et invoque les méthodes appropriées pour mettre en place une configuration alternative. Le problème de maintien de l'état du système est l'une des responsabilité de ce module.

4.2.4 Sous-système *Décision*

Le sous-système *Décision* est composé de trois modules qui participent aux activités des deux phases décisionnelles, à savoir les modules *Estimation*, *Optimisation* et *Décision*.

Module *Estimation*

Le module *Estimation* est chargé de la surveillance et de l'estimation de l'environnement. Il observe continuellement l'arrivée des événements afin d'analyser les fluctuations du comportement de l'environnement dans le but d'estimer les caractéristiques de l'environnement et des perturbations en vigueur. Le module d'estimation et sa méthodologie d'estimation ne sont pas développés par ce projet. Nous avons plutôt recours aux résultats d'études existantes, comme par exemple celle de Golob [20] qui a répertorié 21 environnements différents dans le domaine de la circulation routière. Lorsque le banc d'essai est exploité pour ce contexte, ces 21 environnements seront donc chargés dans la librairie d'environnement illustrée au haut du diagramme de la figure 4.3. Les données historiques de cette étude seront quant à elles chargées dans le dépôt de données historiques illustré à droite de ce même diagramme. Le fonctionnalité du module d'estimation ne sera pas décrite davantage dans ce projet.

Module *Optimisation*

Le module *Optimisation* exécute l'algorithme évolutionnaire PEAP pour l'optimisation multicritères et contrôle la mise à jour de la librairie de Pareto et de la base de règles durant la phase statique de calibration. Ce sous-système collabore avec le sous-système *Décision* duquel il reçoit des requêtes d'optimisation de la configuration de traitement. Il collabore de plus avec le sous-système *Simulation* auquel il soumet des solutions de reconfiguration pour fin d'évaluation dynamique.

Module *Décision*

Le module décisionnel encapsule la fonctionnalité de la prise de décision qui se déroule en deux étapes qui seront expliquées en détails à la section 4.3.

4.2.5 Sous-système *Simulation*

Le sous-système *Simulation* est uniquement actif durant la phase statique de calibration durant laquelle il assume le rôle du sous-système *Traitement*. Il collabore directement avec le sous-système *Optimisation* duquel il reçoit des requêtes pour simuler la performance de solutions de configuration. Il collabore aussi avec un substitut du sous-système *Environnement* qui lui fournit un flux d'événements sous l'une des deux formes suivantes :

- flux de données synthétiques générées par le module *GenPerturb*
- flux de données historiques collectées par observation d'un système adaptatif réel.

4.3 Phase dynamique : tolérance et sélection

Les changements de reconfiguration liés à l'adaptation introduisent des délais supplémentaires de traitement dans le système adaptatif. Les travaux de [5] ont montré qu'il peut y avoir des inconvénients à effectuer des changements de configuration fréquents lors de fortes perturbations et qu'il est parfois préférable de tolérer la situation, c'est-à-dire de ne pas adapter le système, malgré les estimations indiquant qu'un environnement nécessitent des adaptations. Pour cette raison, la décision de reconfigurer le système ne doit pas être nécessairement accordée : il faut tenir compte du contexte dynamique. D'autre part, comme un front de Pareto met plusieurs solutions de reconfiguration optimales à la disposition du mécanisme décisionnel, il faut donc un critère supplémentaire pour effectuer la sélection d'une solution finale unique. Ce critère supplémentaire sera lui aussi relatif au contexte dynamique. Les prochaines sections décrivent comment le processus décisionnel emploie les entrées dynamiques pour appliquer le concept de tolérance et pour obtenir une sélection finale.

4.3.1 Activités de la phase dynamique de la décision multicritères

Les activités de la phase dynamique décisionnelle sont illustrées à la figure 4.4. Un module spécialisé d'estimation évalue périodiquement l'évolution du comportement de l'environnement. en surveillant l'inter-arrivée d'événements à traiter. La phase dynamique est déclenchée lorsqu'un changement significatif de l'environnement est détecté (temps *t1*). Le module produit alors un estimé des caractéristiques du nouvel environnement (temps *t2*). Le mécanisme décisionnel prend une décision en deux étapes. Au temps *t3*, le processus évalue l'état courant du système adaptatif et de l'environnement, puis applique une politique de tolérance conditionnelle (détails à la section 4.3.2). Dans le cas où la décision de reconfiguration en *t3* est positive, le mécanisme applique finalement au temps *t4* une politique de sélection de configuration sur le front de Pareto et produit une sélection de configuration finale (détails à la section 4.3.3). La décision de reconfiguration finale sera transmise au système adaptatif pour sa mise en œuvre sur l'architecture de traitement.

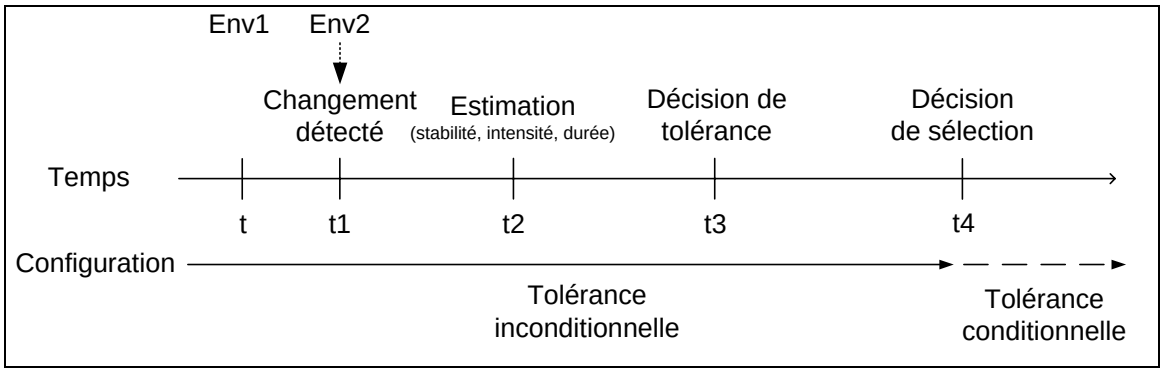

Figure 4.4 Activités de la phase dynamique de la décision multicritères

On notera qu'il pourrait s'avérer nécessaire, dans certains domaines de systèmes adaptatifs où l'impact de la décision est critique, d'ajouter une troisième étape pour la validation de la décision finale par une personne experte. Nous examinons maintenant les détails des deux étapes décisionnelles de la phase dynamique.

4.3.2 Étape décisionnelle #1 : décision de tolérance

Le processus décisionnel de l'étape 1 est modélisé par un automate à états finis (figure 4.5) qui évolue selon des transitions déclenchées par les politiques de tolérance en fonction des méta-événements reçus du module d'estimation. Le processus décisionnel procédera à la seconde étape de la décision uniquement dans l'éventualité où l'état 3 serait atteint. On notera que pour l'instant la transition de l'état 2 directement à l'état 3 (en pointillé) est optionnelle et n'a pas été implémentée dans le mécanisme décisionnel. Ce raffinement dépasse le cadre du présent projet mais pourrait toutefois être considéré lors de travaux futurs.

Le diagramme de séquence de la figure 4.6 illustre une scénario de tolérance durant lequel le système subit un changement d'environnement de *ENV1* à *ENV2* puis revient rapidement au premier environnement *ENV1*. À la détection du premier changement, le module *Estimation* transmet le message *Meta-Evt1* au module *Décision* qui active la première étape décisionnelle menant à la décision de tolérer ou non la configuration actuelle : le module Décision demande d'abord au module *Traitement* de l'informer sur sa configuration courante ; il consulte ensuite la *Base de règles de tolérance* avec les paramètres actuels (le contexte extensionné) ; finalement la base de règles détermine qu'il faut dans ce contexte tolérer la configuration actuelle et conséquemment l'étape décisionnelle #2 (le choix de reconfiguration) n'est jamais activée. Peu de temps après, l'environnement revient à *ENV1* et l'étape 1 de la décision est à nouveau déclenchée (seul le début de cette séquence est illustré dans 4.6).

La décision de tolérance est prise conformément à une politique élaborée selon des entrées exogènes (par exemple les entrées dynamiques fournies par le module d'estimation) ou endogènes (par exemple l'état du système adaptatif ou encore la charge courante de traitement), ou enfin une combinaison des deux. Une politique de tolérance *exogène* sera exclusivement basée sur des paramètres externes au système, tels les caractéristiques de la perturbation dans le cas où il n'y a pas d'effet rétroactif de la performance du système de traitement sur la perturbation. On peut envisager par exemple une politique basée sur la valeur *sigma* d'intensité de la perturbation. Une politique *endogène* sera exclusivement basée sur des paramètres internes au système, tel l'état de l'architecture de traitement. On peut envisager ici une politique de tolérance selon le temps estimé pour atteindre un état de quiétude [31] propice à une

reconfiguration. Finalement, une politique *mixte* fera intervenir les deux types d'entrées. Par exemple dans le cas d'un équipement de télécommunication réseau, ceci équivaut à tenir compte de la charge des tampons internes de l'équipement ainsi que le niveau de congestion du réseau.

