Analyse des critères d'évaluation de systèmes multi-agents adaptatifs

E. Kaddoum* kaddoum@irit.fr
M.P. Gleizes* gleizes@irit.fr
J.P. Georgé* george@irit.fr
P. Glize* pierre.glize@upetec.fr
G. Picard◇ picard@emse.fr

* Institut de Recherche en Informatique de Toulouse,
Université Paul Sabatier - Toulouse III, France

◇ SMA@G2I,
École Nationale Supérieure des Mines de Saint-Étienne, France

Résumé

La complexité croissante des applications actuelles favorise le développement de systèmes multi-agents auto-organisateurs possédant des propriétés self- . Ces systèmes autonomes présentent des capacités intéressantes permettant la gestion de la dynamique endogène et exogène des applications étudiées. De nouveaux critères doivent être étudiés afin de caractériser et évaluer l'apport de ces propriétés self-* et leur influence sur les performances du système. Dans cet article, différentes catégories regroupant les principaux critères d'évaluation sont décrites afin de guider l'évaluation de ce type de systèmes depuis les phases de conception jusqu'aux phases d'exécution : évaluation du système en cours de fonctionnement, caractéristiques intrinsèques et méthodologie de conception.*

Mots-clés : *Système self-* , caractérisation, évaluation empirique, comparaison*

Abstract

In the last few years, the growing complexity of current applications has led to design self-organizing multi-agents systems presenting self- properties. Those systems behave autonomously and must handle the dynamics coming from exogenous or endogenous changes. New or updated characterization and evaluation criteria are required for analyzing the contribution of self-* properties and system performances. This paper aims at guiding the evaluation of this kind of systems from the design phase all the way to the execution results by providing sets of main criteria : evaluation of the system at run-time, intrinsic characterization and development methodologie.*

Keywords: *Self-* systems, characterization, empirical evaluation, comparison*

1 Introduction

La complexité croissante des logiciels favorise la conception de systèmes multi-agents auto-organisateurs présentant des propriétés self-* . Ces systèmes se composent d'un ensemble d'agents autonomes et interagissants, conduisant à l'émergence du comportement collectif. Grâce à leurs propriétés self-* , ces systèmes sont capables d'auto-adaptation et gèrent les dynamiques dues aux changements endogènes et exogènes [34, 1]. Parmi ces propriétés, nous distinguons :

- *l'auto-organisation :* processus par lequel un système change, lors de son exécution, son organisation sans aucune intervention externe [9]. L'auto-configuration permettant au système de devenir conforme aux besoins de l'application, en est un exemple ;
- *l'auto-stabilisation ou homéostasie :* propriété assurant l'atteinte d'un état stable par le système [35] ;
- *l'auto-réparation :* mécanisme permettant la détection et la réparation des erreurs ;
- *l'auto-régulation :* mécanisme permettant l'ajustement autonome des paramètres.

Télécommunications, économie, bio-informatique sont autant de domaines fortement dynamiques avec l'apparition d'événements imprévus. Les systèmes classiques supportent difficilement ces caractérisques mais, contrairement aux systèmes self-* , des preuves formelles concernant leur performance existent. De plus, le fonctionnement de systèmes classiques peut être validé avant même leur déploiement dans des environnements réels. De manière générale, les systèmes classiques sont fonctionnellement adéquats (ils réalisent ce pour quoi ils ont été conçus). Les questions qui se posent à nous sont : comment peut-on mettre en avant les systèmes self-* par rapport aux systèmes classiques sachant leur indéterminisme ? Ou, comment les propriétés self-* garantissent-elles

un bon fonctionnement malgré les perturbations ? Comment prouve-t-on l'adéquation fonctionnelle de ces systèmes ? Puisqu'aucune preuve formelle n'existe pour ce type de système [14, 13, 19], cet article se concentre sur *l'évaluation empirique* qui est primordiale pour notre communauté. Le but étant de souligner les avantages et les caractéristiques des systèmes self-* par rapport aux autres systèmes.

Les rares études et évaluations des systèmes self-* se basent sur un nombre réduit de critères tels que le temps et la charge de communication. Malgré leur importance, ces critères restent insuffisants pour refléter la spécificité de ces systèmes. D'autres critères comme l'homéostasie, le degré de décentralisation, la complexité, etc. sont plus appropriés et doivent être considérés. L'objectif de cet article est de recueillir et analyser les critères essentiels d'évaluation de ces systèmes auto-adaptatifs.

Étant donnée leur diversité, de nouvelles méthodes de caractérisation et d'évaluation sont nécessaires pour étudier leur performance et analyser la contribution des propriétés self-* . Dans ces systèmes, deux types de comportements sont à distinguer : le comportement nominal (N) représentant le fonctionnement normal du système et le comportement self-* (S) permettant la gestion de la dynamique. Cet article traite de l'évaluation de ce type de système depuis les phases de conception jusqu'à leur exécution, en séparant, quand c'est possible, le comportement nominal du comportement self-* .

Dans ce qui suit, nous avons classé les critères identifiés en trois parties : la première concerne l'évaluation du système en cours de fonctionnement, la deuxième étudie ses caractéristiques intrinsèques et la troisième considère la méthodologie de conception. Pour chaque critère, nous donnons une définition et nous fournissons un moyen pour le mesurer. Concernant les comparaisons, nous analysons un ensemble de courbes illustrant des comportements typiques des systèmes self-* .

