THESIS

presented at

Université Paul Sabatier - Toulouse III

U.F.R. MATHÉMATIQUES, INFORMATIQUE ET GESTION

to obtain the title of

DOCTEUR DE L'UNIVERSITÉ DE TOULOUSE

delivered by

UNIVERSITÉ PAUL SABATIER - TOULOUSE III

Mention INFORMATIQUE

by

ELSY KADDOUM

Doctoral school: Informatique et Télécommunication
Laboratory: Institut de Recherche en Informatique de Toulouse
Team: Systèmes Multi-Agents Coopératifs

Optimization under Constraints of Distributed Complex Problems using Cooperative Self-Organization

---

JURY

Tom HOLVOET	*Professor, KULeuven*	(Reviewer)
René MANDIAU	*Professor, Université de Valenciennes*	(Reviewer)
Franco ZAMBONELLI	*Professor, Università di Modena e Reggio Emilia*	(Reviewer)
Thierry DRUOT	*Engineer Airbus, Toulouse*	(Examiner)
Greet VANDEN BERGHE	*Senior Lecturer, KaHo Sint-Lieven*	(Examiner)
Katja VERBEECK	*Lecturer, KaHo Sint-Lieven*	(Examiner)
Marie-Pierre GLEIZES	*Professor, Université de Toulouse III*	(Supervisor)
Jean-Pierre GEORGÉ	*Assistant Professor, Université de Toulouse III*	(Co-Supervisor)

Elsy Kaddoum

Optimisation sous Contraintes de Problèmes Distribués par Auto-Organisation Coopérative

Directeur de thèse : Marie-Pierre Gleizes, Professeur, UPS

Co-Directeur : Jean-Pierre Georgé, Maître de conférences, UPS

Résumé

Les ingénieurs se heurtent quotidiennement, quel que soit leur secteur d'activité, à des problèmes d'optimisation. Il peut s'agir de minimiser un coût de production, d'optimiser le parcours d'un véhicule, d'améliorer les performances d'un produit, d'affiner un modèle de calcul, etc. Ces problèmes se caractérisent par un degré élevé de complexité dû à l'hétérogénéité et la diversité des acteurs en jeu (humains, appareils électroniques, etc.), à la masse importante des données, à la distribution des informations manipulées ainsi qu'à la dynamique des environnements dans lesquels ils sont plongés.

Face à la complexité croissante de ces applications, les approches de résolution classiques ont montré leurs limites. Depuis quelques années, la communauté scientifique s'intéresse aux développements de nouvelles solutions basées sur la distribution du calcul et la décentralisation du contrôle plus adaptées à ce genre de problème. La théorie des *AMAS* (Adaptive Multi-Agents Systems), développée au sein de l'équipe *SMAC*, propose le développement de solutions utilisant des systèmes multi-agents auto-adaptatifs par auto-organisation coopérative. Dans ces systèmes, les agents poursuivent des buts locaux et interagissent d'une manière coopérative. C'est par leur interaction locale que le fonctionnement du système est rendu plus robuste et plus adapté à la dynamique de l'environnement, et ainsi, la fonction globale du système émerge. Suite à plusieurs études, cette théorie a montré son adéquation pour la résolution de problèmes complexes et dynamiques, mais son application reste à un niveau d'abstraction assez élevé.

L'objectif de ce travail est de modéliser et de spécialiser cette théorie pour la résolution de problèmes complexes d'optimisation sous contraintes présentant des critères multidisciplinaires et multi-objectifs. Ainsi, l'utilisation des *AMAS* en sera facilitée et pourra être mise à disposition des ingénieurs ayant à résoudre ce genre de problèmes. Pour cela, le modèle d'agents *AMAS4Opt* avec des comportements et des interactions coopératifs et locaux a été défini. Ce modèle peut être instancié ou étendu pour résoudre divers problèmes d'optimisation. La validation s'est effectuée sur deux problèmes clés d'optimisation : le contrôle manufacturier se caractérisant par un degré élevé de dynamique et la conception de produit complexe basée sur le raisonnement par population se caractérisant par la masse importante de données interdépendantes à manipuler.

Finalement, afin de montrer la robustesse et l'adéquation des solutions développées, un second objectif de mon travail de thèse a été la définition d'un ensemble de critères d'évaluation permettant de souligner les points forts et faibles des systèmes adaptatifs et de les comparer à des systèmes existants.

Elsy Kaddoum

Optimization under Constraints of Distributed Complex Problems using Cooperative Self-Organization

Supervisor: Marie-Pierre Gleizes, Professor, UPS

Co-Supervisor: Jean-Pierre Georgé, Associate Professor, UPS

Abstract

We solve problems and make decisions all day long: at home, at work, while playing. Some problems and decisions are very challenging: What is the best sequence of actions to reach a goal or the best itinerary to deliver orders given the weather, the traffic and the hour? How to choose the best time for a meeting knowing the availability of concerned people and meeting rooms with adequate material? How to improve product manufacturing performances or to refine a computational model given delays to satisfy, interdependencies between parameter and multi-disciplinary aspects? etc. Problems that are characterized by a high level of complexity due to the heterogeneity and diversity of the participating actors such as humans or electronic devises, to the increasing volume of manipulated data and their distribution and to the dynamics of the applications environments.

Classical solving approaches have shown their limits to cope with this growing complexity. Thus, the scientific community has been interested, for the last several years, in the development of new solutions based on computation distribution and control decentralisation, which are more appropriate for solving such problems. The *AMAS (Adaptive Multi-Agent-Systems)* theory developed by the *SMAC* team, proposes to build solutions based on self-adaptive multi-agent systems using cooperative self-organisation. In such systems, cooperative interacting agents pursue local goals. By their interactions, the robustness of the system and its capacities to adapt to dynamic environments are increased. Thus, the global function of the system emerges. This theory have shown its adequacy to solve a large variety of complex and dynamic problems, but it remains at a high abstraction level, requiring *AMAS* experts for its application.

This work proposes a specialisation of this theory for complex optimisation problem solving under constraints characterized by multi-disciplinary and multi-objective criteria. This will make the usage of this theory accessible to different non-*AMAS* experts engineers confronted to such problems. Thus, the *AMAS4Opt* agent model with cooperative, local and generic behaviours and interactions has been defined. Such behaviours and interactions can be instantiated and extended for solving different complex optimisation problems. Once identified, they have been instantiated and tested on two well-known optimisation problems: scheduling in manufacturing control characterised by high level of dynamics and complex product design characterised by the volume of interrelated data.

Finally, in order to show the robustness and adequacy of the developed solutions, a set of evaluation criteria is proposed to underline the advantages and limits of adaptive systems and to compare them with already existing systems.

Many Thanks...

THREE years have passed, three years of hard work but three years of great pleasure, thanks to all who contributed to this work, even indirectly.

First, I would like to thank my jury members. Pr. René Mandiau, thank you for accepting to be the president of my thesis jury. I would also like to thank you, Pr. Tom Holvoet and Pr. Franco Zambonelli for your thorough evaluation and reviewing of my thesis document.

I would also like to thank Greet Vanden Berghe and Katja Verbeeck, not only for accepting my invitation to be part of my jury, but also for my 3 months visit to their laboratory. It was a great experience for me, I have learned a lot from you and from your team.

I also address my thanks to Claudia Raibulet. Our collaboration concerning the evaluation of self-* multi-agents systems brought me a lot. I hope we will be able to continue this collaboration.

My thanks to the Zonta international organization from which I received the Amelia Earhart Fellowships. This reward was for me the recognition of the importance of my work and encouraged me while facing the challenges of my research. A special thank you to Miss Katherine Piquet Gauthier and to every person I met in this organization.

During those three years, I have always felt welcomed in the IRIT laboratory. I would like to thank every person I met or asked for a service. Thank you for being here and for your help.

Special thanks and thanks and thanks to every member of the SMAC team. Marie-Pierre, Pierre and Jean-Pierre, thank you for supervising this work, for being here every time I needed you. I have learned a lot from each of you and hope that every PhD student can have supervisors like you. My colleagues, thanks for your support, for your encouragements, for your joy, for every advice you gave me as teachers or researchers, for the coffee breaks, for every discussion we had, etc. The list is getting long... Thank you for being here.

I would also like to thank every person at Upetec, especially Davy, Sylvain R. and Jean-Pierre. You are a bit far now, but I won't forget all the shared moment, especially the discussions we had at lunch time. It was very nice moments.

I have been in France for eight years now. Thanks to all my friends who I consider as my second family. Fred, Joel, Marion, Georgette, Joseph, Liliane, Cindy and Diana, thanks for everything you have done for us. Elodie, I enjoyed every moment we shared and I hope that you will come back soon to Toulouse. A special thought to the Loustics Team, Nathalie, Stéphanie, Jean-baptiste and Damien, thank you all for your support and the enjoyable parties we had together. Bernard, Michèle, Eric, Fabienne and every member of the Dudouit

family, it is a great pleasure to meet you. Thanks for all your encouragements and advices in this very hard period.

"بابا بابا ردّ عليّ يا بابا حاكيني"

My joy every morning I hear my phone ringing with this song. Mum and Dad, I can't find the words to thank you. You are at my side and encourage me at every moment. Even with the distance, you always found the right words to help me go ahead. I am really happy that you can share this moment with me. I hope that you do not regret the moment you accepted to send us here; I hope that you are proud of your three children.

Sister Larousse, you encouraged me to follow your steps. It was hard at the beginning but you were always here and now I want to thank you for this great experience you shared with me. Thanks for every moment we had and will have together. Thanks for your encouragements especially in those last months of my thesis. Thanks for all the time you spent correcting my documents. You are my "Larousse"! I wouldn't be here without you. Sorry, for every time I annoyed you. I wish you all the best. You deserve it.

Brother Chouchy, my little brother! I hope you enjoy your stay with us. Thanks for all your help and for the great moments (lunches, beers, knacki balls, Hot dog sandwiches, etc.) we shared. Thanks for all your encouraging SMS. Enjoy every moment you can have in your university, I wish you a lot of courage for your studies and the best for your life.

A warm thank for you Stéphane. Thanks for everything you have done for me. Thanks for encouraging me to work hard those last months. I won't have finished this report without you being by my side. You knew how to enter my life and to become an essential part of it.

Contents

Acronyms

ACO Ant Colony Optimization

ADELFE Atelier de Développement de Logiciels à Fonctionnalité Emergente

ADOPT Asynchronous Distributed Constraint Optimization

AGV Automated Guided Vehicle

AMAS Adaptive Multi-Agent System

AMAS4Opt AMAS for Optimisation

AMAS-ML AMAS Modelling Language

API Application Programming Interface

CBR Case-Based Reasoning

COP Constraint Optimization Problem

CSP Constraint Satisfaction Problem

DAMASCOP Distributed AMAS Constraint Optimisation Problem

DBA Distributed Breakout Algorithm

DCOP Distributed Constraint Optimization Problem

DFS Depth-First Search

DynCNET Dynamic Contract-NET

DFJSP Dynamic Flexible Job Shop Problem

ERA Environment, Reactive rules, and Agents

GA Genetic Algorithm

JADE Java Agent DEvelopment Framework

MAS Multi-Agent System

MAY Make Agents Yourself

MDO Multidisciplinary Design Optimization

MOO Multi-Objective Optimization

NCS Non Cooperative Situation

OptAPO Optimal Asynchronous Partial Overlay

PLS Polynomial Time Local Search

PSO Particle Swarm Optimization

SAFlex Self-Adaptive Flexible scheduling

SAPBR Self-Adaptive Population Based Reasoning

SMAC Systèmes Multi-Agent Coopératifs

TOTA Tuples On The Air

TSP Traveling Salesman Problem

Introduction Générale

The introduction in english starts page 7

Introduction à la résolution des problèmes complexes sous-contraintes

Les ingénieurs se heurtent quotidiennement, quel que soit leur secteur d'activités, à des problèmes d'optimisation. Il peut s'agir de minimiser un coût de production, d'optimiser le parcours d'un véhicule, d'améliorer les performances d'un produit, d'affiner un modèle de calcul, etc. Ces problèmes se caractérisent par un degré élevé de complexité dû à l'hétérogénéité et la diversité des acteurs en jeu (humains, appareils électroniques, etc.), à la masse importante des données, à la distribution des informations manipulées ainsi qu'à la dynamique des environnements dans lesquels ils sont plongés.

Ainsi, la résolution des problèmes sous-contraintes a été un des domaines les plus étudiés et ce depuis les débuts de l'informatique. Différentes techniques de résolution ont été développées afin d'assister les utilisateurs dans leurs tâches quotidiennes. Malgré leur adéquation pour la résolution d'un large éventail de problèmes, les approches classiques ont montré leurs limites face à la complexité croissantes des applications actuelles. Cette complexité est principalement due à :

- *La dynamique de l'environnement et du système lui-même (changement de contraintes, modification des acteurs en jeu, etc.);*
- *La diversité des données manipulées et l'augmentation de leur volume;*
- *L'existence d'interdépendances non suffisamment définies entre les paramètres du problème;*
- *La non linéarité des relations entre les paramètres;*
- *Les dimensions multi-objectifs et multi-disciplinaires des problèmes étudiés.*

Un des formalismes les plus étudiés pour la résolution de ce type de problèmes est le Constraint Optimization Problem (COP). Les différentes techniques basées sur ce formalisme ont montré leur adéquation pour la résolution d'un large éventail relativement simple et statique de problèmes sous-contraintes. Malheureusement, ces techniques ont montré leurs limites face à la complexité croissante mentionnée ci-dessus. Ainsi, de nouvelles approches de résolution améliorant ces approches classiques par l'utilisation de la distribution du calcul et la décentralisation du contrôle tentent de répondre à cette complexité. Différentes motivations sont à l'origine de ces deux améliorations. Premièrement, la

réduction du temps de calcul grâce au parallélisme. Deuxièmement, la distribution des compétences et des connaissances. Ainsi, dans une application multi-disciplinaire, chacune des disciplines peut être représentée par une entité spécifique et autonome. La solution est obtenue par les interactions de ces différentes entités. Par cette décomposition, une meilleure représentation de chaque discipline est obtenue améliorant ainsi leur gestion. Finalement, la plus importante motivation est l'augmentation de l'auto-adaptation, la robustesse du système et le traitement local des perturbations. En effet, dans un environnement dynamique, des évènements imprévisibles tels que l'introduction de nouvelles entités, la modification des contraintes, des pannes ou même la détection de situations non prévues à la conception du système peuvent apparaître. Par la distribution du calcul et la décentralisation du contrôle, seules les entités concernées par ces évènements sont perturbées, les autres pouvant continuer leur exécution normalement. En conclusion, ces deux aspects augmentent la flexibilité, la robustesse et l'adaptabilité des systèmes. La plupart des systèmes déployant ces techniques sont connus sous le nom de Systèmes Multi-Agents.

Un système multi-agent se décompose en un ensemble d'entités autonomes et interagissantes appelées agents qui par la coordination de leurs actions locales permettent d'atteindre une solution optimale au niveau global. Différentes techniques de résolution se basent sur les systèmes multi-agents. Leurs différences résident principalement au niveau des mécanismes de coordination et d'interaction entres les agents du système. La théorie "Adaptive Multi-Agent System (AMAS)", utilisée dans ce travail de thèse, propose la conception de systèmes multi-agents adaptatifs en se basant sur la coopération. Cette théorie a montré son adéquation pour la résolution d'un grand nombre de problèmes présentant différents niveaux de complexité. Afin de faciliter la conception de systèmes basés sur cette théorie, la méthodologie ADELFE pour la conception de systèmes à fonctionnalité émergente fut développée. Cette méthodologie se base sur le Rational Unified Process auquel des activités dédiées à l'identification et le développement des agents coopératifs ont été ajoutées. Malgré son adéquation pour guider le développement de systèmes multi-agents coopératifs et adaptatifs, cette méthodologie présente un niveau d'abstraction assez élevé exigeant ainsi la présence d'un expert AMAS pour son utilisation. Dans ce travail, nous proposons un modèle d'agent générique spécialisant l'utilisation de la théorie des AMAS et la méthodologie ADELFE pour la résolution sous-contraintes de problèmes complexes. Le but de ce modèle est de fournir à l'ingénieur une architecture générique d'agents proposant des comportements et des interactions entre agents spécifique à ce type de problèmes.