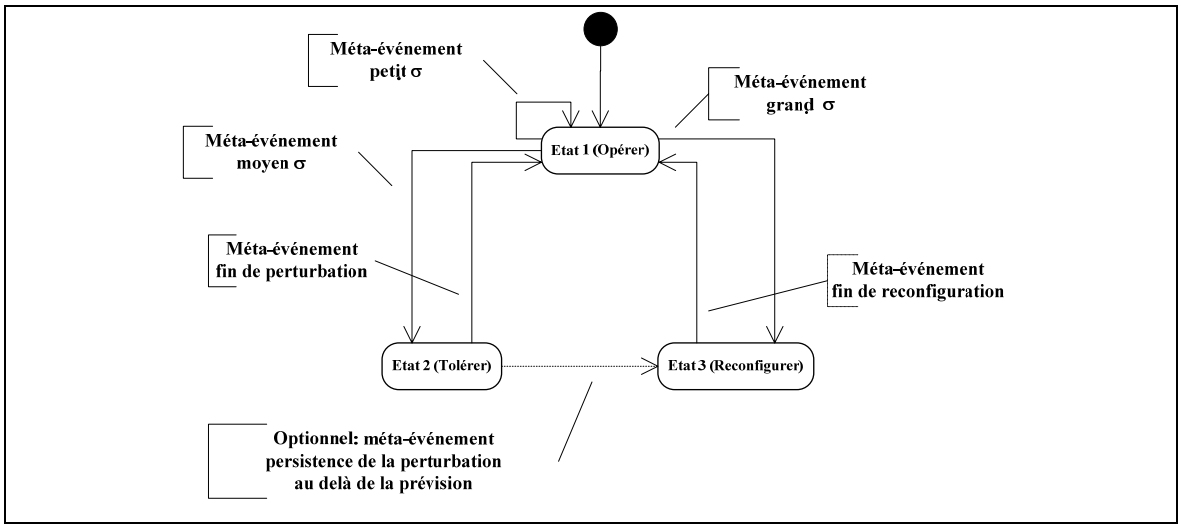

Figure 4.5 Automate décisionnel à états finis

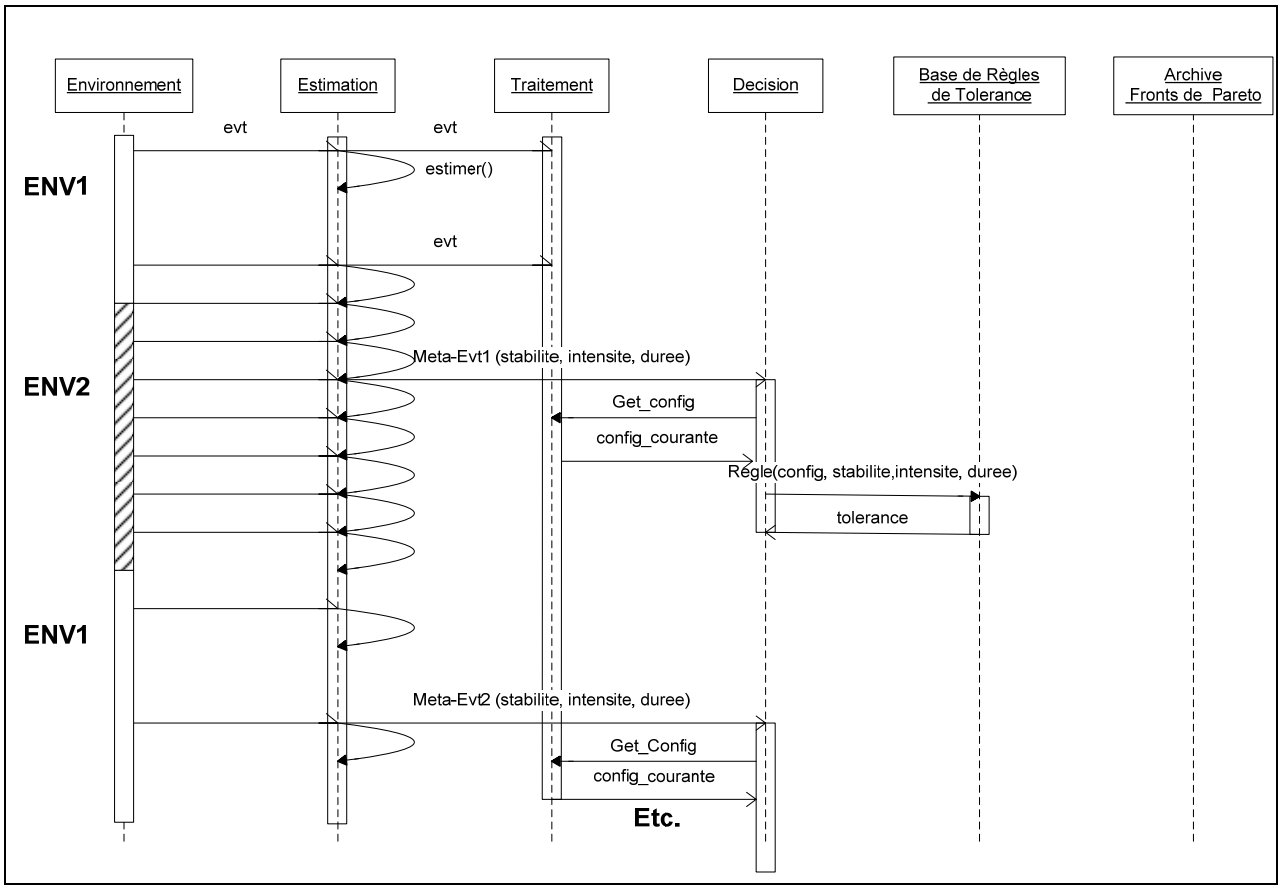

Figure 4.6 Scénario menant à une décision de tolérance

4.3.3 Étape décisionnelle #2 : décision de sélection finale

La politique de sélection finale repose sur une relation bijective entre le domaine de valeur des entrées dynamiques et l'ensemble des solutions contenues sur le front de Pareto.

Cette relation est définie de la manière suivante :

1. On établit une graduation à 'n' niveaux sur le domaine de valeur des entrées. Par exemple, pour une entrée *sigma* d'intensité de perturbation, le domaine de valeurs sera gradué avec n=3 niveaux (stable : *petit sigma*, instable : *moyen sigma*, très perturbé : *grand sigma*).

2. Le front de Pareto est partitionné en 'n' sous-groupes constitués chacun de 1/n des individus (figure 4.8).

3. Chaque segment est réduit par la technique d'amalgamation *k-mean* à un individu unique représentatif de ce sous-groupe. Suite à cette amalgamation le front de Pareto est donc réduit à n solutions.

4. La nième valeur de l'entrée est associée à la nième solution. On se rappellera que chacune de ces solutions est un tuple de deux valeurs f1(a) et f2(a) (une pour chaque critère). La politique de sélection établira un biais en faveur du premier ou du second critère d'optimisation selon la valeur des entrées estimé par le module de surveillance. Par exemple, dans le cas illustré à la figure 4.7, sous une prédiction d'environnement plutôt stable (*petit sigma*), le choix sera biaisé en faveur du critère de qualité de traitement (f2(a)) , alors que sous une prédiction d'environnement plutôt perturbé (*grand sigma*), on favorisera la fonction de rapidité de traitement (f1(a)).

La politique de sélection consiste donc à retourner la solution correspondant à la valeur transmise par le méta-événement.

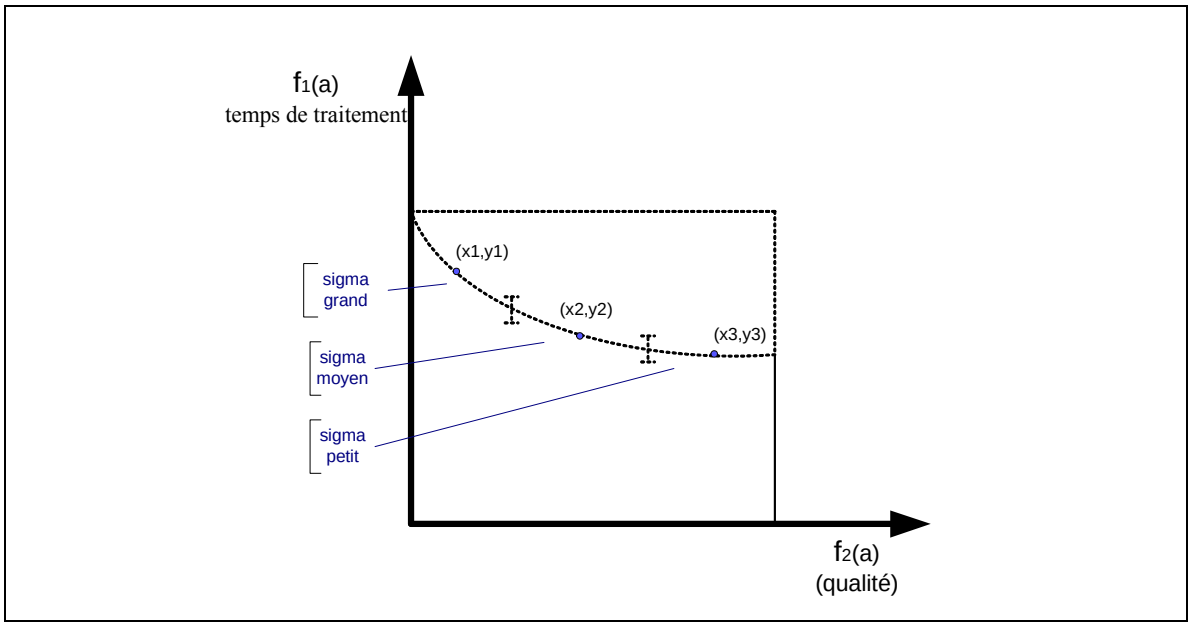

Figure 4.7 Réduction par amalgamation du front de Pareto à trois solutions.