2 Évaluation du fonctionnement du système

Afin de valider le bon fonctionnement d'un système, différentes questions sont à étudier : est-ce que le système est capable de résoudre le problème pour lequel il est conçu ? Est-ce que le système s'auto-adapte ? Est-ce que les propriétés self-* implémentées sont suffisantes pour garantir la robustesse du système ? etc. Pour y répondre, nous considérons les résultats finaux ainsi que le comportement du système en cours d'exécution. Les résultats permettent d'évaluer les performances du système, mais détecter la fin d'une exécution d'un système auto-adaptatif reste difficile. Ces systèmes sont conçus pour s'adapter continuellement à leur environnement et ne jamais s'arrêter. De plus, la nature distribuée de la résolution rend cette détection impossible par le système lui-même, même si un état stable est atteint. Pour cela, des agents observateurs détectant l'atteinte d'états stables et l'arrêt peuvent être envisagés.

2.1 Performances

Dans cette section, nous présentons les critères d'évaluation concernant les performances du système. Les résultats concernant différentes exécutions du système pour un scénario donné sont étudiés. Nous considérons les six critères suivants : le temps, la charge de communication, l'indéterminisme, la précision / qualité, la progression et l'utilisation de la mémoire.

Temps. Différentes mesures ont été définies pour étudier ce critère. [29] utilise le nombre d'opérations atomiques réalisées comme par exemple, le nombre de contraintes vérifiées dans un problème de satisfaction de contraintes. D'autres auteurs mesurent le temps CPU [8, 20, 28] du début de l'exécution jusqu'à la fin. Mais dans ce cas, les caractéristiques des machines utilisées doivent être prises en compte. [10] introduit des ordres de grandeur permettant d'unifier le temps CPU de différentes machines. D'autres études [8, 25] considèrent le nombre de cycles nécessaires aux agents pour atteindre une solution. Une autre définition consiste à mesurer le temps des opérations les plus coûteuses. De notre point de vue, le nombre de cycles nécessaires aux agents pour atteindre une solution est le moyen le plus approprié et facile pour mesurer le temps. En effet, mesurer le temps CPU doit prendre en compte les caractéristiques non seulement des machines mais aussi des plateformes de simulation utilisées. Compter le nombre d'opérations atomiques et mesurer le temps mis par les opérations les plus coûteuses, quant à eux, doivent considérer l'asynchronisme de résolution entre les agents [29].

Le temps d'exécution fut l'un des premiers critères à être utilisé pour l'évaluation des systèmes. Au début des systèmes distribués, le dilemme était de savoir si le système passait plus de temps à communiquer qu'à traiter les don-

nées. Avec les systèmes self-* , ce dilemme se traduit en la séparation entre le temps de travail et d'auto-adaptation. Ainsi, nous proposons quand ceci est possible, de distinguer deux types de temps : NT_s et ST_a. NT_s mesure le temps d'exécution du comportement nominal du système. ST_a mesure le temps mis par le système pour s'auto-adapter aux changements de l'environnement. Par exemple, dans un serveur de mail possédant des propriétés d'auto-réparation, nous pouvons séparer le temps où le serveur exécute son comportement nominal (traitement des messages envoyés et reçus) et le temps où il demande de l'aide suite à une surcharge. Quand cette distinction n'est pas possible, nous proposons d'évaluer la moyenne des différences du temps d'exécution MTD_{ns} du système avec deux scenarii, un sans perturbations et un avec perturbations. Les problèmes de satisfaction de contraintes sont un exemple où cette séparation est difficile [37]. Certains considèrent que le temps passé par l'agent pour affecter sa variable est le NT_s alors que d'autres le considèrent comme étant le ST_a.

Charge de communication. Dans les systèmes multi-agents, les agents communiquent et interagissent afin d'atteindre collectivement une solution ou afin de s'organiser pour résoudre un problème donné. Étudier la charge de communication permet de déterminer la capacité du réseau, en environnement réel. Il permet de connaître la charge du réseau et ainsi éviter de le surcharger. Le nombre de messages échangés est un bon moyen pour mesurer ce critère. Il peut être divisé en deux catégories : le nombre de messages NCL_s pour les phases de résolution nominales et le nombre de message SCL_a pour les phases d'auto-adaptation. Par exemple, dans un serveur de mail possédant des propriétés d'auto-réparation, nous pouvons distinguer entre les messages d'envoi et de réception de mail, et les messages de demandes d'aide quand le serveur est surchargé. Le nombre de messages échangés est fortement lié au degré de décentralisation au sein du système. En général, plus le degré de décentralisation est élevé, plus les agents on besoin d'échanger de messages. Alors que dans les systèmes centralisés, très peu de messages sont nécessaires. Pour cela, ces deux critères (nombre de messages, degré de décentralisation) doivent être étudiés ensemble. Malheureusement, très peu d'études les combinent. D'autres paramètres sont aussi intéressants pour évaluer les charges de communications [4], tels que la taille des messages et leur complexité. Ils permettent de mesurer précisément la charge du réseau et d'évaluer le temps nécessaire pour les agents pour comprendre et traiter ces messages.

Précision / qualité. Ce critère permet d'étudier l'adéquation fonctionnelle d'un système donné. Un système fonctionnellement adéquat est défini comme étant un système utile [32]. Une mesure appropriée pour ce critère est la distance entre la solution obtenue et la solution optimale $\delta(S_{cur}, S_{opt})$. Mais un problème majeur se pose : comment peut-on déterminer cette solution optimale ? Dans certains cas, cette solution est connue pour certains scenarii ou peut être facilement calculée, mais dans le cas général, elle est inconnue. Afin d'y remédier, certains auteurs utilisent des *benchmarks* bien connus et comparent leurs solutions avec les meilleures solutions connues. La qualité de l'auto-adaptation peut être soulignée par l'utilisation de différents scenarii présentant des degrés différents de dynamiques – le but étant de montrer la capacité du système à maintenir un niveau de qualité élevé.