Contributions

Ce travail de thèse propose deux contributions importantes dans le domaine de la résolution sous-contraintes de problèmes complexes utilisant les systèmes multi-agents :

1. *Un modèle d'agent générique guidant le développement de ce type de systèmes et définissant des comportements dédiés d'agents.*
2. *La définition d'un ensemble de critères d'évaluation soulignant l'importance et les points forts de ces systèmes.*

Tout d'abord, nous justifions l'utilisation des technologies basées sur les systèmes multi-agents adaptatifs pour la gestion de la complexité croissante des applications actuelles et futures.

Par la suite, le modèle d'agent générique spécialisant la théorie des AMAS et la méthodologie ADELFE est défini. Ce modèle est direct et présente un niveau d'abstraction soigneusement choisi afin d'être aisément adapté à différents types de problèmes complexes sous-contraintes facilitant ainsi l'utilisation de la théorie des AMAS par des ingénieurs non-experts du domaine. Ce modèle comporte deux types d'agents pour lesquels des comportements et des interactions génériques ont été définis. Chaque agent possède des connaissances et des représentations locales sur le problème. Il est plongé dans un environnement local incluant ses voisins (d'autres agents du système) ou d'autres entités actives du système nécessaire à son processus de raisonnement. Les différents agents du système interagissent d'une manière coopérative avec leur environnement local dans le but de satisfaire leur but local. Pour cela, chaque agent calcule son degré de satisfaction représentant sa difficulté à atteindre son but. Une fois que l'équilibre entre les degrés de satisfaction des différents agents est atteint, la solution globale du problème est obtenue. Les agents conçus sont en relation directe avec les entités du problème ce qui permet de rester fidèle à la définition du problème. Ainsi, aucune transformation du problème dans un cadre formel n'est requise ce qui représente une des principales différences de notre modèle par rapport aux approches classiques. De plus, le modèle défini conserve les avantages des systèmes multi-agents coopératifs et adaptatifs. Il se caractérise par :

- *un traitement local des perturbations se produisant en temps réel telles que les changements de contraintes en cours d'exécution ;*
- *une augmentation du niveau de robustesse ;*
- *sa capacité à maintenir un comportement adéquat et cohérent dans des environnements hautement dynamiques ;*
- *son ouverture. Des nouveaux agents peuvent apparaître ou disparaître du système en cours d'exécution.*

Afin de montrer l'adéquation de ce modèle, deux applications clés du domaine de l'optimisation ont été choisies : une pour la résolution du problème de la gestion de production et une pour la conception de produits complexes en se basant sur des données existantes.

Finalement, comme ces systèmes sont différents des approches classiques existantes et comme il n'existe pas de cadre d'évaluation pour les systèmes adaptatifs, de nouveaux critères d'évaluation et de caractérisation sont requis afin de souligner la spécificité de ces systèmes et leur capacité à s'adapter à des environnements dynamiques. Ainsi, trois catégories de critères ont été définies afin de guider l'évaluation de ces systèmes depuis la phase de conception jusqu'à leur exécution.

Plan du manuscrit

Cette thèse se décomposent en 5 chapitres :

Chapitre 1. *Ce premier chapitre présente un état de l'art des différentes méthodes de résolution existantes. Premièrement, le formalisme COP et son amélioration pour la gestion de la dynamique est détaillé. Par la suite, une introduction au Case-Based Reasoning (CBR) comme technique de résolution est présentée.*

Chapitre 2. *Ce deuxième chapitre est dédié à la présentation de la théorie des AMAS, la méthodologie ADELFE et différents outils utilisés dans la conception de l'architecture d'agents définis dans ce travail.*

Chapitre 3. *Dans ce chapitre, le modèle d'agent générique défini pour la résolution sous-contraintes de problèmes complexes est présenté ainsi que les comportements des agents et leurs interactions.*

Chapitre 4. *Ce chapitre s'intéresse à l'évaluation des systèmes multi-agents adaptatifs. Différents critères d'évaluation soulignant les capacités d'auto-adaptation de ces systèmes ont été définis.*

Chapitre 5. *Dans ce dernier chapitre, deux systèmes multi-agents conçus pour la résolution de deux problèmes clés de l'optimisation: Self-Adaptive Flexible scheduling (SAFlex) pour la gestion de production et Self-Adaptive Population Based Reasoning (SAPBR) pour la conception de produits complexes, sont détaillés. Le modèle d'agents défini dans le chapitre 3 est utilisé pour la conception de chacun de ces deux systèmes. Leur évaluation est adressée en utilisant les critères définis dans le chapitre 4.*

General Introduction

Introduction to Complex Problem Solving under Constraints

We solve problems and make decisions all day long: at home, at work, while playing. Some problems and decisions are very challenging: What is the best sequence of actions to reach a goal or the best itinerary to deliver orders given the weather, the traffic and the hour? How to choose the best time for a meeting knowing the availability of concerned people and meeting rooms with adequate material? How to improve a product manufacturing performances or to refine a computational model given delays to keep, interdependencies between parameters and multi-disciplinary aspects? etc. These questions require thinking, exploration, trial and error.

Since the arising of computer science, the domain of Problem Solving under Constraints has been widely studied and different techniques have been developed to solve problems and to assist users in their daily life. But today's applications are subject to a growing complexity that classical approaches can not manage any more. This complexity is due to:

- The dynamics of the environment and the system itself (constraint changes, modification of the implicated actors, etc.);
- The increasing volume and diversity of manipulated data;
- The existence of insufficiently defined interdependencies between the parameters of the problem;
- The non linearity of the relations between parameters;
- Multi-objective and multidisciplinary dimensions.

The *Constraint Optimization Problem (COP)* formalism is one of the main formalisms used for solving this type of problems. Most of the solving techniques have shown their adequacy to solve a large variety of relatively simple and static problems. But they were unable to handle the aforementioned growing complexity. Thus, researchers have been working on their improvements using the distribution of computation and the decentralization of control. Several aspects motivate taking these directions. The most obvious one is the speed up of the solving thanks to parallelism. Another one concerns the distribution of expertise.

For example, in a multi-disciplinary application, each discipline can be represented by a specialised autonomous entity and the interactions between all theses entities enable a collective solution. This decomposition results in a better management of each discipline. Finally, the most important motivation is the increase of adaptation capabilities in an autonomous manner, robustness and localised perturbations treatment. Indeed, in dynamic environments different unpredictable events occur. Such events can include the introduction of new entities, the modification of constraints, system failures or the detection of situations not predicted at the design level. In such situations, only affected entities are involved while others can continue their execution. In conclusion, the distribution of computation and the decentralization of control provide systems with a high level of flexibility, reliability and adaptability. Most of the systems deploying these techniques are commonly known as Multi-Agent Systems.

A multi-agent system consists in a set of autonomous and interacting entities that coordinate their actions locally in order to reach global optimal performances. Different approaches have exploited the advantages of multi-agent systems. The major differences between them concern the coordination and interactions mechanisms among agents. The *Adaptive Multi-Agent System (AMAS)* theory, on which this work is based, proposes to build adaptive multi-agent systems using cooperation. This theory has shown its adequacy to solve a large number of complex problems with different characteristics and complexity levels. To ease the development of such systems, *ADELFE*, a toolkit to develop software with emergent functionality, has been developed. This toolkit includes a methodology based on the *Rational Unified Process* and adds specific activities dedicated to the identification and development of cooperative agents, but it remains at a high abstraction level. In this work, we propose a generic agent model based on the *AMAS* theory that completes the *ADELFE* methodology providing the engineer with more precise agents models behaviours and interactions for complex problem solving under constraints.

Contributions

This thesis work leads to two main contributions in complex problem solving under constraints using multi-agent system technologies:

1. A generic agent model guiding the development of such systems and defining the behaviours of the agents.

2. The definition of criteria for their evaluation.

After justifying the usage of multi-agent systems technologies for solving current and future complex applications challenges, we propose a generic agent model based on the *AMAS* theory and completing the *ADELFE* methodology.

The proposed model is straightforward and presents a carefully chosen level of abstraction enabling it to be easily adapted to different types of complex problem under constraints. This model is designed in order to reduce the effort that an engineer who is

not an *AMAS* expert would need to design an *AMAS* for the solving of optimisation under constraints problems.

In this model, we define two sorts of agent behaviours and their interactions. The agents of this model are provided with local knowledge and local representations of the problem. Each agent possesses a local environment that includes its neighbours (other agents to interact with) or entities of the problem required for its solving process. The defined agents interact cooperatively with their local environment in order to satisfy their local goal. Each agent computes its satisfaction degree which represents its difficulty to attain its goal. By reaching a satisfaction equilibrium among the different agents a global solution of the problem is obtained. The agents are designed with a strong relationship to the problem entities. By that, we stay close to the problem definition and there is no need to translate the problem into a specific formal framework which is the main difference with classical existing approaches.

In addition to this, our model preserves the advantages of cooperative and adaptive multi-agent systems. The obtained multi-agent system is characterized by:

- local treatment of real-time occurring events such as constraints changes at runtime;
- an increased level of robustness;
- its ability to maintain adequate functioning in highly dynamic environments at runtime when agents want to leave the system or new agents try to enter;
- its openness. New agents can enter and leave the system during runtime.

In order to show the adequacy of this model, we develop two applications: one for solving the well known dynamic flexible job shop problem and one for solving the complex product design problem using existing similar examples.

Because such systems are different from classical ones and because there is no evaluation framework for adaptive systems, new characterization and evaluation criteria are required to underline their specificity and their ability to adapt in dynamic environments. Thus, a number of categorized criteria to guide the evaluation of this kind of systems from the design phase to the execution phase are proposed in this thesis.

Manuscript Organisation

This thesis is divided into 5 chapters:

Chapter 1. This chapter concerns the state of the art of existing solving methods for complex problems under constraints. First, the *Constraint Optimization Problem (COP)* formalism and its enhancement to handle dynamics are detailed. Then an introduction to the *Case-Based Reasoning (CBR)* solving technique is presented.

Chapter 2. In this chapter, the *AMAS* theory, the *ADELFE* methodology and different tools used to design the architecture of the defined agents are presented.

Chapter 3. This chapter highlights the generic agent model we define for complex problem solving under constraints and describes the agents behaviours and interactions accordingly to the *AMAS* theory.

Chapter 4. In this chapter, the evaluation of adaptive multi-agent systems is discussed. Different criteria are defined to underline the importance of self-adaptation.

Chapter 5. In this last chapter, two multi-agent systems designed to solve two different applications are detailed: *Self-Adaptive Flexible scheduling (SAFlex)* for the Dynamic Flexible Job Shop Problem and *Self-Adaptive Population Based Reasoning (SAPBR)* for the Complex Product Design Problem. For each, the approach proposed in chapter 3 is used and the results are discusses accordingly to chapter 4.

1 State Of the Art: Optimization under Constraints

« Il n'est pas de vent favorable pour celui qui ne sait pas où il va. »

Sénèque

Contents

The chapter in english starts page 16.

1

Résumé général du chapitre

Choisir le meilleur chemin pour arriver au travail étant donné l'heure et l'état du trafic, choisir le meilleur moment pour une réunion en considérant la disponibilité des personnes ainsi que celle des salles de réunions et leur équipement, améliorer les performances d'un modèle de calcul, et divers autres problèmes d'optimisation nécessitent de prendre des décisions et faire des choix rarement évidents ou faciles. De part la complexité due à la non-linéarité des interdépendances entre les paramètres ou à l'hétérogénéité des acteurs en jeux ou même à la dynamique inhérente à ces systèmes, nous sommes confrontés à une réduction de la visibilité à long terme d'un choix donné et une demande d'un grand effort d'adaptation en temps réel. D'autre part, ces choix peuvent impacter d'une manière importante l'efficacité et les performances souhaitées et doivent être soigneusement étudiés. Ainsi dès le début des systèmes informatiques, un grand effort a été dédié à l'étude et au développement de systèmes et d'outils afin d'aider les utilisateurs et les assister dans leurs choix.

Un des formalismes les plus important définis pour la résolution sous-contraintes de problèmes d'optimisation complexes est le Constraint Optimization Problem (COP). Dans ce formalisme, avant d'être résolu, un problème doit être traduit dans un cadre formel consistant en un ensemble de variables chacune devant être assignée à une valeur dans un domaine de validité donné afin de minimiser ou de maximiser une fonction objectif. Résoudre ce type de problème revient à parcourir l'espace de recherche (souvent décrit sous la forme d'un arbre de recherche) et à trouver la meilleure affectation possible aux différentes variables.

Différentes techniques visant à parcourir le plus efficacement possible cet espace de recherche ont été définies. Les techniques de ***recherche non-informée*** *furent les premières. Elles nécessitent une exploration complète et exhaustive de l'espace de recherche avant qu'une solution ne soit obtenue. Parmi elles, les* ***algorithmes de recherche en profondeur et en largeur*** *ainsi que leurs améliorations (****Uniform-cost, Depth-limited, Iterative Deepening Depth-First)*** *ont été les plus étudiés. Malheureusement, avec l'augmentation de la taille de l'espace de recherche, ces techniques nécessitent un temps de calcul assez important avant l'obtention d'une solution.*

Afin de répondre à cette limite, des informations supplémentaires sur les caractéristiques des problèmes ont été rajoutées pour accélérer la recherche. Ces techniques sont connues sous le nom de ***recherche informée****. Elles se divisent en deux catégories:* ***méthodes exactes*** *et* ***méthodes approximatives****. Les méthodes exactes comme l'****algorithme par séparation et évaluation****,* ***A**** *ou la* ***programmation dynamique*** *sont complètes et garantissent, si elles existent, l'obtention de solutions optimales, mais pour cela, elles demandent une bonne connaissance du problème à résoudre et peuvent être coûteuses en temps. Les méthodes approximatives quant à elles, ne garantissent pas l'obtention de solutions optimales mais sont capables d'obtenir de bonnes solutions en un temps de calcul raisonnable. Les* ***méta-heuristiques*** *sont les méthodes les plus connues et les plus étudiées.*

Les ***méta-heuristiques*** *sont des algorithmes de résolution incertains et souvent non déterministes. Elles se définissent comme des stratégies de haut niveau visant à guider le processus de recherche en utilisant des informations spécifiques à un problème donné appelées* ***heuristiques****. Leur objectif est d'explorer efficacement l'espace de recherche afin de trouver une solution proche de l'optimal. Pour cela, elles alternent entre deux phases: l'****exploration*** *et l'****exploitation****. La*

première consiste à découvrir de nouvelles zones de l'espace de recherche alors que la deuxième consiste à concentrer la recherche dans les zones prometteuses. Elles sont décrites suivant un niveau d'abstraction indépendant du problème spécifique à résoudre. Nous distinguons entre deux catégories de méta-heuristiques se différenciant par le nombre de solutions explorées en même temps: les ***méta-heuristiques de trajectoire*** *et les* ***méta-heuristiques à base de population****.*

Les méta-heuristiques de trajectoire consistent à faire évoluer une solution en utilisant comme outil de base une procédure de recherche locale. Cette procédure améliore une solution par des déplacements successifs dans un voisinage local. Une des limites principales de cette procédure est sa facilité à être piégée dans des optimum locaux correspondant à des solutions de mauvaises qualités. Différentes stratégies sont développées pour contourner ce problème. Parmi elles, nous distinguons: la ***recherche tabou****, la* ***recherche locale itérée*** *et le* ***recuit simulé****.*

Les méta-heuristiques à base de population sont considérées comme étant des améliorations successives dans une population de solution. Initialement, un ensemble de solutions appelées individus est généré afin de constituer la population. A chaque itération, une nouvelle population est définie en appliquant des procédures de sélection et de remplacement des individus de la population courante. Les différentes méta-heuristiques à base de population se distinguent par leur procédures de sélection et de remplacement. Les ***algorithmes évolutionnistes*** *inspirés par la théorie Darwinienne comme les* ***algorithmes génétiques*** *sont les principaux algorithmes étudiés dans cette classe de méta-heuristiques. Dans ces algorithmes, des procédures de croisement, de mutation et de sélection sont utilisées pour la génération d'une nouvelle population.*

Ces différentes méthodes méta-heuristiques ont montré leur adéquation pour la résolution d'un large éventail de problèmes d'optimisation. Cependant, elles montrent plusieurs limites. Premièrement, elles nécessitent la traduction du problème dans un cadre formel bien déterminé ce qui peut être difficile dans le cadre de problèmes complexes où une spécification globale du problème est souvent difficile à mettre en place. Deuxièmement, ces méthodes nécessitent un grand effort d'ajustement de paramètres comme la taille de la mémoire dans la recherche tabou ou les probabilités de sélection dans les algorithmes évolutionnistes ou la recherche locale itérée. Troisièmement, ces méthodes sont centralisées et leur efficacité repose sur l'évaluation d'une fonction objectif. Ceci réduit leur flexibilité et leur robustesse dans des environnements hautement dynamiques où une adaptation en temps-réel est requise. Finalement, lors de la conception d'une méta-heuristique, il est important de trouver un équilibre entre les phases d'exploration et d'exploitation. D'une manière générale, les méta-heuristiques de trajectoire ont de meilleures performances pendant les phases d'exploitation alors que les méta-heuristiques à base de population sont meilleures pendant les phases d'exploration.