La figure 4.8 illustre une séquence d'événements menant à l'exécution de l'étape 2 pour sélectionner une nouvelle configuration. Au départ la séquence est la même qu'à la figure 4.6. Toutefois dans ce scénario la décision intermédiaire produite par l'étape 1 est de procéder à la reconfiguration du système. Suite à cette décision, le module *Décision* consulte l'archive de fronts de Pareto précalculés avec les paramètres du contexte extensionnée (*Front de Pareto(config, stabilite, intensite, duree)*) et repère le front de Pareto le mieux adapté au contexte défini par les entrées dynamiques. La solution finale est alors immédiatement disponible et prête à être acheminée au module *Traitement*. Ce module se reconfigure immédiatement selon les spécifications de cette nouvelle configuation, tel que l'illustre le changement de texture (de gris pale à gris fonçé) de la ligne d'activation du module *Traitement*.

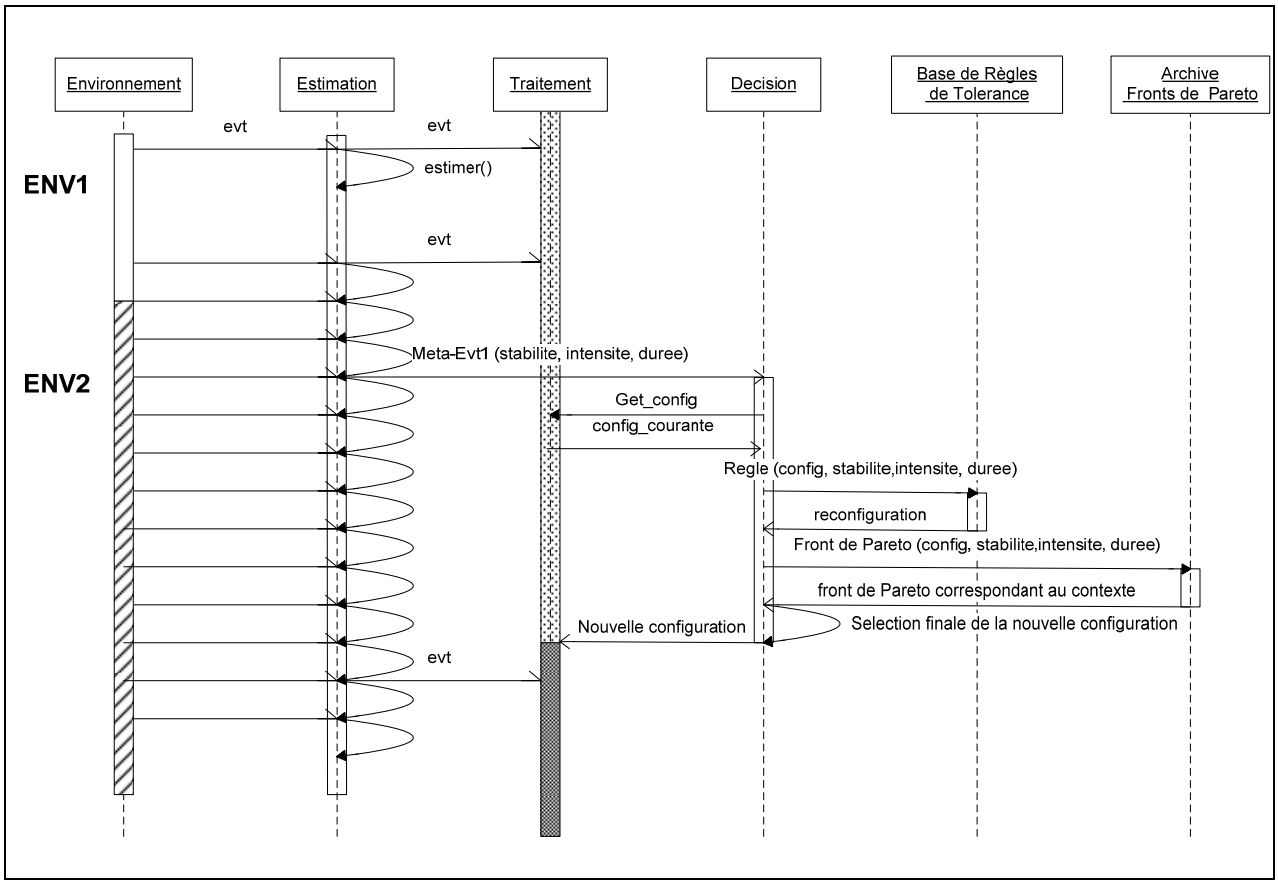

Figure 4.8 Scénario menant à une décision de reconfiguration

4.4 Implantation de l'optimisation PEAP

Notre banc d'essai pour l'optimisation multicritères a été implanté dans le langage de programmation Java. Le domaine public offre plusieurs cadres de programmation évolutive codés en Java et nous avons sélectionné un cadre nommé *Evolution Computing in Java* (*ECJ*) [36]. La principale caractéristique du cadre ECJ est de permettre le chargement dynamique paramétrisé d'un problème évolutionnaire (dans notre cas, d'optimisation multicritères) sans avoir à recoder complètement un algorithme spécialisé pour chaque nouveau système adaptatif. Il s'agit là d'un avantage important pour notre projet, mais qui requiert toutefois une bonne approche d'abstraction dans la conception de l'Agent Complexe.

ECJ met à la disposition des programmeurs de nombreux algorithmes évolutifs dont l'algorithme SPEA2. L'algorithme PEAP est beaucoup plus récent et n'est toujours pas disponible dans les packages de distribution de ECJ. Nous avons donc choisi d'inplanter PEAP en nous inspirant de l'implantation existante de ECJ pour SPEA2.

Notre conception et notre inplantation de l'Agent Complexe emploie quatre niveaux d'abstraction (figure 4.9). Selon ce découpage fonctionnel, l'implantation d'un mécanisme décisionnel dans un domaine spécifique de système adaptatif ne nécessitera que des ajustements au dernier niveau. Nous décrivons brièvement le rôle organique de chaque niveau.

Classes de niveau 1

A un premier niveau, on retrouve les classes génériques du cadre *ECJ* qui supportent les fonctions générales de toutes les catégories d'algorithmes évolutionnaires.

Classes de niveau 2

Au second niveau, on retrouve les classes qui mettent en œuvre une catégorie spécialisée d'algorithme évolutif, dans notre cas la catégorie évolutionnaire *multicritères*.

Classes de niveau 3

Au troisième niveau, on retrouve les classes dotées du préfixe *PEAP* qui mettent en œuvre un algorithme spécifique d'évolution multicritères, dans notre cas l'algorithme *PEAP*.

Classes de niveau 4

Au quatrième et dernier niveau, on retrouve les classes dotées du préfixe *Opt* et qui sont spécifiques à un problème spécifique d'optimisation multicritères, par exemple *vitesse versus critère de qualité de traitement*. Les classes du niveau 4 sont des classes spécialisées et encapsulées dans un *package* Java (*Optic*) et peuvent être chargées dynamiquement dans le cadre *ECJ*. C'est à ce niveau que l'on retrouve la mise en œuvre de la fonction objective. Les classes des autres niveaux n'ont pas à être modifiées pour la mise en place d'un nouveau mécanisme décisionnel.