Indéterminisme. Au sein des systèmes multi-agents auto-organisateurs, les informations sont distribuées entre les agents qui possèdent chacun une vision locale de leur environnement. Ces caractéristiques intéressantes pour le traitement des problèmes complexes, rendent la résolution non déterministe. Pour un scénario donné, la solution peut différer d'une exécution à une autre, ce qui soulève le problème de l'exécution d'une série de simulations [16]. Étudier ce non déterminisme en mesurant la distance entre ces différentes solutions renseigne sur la stabilité du système, et facilite sa comparaison avec les approches exactes de résolution. Des mesures de tendance centrale telles la moyenne ou la médiane et des mesures de dispersion telles l'écart-moyen ou la variance sont généralement utilisées [7].

Progression. Ce critère, noté NP_s, permet de savoir comment le système étudié atteint une solution. Pour cela, nous mesurons le pourcentage du but atteint à chaque étape. Le progrès concerne l'activité du système. Aucune connaissance de la solution optimale n'est requise. Par exemple, dans [8], le progrès est mesuré comme étant le nombre total de conteneur usinés à la fin de chaque étape. Ce critère doit être étudié tout au long de l'exécution sous forme de courbes représentant l'activité du système. Ces courbes auront des formes spécifiques (linéaire, exponentielle, logarithmique) (cf. section 5). Mais comment l'auto-adaptation influence cette progression : est-ce qu'elle la maintient ou l'améliore (situation idéale), la ralentit, voire la diminue. La progression SP_a concernant l'auto-adaptation suite à une perturbation peut être éva-

luée de la même façon que le NP_s.

Utilisation de la mémoire. Un autre critère de performance consiste à mesurer le taux de mémoire utilisée par chaque agent pour les phases de résolution NM_s et les phases d'auto-adaptation SM_a. Ce critère influence directement la complexité algorithmique (cf. section 3.1). Par exemple, la complexité algorithmique d'un système où chaque agent doit traiter les informations provenant de tous les autres agents, est plus élevée que celle d'un système où chaque agent doit traiter les informations provenant d'un voisinage réduit. Le déploiement du système dans un environnement réel est aussi affecté par ce critère. C'est le cas des applications d'intelligence ambiante où des appareils ayant des capacités réduites de mémoire doivent être utilisés.

2.2 Homéostasie & Robustesse

Une des principales caractéristiques des systèmes multi-agents ayant des propriétés self-* , est leur capacité à maintenir leur fonctionnement dans des environnements hautement dynamique. En d'autres termes, ces systèmes possèdent des capacités de robustesse et d'homéostasie ou auto-stabilisation. La première concerne leur aptitude à maintenir leur comportement quand des perturbations se produisent [34]. La deuxième, quant à elle, est la capacité du système à retrouver, une fois perturbé, son comportement normal. [35] définit l'homéostasie comme étant la capacité à revenir dans un état idéal dans lequel le système est au maximum de ses performances. L'étude de ces deux critères permet de comprendre comment le système gère les changements dynamiques de son environnement et ainsi valider son adéquation fonctionnelle pour la résolution du problème.

Robustesse. Les problèmes complexes présentent un taux élevé de dynamique car des événements se produisent à des étapes différentes et sont souvent imprévisibles. Un système robuste est un système capable de maintenir un comportement stable malgré les perturbations. Pour mesurer ce critère, on peut étudier :

- si le système maintient son adéquation fonctionnelle sous perturbations ($\Delta Q \prec \epsilon$) ;
- si la différence entre le nouvel état (solution) atteint par le système et son ancien état est minimale ;
- si les changements intervenus au sein du système pour atteindre ce nouvel état sont minimaux.

Temps d'adaptation. Ce critère mesure le temps nécessaire pour le système pour retrouver un état stable après l'apparition d'une perturbation. En mesurant le temps (cf. section 2.1), nous avons introduit ST_a le temps mis par le système pour s'auto-adapter tout au long de son exécution. Ici, nous définissons le temps STA_{lc} nécessaire pour la prise en compte d'un type de changement dynamique. Le nombre d'étapes dont les agents ont besoin pour revenir à un état de fonctionnement normal est un moyen de mesure approprié pour ce genre de critères. Deux niveaux sont à distinguer : le niveau local et le niveau global. Le premier considère le nombre de cycles de vie de chaque agent. Il permet d'examiner l'auto-adaptation des agents. Pour le deuxième, c'est le nombre d'étapes nécessaires au système pour retrouver son comportement normal qui est mesuré. Il permet d'étudier l'auto-adaptation du système global (cf. figure 3).

Mesure pour l'évaluation du fonctionnement du système

- Le temps pour l'auto-adaptation ST_a, le temps de résolution NT_s, le temps d'adaptation STA_{lc} ou la moyenne des différences de temps entre différents degrés de perturbations MTD_{ns}.
- Le nombre de messages pour la résolution NCL_s et celui pour l'auto-adaptation SCL_s.
- La qualité de la solution ($\delta(S_{cur}, S_{opt})$) ainsi que le degré d'indéterminisme de résolution.
- La progression de la résolution NP_s et celle de l'auto-adaptation SP_a.
- Le taux de mémoire utilisée pour les phases de résolution NM_s et de celle utilisée pour les phases d'auto-adaptation SM_a .
- Le niveau de robustesse du système ($\Delta Q \prec \epsilon$).