Les méta-heuristiques hybrides adressent cette dernière limite. Elles consistent à combiner deux ou plusieurs méta-heuristiques allant d'une simple séquence entre deux méta-heuristiques s'exécutant l'une après l'autre, à la mise en coopération de plusieurs méta-heuristiques s'exécutant en parallèle. Lors de la conception de ces méthodes, il est important de faire attention à comment les combiner afin d'éviter une convergence prématurée et maintenir l'équilibre entre les phases d'exploitation et d'exploration. L'introduction du parallélisme (la distribution du calcul) d'une part et de la coopération entre plusieurs méta-heuristiques (décentralisation du contrôle) de l'autre ont permis l'amélioration des résultats obtenus par ces méthodes et ont par ainsi montré leur importance pour résoudre des problèmes présentant une complexité croissante.

Afin d'intégrer ces notions de distribution et de décentralisation, le formalisme du Distributed

Constraint Optimization Problem (DCOP) est introduit. Un DCOP est un COP où chacune des variables est gérée par un entité autonome appelée agent. Les différents agents du système doivent coordonner leurs choix et leurs actions afin de satisfaire les contraintes de leur variable et d'optimiser une fonction objectif. Ainsi, le processus de résolution est distribué sur un ensemble d'agents. Ces méthodes de résolution sont appelées les Systèmes Multi-Agents. Nous distinguons trois techniques pour l'identification des agents: ***agentification des variables***, ***inspiration naturelle*** *et* ***agentification des entités du domaines****.*

La technique d'agentification des variables consiste à distribuer un problème défini en tant que COP. Chaque agent est responsable de l'affectation de sa variable en respectant ses contraintes avec son voisinage. La plupart des méthodes basées sur cette technique organise les agents selon une structure hiérarchique dictée par les contraintes. Parmi elles nous distinguons ***Asynchronous Distributed Constraint Optimization (ADOPT)*** *et* ***Optimal Asynchronous Partial Overlay (OptAPO)****. Dans ADOPT, une structure d'arbre est utilisée pour définir les échanges de messages entre les différents agents. Cette structure rigide confère à l'algorithme plusieurs limites notamment l'****existence d'un noeud central*** *et la* ***difficulté d'adaptation*** *aux changements de contraintes. OptAPO contourne cette difficulté en augmentant le degré de décentralisation. Dans cet algorithme, un agent est élu comme agent de médiation dans un voisinage donné. Cet agent a la charge de résoudre le problème entre les différents agents de son voisinage direct (agents avec lesquels un lien direct existe) en respectant leurs contraintes vers les agents non inclus dans son voisinage. Si aucune solution n'est trouvée, l'agent ajoute à son voisinage les agents externes avec lesquels des conflits existent. Ainsi, les parties difficiles du problème sont découvertes et leur résolution est centralisée au niveau d'un agent de médiation. Lors des phases de médiations, l'algorithme par séparation et évaluation est utilisée afin de déterminer l'existence d'une solution. La limite principale de cette approche est l'****augmentation du nombre d'agents*** *impliqués dans une phase de médiation donnée. En effet, dans un problème où les variables sont fortement couplées, le voisinage d'un agent donné peut augmenter jusqu'à inclure tous les agents du système. Dans ce cas, OptAPO se résume à une résolution centralisée utilisant l'algorithme par séparation et évaluation coûteux en temps. Cependant, l'augmentation du degré de décentralisation* ***améliore la robustesse*** *de cet algorithme vis-à-vis de la dynamique.*

Les algorithmes à inspiration naturelle s'inspirent du fonctionnement du comportement collectif d'insectes sociaux tels que les oiseaux, les abeilles, les fourmis, les poissons. Les algorithmes ***d'optimisation par colonies de fourmis et par essaims particulaires*** *sont les plus étudiés de cette catégorie. Dans ces approches, un agent est associé à la construction ou à l'amélioration d'une solution unique. Les interactions entre agents visent à échanger des informations à propos des zones prometteuses de l'espace de recherche, de manière à rendre l'exploration plus efficace. Cette coopération entre agents est mise en oeuvre de deux manières différentes. Dans l'optimisation par colonies de fourmis, le comportement des agents est inspiré du comportement d'une fourmilière. De ce fait, les agents coopèrent par le biais de la matrice de phéromone. Cette situation particulière de coopération entre agents interagissant de manière indirecte via leur environnement se nomme la* ***stigmergie****. Dans l'optimisation par essaims particulaires, le comportement des agents est inspiré par le déplacement d'un groupe d'oiseaux chacun influençant sa position et sa vitesse en fonction des autres. Les agents de ce modèle* ***interagissent directement*** *en échangeant des informations concernant les meilleures positions visitées attirant ainsi les uns et les autres vers les zones prometteuses de l'espace de recherche. Ces techniques sont similaires aux méta-heuristiques à base de*

population mais se différencient par la ***distribution de la population sur un ensemble d'agents*** *chacun faisant évoluer sa solution d'une manière* ***autonome****. De ce fait, nous pouvons noter l'****absence d'un contrôle global*** *dans ce type d'algorithmes. Malheureusement, ces algorithmes utilisent à l'image des méta-heuristiques à base de population des* ***fonctions objectifs globales*** *limitant leur utilisation pour les problèmes possédant ce type de fonction. De plus, ils nécessitent l'****ajustement de paramètres*** *tels que le nombre de fourmis, les stratégies de dépôts et d'évaporation des phéromones ou la modification de la vitesse des agents. Ces méthodes montrent l'intérêt de la distribution du calcul et la décentralisation du contrôle mais leur usage reste limité à des problèmes spécifiques.*

Les approches basées sur l'agentification des entités du domaine consistent à modéliser les agents en partant de la ***description naturelle du problème indépendamment de tout formalisme****. Deux types d'agentification sont à distinguer. La première consiste à agentifier les* ***entités physiques*** *du domaine. Dans la deuxième, les* ***entités fonctionnelles*** *du problème sont représentées par des agents. Avec ces approches, le système multi-agent conçu est proche de la définition du problème et aucune transformation ou traduction n'est à faire. Chaque agent représente une entité du problème et possède ces caractéristiques. Il est conçu avec un objectif local et son comportement est dicté par le fonctionnement de l'entité qu'il représente. Par leurs interactions, les agents coordonnent leurs actions afin de satisfaire leurs buts locaux menant ainsi le système vers une solution globale. Différentes techniques d'interaction sont utilisées comme la* ***communication directe par l'utilisation de messages****, la* ***communication indirecte via l'environnement*** *inspirée par l'optimisation par colonies de fourmis ou l'****apprentissage par renforcement****. Dans cette dernière technique, un agent doit choisir les actions à réaliser en fonction de la situation, de façon à maximiser son gain. Aucune communication avec les autres agents du système n'est utilisée. Ces techniques ont montré leur adéquation pour la résolution de différents types de problèmes avec différents niveaux de complexité. Ces approches permettent grâce aux processus de* ***décision locale*** *une* ***adaptation rapide et en temps réel*** *aux changement dynamiques et aux perturbations. L'autonomie donnée à chaque agent permet l'obtention d'une architecture distribuée très peu couplée rendant ces systèmes plus robustes.*

Dans certains domaines d'applications, la résolution d'un problème dépend de l'expérience et l'expertise des ingénieurs. En effet, dans ces cas, les ingénieurs ne sont pas capables d'exprimer d'une manière complète le processus de résolution et les choix effectués. Ainsi, il est impossible de modéliser ou de définir le problème afin de le résoudre avec les techniques énoncées ci-dessus. Pour cela, un nouveau formalisme nommé le ***raisonnement par cas*** *est développé. Le raisonnement par cas consiste à résoudre un problème en utilisant les solutions déjà utilisées dans des cas similaires. La connaissance de l'expert ou de l'ingénieur est formulée sous la forme d'un ensemble de cas. Pour chacun, une description détaillée du problème est donnée ainsi que de la solution suivie et les étapes qui y ont mené. Les différentes techniques utilisant ce formalisme suivent un processus de résolution de quatre étapes:* ***recherche de cas similaires, réutilisation, révision et apprentissage****. Avec l'augmentation de la taille des données et la croissance de la complexité de cas et leur description, la distribution du calcul et la décentralisation du contrôle sont utilisées pour l'amélioration des résultats et des performances. Ces approches présentent différentes limites notamment durant l'étape de recherche de cas similaires. En effet, les résultats de cette étape dépendent* ***fortement de la description des cas*** *(souvent difficile dans les cas complexes) et des fonctions de similarités utilisées. Une autre limite concerne la* ***distribution de la résolution*** *qui consiste à diviser le problème sur*

plusieurs agents et combiner les résultats obtenus. Dans ce cas, chaque agent est capable d'obtenir la meilleure solution possible mais leur combinaison ne conduit pas nécessairement à une bonne solution globale.

En conclusion, nous déduisons que face à la complexité croissante des applications actuelles, ***les systèmes multi-agents par leur distribution du calcul et décentralisation du contrôle permettent l'obtention de systèmes robustes, flexibles et ayant la capacité de s'adapter rapidement à la dynamique*** *de leur environnement. Ces mécanismes nécessitent néanmoins la mise en place d'****interactions locales*** *entre les agents leur permettant de coordonner leurs actions localement afin de produire une solution au niveau global. En analysant toutes les techniques de résolution utilisant ces mécanismes, nous notons que la* ***coopération est une notion fondamentale*** *régissant ces interactions et augmentant la qualité des solutions obtenues. De ce fait, nous proposons la définition d'un modèle d'agent générique comportant des comportements et des interactions basés directement sur la coopération pour la résolution sous-contraintes de problèmes complexes.*

1.1 Introduction

Optimisation problems are omnipresent nowadays. Indeed, people, teachers, engineers must constantly solve complex problems and make decisions. Choosing the best way to reach work given the hour and the traffic, establishing the best timetabling given the availability of each participant or improving the performance of a designed model are different complex problems. Their complexity is due to the heterogeneity and diversity of the participating actors, their evolving constraints or the interdependency of the involved parameters making a global comprehension of the problem difficult or even impossible. Thus, choices must be done to solve these problems. Such choices are very important and challenging as they greatly influence the obtained results.

Since the arising of computer science, a huge effort has been dedicated to solve such problems and assist users in their choices. Different formalisms have been developed for solving complex optimization problems under constraints. Among them, the *Constraint Optimization Problem (COP)* formalism is the most widely studied. In this formalism, problems are translated in a given framework where a set of variables (problem entities) must be assigned a value of a given domain so as to minimize or maximise an objective function. Solving such problems consists in exploring the search space and finding the best assignment to the variables. In the earlier days of computer science, *Uninformed Search* techniques have been studied. These techniques use a complete and exhaustive exploration of the search space before a solution is obtained. Given the increasing size of the search space, those methods require a prohibitive amount of time before a solution can be obtained. Thus, the *Informed Search* techniques have been introduced. In these techniques additional information on the problem is used to speed the search. Two information types can be distinguished dividing by that the *Informed Search* techniques into two categories *Exact Methods* and *Approximate Methods*. The former are complete and guaranteed to reach an optimal solution but require additional solving time and a good knowledge of the problem to solve. The latter are approximate but have shown their adequacy to solve a large variety of *COP* problems. The most important class of approximate methods is *Meta-heuristics*.

Still, those techniques have shown their limits to handle the growing complexity of current applications, especially when confronted to unpredictable and changing events that produce dynamics in the system. Thus, researchers have been working on their improvements using the distribution of computation and the decentralisation of control. A new formalism, *Distributed Constraint Optimization Problem (DCOP)* has then been defined, and different solving techniques commonly known as *Multi-Agent Systems* have been proposed.

In some domains, translating the problem into the *COP* formalism is impossible as the solving process depends on the experts knowledge and experience. Indeed, experts find it hard to articulate their thought processes when solving problems. This is because knowledge acquisition is extremely difficult in these domains, and is likely to produce incomplete or inaccurate problem specifications. In such cases, expertise is embodied in a library of past cases rather than being encoded in classical rules or constraints. The common techniques to solve such problems are known as *Case-Based Reasoning (CBR)* techniques.

The chapter is organised as follows: in section 1.2 the *COP* framework is defined, sections 1.2.1.1, 1.2.1.2 and 1.2.2 presents the different techniques developed for its solving. The *DCOP* framework is presented in section 1.3. Its solving techniques are introduced in sections 1.3, 1.3.1, 1.3.2 and 1.3.3. Section 1.4 introduces the *CBR* techniques and underlines their limits and enhancements for complex problem solving under constraints. A global discussion (section 1.5) concludes the chapter and introduces the main contribution of this work, as a way to address the limitations highlighted in this chapter.

1.2 Constraint Optimization Problem

The *Constraint Satisfaction Problem (CSP)* framework is a mean to model problems to solve in Artificial Intelligence or Operations Research. It consists in a set of variables ($X = x_1, ..., x_n$), which take values in specific domains ($D = D(x_1), ..., D(x_n)$) and are restricted by constraints ($C = c_1, ..., c_k$). We distinguish between a partial or complete assignment (or solution) where some or all of the variables are assigned. Solving the problem consists in finding a complete assignment with no constraint violation. Such an assignment is called consistent or legal.

In different *CSPs* such as Scheduling Problems [Brandimarte, 1993] constraints are associated to cost functions (i.e. penalties on non respected delays). In addition to the verification of the constraints, a solution must minimize the costs for these problems. Unfortunately, some *CSPs* are over-constrained and no solution exists for them. In such problems, constraints are also associated to cost functions and solutions must minimize the cost of the violated constraints. In both cases, problems are known as *Constraint Optimization Problem (COP)*.

In brief, a *COP* is a *CSP* that requires a solution that optimizes (maximizes or minimizes) an *objective function* also called *a fitness function*. This objective function usually combines different costs functions.

Such problems are commonly presented using constraints graphs. Their search space

is represented using a *tree* where each node is associated to a specific partial solution of the problem. The root node is called the *initial state*. The final state (or *leaf*) is called *goal state*. A goal test function is usually used to determine if a given state is a goal or not. A *successor function* enables the exploration of the graph by indicating from a given state which states are possible to reach. Nodes are connected by arcs each associated to a cost. In some cases, different goal states can exist each reached with having a different cost path. Solving the problem consists in finding the global optimal path called *the global optimal solution* (Definition 1) from the initial state to a goal state, other paths are called *local optimal solutions* (Definition 2).