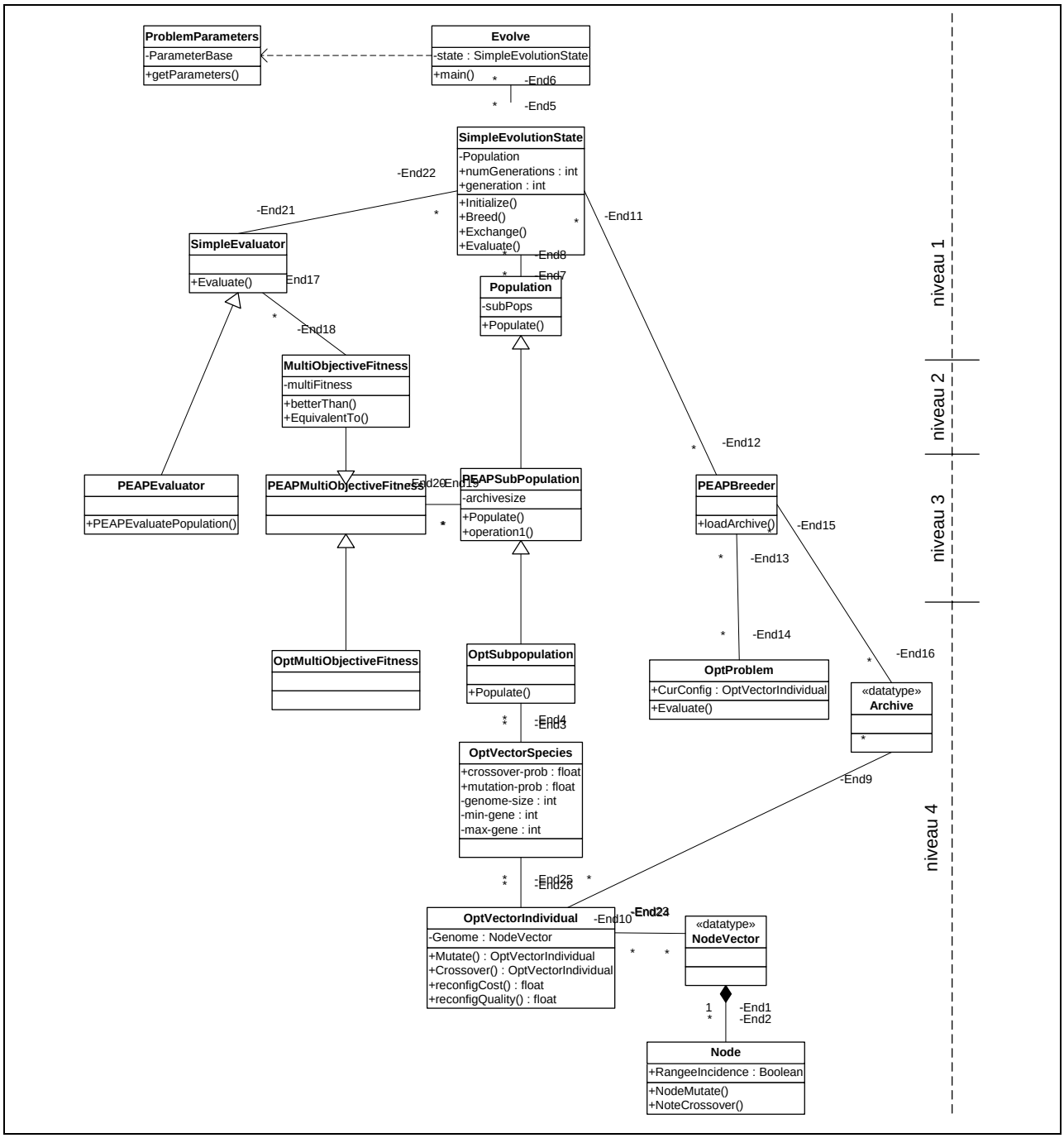

Figure 4.9 Diagramme de classes – optimisation PEAP

4.5 Étude de cas dans le domaine des transports intelligents

Nous présentons maintenant une étude de cas démontrant la mise en œuvre de notre cadre et de son mécanisme décisionnel dans le domaine des systèmes de transport intelligents (ITS) [28] dans le but d'améliorer la gestion de la sécurité routière. Notre objectif est de valider qu'une décision de reconfiguration du réseau routier ne mène pas à une déstabilisation du réseau ni à une perte de confiance des conducteurs relativement aux directives qui leur sont fournies car ces deux résultats peuvent diminuer la sécurité globale du réseau. Nous souhaitons en particulier illustrer l'impact de la notion de tolérance lors de la prise de décision de reconfiguration.

4.5.1 Présentation du scénario de reconfiguration

Soit un réseau de transport urbain tel que celui schématisé à la figure 4.9. Ce réseau est sujet à l'arrivée d'un flux de véhicules automobiles qui le traversent d'ouest en est. Dans des conditions stables la circulation est fluide c.à.d. que le nombre de véhicules quittant le réseau équivaut au nombre de véhicules qui y entrent. Cette stabilité peut néanmoins être affectée par divers changements d'environnement qui viendront perturber les flux d'entrée et la capacité de passage en certains points du réseau routier. Dans de tels cas il y a possibilité de congestion et la circulation devient alors plus risquée [42]. Pour pallier à ces risques, un système ITS est déployé sur le réseau et le supervise par l'entremise de senseurs de circulation et de panneaux d'affichage.

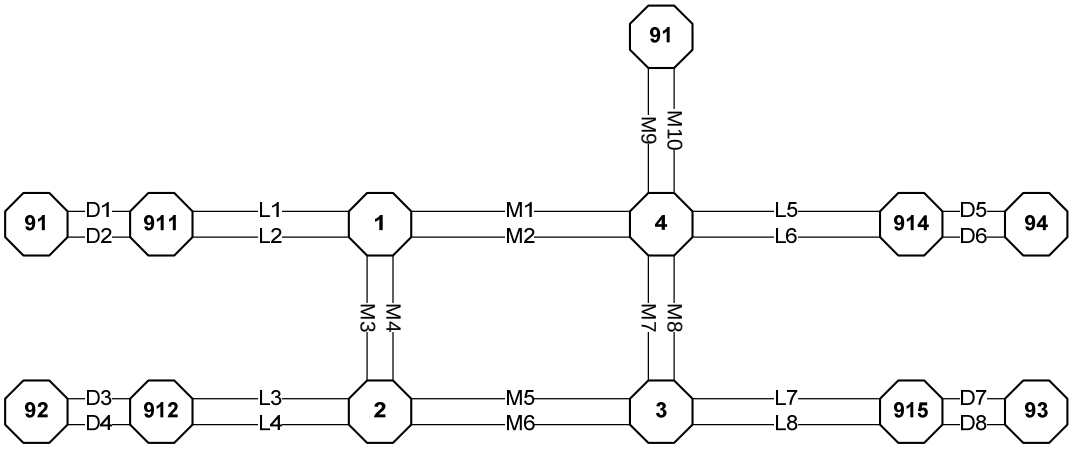

Figure 4.9 Schéma d'un de réseau de transport urbain

Le système ITS collecte et analyse en tout temps les données fournies par divers senseurs afin de détecter les changements dans le comportement de la circulation, d'estimer la durée probable des

perturbations et d'agir sur le réseau à l'aide de « messages de reconfiguration » qui incitent les conducteurs à modifier leur route pour pallier aux risques et aux délais engendrés par les perturbations.

Notre étude de cas évaluera la qualité des actions engendrées par le système selon que ces actions sont déclenchées automatiquement ou qu'elles résultent d'un processus décisionnel tel que celui mis en œuvre par notre cadre. A cette fin on propose de tester le scénarios dans le contexte d'une perturbation de courte durée, en comparant une approche par reconfiguration réactive versus notre approche par décision de reconfiguration avec tolérance. Notre objectif est de déterminer si les instructions déclenchées par notre mécanisme décisionnel donnent lieu à une sécurité routière accrue.

4.5.2 Instanciation du cadre pour les ITS

L'architecture de notre cadre ITS peut être vue selon trois couches logicielles (fig. 4.10). La couche supérieure est représentée par un agent unique, l'Agent Superviseur, qui a la responsabilité d'équilibrer la charge de traitement des données parmi les agents des couches inférieures. La couche intermédiaire est constituée de multiples Agents Complexes (AC) dont le rôle est d'une part de traiter l'information générée localement par de multiples senseurs dispersés à travers l'infrastructure physique du réseau routier, et d'autre part de la transmettre aux multiples actuateurs des décisions de contrôle et de gestion de la performance du réseau routier en termes de sécurité routière. La couche inférieure est constituée d'une myriade d'Agents Environnement matériels ou embarqués tels des senseurs (ex. boucles de détections, caméras vidéo, véhicules) ou des actuateurs (ex. panneaux d'affichage, feux de signalisation). Les Agents Environnement permettent la communication bi-directionnelle entre le réseau routier et le cadre ITS par l'entremise d'un couche télécommunication non illustrée (hors propos). Ils relaient en temps réel les événements décrivant l'état du réseau, à partir des senseurs jusqu'à la couche de traitement (c.à.d. les ACs), ; en sens inverse ils relaient les décisions de gestion du réseau routier de la couche traitement vers les actuateurs (ex. panneaux d'affichage) impliqués par ces décisions. La figure 4.10 illustre les interactions entre les trois couches.

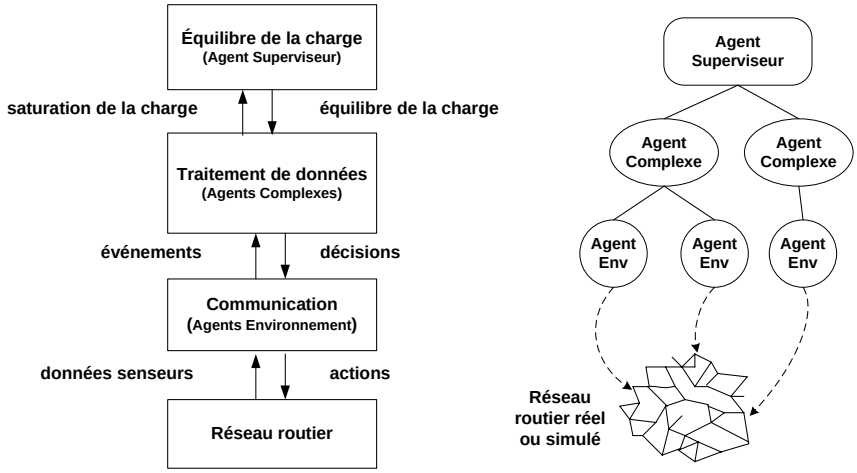

Figure 4.10 Disposition en trois couches du cadre ITS

4.5.3 Séquence décisionnelle du cadre ITS

Dans le contexte des ITS, la réduction des impacts négatifs sur la performance est transposée en termes de sécurité routière. Nous adaptons maintenant notre description du processus décisionnel décrit à la section 4.3 au contexte de l'étude de cas. Dans ce contexte, l'adaptation menant à une sécurité accrue se traduit par diverses actions de reconfiguration des actuateurs dispersés le long du réseau routier.

Dans notre cadre toute reconfiguration est d'abord précédée par un processus décisionnel impliquant deux décisions distinctes (index temporels *t3* et t*4* de la fig. 4.11).Dans un premier temps l'*Agent Complexe* décide comment optimiser son architecture de traitement selon le nouveau « Régime » (comportement) du réseau (l'environnement). Dans un second temps, l'architecture de traitement produit une décision de reconfiguration et décide de la mettre en œuvre ou non en appliquant les règles de tolérance. S'il y a mise en œuvre de la reconfiguration, la décision se traduit par des instructions de contrôle acheminées (ex. message de redirection de la circulation) à l'index temporel *t4* aux *Agents Environnement* concernés.