3 Caractéristiques intrinsèques du système

Les critères présentés ci-dessus ne peuvent être mesurés que pendant ou après l'exécution du système. De ce fait, un nombre limité de scenarii ou de situations peuvent être envisagés. Malgré leur importance, ces critères ne garantissent pas les qualités du système pour toutes les situations et tout au long de son existence. Le but de cette section est de compléter la précédente en définissant et discutant comme critères d'évaluation certaines caractéristiques des

systèmes self-* . Ces critères peuvent être étudiés sans exécuter le système. Connaissant la nature du système et son fonctionnement, nous pouvons déduire des informations concernant sa complexité algorithmique, son taux de décentralisation, ses algorithmes locaux ou l'influence du nombre d'agents ainsi que leur besoin calculatoire. Ceci permet une première évaluation et comparaison de ces systèmes.

3.1 Complexité algorithmique

Dès les premiers systèmes informatiques[1] [15], les concepteurs ont cherché à mesurer le temps et la mémoire nécessaire à leur programme ou algorithme. Généralement, il est facile de prévoir l'exécution d'un algorithme classique dans un scénario du pire-cas et en étendant ses entrées à un problème réel, nous pourrons prédire l'échec de l'algorithme à fournir une réponse en un temps convenable. Ce problème d'extensibilité a été largement étudié durant les quarante dernières années et un cadre théorique[2] et formel a été développé [36]. Avec ces études, les notions bien connues mais difficiles à maîtriser d'*ordre de complexité* et de *NP-difficile* furent introduites. L'idée principale étant l'aveu de l'existence de problèmes impossibles à résoudre en un temps polynomial. Dans cet article, nous ne nous intéressons pas à la théorie de complexité, mais la question relative à l'évaluation des systèmes self-* qui est : comment peut-on dire que notre système est efficace en n'analysant que son code ? Dans le cas des systèmes ayant un contrôle global, les outils d'évaluation des algorithmes classiques peuvent être appliqués. Ceci est un avantage intéressant pour l'évaluation, mais un inconvénient majeur pour les systèmes s'attaquant à des problèmes avec des niveaux élevés de complexité.

3.2 Décentralisation et résolution locale

La plupart des systèmes self-* conçus pour la résolution des problèmes présentant des niveaux de complexité réelle, utilise la distribution du calcul, la décentralisation du contrôle et des mécanismes simples et locaux d'exploration de l'espace de recherche. Sachant qu'un seul changement local (aussi minime soit-il) peut avoir une répercussion non-linéaire au sein du système, comment peut-on calculer la complexité algorithmique du système dans son intégralité ? Des travaux sur la théorie de complexité des algorithmes de recherche locale existent [31] . Ils définissent la classe des recherches locales en temps polynomial (*PLS : Polynomial Time Local Search*) mais présentent des limites strictes.

Complexité algorithmique locale. Ce critère consiste à analyser la complexité algorithmique des comportements locaux de chaque agent, ou parties du système. Malgré sa simplicité, il permet déjà de disqualifier les algorithmes locaux nécessitant beaucoup de calculs.

Décentralisation. La complexité algorithmique peut être complétée par l'étude d'un ratio entre la décentralisation et le taux de calcul local nécessaire. D'un côté, un système ne présentant pas un degré de décentralisation approprié au problème traité, peut appartenir à la classe des problèmes NP-complets. D'un autre côté, dans un système présentant un degré important de décentralisation mais un haut niveau de complexité du calcul local, chaque décision locale peut devenir NP-difficile. En résumé, de combien d'agents a-t-on besoin ? Quel devrait être leur degré de distribution et leur simplicité ? Avec combien d'agents, un agent devrait interagir ? Ces deux critères doivent être étudiés simultanément. Si l'on prend par exemple, un algorithme local sélectionnant aléatoirement une action pour un agent, la complexité algorithmique est très faible, mais le système résultant est très basique. Par contre, ce critère peut être utilisé pour filtrer les algorithmes ayant une complexité très élevée. Par exemple, un algorithme où un agent doit vérifier les propriétés de tous les autres agents et de plus, prédire l'effet de son action après n étapes, est une approche naïve.

Effet des actions locales. Un autre critère intéressant à étudier est l'effet d'une action locale sur le reste du système. Les systèmes self-* qui en général sont des systèmes complexes, sont sujets à l'effet *papillon* bien connu au sein des systèmes utilisant des mécanismes *gossip* ou la propagation dans les réseaux de neurones [22]. Ce critère peut être étudié en examinant les stratégies de recherche de chaque agent. Est-ce qu'un agent diffuse (*broadcast*) systématiquement ces informations et ces besoins à tous les autres agents du système ? Ou procède-t-il par sélection des agents les plus appropriés ? Par exemple, les heuristiques de type *minconflict* [30] essaient de choisir, à chaque étape, la décision ayant un effet minimal, réduisant ainsi la complexité globale.

Connaissance initiale et acquise. Certains algorithmes requièrent un ensemble d'informations avant même le lancement de l'exécution.

[1] http ://people.cs.uchicago.edu/ fortnow/papers/history.pdf
[2] http ://www.math.upenn.edu/ wilf/AlgComp3.html

Par exemple, dans différentes métaheuristiques comme les approches ACO ou PSO, une analyse préliminaire de l'espace de solutions doit être effectuée afin de fournir les bons paramètres tels le nombre d'agents ou la probabilité de distribution caractérisant leur comportement [12]. D'autres algorithmes ont besoin d'acquérir des informations durant leur activité afin d'être efficace. L'étude de ce critère nous permet de distinguer ces deux types d'algorithmes et nous renseigne sur leur déploiement (les algorithmes appartenant à la deuxième catégorie étant plus difficiles à déployer).