Definition 1. ***Global optimal solution [Glover and Kochenberger, 2003, chap. 6]***

A global optimal solution is a solution that has a better objective function than all solutions of the search space.

Definition 2. ***Local optimal solution [Glover and Kochenberger, 2003, chap. 6]***

A local optimal solution is an optimal solution in a given region of the search space.

1.2.1 Solving Techniques Overview

Different solving techniques have been developed. [Russell and Norvig, 2003] divided them into two types: *Uninformed Search* and *Informed Search*.

1.2.1.1 Uninformed Search

Also called *Blind Search*, the uninformed search concerns techniques that do not have any additional information on the problem beyond that provided in the problem definition. They are able to expand the search tree by generating *successors* and determine if a given state is a goal or not. The most common *Uninformed Search* algorithms are: *Breadth-first Search, Uniform-Cost Search, Depth-First Search, Depth-Limited Search* and *Iterative Deepening Depth-First Search* [Russell and Norvig, 2003].

The Breadth-First Search expands all the nodes of a given level before any nodes of the next level are expanded. If an expanded node generates a goal state, the solution is returned and the algorithm stops. Indeed, the cost of each path is not considered when expanding a node. Thus, the *Breadth-First Search* is more considered as a *CSP* solver algorithm than a *COP* algorithm. It gives an optimal solution when all arc costs are equal.

The Uniform-Cost Search is an extension of the *Breadth-First Search* fixing this drawback as instead of expanding the first available node, it expands the node with the lowest path cost which lead to optimal solutions whatever the arc costs are.

In the Depth-First Search nodes are expended in depth. Thus, the complete branch of a given node is explored before a new node of the same level is expanded.

The Depth-Limited Search is a variant of the *Depth-First Search* where the nodes at a specific depth l are treated as if they have no successors. The definition of this limit l is

very important as it can be source of incompleteness. Indeed, if a goal state is at a depth d unknown at the beginning, choosing $l < d$ won't lead to a solution while choosing $l > d$ is non-optimal as it costs time and memory.

The Iterative Deepening Depth-First Search mitigates this drawback by repeating the *Depth-Limited Search* for different value of l. Algorithm 1.1 shows how the *Iterative Deepening Depth-First Search* applies repeatedly the *Depth-Limited Search* with increasing limits until a solution is found or until the *Depth-Limited Search* indicates that no solution exits.

Algorithm 1.1: The Iterative Deepening Depth-First Search Algorithm [Russell and Norvig, 2003]

```
Input: problem
for depth ← 0 to ∞ do
    result ← Depth-Limited-Search(problem, depth)
    if result ≠ stop then
        return result
    end
end
```

ANALYSIS

As no additional information are given, the uniformed search techniques usually need to explore all the states of the search space before a solution is obtained. Thus, they usually provide good results but their solving time is exponential and they require high memory space. In addition to this, such techniques are highly dependent on the problem description and on the constructed search tree. Thus, they cannot be used in dynamic environment where the structure of the problem and the constraints evolve during time. These techniques have been widely improved using the *Informed Search* methods.

1.2.1.2 Informed Search

The Informed Search algorithms use problem specific knowledge and thus, have the ability to find good solutions more efficiently. They are provided with evaluation functions informing on how much a reached solution (partial or complete) is promising. Such functions incorporate additional knowledge on the problem which reduces the size of the search space and increases the exploration speed. In figure 1.1, such optimization methods are divided into two sets: *Exact Methods* and *Approximate Methods*. As *Meta-heuristics* (highlighted in the figure) have shown their adequacy to solve a large variety of problems presenting different characteristics that this work is interested in, they are presented in a separate section (1.2.2).

A. EXACT METHODS The Exact methods also named *Complete Algorithms* provide optimal solutions and guarantee their optimality. They are based on enumerative methods that can be viewed as tree search algorithms that do a complete exploration but only on

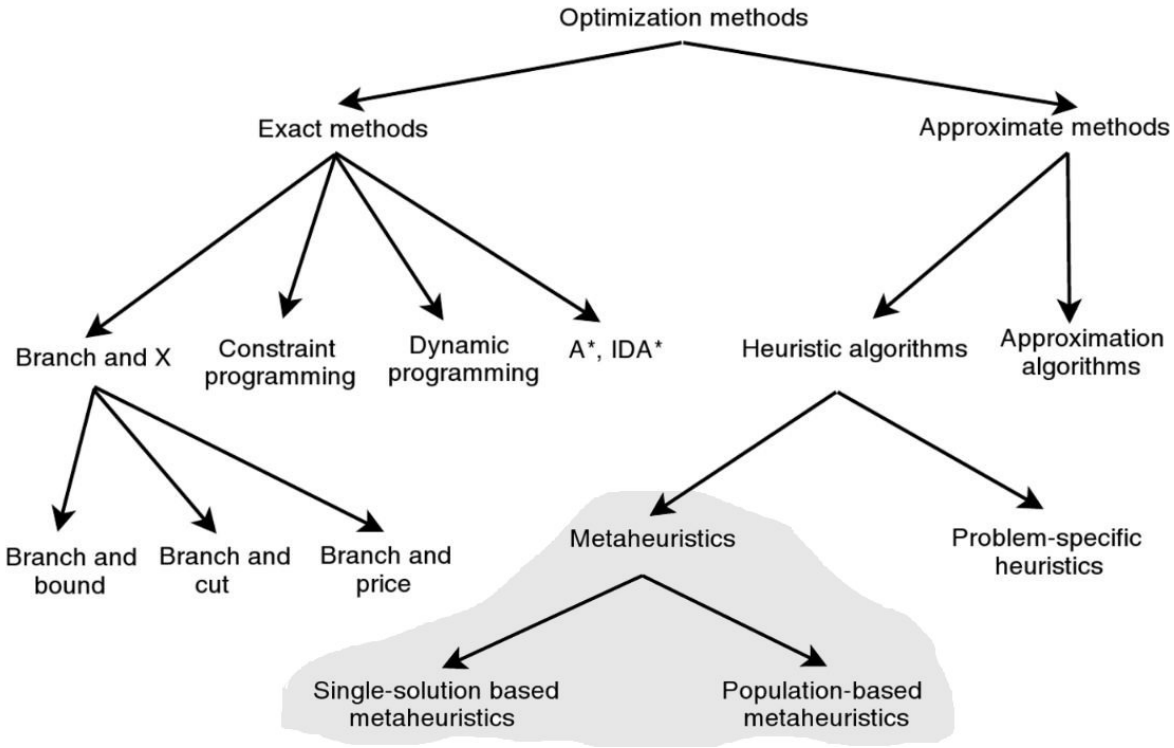

Figure 1.1 — Classical Optimization Methods Classification [Talbi, 2009]

the interesting parts of the search space. The *Branch and X* family algorithms and *A** enumerate all optimal solutions of the considered problem. They use a bounding function in order to prune subtrees that do not contain an optimal solution. Constraint Programming algorithms use propagation and filtering techniques to eliminate non feasible solutions. *Dynamic Programming* is based on the recursive division of a problem into simpler subproblems. Among those algorithms, a presentation of the *Branch and Bound, A** and *Dynamic Programming* algorithms is given.

The Branch and Bound algorithm is one of the most popular enumerative methods used to solve optimisation problems in an exact manner. This algorithm explores the search space by building a tree representing the problem being solved and its associated search space. The leaf nodes are the potential solutions while the internal nodes are subproblems of the total solution space. Two main operators are used to construct such tree: *branching* and *pruning*. The algorithm proceeds in several iterations during which both operators intervene. The *branching* operator indicates the order in which the branches are explored *depth-first* or *breadth-first*. The *pruning* operator eliminates the partial solution that does not lead to optimal solutions. Figure 1.2 shows an execution of the *Branch and Bound* algorithm considering the *Traveling Salesman Problem (TSP)* problem with 4 cities. A solution to the problem is an optimal path connecting the four cities and visiting each city only once. For the considered problem, the optimal path is {1;2;3;4;1}. The right side of the figure shows the cost of the different connections between the cities. The left side shows the tree search constructed by the *Branch and Bound* algorithm. We can see that after expanding the first path a cost value of 70 is found. While exploring the second, third, forth and sixth paths,

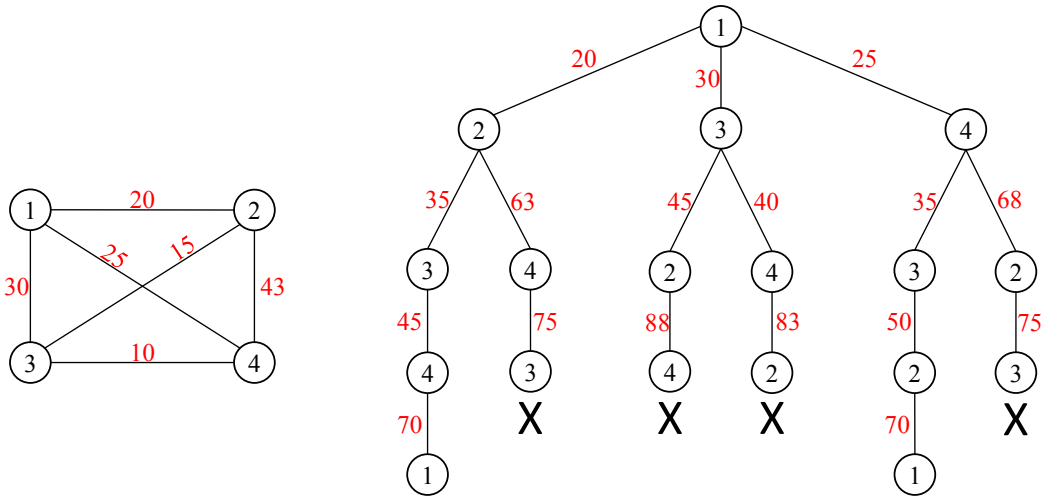

Figure 1.2 — Example of an execution of the *Branch & Bound* on the Travelling Salesman problem with 4 cities

before finishing the construction of the solution, the algorithm cuts those paths as a higher cost value is computed.

In the *A algorithm**, the tree search is constructed using an evaluation function that combines the exact cost of the expanded node (from the start node to the current one) and the estimated cost to the goal node (from the current node to the goal). This estimation is done using a function called a *heuristic*. *A** is guaranteed to provide optimal solutions if and only if this *heuristic* is *admissible*, in the sense that it does not over-estimate the cost to reach the goal. Thus, it is not possible to miss an optimal solution and the interesting parts of the search space are explored.

In Dynamic Programming, the problem is recursively divided into simpler subproblems and the solution is obtained after a sequence of partial decisions. Such techniques are applied to solve search and optimization problems which exhibit the characteristics of *overlappling subproblems* and *optimal substructure*. The *overlapping subproblems* characteristic concerns the ability of the problem to be divided into interdependent subproblems, while, the *optimal substructure* means that the solution to the problem can be obtained by the combination of optimal solutions to its subproblems. Thus, their usage is limited to specific type of problems.

Analysis

Unfortunately, despite the guarantee of optimality, those techniques fail to solve large instances of difficult problems. For example, in 1998, [Applegate et al., 1998] reported that the maximum size of the *TSP* problem that was solved to optimality is 13.509 cities with a running time approximated to 10 years[1]! In 2006, given the evolution of the search strategies, the size of the largest *TSP* problem solved to optimality is 85.900 cities with a running time approximated to 136 CPU-years [Applegate et al., 2006]. It is commonly

[1]The reported time is an estimation of the cumulative CPU time spent on the 48 workstations used to run the simulation

accepted that for complex applications, such techniques cannot be efficiently used to adapt to the dynamics and provide fast and good solutions.

B. APPROXIMATE METHODS

The Approximate methods generate high quality solutions in a reasonable time for practical use, but there is no guarantee of finding a global optimal solution. They are divided into two subclasses: *Approximation Algorithms* and *Heuristic Algorithms*. *Approximation Algorithms* provide a worst-case performance guarantee in both computational time and solution quality. *Heuristics* have acceptable performance in a wide range of problems but do not guarantee to find global optimal solutions or any lower bound quality for the obtained solution. In *Approximation Algorithms*, a study of the problem structure is required for the guarantee of performance which limits their applicability. For many real-life applications with an evolving dynamic environment where the structure of the problem is discovered during time, such algorithms are not applicable. As *Heuristic Algorithms* have shown their effectiveness to solve large and complex problems, this state of the art (section 1.2.2) concentrates on them and their evolution to integrate complexity.

1.2.2 Heuristics and Meta-Heuristics

As shown in figure 1.1, *Heuristic Algorithms* are classified into two subclasses, *Problem-Specific Heuristics* (Definition 3) and *Meta-heuristics* (Definition 4) [Blum and Roli, 2003].

Definition 3. ***Problem-Specific Heuristics***

Problem-Specific Heuristics are search strategies that are designed to efficiently solve a very specific problem or instance.

Definition 4. ***Meta-Heuristics***

Meta-heuristics are high-level strategies that use other problem specific heuristics. They aim at finding the best combination between the ***exploration*** *(Definition 5) and* ***exploitation*** *(Definition 6) phases of the search space.*

Definition 5. ***Exploration***

In the exploration phase, new regions of the search space are discovered.

Definition 6. ***Exploitation***

The exploitation phase consists in exploiting locally the best discovered solutions and trying to improve them.

Meta-heuristics are characterized by the following fundamental properties:

- Meta-heuristics are approximate and usually non-deterministic strategies that guide the search process to efficiently explore the search space.

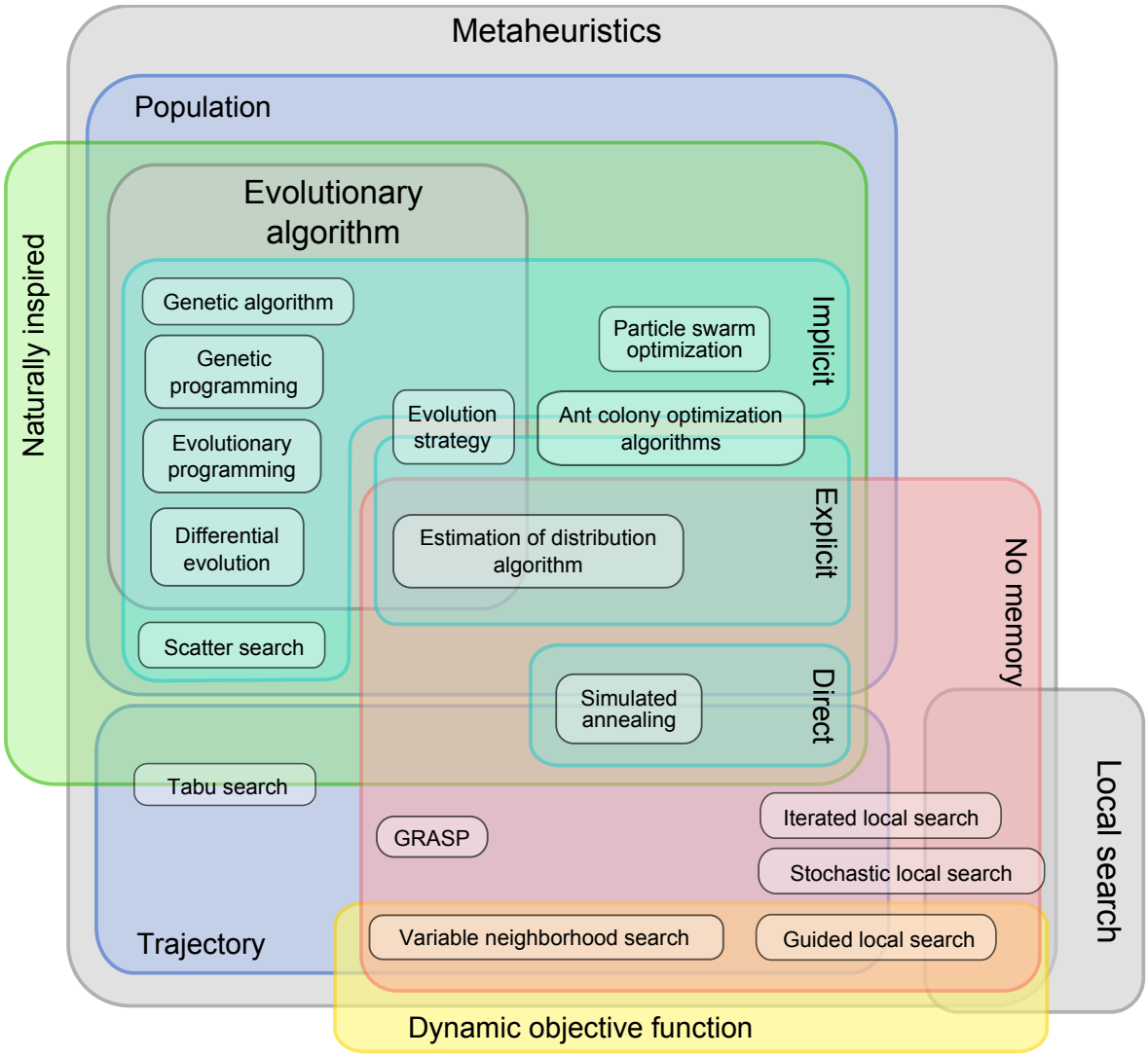

Figure 1.3 — Meta-heuristics Classification from Johann Dréo via Wikimedia Commons

- ▷ Meta-heuristics are not problem-specific. They may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- ▷ The basic concepts of meta-heuristics permit an abstract level description.
- ▷ Techniques which constitute meta-heuristic algorithms range from simple local search procedures to complex learning processes.