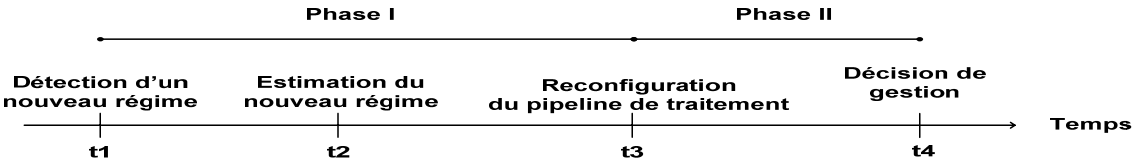

Figure 4.11 Séquence décisionnelle pour le cadre ITS

L'architecture de traitement de l'*Agent Complexe* est conçue tel un pipeline de classes de traitement ou chaque classe est instanciée par au plus un composant. Dans le domaine de la sécurité routière, on pourra par exemple considérer les trois classes suivantes : filtrage/échantillonage des données de senseurs (boucles de détection, caméra vidéo), estimation du régime de circulation selon par exemple la classification de [20], algorithmes de routage des véhicules basés sur des heuristiques ou des métaheuristiques [8].

La performance individuelle de chaque composant, en termes de qualité de traitement et de rapidité de traitement, est liée au contexte de traitement dans lequel on l'applique ainsi qu'à la composition globale du pipeline. Le pipeline optimal pour le contexte courant (régime, perturbation, etc.) aura été déterminé au préalable par simulation sur le banc d'essai adapté aux transports routiers et que nous décrirons sous peu.

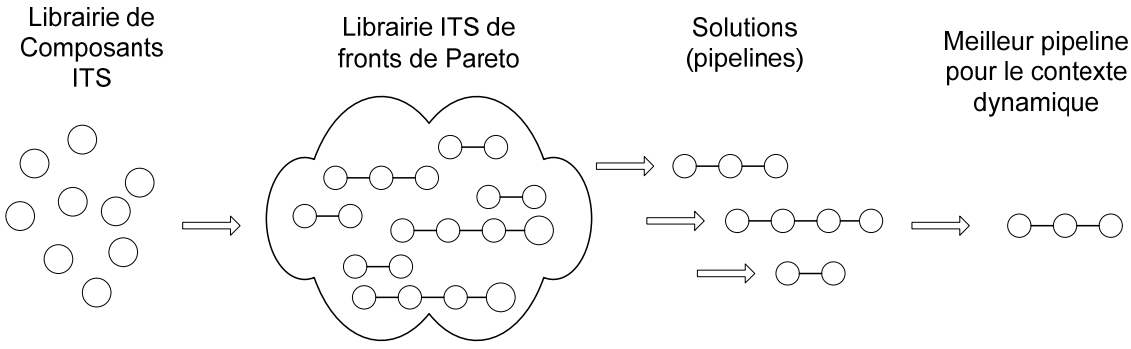

Figure 4.12 Reconfiguration du pipeline de traitement de l'Agent Complexe

4.5.4 Environnement d'expérimentation SUMO

Pour la phase expérimentale de notre étude, nous avons choisi un environnement de simulation de réseau routier appelé SUMO (Simulation of Urban MObility) [54] que nous avons connecté au banc d'essai (fig. 4.3) de notre cadre ITS selon l'architecture illustrée par la figure 4.13. SUMO est un environnement autonome qui exécute des simulations routières microscopiques, c.a.d. que le comportement de chaque véhicule est géré individuellement selon un modèle de poursuite routière (*car following model* en anglais). SUMO offre une interface permettant d'opérer les simulations en mode client-serveur. Dans ce mode un système externe tel que notre banc d'essai ITS peut contrôler dynamiquement l'exécution pas à pas d'un scénario de simulation routière par l'entremise d'une interface de contrôle de trafic dénommée TraCI (*Traffic Control Interface* en anglais) et implémentée à l'aide du protocole TCP/IP. Alternativement les simulations SUMO peuvent aussi être effectuées en mode autonome.

En mode serveur, SUMO initialise une simulation en chargeant d'abord deux fichiers qui définissent la configuration du réseau et le trafic routier. Le simulateur se met ensuite en mode serveur et attend les commandes de contrôle qui lui seront soumises via l'interface *TraCI*. Ces commandes peuvent contrôler l'avancement de l'horloge, modifier les comportements des éléments physiques du réseau (ex. véhicules, feux de signalisation, voies de circulation). Le client (dans ce cas, le banc d'essai) peut superviser à chaque pas d'horloge l'état de la simulation et interroger l'état général du réseau ou des ses éléments individuels de manière à obtenir par exemple la position et la vitesse des véhicules, le taux d'occupation des voies, l'état d'un feu de signalisation. Les actions de contrôle et les requêtes de données d'état sont gérées par les *Agents Environnement* de notre cadre ITS.

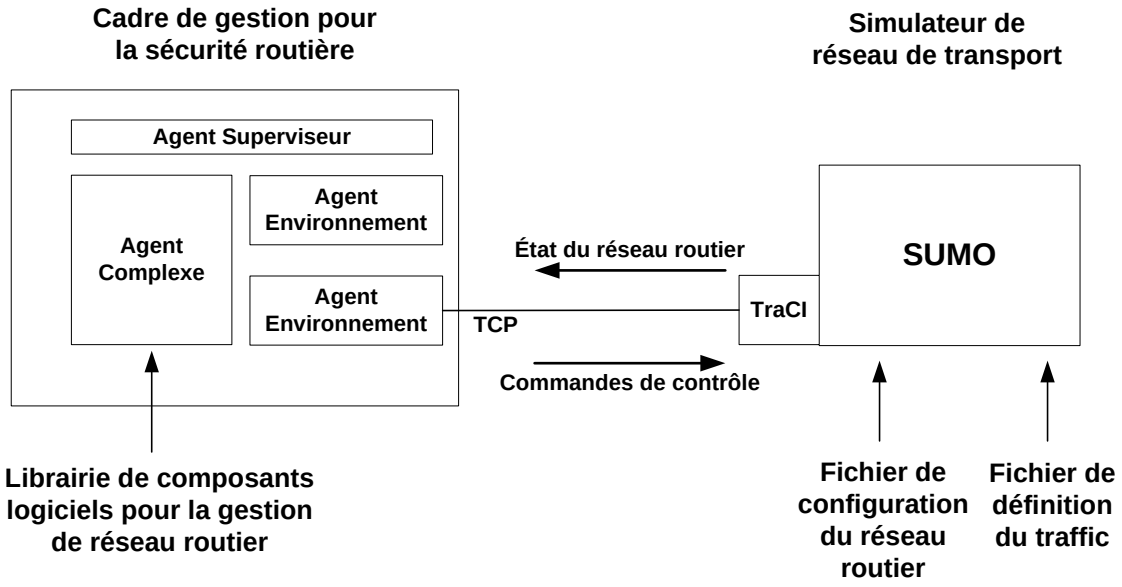

Figure 4.13 Interface entre le cadre ITS et le simulateur SUMO

4.5.5 Présentation des résultats du scénario d'essai

La figure 4.14 est une représentation graphique générée par SUMO pour le schéma de réseau routier de la figure 4.9. Le réseau comprend divers éléments physiques disposés le long des voies : véhicules en jaune, boucles de détection en bleu et en rouge, panneaux de messages dynamiques (VMS), etc. Une boucle de détection positionnée sur la voie M2 est supervisée par l'*Agent Environnement* #1. Les voies L2 et L4 sont équipées de panneaux VMS qui informent les conducteurs parcourant ces voies de l'état du trafic en aval et de la durée probable de toute congestion qui y serait détectée.

Au temps de simulation correspondant à la figure 4.14 le trafic vient tout juste de changer du régime 'light free flow' au régime 'congested flow' en raison de la fermeture du centre d'achat à proximité de la voie M2. Ce changement de contexte a été détecté par le module d'Estimation de l'AC gérant cette portion du réseau. L'estimation obtenue indique que la perturbation sera de courte durée. Bien qu'éphémère, ce nouveau régime pourrait néanmoins affecter la sécurité des voyageurs en amont qui s'acheminent vers M2. La décision de l'AC et du mécanisme décisionnel qu'il incorpore n'est jamais simplement réactive. Le cadre ITS initie plutôt une décision en deux temps et contrainte dans le temps (échéance t4 de la figure 4.11). Dans cette ronde de simulation particulière, l'AC décide que le reroutage des véhicules n'est pas indiqué et

opte plutôt pour une mesure d'atténuation qui consiste à informer les conducteurs se dirigeant vers M2 que cette voie est sujette à une condition de trafic lourd. Cette action est relayée par l'AC aux agents d'environnement liés aux panneaux d'affichage dynamique positionnés en amont sur les voies L2 et L4.