Influence du nombre d'agents. Le nombre d'agents au sein d'un système affecte considérablement l'efficacité de ce dernier. C'est un critère important pour assurer l'optimalité de l'organisation, comme pas exemple, dans le déploiement d'agents satellites coûteux [3]. Plus intéressant pour les systèmes self-* , ce critère permet d'étudier le passage à l'échelle d'un système pouvant évoluer tout au long de son exécution. Il peut être mesuré lors du fonctionnement du système mais aussi en analysant comment ce dernier est construit : est-il facile d'introduire de nouveaux agents (système ouvert) ? Les tâches des agents peuvent-elles être facilement déléguées à d'autres agents ? Existe-t-il des capacités d'auto-organisation étendues ? etc.

Critères de complexité locale et de décentralisation
- Calculer la complexité locale des processus de décision.
- Évaluer le degré de décentralisation.
- Étudier l'effet d'une action d'un agent donné sur le reste du système.
- Évaluer la difficulté de fournir des connaissances initiales au système et d'en acquérir de nouvelles tout au long de son exécution.
- Analyser le passage à l'échelle de ces systèmes.

4 Caractérisation de la méthodologie de développement

Un autre point à analyser avant l'implémentation d'un système self-* est sa méthode de développement. Plus précisément, le processus à suivre ainsi que les concepts à manipuler tout au long des phases de développement (analyse, conception, implémentation, etc.). En effet, les différentes approches de conception ne sont pas équivalentes de ce point de vue. Les critères permettant d'étudier une méthode sont plus qualitatifs que quantitatifs, mais sont utiles pour les prochains développements basés sur la même approche. Parmi eux, nous distinguons les phases dédiées aux systèmes self-* , à savoir : l'identification des agents, la distribution, la généricité et la flexibilité.

Analyse et identification des agents autonomes. Durant les phases d'analyse, plusieurs éléments sont à prendre en compte, ce qui accentue la difficulté des tâches à réaliser. Dans certains cas, le développement est simple et direct, alors que dans d'autres, des efforts supplémentaires sont à fournir et des connaissances d'experts sont requises. Prenons comme exemple la résolution dans le cadre des problèmes de satisfaction de contraintes (CSP). Pour développer un solveur distribué de CSP utilisant le *Ditributed Breakout (DBA)* [21] ou ERA [24], il suffit de traduire le problème à résoudre dans le cadre formel choisi. Sachant que les décisions prises à ce stade influencent directement les performances du systèmes (cf. section 2.1), cette traduction peut être difficile. Mais la spécification du problème mènera directement à une topologie bien spécifique (i.e les groupes de voisins sont formés en se basant sur le type de contraintes). Dans ce cas, l'analyse est simple. Si nous prenons le cas des approches ACO [11] ou PSO [23], la traduction d'un problème donné se base aussi sur un cadre formel particulier, mais requiert un certain degré d'expertise pour le développement des solveurs des sous-problèmes (i.e fourmis ou particules) et plus particulièrement, pour la définition de l'ensemble des paramètres. La conception de système self-* devient alors un problème de réglage et d'ajustement de paramètres où simulation et analyse sont requises en plus de la modélisation. Ce type de méthodologie est appelé *living design* dans [2].

D'autres types d'approches n'utilisent pas de cadre formel ou semi-formel pour la résolution des problèmes. Nous prenons l'exemple de la théorie des AMAS (*Adaptive Multi-Agent Systems*) [33] où l'on se focalise sur les propriétés et comportements des agents (i.e coopération). Dans cette théorie, le concepteur doit identifier les entités à agentifier pour construire son système self-* . D'autres approches nécessitent aussi cet effort d'analyse. Nous citons les approches bio-inspirés [27] où les concepteurs doivent identifier les agents [6] (guêpe, araignée, etc.), les mécanismes de coordination (taux d'évaporation des phéromones, taux

d'oubli, etc.) et leur sémantique (que représente chaque phéromone) [32].

Un autre critère à étudier est également la simplicité du système à développer. Généralement, les algorithmes sont faciles à implémenter mais les mécanismes de coordination ou les comportements coopératifs sont plus difficiles à mettre en place.

Facilité de déploiement et de distribution. Le déploiement et la distribution d'un système donné dépendent fortement des paradigmes utilisés pour son développement. Les approches basées sur la stigmergie tel ACO, requièrent le développement d'environnement spécifique capable de gérer le dépôt de phéromones et leur mise à jour. Pour cela, des efforts supplémentaires sont nécessaires pour le déploiement de tel système. Notons l'existence de plusieurs environnements distribués répondant à ces besoins comme les intergiciels *tuple spaces* tels *Linda* [17] ou plus récemment *SwarmLinda* [5] ou l'intergiciel TOTA [26] pour les applications pervasives.

Les approches non basées sur la stigmergie mais sur la communication directe, nécessitent, elles aussi, l'utilisation de frameworks dédiés. Par exemple, le déploiement de DBA [21] requiert l'installation et la configuration d'un intergiciel multi-agent comme Jade[3] ou MAY[4]. Il requiert également le développement des entités en tant qu'agents et l'utilisation de librairies dédiées. À part cette tâche technique, le développement des agents reste facile à réaliser surtout lorsque les algorithmes existants sont clairement définis comme dans [37].

Généricité. Les approches de développement des systèmes self-* présentent différents degrés de généricité. Ce degré s'exprime par l'effort nécessaire pour adapter une approche à la résolution d'un problème particulier. Par exemple, les approches CSP distribués tel que DBA ou ACO sont complètement génériques, dans le sens où elles fournissent un solveur distribué pour tout problème exprimé sous forme d'un CSP. Cependant, certains paramètres restent à fixer comme le temps pour DBA , la probabilité de distribution des comportements dans ERA, ou l'ensemble des phéromones et les paramètres globaux pour ACO. Ceci est l'inconvénient majeur de ces approches métaheuristiques difficilement flexibles et réutilisables pour des problèmes dérivés.