Figure 1.3 summarizes the different categories used to classify meta-heuristics. The most common and used one is the *Single Point Search vs. Population-Based*, which differ by the number of solutions explored at the same time.

1.2.2.1 Single Point Search Algorithms

Single Point Search Algorithms, also called *Trajectory Algorithms*, work on a single current solution and evolve it using local search procedures that improve the solution using a

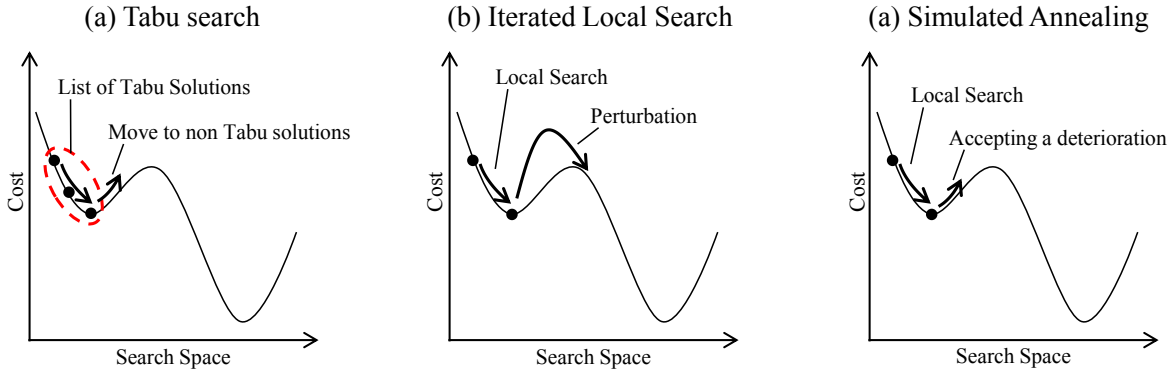

Figure 1.4 — The principle of different *Single Point Search* Techniques [Meignan, 2008]

local neighbourhood. Algorithm 1.2 shows the main step of algorithms based on these techniques. First, a set of candidate solutions $C(s)$ called *neighbourhood* is generated by applying local transformation of the current solution (s). Then a new solution is selected among the set of candidate solutions. This selection strategy is based on the objective function to optimise. These generation and replacement phases can store some history of the search in a memory that can be used in the generation of the candidate list of solutions and the selection of the new solution. The memory usage influences the speed of the search. The main problem of this procedure is that it is easily trapped in a local optimum. To avoid this problem, different strategies have been developed such as *Tabu Search, Iterated Local Search* and *Simulated Annealing* (figure 1.4).

Algorithm 1.2: High level template of *Single Point Search Algorithms* [Talbi, 2009]

Input: Initial solution s_0
Output: Best Solution found
$t = 0$
repeat
  /*Generate candidate solutions from s_t*/
  Generate($C(s_t)$)
  /*Select a solution from C(s) to replace the current solution s_t*/
  s_{t+1} = Select($C(s_t)$)
  $t = t + 1$
until *Stopping criteria satisfied*

While exploring the neighbourhood, the **Tabu Search** [Glover and Laguna, 1993] avoids going back to early explored solution using a tabu list. This list only registers recent found solutions and by that it is considered as a short-term memory. To improve *Tabu Search* efficiency, techniques using mean and long-term memory have been introduced [Glover and Kochenberger, 2003, chap. 2].

The **Iterated Local Search** strategy [Glover and Kochenberger, 2003, chap. 11] consists in perturbing the solution whenever a potentially local optimum is reached. It alternates between two cycles: finding a solution that is a local optimum in a given neighbourhood and resetting the current affectation to another point of the search space. These two cycles

are repeated until a termination condition is met. The changes applied to the local optimum are neither too small nor too large. If they are too small, few new solutions will be explored. If they are too large, a random restart algorithm is obtained.

Simulated Annealing introduced by [Kirkpatrick et al., 1983] is a probabilistic method that works by emulating the physical process whereby a solid is slowly cooled so that when eventually its structure is "frozen", this happens at a minimum energy configuration. Thus, the parameters of this technique are: energy and temperature. The energy parameter represents the cost of the solution. A new solution is accepted only if it improves the cost of current solution. Otherwise, its acceptance depends on a probability measure computed using the temperature parameter. During the search process, this temperature decreases which decreases the acceptance probability too. Thus, at the beginning of the search, solutions with less quality are easily accepted while at the end, only better solutions are accepted.

1.2.2.2 Population-Based Search Algorithms

Population-Based meta-heuristics could be viewed as an iterative improvement in a population of solutions [Talbi, 2009, chap. 3]. First, an initial population of solutions called individuals is generated. Then, it is iteratively improved by the generation of a complete new population and the replacement of the current one using selection procedures. Such selection procedures are carried out from both current and new generated populations. Population-Based meta-heuristics differ in the way they perform the generation and the replacement procedures. Most of them are nature-inspired algorithms. The most important category of *Population-Based Search Algorithms* are *Evolutionary Algorithms*.

Evolutionary Algorithms are inspired by the theory of evolution presented by C. Darwin [Darwin, 1859]. In theses algorithms, crossover, mutation and selection procedures are used for the population improvements. The crossover procedure enables the generation of new solutions by crossing two current individuals that play a major role (i.e. having the best cost performance). The mutation procedure is used to promote diversity by the random modification of the contents of an individual. The most well-known *Evolution Algorithm* is the *Genetic Algorithm (GA)*. It follows the standard procedure shown in algorithm 1.3 where each individual is a particular chromosome regrouping a set of genes called genotype. The fitness function evaluate each chromosome by computing the characteristics of the genotype. In a realistic problem, it combines several measures to optimise and thus, it encapsulates the objective function of the problem.

1.2.2.3 Analysis of Meta-heuristics

These classical *meta-heuristics* are approximation methods that have shown their adequacy to solve a large set of types of problems with varying levels of complexity. Despite their adequacy, they show several limits.

These methods rely on the *COP* formal framework in which a given problem must be translated. For some problems the translation is straightforward but still a difficult task, especially since choices made during this phase may strongly impact the performances of

Algorithm 1.3: A genetic Algorithm outline [Glize and Picard, 2012]

```
foreach chromosome i do
    initialise x_i by the set of genes
    x_i.fitness ← f(x_i)
end
while termination conditions not met do
    compute the fitness ∑ x_i.fitness of the overall new population
    foreach individual x_i of the population do
        select two individuals (x_i, x_j)
        (x'_i, x'_j) ← crossover(x_i, x_j)
        (x'_i, x'_j) ← mutation(x'_i, x'_j)
        x'_i.fitness ← f(x'_i)
        x'_j.fitness ← f(x'_j)
        insert offspring in new generation population
    end
    replace the current population with the new population
end
```

the solving process. Moreover, for complex problems with different interacting entities and dynamic events, problems are easily described by their entities and their interactions. Thus, having a global specification of the problem with a complete description of the constraints is difficult or even impossible. That is why problems are usually simplified to fit the framework. In addition to this, the *COP* framework requires the definition of a global objective function that in complex problems is difficult to define and can change during the execution.

Another limit of these approaches is the need for parameter tuning. For example, in the *Tabu Search* algorithm, the size of the memory plays an important role in the exploration phase. Ideally, we would like to store all the tested solutions for a better exploration of the search space. This results in a high memory usage and increases the solving time as additional time is required to access the memory. *Tabu search* using short-term memory, reduces the memory usage but needs additional computational time for the exploration phase. Thus, the question is: what is the right memory size? In the same manner, genetic algorithms require the tuning of probability parameters to improve individuals selection, crossing, and mutation procedures. Those tuning are highly problem specific and may even require to change during runtime when confronted to evolving problems.

COP are known to be NP-Hard. Thus, the search space grows exponentially with the growth of problem instances. As these approaches are centralized and non-distributed, additional computational time is required for the exploration phase.

This centralized control depends on a global objective function for deciding whether a solution is consistent or not. In mono-objective problems, this results in obtaining optimal solutions. In multi-objective problems, the different objectives are conflicting and aggregating them in one global objective function results in non adequate solutions. The

Pareto approach solves multi-objective problems by proposing a set of solutions representing seemingly equivalent solution with regard to the objectives. When this approach relies on classical meta-heuristics, it suffers from their limits.

In addition to this, the control centralization results in non-flexible methods hardly adaptable to dynamic events. When dynamic events such as change of constraints or the arrival of new entities happen, theses methods fail to adapt in real time, and a new solving from scratch is required. This centralization also leads to bottlenecks and lack of robustness needed in real world applications.

Finally, when designing a meta-heuristic, it is important to find the right equilibrium between the exploration and exploitation phases to be sure that all regions of the search space are explored and the best discovered solutions are well exploited without being trapped in a local optimum. In the above presented methods, some perform better during the exploration phase such as *Population-Based Search Algorithms* while other perform better during the exploitation phase such as *Single Point Search Algorithms*. Thus, an effort has been done to combine the advantages of two or more methods in one method. Such mixed methods are called *hybrid meta-heuristics*.

1.2.3 Hybrid Meta-Heuristics

The most commonly studied hybrid meta-heuristics consider a combination between *Population-Based Search Algorithms* and *Single Point Search Algorithms*.

One of the most well known hybridization algorithms is the *Memetic Algorithms* [Glover and Kochenberger, 2003, chap. 5]. They consider a combination of *Evolutionary Algorithm* such as *GA* with *Local Search* techniques. They aim to combine the strength of *GA* which is the ability to find promising areas of the search space (*exploration*) with the strength of local search which is their capabilities of quickly finding better solutions in the neighbourhood of a given starting solution [Blum et al., 2011]. Such techniques consist in improving the solutions obtained by the *crossover* and *mutation* procedures by applying on them a *local search* such as a *Tabu Search* [Glover et al., 1995] or *Simulated Annealing* [Tamilarasi and Kumar, 2010].

Another type of hybrid algorithms are self-contained meta-heuristics that are executed in sequence. *Population-Based Search* algorithms where the initial population has been generated using a *Greedy Heuristics*[2] [Ahuja et al., 2000] belong to this class. Another example of such hybrid meta-heuristics is the application of a *Local Search* to improve the results obtained by a *GA* [Hageman et al., 2003; Jat and Yang, 2010]. The main problem of these techniques is the size of the population that must be generated by the *Greedy Heuristics* at the beginning of the search or that must be improved by the *Local Search* which can be time consuming.

Thus, a new type of *hybrid algorithms* has been studied. It concerns self-contained meta-heuristics being executed in parallel and cooperating to find the optimal solution. An example of algorithms of this class, is the *Island model* for *GA* where the population is divided

[2] *Greedy Heuristic* are stepwise, iterative procedures that create the population by making seemingly best choices at each step until all the population is generated.

into different subpopulations each being evolved by a *GA*, and individuals can migrate between subpopulations. This algorithm and different other parallelization techniques of the *GA* are presented in [Nowostawski and Poli, 1999]. The idea is to perform distributed search by different meta-heuristics and making them cooperate by exchanging the best found solutions. These techniques have been also developed using different *Single-Point Search algorithms* such as *Tabu Search* [James et al., 2009].

1.2.3.1 Analysis of Hybrid Meta-Heuristics

A first point to underline is that when combining the functionalities of different *meta-heuristics*, designers must pay attention to when to apply the hybridization. Indeed, if it is applied at each iteration, very competitive solutions will be generated in the beginning of the search, limiting the abilities of the *GA* to explore distant points of the search space as it is drown to these local attractors.

The introduction of the parallel meta-heuristics address this centralization problem by using cooperation between different meta-heuristics which improves the results. Such techniques enable to speed up the search and solve large-scale problems as the search space is divided into different smaller search spaces and to improve the quality of the obtained solutions as the different meta-heuristics cooperate and exchange information about the best found solutions. But, they possess different limits. They require to tune the parameters of each used meta-heuristic, to decide when and what information must be exchanged which can influence the communication costs and how the communication must be done (the exchange topology) in order to maintain the equilibrium between the exploration and exploitation phases and avoid premature convergence.

In brief, in spite of their adequacy for specific problems, these methods need to be improved in order to tackle the growing complexity of today's applications. Different approaches based on the distribution of computation and control decentralization have been proposed. The next section presents a selective review of these approaches.

1.3 Distributed Constraint Optimization Problem

A *Distributed Constraint Optimization Problem (DCOP)* [Matsui et al., 2008] is a *COP* where the variables are managed by agents that control the value they take. Agents must coordinate their choice of values so that a global objective function is optimized. The global objective function is modelled as a set of constraints, and each agent knows the constraints in which its variables are involved. Thus, the solving process is distributed among a set of agents. Such solving techniques are known as *Multi-Agent Systems (MASs)*.

A *MAS* is defined as a system in which several interacting, intelligent agents pursue a set of goals or perform a set of tasks [Weiss, 1999]. Different solving techniques are designed using *MAS* approaches. Such techniques differ by agent identification, agent interaction and agent behaviour.

The most commonly used definitions of the term *Agent* are:

[Ferber, 1995] definition:
An agent is an *autonomous* physical or virtual entity able to *act* (or communicate) in a given *environment* given *local perceptions* and *partial knowledge*. An agent acts in order to reach a *local objective* given its *local competences*.

[Weiss, 1999] definition:
An agent is a computer system that is situated in some *environment*, and that is capable of *autonomous actions* in this environment in order to meet its design *objectives*.

These definitions agree on the fact that an agent is autonomous, acts in an environment and possesses an objective to reach. By autonomous, we mean that an agent is able to control its own state and behaviour. Its existence is independent from the existence of other agents. It is able to maintain its behaviour in different types of environment such as dynamic, inaccessible and non-deterministic environments [Russell and Norvig, 2003; Weiss, 1999]. The agent environment concerns all the objects and other agents that influence the agent behaviour. Its behaviour is divided into three steps: *Perception, Decision* and *Action*. In the *Perception* step, the agent reads and interprets messages it receives from its environment in order to update its local representations. The *Decision* step concerns two points: given the updated information and its local knowledge and skills, the agent decides the set of possible actions that must be performed in order to respond to a situation, to answer a request or to anticipate future situations. Once this set is defined, the agent decides, given its capacities, skills and resources, what is the subset of actions that it will effectively perform. Finally, in the *Action* step, the agent performs the selected actions.