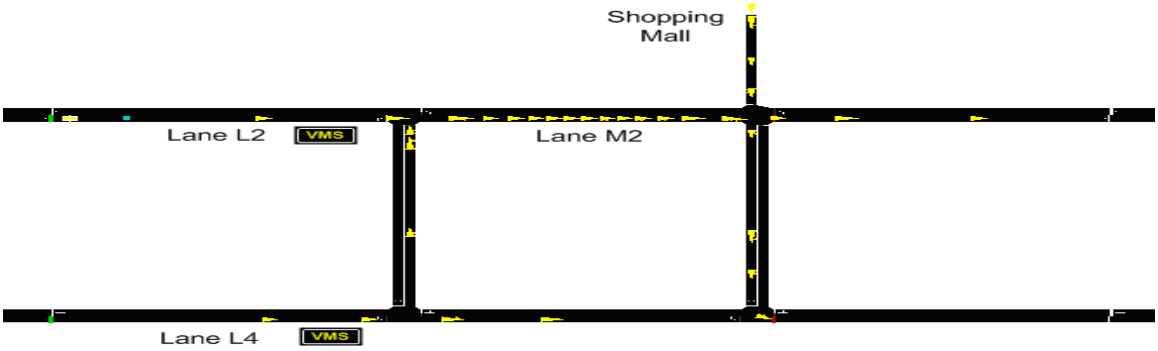

Figure 4.14 Congestion détectée sur la voie M2.

La figure 4.15 présente la séquence décisionnelle impliqué dans ce scénario. Notons que dans ce diagramme, les éléments *Estimation, Traitement, Décision, Base de règles de tolérance* et *Archive de Fronts de Pareto* illustrés par les diagrammes de séquence des figures 4.6 et 4.8 sont tous contenus dans l'Agent Complexe. À l'index temporel *t1*, l'AC détecte le changement de régime, un changement qui risque de faire évoluer le réseau vers un régime congestionné. Cette condition est jugée non sécuritaire [20]. En *t2,* le module *Estimation* fournit les paramètres supplémentaires tels l'intensité et la durée probable de la perturbation. L'AC déclenche alors en *t3* une décision multicritères afin d'optimiser la configuration de son architecture interne de traitement (c.a.d. de son pipeline). Plusieurs solutions sont évaluées, un front de Pareto et la meilleure solution sont identifiés à *t4*. La seconde étape de la décision s'amorce alors et décide de mettre en œuvre l'action de reconfiguration du réseau. Cette action est transmise aux agents d'environnement #2 et #3 (non détaillés sur le diagramme de séquence) afin d'afficher un message de redirection sur les voies L2 et L4.

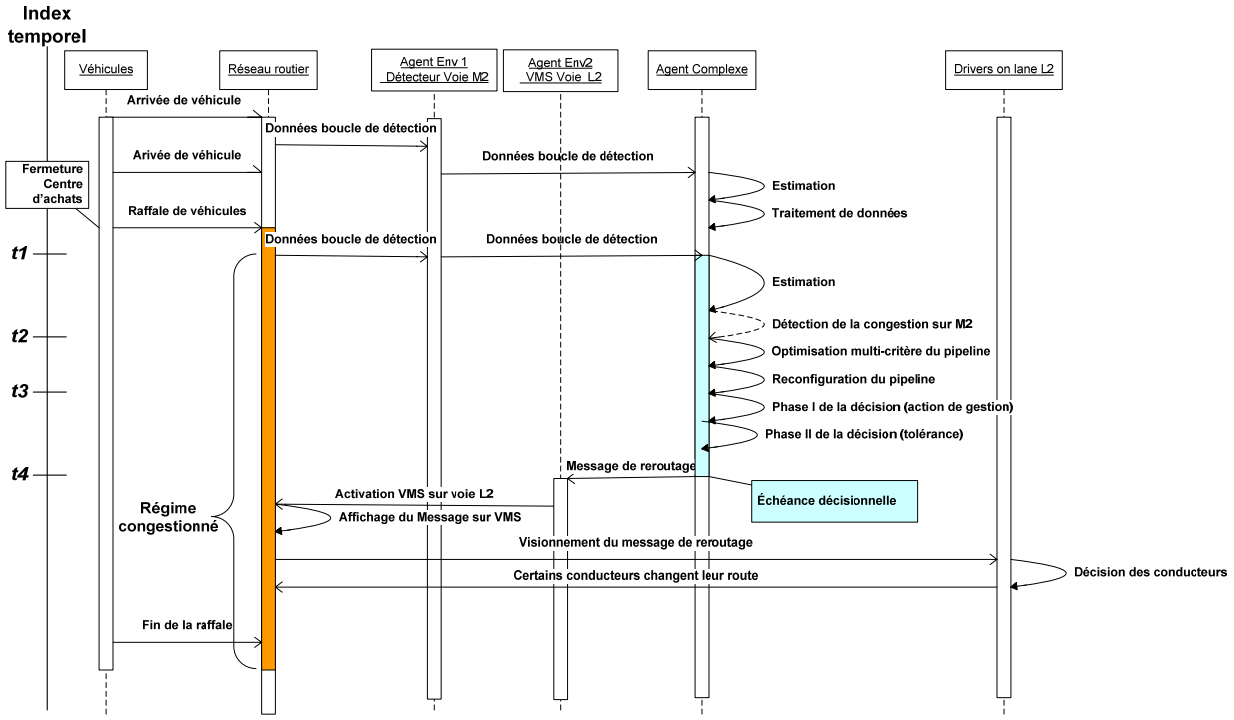

Figure 4.15 Diagramme de séquence du mécanisme décisionnel

4.5.6 Conclusion de l'étude de cas

Une décision réactive, pré-calculée en fonction de données historiques, sera hors contexte relativement à une perspective dynamique, et pour cette raison risque généralement d'être sous-optimale. Dans le scénario présent, le reroutage des véhicules en amont n'est pas nécessaire et peut de fait introduire de l'instabilité dans d'autres sections du réseau routier. Les résultats présentés à la figure 4.16 démontrent en effet que pour une intensité de perturbation située entre 50 et 70 véhicules introduits brusquement dans le réseau près de M2, une décision de reroutage exposera davantage de véhicules en amont à une condition de congestion qu'en l'absence de reroutage, avec pour conséquence que les risques globaux de collision en seront accrus. En outre, il est probable que les conducteurs qui sont soumis à des directives de circulation erronées opteront éventuellement d'ignorer ces messages et adopteront ainsi un comportement imprévisible et donc plus risqué.

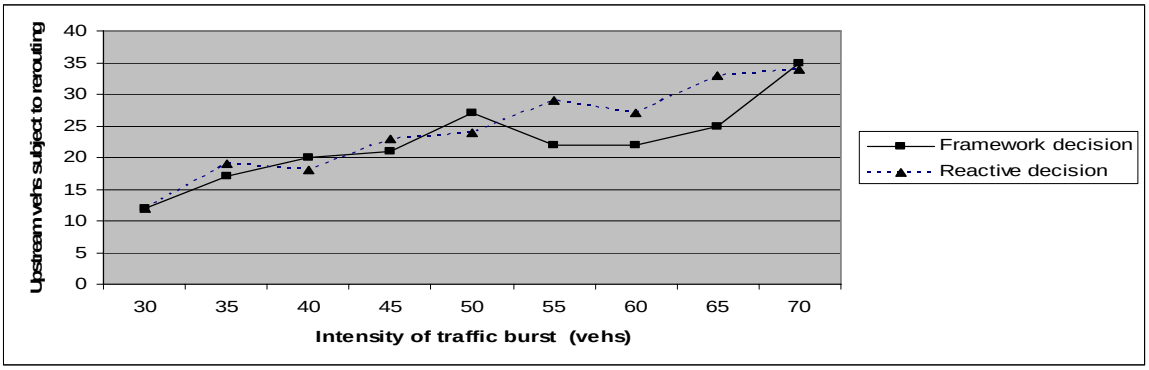

Figure 4.16 Nombre de véhicules en amont se dirigeant vers une congestion en aval.

Le cas d'essai présenté à la section 4.5 confirme l'applicabilité de ce cadre et de son mécanisme décisionnel à des domaines spécifiques tel que celui de la sécurité dans les systèmes de transport intelligents. Nous constatons que la stratification en trois couches d'agents procure la flexibilité nécessaire pour s'ajuster à différentes approches de simulation (ex. SUMO, J-Sim), la simulation étant une étape fondamentale de la mise en œuvre du cadre. En effet, la couche "Agent Environnement" peut aisément se spécialiser à un domaine applicatif spécifique, puis faire la transition du contexte de simulation vers le contexte d'exploitation.

Chapitre 5 Conclusion, contributions et travaux futurs

5.1 Conclusion

Les systèmes distribués opérant dans les réseaux étendus modernes sont habituellement tenus de fournir un niveau convenu de qualité de service à leurs usagers, indépendamment, dans la mesure du possible, des changements de comportement de leur environnement. L'adaptation dynamique par la reconfiguration de l'architecture de traitement est une stratégie reconnue pour le maintien de cette performance lorsque le comportement de leur environnement subit des perturbations. L'approche par la reconfiguration dynamique fait face à deux difficultés importantes.

La première difficulté est d'éviter que la reconfiguration introduise une dégradation supplémentaire de la performance ou pire, une déstabilisation du système si les comportements changent trop rapidement.

La seconde difficulté découle du fait qu'aucun des comportements passés de l'environnement ne correspond parfaitement bien à celui de l'environnement actuel. Nous avons souligné qu'il existe des correlations entre les événements de l'environnement selon des patrons qui singularise chaque comportement et que la performance d'une configuration donnée est étroitement liée à ces comportements. L'identification de l'environnement répertorié le plus similaire à l'environnement actuel est donc en fait une approximation et la sélection d'une solution architecturale qui en résulte ne sera donc probablement pas optimale.