[3] http://jade.tilab.com/

[4] http://www.irit.fr/MAY,774

Critères pour l'évaluation de la méthodologie de développement

- Combien d'effort est requis pour adapter le concept général à un problème donné ?
- Combien d'effort est requis pour la distribution de la résolution sur un ensemble d'agents ?
- Quel est l'effort de déploiement nécessaire ?
- Pour quel type de problème cette méthode peut être utilisée ?
- Quel est la difficulté pour faire évoluer un système en lui ajoutant par exemple, de nouvelles contraintes ?

5 Discussion

La caractérisation et l'évaluation d'un système fournissent un ensemble de valeurs qualitatives et quantitatives permettant non seulement d'évaluer le système mais aussi de le comparer avec d'autres systèmes pour souligner ses limites et ses avantages. Dans cette section, nous proposons un ensemble de moyens facilitant cette comparaison, et nous présentons un panel de comportements typiques associés aux processus d'auto-adaptation.

5.1 Évaluation comparative

L'évaluation des performances d'un système permet de montrer ses points forts et faibles par rapport à d'autres systèmes. Les critères présentés dans les sections 2, 3 et 4 peuvent être utilisés pour établir la comparaison entre plusieurs systèmes.

L'interdépendance de ces différents critères fait qu'une vue multi-objectif est plus appropriée pour leur comparaison. Cette vue ne peut pas être réduite à une seule dimension les combinant, à cause de leur sensibilité au contexte. Ces critères dépendent du domaine et des exigences initiales. Par conséquent, nous proposons un radar graphique pour valider l'adéquation fonctionnelle d'un système self-* . Un système "'idéal"' surpassant tous les autres systèmes se situerait donc, au centre de ce radar. Mais les systèmes actuellement développés ressemblent à ce que l'on peut voir dans la figure 1 où un système est meilleur pour une dimension (un critère) et non pour une autre.

5.2 Analyses

Cette vue multi-objectif permet d'avoir une vision globale de la comparaison. Ceci étant, il

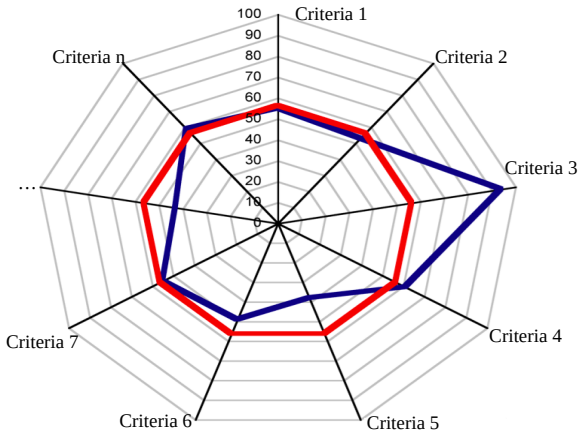

FIG. 1 – Un radar graphique pour la comparaison de systèmes self-⋆

est aussi intéressant de regarder en détail chacun des critères séparément. Dans ce qui suit, nous présentons un ensemble de courbes représentant des comparaisons entre différents systèmes. Chaque courbe représente un comportement typique du système pour un critère donné.

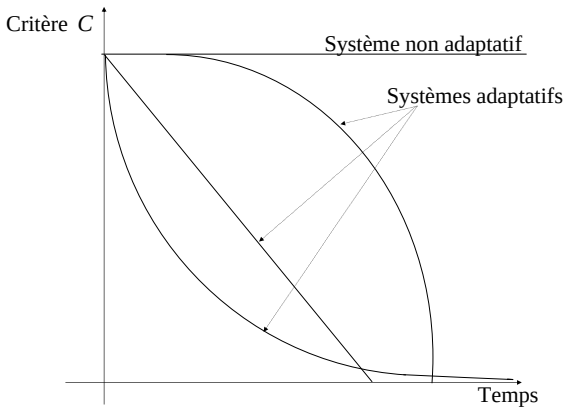

FIG. 2 – Modes de convergence vers une solution

Un premier critère à étudier est l'amélioration de la qualité de la solution au cours du temps C=$\delta(S_{cur}, S_{opt})$ (cf. section 2.2). Nous différencions les systèmes non-adaptatifs des auto-adaptatifs. Les premiers n'étant pas capables de prendre en compte les changements, n'améliorent pas leur solution. Pour les deuxièmes, nous distinguons trois types de réponses (cf. figure 2 :

- la ligne diagonale quand le système améliore sa solution proportionnellement au temps ;
- la courbe du bas représentant le meilleur cas où le système est capable de s'adapter rapidement et atteindre une bonne solution ;
- la courbe du haut représentant un système ayant besoin de beaucoup de temps avant de s'adapter et de trouver une bonne solution.

De la même façon, cette figure peut être utilisée pour analyser la réponse du système à un type de changement dynamique particulier (l'événement se produit au temps $t = 0$).

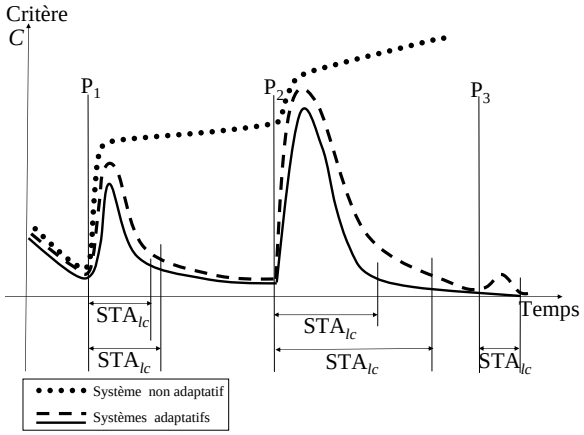

FIG. 3 – Modes d'adaptation aux changements

Dans la figure 3, nous illustrons la réponse d'un ensemble de système à différents types de perturbations. Considérons comme critère C *la qualité de la solution*. Nous pouvons voir qu'au début de l'exécution, les trois systèmes étudiés commencent par améliorer leur solution. Quand la première perturbation apparaît, la distance entre la solution courante et la solution optimale croît considérablement pour chacun d'eux. Pour le système non adaptatif, la qualité de la solution diminue de plus en plus avec chaque nouvelle perturbation. Alors qu'un système auto-adaptatif est capable d'améliorer sa solution et de revenir dans un état stable.