Concerning the agent identification, three different techniques are commonly used: *Variables Agentification* approaches, *Nature Inspired* approaches and *Domain Entities Agentification*.

1.3.1 Variables Agentification

These approaches propose a simple distribution of the problem, when given a *COP* description. Each agent is responsible for the affectation of it is own variable while respecting its constraints with its neighbourhood. Most of the algorithms developed following this approach organize agents into a hierarchy based on the constraint definition. *Asynchronous Distributed Constraint Optimization (ADOPT)* and *Optimal Asynchronous Partial Overlay (OptAPO)* belong to the well-known and studied algorithms of this approach.

1.3.1.1 Asynchronous Distributed Constraint Optimisation (*ADOPT*)

ADOPT [Silaghi and Yokoo, 2009] is an asynchronous *DCOP* solver, which is guaranteed to find an optimal solution. It orders agents using a *Depth-First Search (DFS)* tree deduced from the graph of constraints, and restricts communication to parent/child relationships. Constraints exist between a variable and any of its ancestors or descendants, but not between variables in separate sub-trees. Agents interact using three kinds of messages (figure 1.5):

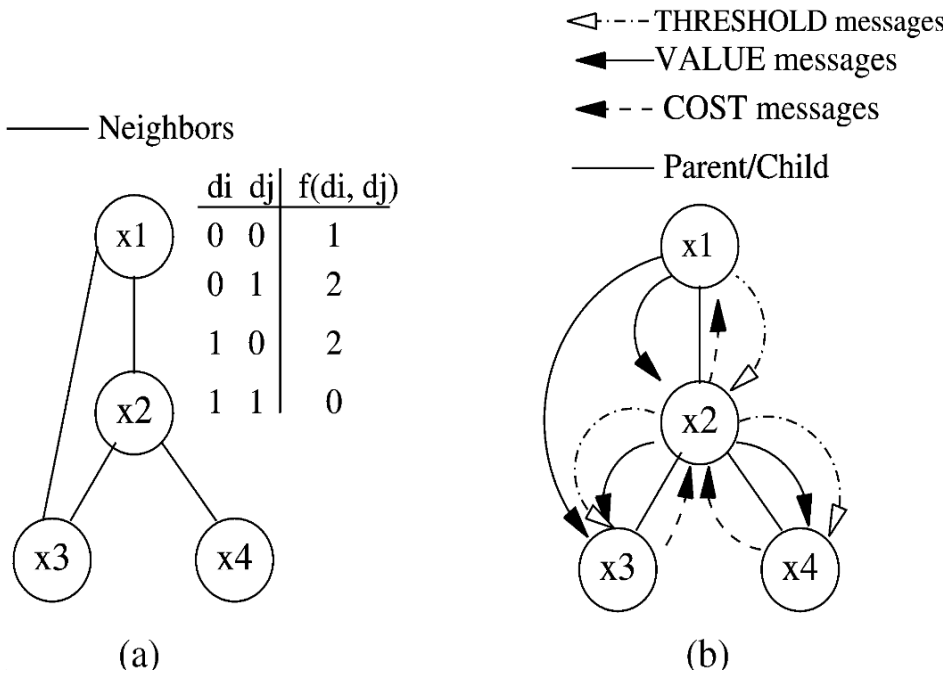

Figure 1.5 — From a *DCOP* (a) description to the *ADOPT* (b) *DFS* tree and communication graph

1. **Value** messages that communicate the assignment of a variable from ancestors to descendants that share constraints with the sender. At start, each agent randomly chooses a value for its variable and sends appropriate VALUE messages.

2. **Cost** messages, sent from a child to its parent, indicate the cost of the subtree rooted at the child.

3. **Threshold** messages contain Lower and Upper costs bounds. Sent from a parent to its child, they guide agents choosing their values. They are used to detect the termination of the execution. Indeed, when the upper bound and the lower bound meet at the root agent, then a globally optimal solution is found.

1.3.1.2 Optimal Asynchronous Partial Overlay (*OptAPO*)

OptAPO introduced in [Mailler and Lesser, 2006] increases the decentralisation using cooperative mediation. *OptAPO* is a sound and optimal algorithm where agents extend and overlap the context that they use for making their local decisions as the problem solving unfolds. Thus, this algorithm discovers difficult portions of a shared problem through trial and error and centralizes these sub-problems into a mediating agent in order to take advantage of a fast centralized solver. Agents of this algorithm work by constructing a *good_list* and maintaining a structure called the *agent_view*. The *good_list* holds the names of the variables that are known to be connected to the owner by a path in the constraint graph. The *agent_view* holds the names, values, domains and constraints of variables to which an agent is linked. Whenever the agent receives information on other agent variables,

it records them in the *agent_view*. The variable is added to the *good_list* only if the graph created by all the variables of the list remains connected, ensuring that only agents having an interdependency link with the owner are added. As the problem solving unfolds, each agent tries to solve the problem it has centralized within its *good_list*. To do this, an agent acts as a mediator and computes a solution to a portion of the overall problem and recommends value changes to the agents involved in the mediation session (agents in its *good_list*). When different linked agents want to mediate, the mediator is selected using a priority value based on the size of the *good_list*. Thus, the most knowledgeable agent is selected to solve the subproblem. As a mediator, the first attempt of the agent is to solve conflicts by changing its own value. If no possible value is found, the agent starts the mediation session and uses a *Branch and Bound* algorithm to solve its subproblem. If no solution is possible without causing a violation with agents outside of the session, the mediator adds links to those agents assuming that they are linked to its variable, increasing by that the size of its *good_list*.

Figure 1.6 presents an illustration of *OptAPO* execution for solving the 3-colouring graph problem. This problem consists in assigning a color from a set of three colors {dark red; blue; white} ({dark grey; light gray; white} for wite & black figures) to each region of a map such as two adjacent regions cannot have the same color. The instance shown in the illustration considers a map of 6 regions {A;B;C;D;E;F} represented by agents and sharing 8 constraints. At the start of the algorithm, each agent chooses a color to its region (as shown in the left side of the figure) and sends this information to agents having a constraint with it. Each agent, after updating its *good_list* and *agent_view*, checks its view. In this example two conflicts are detected: (A;B) and (C;D). Agent C having the higher priority value among its neighbours and being unable to solve its conflict locally, sends an invitation to {A;B;D;E} to start a mediation session. Thus, each agent labels its domain elements (here the three possible colors) and sends it back to C :

- A - blue conflict with C, dark red conflict with B, white no conflict
- B - blue conflict with C & D, dark red conflict with A, white no conflict
- D - blue conflict with C, dark red conflict with B, white conflict with E
- E - blue conflict with C & F, dark red no conflict, white no conflict

Once all of the responses are received, the mediator C conducts a *Branch and Bound* search that attempts to find a satisfying assignment to its subproblem and minimizes the amount of conflict that would be created outside of the mediation (here with F). In this example, C finds a solution that consists in changing B to green and D to red (left side of the figure). This solution is sent to agents and the problem is solved.

1.3.1.3 Analysis

ADOPT is proved to be complete. Nevertheless, like several algorithms in this approach, it suffers from the hierarchy structure between the agents which leads to different drawbacks. The primary drawback, is the existence of a central node which constitutes a

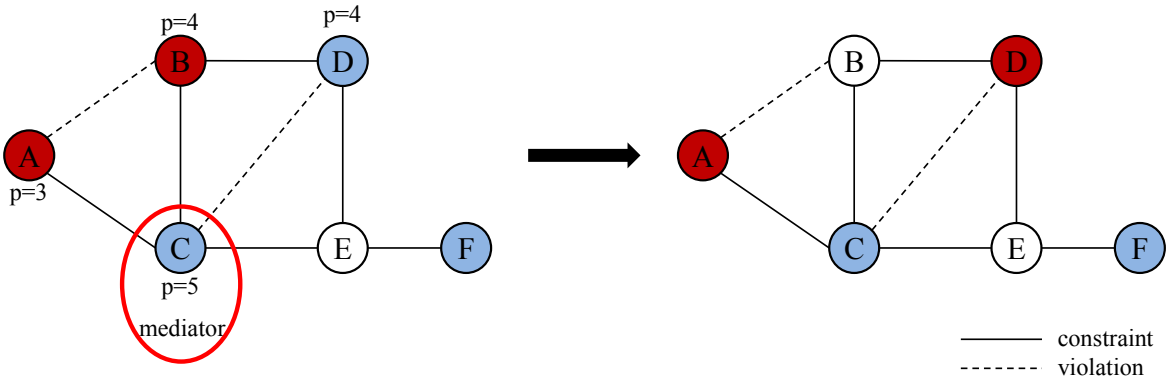

Figure 1.6 — Resolution of the 3-colouring graph problem using *OptAPO*

bottleneck, especially for large problem instances. Furthermore, as the structure is based on the constraints, whenever a constraint changes or new constraints are defined, a new structure must be established. In general, approaches based on centralization present a number of drawbacks limiting their usage in dynamic environment. [Ouelhadj and Petrovic, 2009] discusses several drawbacks of such approaches applied to dynamic scheduling in manufacturing systems.

The performances of *OptAPO* are closely related to the *Branch & Bound* algorithm. Indeed, each time an agent decides to mediate in *OptAPO* it conducts a *Branch & Bound* search in its *good_list*. As the size of the *good_list* grows, the size of the *Branch & Bound* search space increases dramatically. In addition to this, during the *OptAPO* execution, the *good_list* tends to increase as additional links are created due to external conflicts. So, there is a tendency of at least one agent achieving complete centralization which leads to executions of the *Branch & Bound* algorithm that involve all variables [Davin and Modi, 2005]. Thus, this algorithm is hardly applicable to real-world applications.

In *OptAPO*, different messages are exchanged between agents during the solving. These messages include information on the values taken by agents and their constraints. For example, when requested for a mediation session, each requested agent labels each of its domain values with the names of the agents that it would be in conflict with it if it were asked to take that value and return this information in an *evaluate!* message. The size of this message is related to the number of variables and the size of the agent's domain, which can increase dramatically and causes network overload. In addition to this, privacy and security problems can be underlined.

Finally, as discussed in section 1.2.2.3, these approaches require a translation of the problem to the *DCOP* formal framework which also limits their usage.

1.3.2 Nature Inspired

Known as *Swarm Intelligence*, algorithms of this approach are inspired from the collective behaviour of more or less social species such as ants, bees, wasps, termites, fishes and birds [Bonabeau et al., 1999]. The most studied and used nature inspired optimization algorithms are *Ant Colony Optimization (ACO)* and *Particle Swarm Optimization (PSO)* [Talbi, 2009, chap.

3].

1.3.2.1 Ant Colony Optimization

ACO algorithms are considered as *MASs* where each agent imitates the cooperative behaviour of real ants [Dorigo et al., 2000]. The main idea of *ACO* is to mimic the simple ant behaviour using simple communication mechanisms and performing complex tasks such as finding shortest paths to food sources (figure 1.7). Like ants, agents interact indirectly via the environment by tagging promising regions. Such tags are called *Pheromones* like the chemical olfactive and volatile substance left by biological ants to guide other ants toward a target point. Thus, *Pheromones* are subject to evaporation and reinforcement processes. In *ACO* each agent constructs a solution incrementally from partial solutions. Each agent decides to add a feasible solution component from the set of feasible neighbours with respect to the current partial solution. At each step, the agent drops pheromone in the visited position. The *ACO* algorithm requires the initialisation of different parameters such as the number of artificial ants to work in the search space which influences the speed of the search. Other parameters concern the initial value of the pheromone and the pheromone evaporation rate. The basic common approach for the pheromone update consists in having the agent increase the pheromone values of a position when visiting this position of the search space, while the pheromone values decrease by a fixed proportion as no agent visits the position. Thus, promising positions presenting parts of the better solutions contain high pheromone values and are more attractive. Those parameters are important as the pheromone guides agents near the paths to good solutions and avoids older paths by the evaporation process. This can be seen in figure 1.7. The first ant finds the food source (F), then returns to the nest (N) leaving behind a trail of pheromone (b). The other ants of the nest indiscriminately follow the different possible ways leaving trail of pheromones. As the pheromones evaporate during time, long portions of other ways lose their trail pheromones while the trail pheromone of the shortest route is strengthened which makes it more attractive (2). Thus, the shortest path connecting the nest to the food source emerges (3).

1.3.2.2 Particle Swarm Optimization

In *PSO* algorithms, agent behaviour is inspired from the social behaviour of natural organisms such as bird flocking and fish schooling [Kennedy, 2010]. Each agent is a candidate solution of the problem called *particle*. The *PSO* algorithms find optimal regions of complex search space through the interaction of individual agents in the population. It shares many similarities with evolutionary algorithm such as *GA*. However, it has no evolution operators. Each agent (*particle*) is characterized by its own position x_i and velocity v_i represented as vectors and indicating the flying direction and the movement to perform. Each agent possesses a neighbourhood which denotes the social influence between them.

Algorithm 1.4 presents an outline of a classical *PSO*. At each step, the fitness function $f(x_i)$ determines which particle has the best value in the swarm (g_{best}) and the best position ever visited by each agent (l_{best}). Each agent keeps track of the best explored solutions and

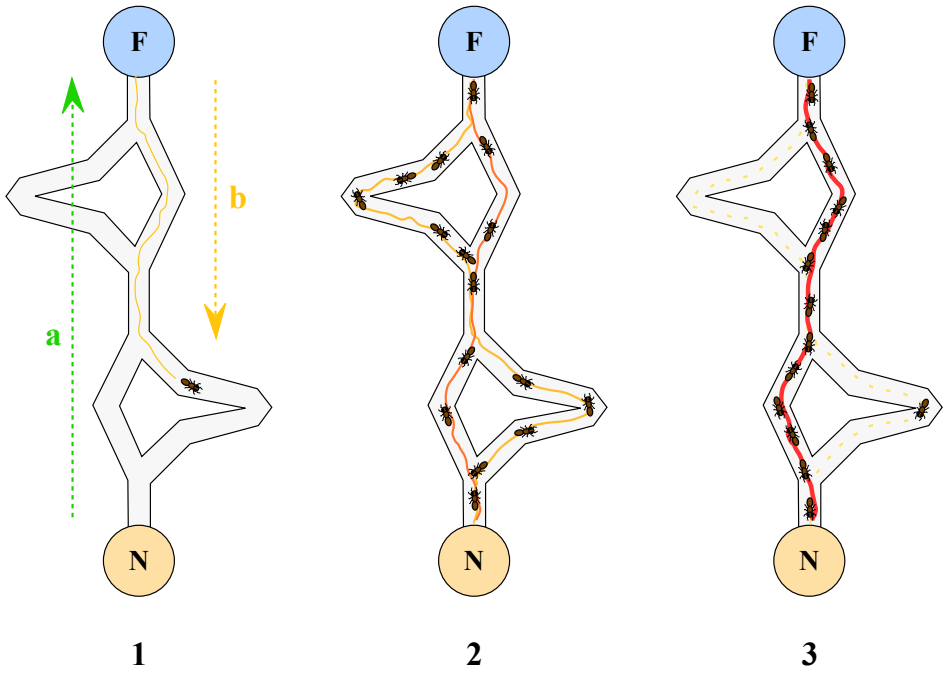

Figure 1.7 — Shortest path fund by an ant colony

updates its velocity toward its local and global best positions, updating by that its position. As agents exchange information on the best visited position, when an agent detects that it has a better quality solution, all the other agents move closer to it for better exploration of the region.