Le présent projet a démontré qu'il est possible de contrer ces deux difficultés en définissant une gestion adéquate du mécanisme de reconfiguration qui intégre au processus décisionnel des entrées dynamiques et qui module sa décision avec des règles de tolérance. Nous avons présenté une approche décisionnelle employant l'optimisation multicritères pour trouver à priori les solutions architecturales de traitement les mieux adaptées à des comportements répertoriés. Chaque comportement étant unique, nous avons démontré comment l'inclusion d'entrées dynamiques nous permet de raffiner la connaissance de l'environnement actuel et donc de prendre des décisions de meilleure qualité grâce à des informations plus complètes.

5.2 Contributions

La principale contribution de ce projet, propre au domaine des mécanismes décisionnels, est l'introduction du concept de **tolérance** lors la prise de décision. Ainsi :

> *Dans notre mécanisme décisionnel, la décision multicritères de reconfiguration, modulée par des règles de tolérance, ne mène pas toujours à une reconfiguration.*

Nous avons démontré et illustré par un cas d'essai que le mécanisme décisionnel produisant une sous-décision de tolérance peut effectivement produire une performance supérieure à une décision de reconfiguration précalculée et réactive, comme l'illustre le cas d'essai avec un scénario de redirection des véhicules en situation de congestion temporaire dans un réseau routier.

Une contribution secondaire intéressante, propre au domaine du génie logiciel, est la démonstration de facto que :

> *L'implantation actuelle du cadre de gestion et de son algorithme PEAP est générique et s'ajuste aisément à différents domaines applicatifs sans changement significatif au niveau du code.*

Ceci résulte en partie de la segmentation selon trois couches fonctionnelles (Agent Superviseur, Agent Complexe et Agent Environnement) qui isolent les spécificités physiques du domaine applicatif dans les Agents Environnements et les spécificités architecturales et décisionnelles propres à un domaine d'application dans des classes spécialisées de l'algorithme d'optimisation PEAP. De plus le cadre facilite l'intégration de simulateurs spécialisés (ex. J-Sim pour les applications réseaux de télécommunications, SUMO pour les réseaux de transport routier) dans la mesure où ces simulateurs possèdent une interface de contrôle accessible à Java ou via le protocole TCP/IP. Conséquemment, le cadre peut être déployé dans un contexte opérationnel en substituant l'environnement réel à l'environnement simulé et en remplaçant les agents environnements concernés.

.

5.3 Travaux futurs

Une fois déployé dans un contexte opérationnel, le mécanisme décisionnel dispose d'une librairie de fronts de Pareto essentiellement statique et qui n'évolue pas au fil des nouveaux comportements rencontrés. Il serait intéressant d'explorer en continuité du présent projet une approche pour ajouter une capacité d'apprentissage en ligne permettant de faire évoluer la librairie de Pareto sans avoir à resoumettre le cadre à la phase hors ligne de calibration sur banc d'essai.

D'autre part, on peut aussi envisager d'explorer et de tester des alternatives à l'algorithme PEAP. On suppose qu'il peut exister une corrélation quelconque entre un domaine applicatif et un type d'algorithme évolutif et de représentation génétique. Les domaines de la programmation génétique et évolutionnaire sont en pleine effervescence et de nouveaux algorithmes plus performants sont conçus régulièrement. Certains chercheurs s'intéressent présentement à une catégorie d'algorithmes évolutionnaires qui dans certaines ciconstances convergent très rapidement (moins de dix générations) vers une solution optimale. On pourrait envisager que de tels algorithmes puissent constituer la base d'un processus d'apprentissage. Dans ce cas, non seulement la librairie pourrait évoluer au fil de nouveaux environnements rencontrés mais elle pourrait aussi s'ajuster automatiquement suite à l'ajout de nouveaux composants de traitements ou de versions plus performantes de composants existants.

Bibliographie

[1] C.W. Ahn, S. Oh et R.S. Ramakrishna. *On the Practical Genetic Algorithms*. In Proceedings of the 2005 conference on Genetic and evolutionary computation (GECCO'05), pages 1583-1584, Washington, DC, USA, juin 2005.

[2] O. Angin, A.T. Campbell, M. E. Kounavis et R. R.-F. Liao. *Open Programmable Mobile Networks*. In Proceedings of the Eighth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'98), Cambridge, England juillet 1998.

[3] I. Benyahia et M. Hilali. *An Adaptive Framework for Distributed Complex Applications Development*. IEEE TOOLS USA 2000 International Conference on Technology of Object-Oriented Languages and Systems, pages 339-349, 2000.

[4] I. Benyahia et P. Kouamé. *Design Methodology For An Adaptive ATM Switch*. 7th IFAC Symposium on Cost Oriented Automation (COA 2004), pages 301-304, 2004.

[5] I. Benyahia, J.-Y. Potvin et Y. Xu. *A Complex Applications Framework Supporting Tolerant Dynamic Vehicle Dispatching*. IEEE International Conference on Computer Systems and Applications, 2006 (AICCSA'2006), pages 875-881, 2006.

[6] I. Benyahia et V. Talbot. *Optimizing the Architecture of Adaptive Complex Applications Using Genetic Programming*. In Proceedings of the Fourteenth International Conference on Distributed Multimedia Systems (DMS 2008), pages 27-31, Boston, 2008.

[7] A. Berro. *Optimisation multiobjectif et stratégies d'évolution en environnement dynamique*. Thèse de doctorat. Université des Sciences Sociales, Toulouse, 2001.

[8] L. Bianchi, M. Birattari, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria et T. Schiavinotto. *Hybrid Metaheuristics for Vehicle Routing Problem With Stochastic Demands*. Technical report no IDSIA-06-05. Dalle Molle Institute for Artificial Intelligence, Switzerland, 2005.

[8a] C. Bohoris, G. Pavlou et A. Liotta. *Mobile Agent-based Performance Management for the Virtual Home Environment*. Journal of Network and System Management (JNSM), Vol. 11, No. 2, pages 133-149, Plenum Publishing, juin 2003.

[9] G. S. Blair, T. Coupaye et J.-B. Stefani. *Component-based architecture: the Fractal initiative*. Annales des Télécommunications, volume 64, no 1-2: pages 1-4, 2009.

[10] E. Bruneton, T. Coupaye et M. Leclercq. *The FRACTAL component model and its support in Java*. Software – Practice and Experience (SP&E), volume 36, no 11-12, pages 1257-1284, 2006.

[11] I. Cardei, R. Jha, M. Cardei et A. Pavan. *Hierarchical Architecture For Real-Time Adaptive Resource Management*. The IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, volume 1795, pages 415-434, New York, 2000.

[12] M. Castaldi, A. Di Marco et P. Inverardi. *Data driven reconfiguration for performance improvements: a model based approach*. In Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS '04), Edinburgh, Scotland, UK, 2004.

[13] D. W. Coit et A. E. Smith. *Penalty Guided Genetic Search For Reliability Design Optimization*. Computers and Industrial Engineering. Volume 30, pages 895-904, 1996.

[14] N. Colson, A. Kountouris, A. Wautier et L. Husson. *Autonomous Decision Making Process for the Dynamic Reconfiguration of Cognitive Radios*. ICCCN 2008, pages 497-502, Virgin Island USA, 2008.

[15] L. Console, C. Picardi et D. Theseider Dupré. *Temporal Decision Trees: Model-based Diagnosis of Dynamic Systems On-Board*. Journal of Artificial Intelligence Research 19, pages 469-512, 2003.

[15a] N. Davies, A. Friday, G. Blair et K. Cheverst. *Distributed Systems Support for Adaptive Mobile Applications*. ACM Mobile Networks and Applications, Special Issue on Mobile Computing System Services, Vol. 1, No. 4., 1996.

[16] C. Dellarocas, M. Klein et H. Shrobe. *An Architecture for Construction Self-Evolving Software Systems*. In Proceedings of the Third International Software Architecture Workshop, pages 29-32, 1998.

[17] J. E. Fieldsend. *A Multi-Objective Algorithm based upon Particle Swarm Optimisation, an Efficient Data Structure and Turbulence*. In Proceedings U.K. Workshop Computational Intelligence (UKCI'02), pages 37-44, Birmingham, U.K. Sept 2-4, 2002.

[18] A. Globus, S. Atsatt, J. Lawton et T. Wipke. *JavaGenes: Evolving Graphs with Crossover*. Publié sur http://www.nas.nasa.gov/~globus/papers/JavaGenes/paper.html, 2000.

[19] L. Godard. *Distributed and Hierarchical Management Modeling for Reconfiguration and Decision Taking of Cognitive Radio Equipments*. Thèse de doctorat, IETR - Institut d'Electronique et de Télécommunications de Rennes, 2008.

[20] T.F. Golob et W.W. Recker. *A Method for Relating Type of Crash to Traffic Flow Characteristics on Urban Freeways*. Transportation Research Part A: Policy and Practice, volume 38, no 1, pages 53-80, janvier 2004.

[21] H. Gomaa. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2000.

[22] O. Gonzalez, H. Shrikumar, J.A. Stankovic et K. Ramamritham. *Adaptive fault-tolerance and graceful degradation under dynamic hard real-time scheduling*. In Proceedings of the 18th IEEE Real-Time Systems Symposium, pages 79--, 1997.

[23] R. S. Gray, G. Cybenko, D. Kotz, R.A. Peterson et D. Rus. *D'Agents: Applications and performance of a mobile-agent system*. Software: Practice and Experience, volume 32, pages 515-541, 2002.