Une autre information présentée par ces courbes est le *temps d'adaptation* STA_{lc} ((cf. section 2.2). Un système auto-adaptatif est capable de revenir dans un état optimal plus ou moins rapidement (cf. courbe pleine et courbe en pointillés de la figure).

Un autre critère pouvant être examiné est la *robustesse* du système. Par exemple, dans la figure 3, le système représenté par la courbe pleine est plus robuste que celui représenté par la courbe en pointillés. Sa réaction aux perturbations est plus faible, plus particulièrement si on considère la perturbation P_3. Ceci permet d'illustrer également la capacité de certains systèmes self-⋆ à apprendre comment s'adapter face à un certains types de perturbations, et ainsi améliorer leur réaction et répondre plus efficacement lorsque ce type de perturbations survient.

Un dernier critère illustré par ces courbes est le

nombre de messages SCL_a utilisés pour l'adaptation. Un système non adaptatif maintient le même nombre de messages malgré les perturbations (dans notre cas, il est égal à zéro et est représenté par l'axe des x). Dans un système auto-adaptatif, le nombre de messages croît suite aux perturbations et diminue au fur et à mesure que le système s'adapte. Le fait que le système représenté par la courbe pleine ait besoin de moins de messages que celui représenté par la courbe en pointillés, peut s'expliquer par le degré de décentralisation au sein de chacun des systèmes (cf. section 3.2).

5.3 Systèmes self-* *vs.* Systèmes classiques

Une limite très importante apparaît lorsque nous essayons de montrer le bon fonctionnement d'un système self-* . Elle provient de la différence majeure entre ces systèmes et les systèmes classiques : leur capacité à changer, s'adapter et évoluer. La complexité algorithmique et la décentralisation peuvent toutes deux changer complètement quand le système est sujet à d'importantes modifications dans sa structure, sa façon de traiter les informations, ses communications, etc. L'unique moyen de mesure sera de stopper le système, de prendre un cliché (*snapshot*) et d'étudier sa complexité algorithmique et sa distribution à un moment bien précis. Ces différents clichés pris régulièrement constituent en eux-mêmes un nouveau critère : est-ce que le système évolue comme un cancer ou reste-t-il fonctionnel ? Est-ce que le système se simplifie lui-même s'il en a besoin ?

Notons aussi que cette différence présente également leur principale force. Quand la résolution d'un problème nécessite une classe d'algorithmes complexes, les systèmes self-* sont capables de simplifier l'espace de recherche et aboutir à des états bien spécifiques. Pour certains critères, ceci peut représenter un système optimal. C'est le cas des applications auto-adaptatives capables de construire d'une manière autonome des modèles adaptés pour les systèmes complexes (phénomènes naturels, l'activité humaine) [18].

6 Conclusion

Tout au long de cet article, nous avons présenté un ensemble de critères permettant l'évaluation des systèmes multi-agents auto-organisateurs présentant des propriétés self-* . Nous proposons de les observer et analyser selon trois points de vue :(i) évaluer le fonctionnement du système et sa réaction aux changements endogènes et exogènes, (ii) étudier les propriétés intrinsèques du système en se focalisant sur la décentralisation et ses propriétés locales, et (iii) caractériser la méthodologie de développement depuis les phases de conception jusqu'aux phases de déploiement. Une fois mesurés, ces critères doivent être analysés d'un point de vue global. Cependant, regrouper des critères assez différents n'est pas trivial. Pour cela, une vue multi-objective est plus appropriée. Ce point a été discuté en illustrant un ensemble de critères par des graphes théoriques.

Ce travail est un premier pas pour évaluer les systèmes self-* , et pour mettre en avant les conséquences de leur comportement, et spécialement leur capacité d'auto-adaptation non présente dans les systèmes classiques. Néanmoins, nous sommes conscients qu'une validation formelle est fondamentale pour promouvoir ces systèmes dans l'industrie. Actuellement, nous ne sommes pas capables de prouver le comportement complet de ces systèmes. Pour cela, nous pensons qu'une voie de recherche à poursuivre serait d'associer la simulation et les preuves formelles classiques.

Références

[1] C. Bernon, M.-P. Gleizes, S. Peyruqueou, and G. Picard. Adelfe : a methodology for adaptive multi-agent systems engineering. In *Third International Workshop on Engineering Societies in the Agents World (ESAW-2002)*, LNAI 2577, pages 156–169. Springer, September 2002.

[2] C. Bernon, M.-P. Gleizes, and G. Picard. Enhancing Self-Organising Emergent Systems Design with Simulation. In *International Workshop on Engineering Societies in the Agents World (ESAW)*, LNCS 4457, pages 284–299. Springer, septembre 2007.

[3] G. Bonnet and C. Tessier. Multi-agent collaboration : A satellite constellation case. In *STAIRS*, volume 179, pages 24–35. IOS Press, 2008.

[4] I. Brito, F. Herrero, and P. Meseguer. On the evaluation of discsp algorithms. *The Fifth International Workshop on Distributed Constraint Reasoning (DCR'04 at CP'04)*, (5) :142–151, 2004.