Algorithm 1.4: Particle Swarm Optimisation outline

foreach *particle i* **do**
  initialise position x_i and velocity v_i
  initialise the local best value: $l_{best} \leftarrow x_i$
  initialise the global best value: $g_{best} \leftarrow$ argmin $f(x_i)$
end
while *termination conditions not met* **do**
  foreach *particle i* **do**
    update velocity: $v_i = w.v_i + c_g.rand(g_{best} - x_i) + c_l.rand(l_{best} - x_i)$
    update position dimension: $x_i = x_i + v_i$
    update the local best value: $l_{best} \leftarrow$ argmin$[f(l_{best}, f(x_i))]$
    update the global best value: $g_{best} \leftarrow$ argmin$[f(g_{best}, f(x_i))]$
  end
end

1.3.2.3 Analysis

Both algorithms are similar to *Population-Based Search* meta-heuristics as each agent handles and evolves a complete solution of the considered problem. Thus the multi-agent

system is a population of solutions. Nevertheless, they differ from *Population-Based Search* meta-heuristics by the distribution of the population on a set of agents, and the individuals evolution. While in *Population-Based Search*, selection and evolution procedures are used considering all the population, in *ACO* and *PSO* this evolution is carried out by each individual. Indeed, each agent decides how to modify its solution given its representation of the environment and its interactions with other agents.

A first point to notice in these approaches is **the absence of a global control**. There is no central agent to decide if a global optimal solution is obtained. The behaviour of the whole system emerges from the interaction of simple agents.

Such techniques are considered as approximate methods with no quality guarantee on the obtained solution. But, given their decentralization, they are more robust to dynamics, such as the arrival of new agents. Unfortunately, as *Population-Based Search* meta-heuristics, **agents use global objective functions** to evaluate their costs, this limits the usage of these techniques to problems where such functions exist. In addition to this, *ACO* algorithms efficiently solve distributed network problems where shortest paths are required, but are hardly adaptable to other type of problems as for instance, they are **inappropriate for continuous problems** when the domain of variables is large as they require to modify the structure of the problem in a combinatorial one (by splitting the domains of the variables to intervals) or to change the pheromone model by a continuous one (as the pheromone trails are associated to a finite set of values related to the decisions that the ants make).

Another point to consider is **the need for parameters tuning** such as the number of agents in the ant colony, pheromone influence and evaporation rate in *ACO* or the velocity modification in *PSO*. Thus, solutions accuracy are sensitive to the dynamics present in recent applications (evolving of number of variables and constraints). The more the current problem configuration is far from the initial one, the less the initial parameter values are relevant and the more the result is far from the optimum.

In brief, these methods demonstrate that the distribution of computation and the control decentralization are required to efficiently solve dynamic problems, but suffer from several drawbacks that limit their usage to specific problems.

1.3.3 Domain Entities Agentification

These approaches consist in modelling agents from the domain description, independently from any *COP* formalisation. Software agents represent entities of the problem to solve or tools for coordination.

These approaches have been widely used to solve a large variety of complex problems and different types of agents have been developed. We point out two types of entities agentification. The first one considers the physical entities of the problem as agents. In the second one, the functional entities of the problem are agentified. For instance, considering the manufacturing process planning and scheduling problem (figure 1.8), in a functional decomposition approach, agents represent functional modules such as routing, materials handling, transportation management or load management, while in a physical decomposition approach, agents are used to represent entities in the physical world such as

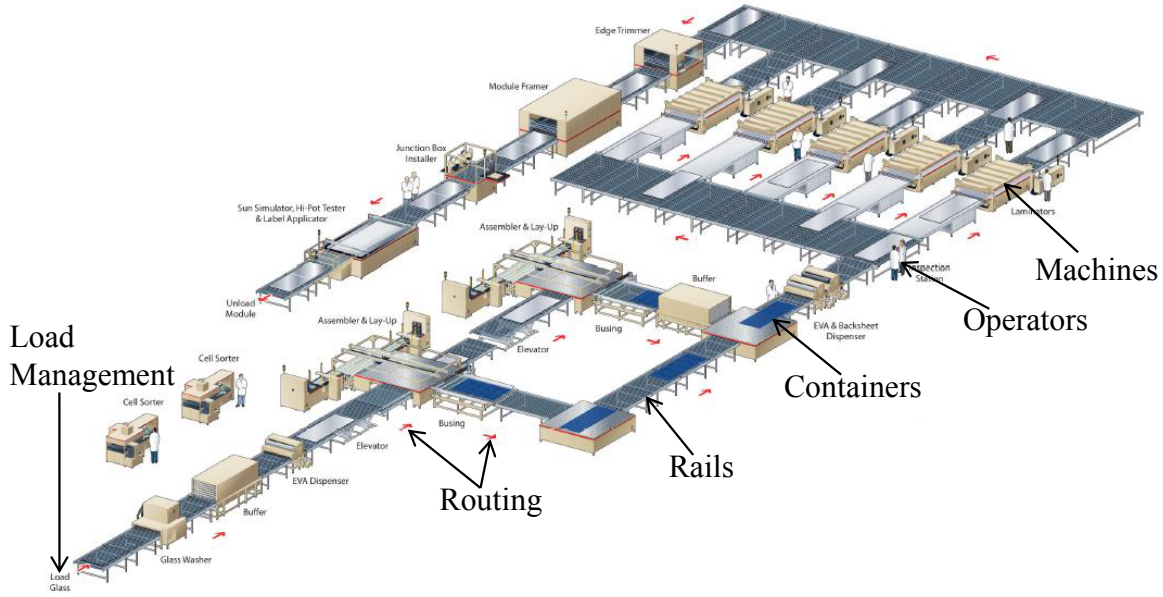

Figure 1.8 — *Functional & Physical* decomposition of the manufacturing control problem

operators, machines or tools. Considering a multi-disciplinary optimization problem such as complex product design, in a functional decomposition approach, agents represent the links between the different disciplines while in a physical decomposition approach, agents represent the considered disciplines.

In such a context, the designed *MAS* is close to the problem definition and no transformations are needed. Each agent represents an entity of the problem. Thus, agents possess the characteristics of the entity they represent and are designed with a corresponding local objective. Their behaviour is dictated by the description of the problem.

To reach their objective, agents use their local knowledge and interact with each other. By their interactions, they coordinate their actions and behaviour resulting in more coherent systems. [Weiss, 1999] defines coordination as *the property of a system of agents performing some activity in a shared environment*. This coordination enables the agents to self-organize their actions. From the obtained organization, the global function of the system emerges without explicit global control.

Different other optimization problems under constraints were solved using domain entities agentification techniques such as road traffic simulation [Doniec et al., 2008], vehicle routing [Weyns et al., 2008], robot task allocation [Lacouture et al., 2011; Gerkey and Mataric, 2002] or complex product design [Welcomme et al., 2006].

This variety of applications show us the adequacy of theses techniques to solve complex problems. Different from centralized approaches, agent-based approaches can **respond quickly to dynamic changes** and **disturbances through local decision making**. Thanks to the autonomy given to each individual agent resulting in a distributed loosely coupled architecture, these approaches provide **better fault tolerance** than traditional approaches, increasing by that **robustness** and **adaptivity**.

One of the most studied problem in the field is *intelligent manufacturing*. [Shen et al., 2006]

presents a detailed review of agent-based approaches used to solve different applications of the domain. In our case, we consider the manufacturing process planning and scheduling which is one of the well studied optimization problems. Different solving techniques have been investigated by the *CollInE*[3] group in France. Mostly all multi-agent solving systems agentify physical entities such as jobs and machines. The underlined differences between them pertain to the negotiation and cooperation processes used by agents to coordinate their activities. In the following, three different coordination mechanisms are presented: direct communication using *Dynamic Contract-NET (DynCNET)*, indirect communication using *stigmergy* and environment feedback using *Reinforcement Learning*.

1.3.3.1 Direct Communication: *Dynamic Contract-Net Protocol*

In some systems, direct communication between agents such as *Contract-Net Protocol* and its extended versions [Baker, 1991; Parunak, 1987; Smith, 1980] is used. [Weyns et al., 2007] introduces the *DynCNET* protocol, a dynamic extension of the *contract-net protocol* where already established contracts can be dynamically changed to support dynamic changes of the environment. *DynCNET* is a *m* x *n* protocol where an initiator that offers a task can interact with *m* participants and a participant that is searching for a task interacts with *n* initiators. The default message sequence of *DynCNET* shown in figure 1.9 consists in four steps:

1. after the initiator sends a call for proposals,
2. it selects among the received proposals,
3. a provisional winner that
4. informs the initiator that the task is started.

By having the possibility of revising provisional task assignment between step 3 and 4, the *DynCNET* ensures the management of dynamics. Indeed, even if the initiator selects a winner, it can change its selection if another more appropriate participant is found.

The *DynCNET* was tested on the *Automated Guided Vehicles (AGVs)* to transport loads in an industrial environment. In this problem, an *AGV* must drive to a load before it can pick it up and transport it to the destination. During this drive, different events may occur such as new loads entering the system that are more suitable for the *AGV* to transport or new *AGVs* more suitable to perform the transportation may become available. Thus, a flexible task assignment approach is required for this problem. To apply *DynCNET*, two physical agents are identified: *AGV* agents that correspond to participants and transported agents (loads) that corresponds to initiators. They interact using the default message sequence, and before the effective start of a transportation, both agents can change their affectations if new more suitable agents are discovered.

ANALYSIS

[3] `http://www.irit.fr/COLLINE/Presentations.html`

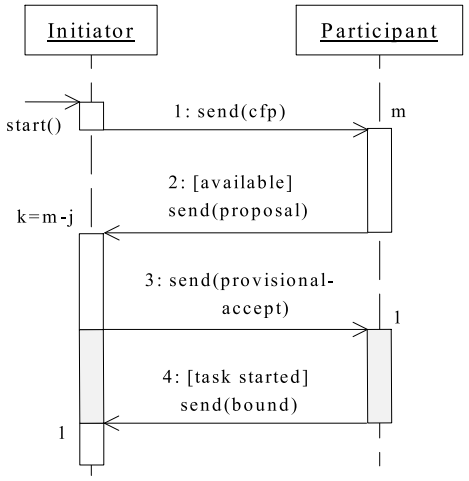

Figure 1.9 — DynCNET Basic Protocol [Weyns et al., 2007]

An important point to underline in this protocol, is the possibility given to agents to revise already taken decisions. This revision is based on the discovery of *more suitable* agents which still is an application dependent notion.

1.3.3.2 Indirect Communication: *Stigmergy*

In other systems such as [Rajabinasab and Mansour, 2011; Xiang and Lee, 2008; Valckenaers et al., 2006], agents communicate indirectly via the environment using markers. For example, [Valckenaers et al., 2006] presents a system where coordination and control are inspired by food foraging behaviours in ant colonies. The developed system is applied to scheduling in the manufacturing control problem. In this system, three types of agent are used.

- The *Resource Agent* controls a resource in the system. It has deep and detailed knowledge about the resource it controls and can provide information concerning the different states of the resource.
- The *Product Agent* corresponds to a product model in the system. It hold the knowledge on how it can be produced. In other terms it knows the list of operations to be executed and the qualified resources to execute them.
- The *Order Agent* corresponds to a task that needs to be executed. The agent must perform correctly the assigned work on time. It interacts with the product agent to discover what valid sequences are available.

Each task of the system is controlled by an order agent that must find a sequence of suitable resources to treat it. Two steps are required by the order agent to realise its goal. First, it creates *Explorer agents* that each search for a good solution to accomplish the given task. When an explorer agent finds a solution, it returns it to the order agent that maintains a set of candidate solutions (left side of figure 1.10). These candidate solutions are marked and reinforced regularly by exploring agents that can rediscover the most

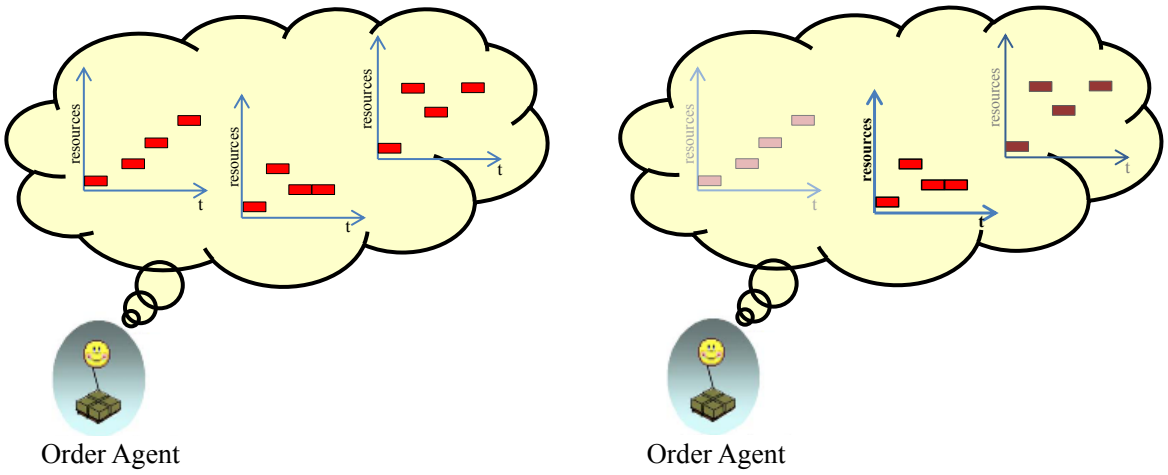

Figure 1.10 — The marking of the most attractive candidate solution and the evaporation of those not rediscovered

attractive candidates with a high probability. The markers of candidate solutions not rediscovered evaporate during time, decreasing their adequacy for the order agent (right side of figure 1.10). Second, after this set of candidate solutions has been constructed and the estimated starting time for the processing of the task approaches, the order agent selects the most adequate solution that becomes its intention. To execute its intention, the order agent generates intention agents, at a given frequency. Those agents follow and update the selected candidate solution. Indeed, they are able to mark the resource agent by the intention of their order agent. This action has two folds. It enables the resource agent to predict performance more accurately to its visitors (explorer and intention agents). Such performance are returned to the order agent that can update the selected candidate solution. The marks left by the intention agents evaporate. Thus, order agents must create intention agents at a sufficiently high frequency to maintain its booking intentions. This refreshing mechanism is required as order agents can adapt their candidate solution given the update information about the evolution of its current intention and the resources states.

Analysis

In brief, in this system, two coordination mechanisms inspired by ant colonies are underlined. The first one concerns the marking of the candidate solution while exploring the search space. The second concerns the marking of the resource agents while executing the treatment of the tasks. Both mechanisms imply the generation of different *Explorer* and *Intention* agents to maintain the information up to date. The system is **robust to dynamics** such as the arrival of new tasks as new order agents independent from the already existing ones are created, or resources breakdown as agents can always have updated informations concerning the resource states. In addition to this, the main advantage of this technique is the **flexible emergent forecasting**. Indeed, the discovering of different candidate solutions and the reservation of resources by intention agents facilitate the prediction of the performance of the manufacturing system while being flexible and robust

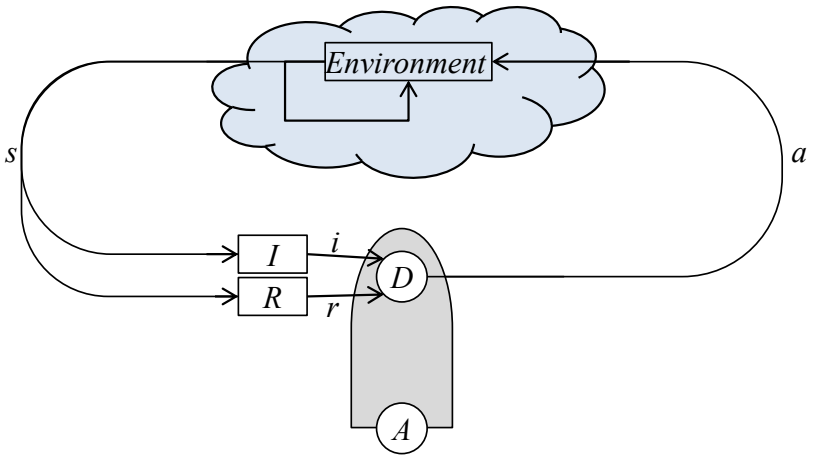

Figure 1.11 — The Reinforcement Learning mechanism where an agent *A* receives an *input (i)* describing the current *state (s)* of the environment. Given its *Decision (D)* module, the agent chooses an *action (a)* that changes the state of the environment. The value of this state transition is then sent back to the agent through a *reinforcement signal (r)*

to dynamics. Still, such mechanisms **require the tuning of the evaporation rates and the marking strategies**. In addition to this, as different agents must be created by each order agent for the exploration and treatment phases, **scalability issues** may be a limit for such techniques as the number of agents is higher than in other approaches.