[24] J. Hillman et I. Warren. *An Open Framework for Dynamic Reconfiguration*. In Proceedings of 26th International Conference on Software Engineering (ICSE'04), pages 594-603, 2004.

[25] M. A. Hiltunen et R. D. Schlichting. *Adaptive distributed and fault-tolerant systems*. International Journal of Computer Systems Science and Engineering, volume 11, pages 125--133, Septembre, 1996.

[26] R. Hubley. *Multiobjective Analyzer for Genetic Marker Acquisition*. Publié sur http://snp-magma.sourceforge.net, 2002.

[27] L. Ismail et D. Hagimont. *A performance evaluation of the mobile agent paradigm*. In Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications, pages 306-313, 1999.

[28] ITS Canada. Regional ITS Architecture Guidance for Canada. Publié sur http://www.tc.gc.ca/innovation/its/eng/architecture/documents/regional_its_architecture_guidance/pdf/regional_its_architecture_guidance.pdf , 2010.

[29] Esko K. Juuso. *Integration of Intelligent Systems in Development of Smart Adaptive Systems*. International Journal of Approximate Reasoning, volume 35, no. 3, pages 307-337, 2004.

[30] H. Katagiri, K. Hirasawa, K., Jinglu Hu et J. Murata. *Comparing Some Graph Crossover in Genetic Network Programming*. SICE 2002. In Proceedings of the 41st SICE Annual Conference, pages 1263-1268, Osaka, 2002.

[30a] J.R. Koza. *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*, Stanford University Computer Science Department technical report STAN-CS-90-1314 ,1990.

[31] J.Kramer et J.Magee, *Dynamic Configuration for Distributed Systems*. IEEE Transactions on Software Engineering, SE-11 (4), pages 424-436, 1985.

[32] O. Layaïda, S. Ben Atallah et D. Hagimont. *Reconfiguration-based QoS Management in Multimedia Streaming Applications*. In Proceedings of the 30th IEEE/EUROMICRO Conference, Track on "Multimedia & Telecommunications: Challenges in Distributed Multimedia Systems", pages 248-25, Rennes, 2004.

[33] B. Li et K. Nahrstedt. *A control theoretical model for quality of service adaptations*. (IWQoS 98) Sixth International Workshop on Quality of Service, pages 145-153, 1998.

[34] B. Li, W. Jeon, W. Kalter, K. Nahrstedt et J.-H. Seo. *Adaptive Middleware Architecture for a Distributed Omnidirectional Visual Tracking System*. In Proceedings of SPIE Multimedia Computing and Networking 2000, pages 101-112, 2000.

[35] C. Lu, J.A. Stankovic, T.F. Abdelzaher, G. Tao, S.H. Son et M. Marley. *Performance Specifications and Metrics for Adaptive Real-Time Systems*. In Proceedings of the 21st IEEE conference on Real-time systems symposium, pages 13-23, Washington, DC, USA, 2000.

[36] S. Luke, L. Panait, G. Balan, S. Paus, Z. Skolicki, E. Popovici, K. Sullivan, J. Harrison, J. Bassett, R. Hubley, A. Chircop, J. Compton, W. Haddon, S. Donnelly, B. Jamil, J. Zelibor, E. Kangas, F. Abidi, H. Mooers et J. O'Beirne. *ECJ 20, A Java-based Evolutionary Computation Research System*. Publié sur http://cs.gmu.edu/~eclab/projects/ecj/ , 2010.

[36a] N. Migas, W.J. Buchanan et K.A. McArtney. *Mobile Agents for Routing, Topology Discovery, and Automatic Network Reconfiguration in Ad-Hoc Networks*. 10th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03), pages 200-206, Huntsville, Alabama, 2003.

[37] P. McKinley, C. Tang, et A. Mani. *A Study of Adaptive Forward Error Correction for Wireless Collaborative Computing*. IEEE Transactions on Parallel and Distributed Systems, volume 13, pages 936-947, Septembre 2002.

[38] P. K. McKinley, S.M. Sadjadi et R.E.K. Stirewalt. *A Taxonomy of Compositional Adaptation*. Technical Report MSU-CSE-04-17, Department of Computer Science and Engineering, Michigan State University, 2004.

[39] D. Nicol et J. P.F. Reynolds. *Optimal Dynamic Remapping of Data Parallel Computations*. IEEE Trans. Computers, volume 40, no. 3, pages 295-306, 1991.

[40] M. O'Neill, C. Ryan, M. Keijzer et M. Cattolico. *Crossover in Grammatical Evolution*. Genetic Programming and Evolvable Machines, volume 4, no. 1, Kluwer Academic Publishers Hingham, MA, USA, 2003.

[41] L. J. Osterweil, A. Wise, J.M. Cobleigh, L.A. Clarke et B.S. Lerner. *Architecting Dynamic Systems Using Containment Units*. Department of Computer Science, University of Massachusetts, Amherst, UM-CS-2002-029, 2002.

[42] M.-H. Pham, A. Bhaskar, E. Chung and A.-G. Dumont. *Toward a Risk-Sensitive Active Traffic Management Syste*m. The IET Road Transport Information and Control Conference and the ITS United Kingdom Members' Conferenc*e*, pages 1-9, London, May 25-27, 2010.

[43] R. Poli. *Evolution of Graph-Like Programs with Parallel Distributed Genetic Programming*. In Proceedings of the 7th International Conference on Genetic Algorithms, pages 346-353, East Lansing, MI, USA, juillet 1997.

[44] J.C. Pomerol. *From Aggregating By Rules to the Integration of Expert Systems in Multicriteria DSSs*. IEEE SMC, volume 1, pages 483-488, Le Touquest, France, 1993.

[45] J. Pomerol, P. Brézillon, L. Pasquier. *Operational Knowledge Representation for Practical Decision Making.* hicss, vol. 3, page 3023, 34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 3, 2001.

[46] S. Porcarelli, M. Castaldi, F. Di Giandomenico, A. Bondavalli et P. Inverardi. *A Framework for Reconfiguration-based Fault Tolerance in Distributed Systems*. 2nd book on Architecting Dependable Systems, Lecture Notes in Computer Science Series, LNCS 3069, Springer, pages 167-189, 2004.

[47] A. Rasche et A. Polze. *Dynamic Reconfiguration of Component-based Real-time Software*. In Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2005), pages 347-354, Sedona, Arizona, USA, février, 2005.

[48] D. Rosu, K. Schwan, S. Yalamanchili et R. Jha. *On Adaptive Resource Allocation for Complex Real-Time Applications*. In Proceedings of the 18th IEEE Real-Time Systems Symposium, pages 320--, décembre 1997.

[49] L. P Santos et A. Proenca. *A Bayesian RunTime Load Manager on a Shared Cluster*. In Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID'01), pages 674-679, 2001.

[50] E. Schneider, F. Picioroagacaron et U. Brinkschulte. *Dynamic reconfiguration through OSA+, a real-time middleware*. In Proceedings of the 1st international Doctoral Symposium on Middleware, pages 319-323, Toronto, Ontario, Canada, Octobre 2004.

[51] F. Siqueira et V. Cahill. *Quartz: A QoS Architecture for Open Systems*. In Proceedings of the The 20th International Conference on Distributed Computing Systems, pages 197--, 2000.

[52] M. Shoenauer. *Optimisation évolutionnaire multi-objectif*. Publié sur http://tao.lri.fr , Yravals, 2007.

[53] K. Sullivan, J.C. Knight, X. Du et S. Geist. *Information Survivability Control Systems*. In Proceedings of the 21st international conference on Software engineering, pages 184-192, 1999.

[54] *Simulation of Urban Mobility*. Institute of Transportation Systems. Publié sur http://sourceforge.net/apps/mediawiki/sumo/index.php?title=Main_Page , 2010.

[55] D. Sykes, W. Heaven, J.Magee et J. Kramer. *From goals to components: a combined approach to self-management*. In Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems, pages 1-8, 2008.

[56] S. Tak, P. Prathombutr et E. K. Park. *An efficient technique for a series of virtual topology reconfigurations in WDM optical networks*. Computer Communications, volume 30, no 6, pages 1301-1314, 2007.

[57] C.K. Toh et Wei K. Tsai. *Intelligent Control of QoS Adaptation in Next Generation Wireless Broadband Networks*. Journal of High Speed Networks, volume 10, pages 37-58, 2001.

[58] H.-Y. Tyan, A. Sobeih et J. C. Hou. *Design, Realization and Evaluation of a Component-based, Compositional Network Simulation Environment*. Simulation, volume 85, no 3, pages 159-181, 2009.

[59] I. Warren, J. Sun, S. Krishnamohan, T. Weerasinghe. *An Automated Formal Approach to Managing Dynamic Reconfiguration*. In Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, pages 37-46, 2006.

[60] M. Wermelinger. *A hierarchic architecture model for dynamic reconfiguration*. In Proceedings of the Second International Workshop on Software Engineering for Parallel and Distributed Systems, pages 243-254. IEEE Computer Society Press, 1997.

[61] T1A1.2 Working Groupd on Network Survivability Performance. *A Technical Report on Network Survivability Performance*. 1993.

[62] D. Paul, S. Yalamanchili, K. Schwan et R. Jha. *Decision models for adaptive resource management in multiprocessor systems*. Publié sur http://www.htc.honeywell.com/projects/arm/ , 1998.

[63] E. Zitzler, M. Laumanns et L. Thiele. *Spea2: Improving the strength Pareto evolutionary algorithm*. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, 2001.