[5] A. Charles, R. Menezes, and R. Tolksdorf. On the implementation of swarmlinda. In *Proceedings of the 42nd Annual Southeast Regional Conference (ACM-SE)*, pages 297–298. ACM Press, 2004.

[6] V. Chevrier. From Self-Organized Systems to Collective Problem Solving . In *Engineering Societies in the Agents World V*. Springer, 2004.

[7] M. Chiarandini. *Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems*. PhD thesis, Udine, Italy, 2005.

[8] G. Clair, E. Kaddoum, M.-P. Gleizes, and G. Picard. Self-Regulation in Self-Organising Multi-Agent Systems for Adaptive and Intelligent Manu-

facturing Control. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE Computer Society, 2008.

[9] G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos. Self-Organisation and Emergence in Multi-Agent Systems : An Overview. *Informatica*, 30(1) :45–54, janvier 2006. ISSN 0350-5596.

[10] Jack J. Dongarra. Performance of various computers using standard linear equations software in a fortran environment. *SIGARCH Comput. Archit. News*, 16(1) :47–69, 1988.

[11] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, 2004.

[12] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard. *Metaheuristics for Hard Optimization : Methods and Case Studies*. Springer, 2005.

[13] B. Edmonds. Engineering self-organising systems, methodologies and applications. In *Engineering Self-Organising Systems*, LNCS 3464. Springer, 2005.

[14] B. Edmonds and J. Bryson. The insufficiency of formal design methods - the necessity of an experimental approach - for the understanding and control of complex mas. In *AAMAS*, pages 938–945, 2004.

[15] L. Fortnow and S. Homer. A short history of computational complexity. In *The History of Mathematical Logic*. North-Holland, 2003.

[16] F. Gaillard, Y. Kubera, P. Mathieu, and S. Picault. A reverse engineering form for multi agent systems. In *Proceedings of the 9th International Workshop Engineering Societies in the Agents World (ESAW'2008)*, 2008. Actes électroniques uniquement.

[17] D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1), 1985.

[18] J.-P. Georgé, S. Peyruqueou, C. Régis, and P. Glize. Experiencing Self-Adaptive MAS for Real-Time Decision Support Systems. In *International Conference on Practical Applications of Agents and Multiagent Systems (PAAMS)*, pages 302–309. Springer, mars 2009.

[19] M.-P. Gleizes, V. Camps, J.-P. Georgé, and D. Capera. Engineering Systems which Generate Emergent Functionalities. In *Engineering Environment-Mediated Multiagent Systems - Satellite Conference held at The European Conference on Complex Systems (EEMMAS)*, LNAI 5049. Springer, juillet 2008.

[20] K. M. Hansen, W. Zhang, and M. Ingstrup. Towards self-managed executable petri nets. In *SASO '08 : Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 287–296. IEEE Computer Society, 2008.

[21] K. Hirayama and M. Yokoo. The distributed breakout algorithms. *Artificial Intelligence*, 161(1-2) :89–115, 2005.

[22] M. Jelasity and O. Babaoglu. T-Man : Gossip-Based Overlay Topology Management. In *Engineering Self-Organising Systems : Third International Workshop (ESOA'05)*, LNCS, pages 1–15. Springer, 2006.

[23] J. Kennedy and R. C. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[24] J. Liu, H. Jing, and Y. Y. Tang. Multi-agent Oriented Constraint Satisfaction. *Artificial Intelligence*, 136(1) :101–144, 2002.

[25] N. Lynch. *Distributed Algorithms*. Morgan-Kaufmann, 1996.

[26] M. Mamei and F. Zambonelli. Programming Stigmergic Coordination with the TOTA Middleware. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'05)*, pages 415–422. ACM, 2005.

[27] J.-P. Mano, C. Bourjot, G. Lopardo, and P. Glize. Bio-inspired mechanisms for artificial self-organised systems. *Informatica*, 30(1) :55–62, 2006.

[28] D. Meignan, J.-C. Creput, and A. Koukam. A coalition-based metaheuristic for the vehicle routing problem. pages 1176–1182, June 2008.

[29] A. Meisels, E. Kaplansky, I. Razgon, and R. Zivan. Comparing performance of distributed constraints processing algorithms. In *AAMAS-02 Workshop on Distributed Constraint Reasoning*, number 5, pages 86–93, 2002.

[30] S. Minton, M.D. Johnston, A.B. Philips, and P. Laird. Minimizing Conflicts : a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Constraint-Based Reasoning*, 58(1-3) :161–205, 1994.

[31] C. H. Papadimitriou, A. A. Schäffer, and M. Yannakakis. On the complexity of local search. In *STOC '90 : Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 438–445. ACM, 1990.

[32] H.V.D Parunak and S. Brueckner. Engineering swarming systems. *Methodologies and Software Engineering for Agent Systems*, pages 341–376, 2004.

[33] G. Picard and P. Glize. Model and analysis of local decision based on cooperative self-organization for problem solving. *Multiagent and Grid Systems (MAGS)*, 2(3) :253–265, 2006.

[34] P. Robertson, R. Laddaga, and H. Shrobe. Introduction : the first international workshop on self-adaptive software. In *Proceedings of the 1st IWSAS*, LNCS 1936, pages 1–10. Springer, 2000.

[35] R. P. Würtz. *Organic Computing*. Springer, 2008.

[36] Herbert S. Wilf. *Algorithms and Complexity*. Prentice-Hall, 1986.

[37] M. Yokoo. *Distributed Constraint Satisfaction : Foundations of Cooperation in Multi-Agent Systems*. Springer, 2001.