1.3.3.3 Environment Feedback: Reinforcement Learning

Another studied coordination technique in complex problem solving under constraints is *Reinforcement Learning*. In this technique, a *feedback* is sent by the environment of the *MAS* indicating the performance level achieved so far. Reinforcement learning concerns the problem of a learning agent interacting with its environment to achieve a goal [Sutton, 1999]. The standard reinforcement-learning model is presented in figure 1.11. At each interaction step, the *Agent (A)* receives an *input (i)* describing the current *state (s)* of the environment. Given its *Decision (D)* module, the agent must choose an *action (a)*. This action changes the state of the environment. The value of this state transition is then sent back to the agent through a *reinforcement signal (r)* [Kaelbling et al., 1996]. Instead of being given examples of desired behaviour, the learning agent must discover by trial and error how to behave in order to get a higher reward. Thus, learning requires *exploration and exploitation* phases. Indeed, it is not enough for an agent to select the better already tested actions (*exploitation*) because then no returns can be obtained for other actions that may perform better. Note that, for the exploitation phase, a memory space is required by the agent to memorize the reward obtained given a state (s) and an action (a). Different techniques and algorithms such as *Learning Automata* or *Q-Learning* have been developed in order to help the agent discover the right behaviour. A complete presentation of *Reinforcement Learning* and the developed algorithms can be found in [Verbeeck, 2004].

This technique has also been applied to scheduling in the manufacturing control problem [Aydin and Öztemel, 2000; Martinez et al., 2010; Wang and Usher, 2005]. In

these systems, stations are considered as agents having a list of operations to treat. Stations are provided with a set of dispatching rules to decide which operation to treat first. *Reinforcement Learning* is used by the stations to learn which dispatching rule is the most appropriate to use given the current state of the environment and an objective function to minimize. *Reinforcement Learning* techniques have shown their adequacy to solve scheduling problems under dynamics. The most studied source of dynamics in such systems is the arrival of new operations.

ANALYSIS

Still, the performance of these techniques are highly related to the structure of the problem and the dynamics occurrence. Indeed, in stable environments where dynamics occurrence follows a smooth distribution law such as an *exponential* or *poisson distribution*, agents are able to learn the structure and choose appropriate actions to perform. Such problems are characterized by the *Markov property* that can be learned by agents. These techniques also perform well in environments that switch from a stable state to another for a given period. In such environments, the agent behaviour is chaotic for a certain period corresponding to the state changes. After agents learn to act in this new state, performances are increased. But, in completely changing environments or random occurrence of dynamics, the learning process in fact prevents the agents to act efficiently.

1.3.4 *MAS* and Meta-heuristics

Given their distribution and decentralisation characteristics, *MAS* have been used for the implementation of *hybrid meta-heuristics* relying on parallelism described in section 1.2.3.

A composition of these two techniques has been also investigated. The *Beam-ACO* search algorithm [Blum et al., 2011] is an example of such composition where the probabilistic construction of the solution provided by *ACO* is replaced by the solution construction mechanism used by a probabilistic *Beam Search*[4]. *OptAPO* presented in section 1.3.1 is another example where the *Branch and Bound* algorithm is used by agents to perform the cooperative mediation session.

In domain entities agentification, some agents can use meta-heuristics to locally search for optimal solutions. It is the case of the multi-agent system developed in [Madureira et al., 2009] for scheduling in manufacturing control where resource agents use a *Genetic Algorithm* and *Tabu Search* to locally schedule the received operations and then they negotiate with other agents in order to overcome inter-agent constraints and achieve a global schedule.

[4]The *Beam Search* algorithm is an optimization of *best-first search* algorithm which is a graph search that orders all partial solutions according to a heuristic that predicts how close a partial solution is to a complete solution. In *Beam Search*, only a predetermined number of best partial solutions are kept as candidates called the *beam width*.

1.4 Case Based Reasoning Solving Technique

In some domains such as complex product design, the solving process depends on the experts knowledge and experience. Indeed, experts find it hard to articulate their thought processes while solving problems. This is because knowledge acquisition is extremely difficult in such domains, and is likely to produce incomplete or inaccurate results.

In *Case-Based Reasoning (CBR)* systems, expertise is embodied in a library of past cases rather than being encoded in classical rules. The knowledge and reasoning process used by an expert to solve the problem is not recorded, but is implicit in the case description.

To solve a given instance of a problem, all case-based reasoning methods such as [Bonzano et al., 1996], [Maher and de Silva Garza, 1997] and [Plaza et al., 1997] have in common the following process (figure 1.12):

- **retrieve** the most similar case (or cases) comparing the case to the library of past cases;
- **reuse** the retrieved case to try to solve the current problem;
- **revise** and **adapt** the proposed solution if necessary;
- **retain** the final solution as part of a new case.

Figure 1.13 shows the different tasks to accomplish at each step. Each task can itself be divided into a set of tasks (plain lines). The stippled lines link each task to alternative methods that can be applied for solving it.

Like in the earlier *COP* approaches, centralization is the main limit of the first solving *CBR* techniques. Indeed, such approaches must handle huge volumes of data or manipulate information that can be distributed as is the case of networked information for instance. Thus, *Distributed Case Based Reasoning* techniques have been studied. Indeed, the distribution of resources within case-based reasoning architectures is beneficial in a variety of application contexts. For instance, [McGinty and Smyth, 2001] used a *Collaborative Case-Based Reasoning* for better route planing systems, [Prasad et al., 1996] introduce the *CBR-Team* system where heterogeneous agents, each responsible for a particular component design task, cooperate to resolve design conflicts. Different other systems are discussed in [Plaza and McGinty, 2005].

1.4.1 Analysis

[McGinty and Smyth, 2001] underlines different advantages for the usage of *Distributed Case-Based Reasoning* strategies for problem solving. We underline the fact that these strategies where problems are solved by the combined effort of multiple, independent *CBR* agents, have the potential to improve the performance and maintainability of real-world case-based systems. In addition to this, since each agent is treated as an independent problem solving entity, overall maintenance is made easier as agents may be locally adapted independently of the other agents.

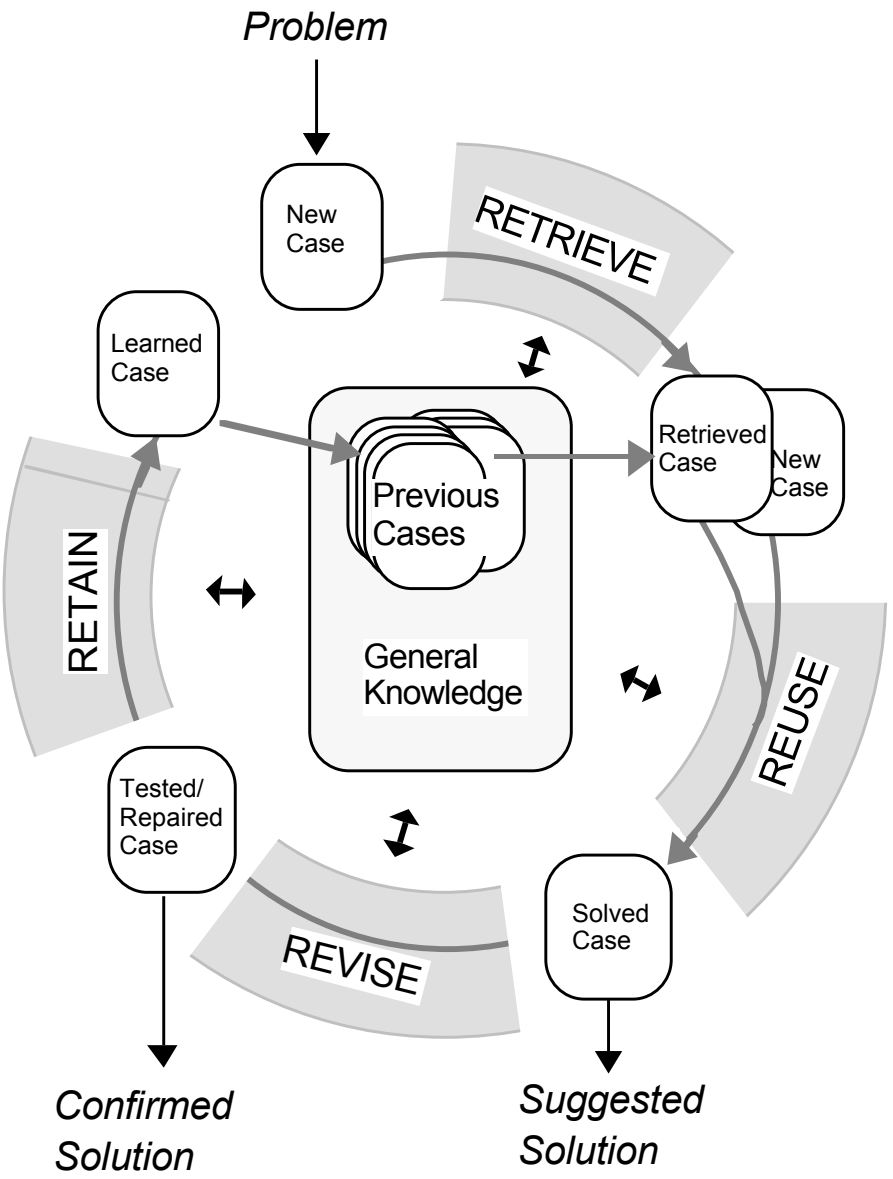

Figure 1.12 — The Case-Based Reasoning Cycle [Aamodt and Plaza, 1994]

Still, these techniques present several limits such as the cases description. Indeed, the description of each case includes the problem and its solution. For complex problems, the solution is usually hard to formulate. For instance, in complex product design, each product can be described as a set of characteristics with not well-known dependency links. The solution for such problems cannot be formulated separately, it is included in the characteristics description. In addition to this, the first step of the *CBR* process is *retrieving* cases that match the problem to solve as accurately as possible. The *retrieve case* step is highly dependent on the case description and the distance functions used. Thus, whenever cases are not well described, the complexity of this step is increased while its performance is decreased.

Another point to consider is that in a distributed environment, the combination of the results obtained by each agent remains difficult to perform. For example, in a distributed

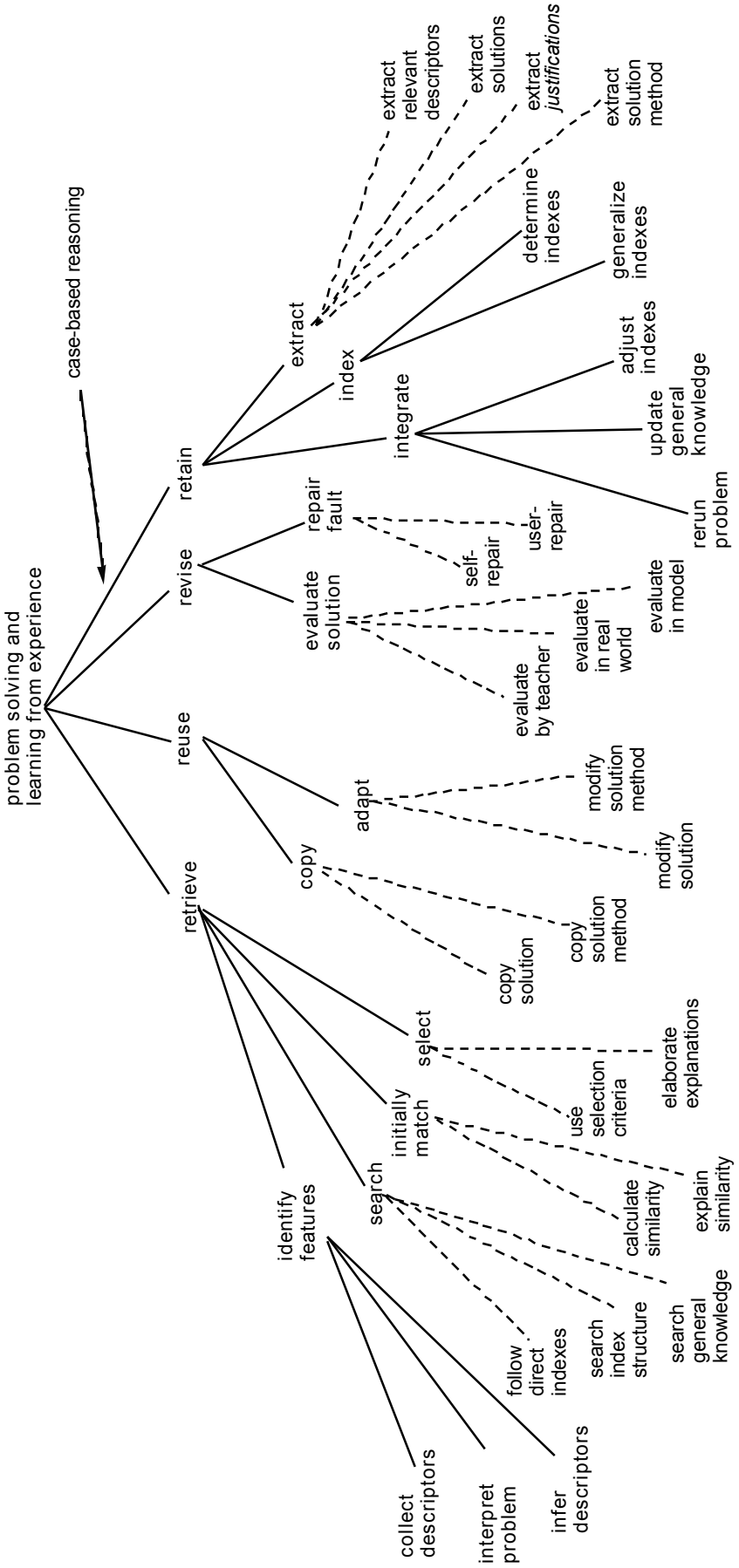

Figure 1.13 — A task-method decomposition of the four steps of the *CBR* process [Aamodt and Plaza, 1994]

case-based system, the problem description can be divided into several sub-problems each handled by an agent. The partial view of each agent may result in the best local cases but their combination may not result in the best overall case [Prasad, 2000]. [Plaza et al., 1997] underlines four major advantages for making agents cooperate in *CBR*. It states that cooperation is required in order to improve the individual performance, the quality of solutions, the efficiency in achieving solutions and especially to achieve tasks that could not be solved in isolation. In such cases, the *CBR* techniques can be seen as the cooperation between existing cases in order to solve new cases.

1.5 Conclusion & Discussion

In this chapter, the main solving techniques used for solving optimisation problems under constraints are presented. Different limitations and advantages have been underlined.

A first limit concerns the necessity to translate problems in **specific frameworks** such as *COP* or *DCOP* so that already defined solving techniques can be applied. Another point concerns the **rigid tree structure** used in several solving techniques (*Branch and Bound*, *ADOPT*, etc.). In addition to this, mostly all the defined solving techniques require the **definition of a specific objective function** used as a **fitness evaluation for the system**, and/or **parameter tuning** such as in *GA*, *ACO* or *PSO*.

The presented evolution and improvements of the solving techniques have underlined the **importance of the distribution of computation and the decentralisation of control** in order to speed up the search and **increase the flexibility** of the developed systems. Multi-agent systems approaches by integrating both concepts have shown their adequacy to efficiently solve problems with growing complexity. Specifically, domain entities agentification approaches have shown that using bottom up approaches to design systems by **being close to the problem definition** is more appropriate than considering a global solution when solving such problems. This is justified by the fact that being close to the problem definition enables to **better understand the functioning** of the different entities that can increase by their **local treatment the adaptivity** of the system and thus **its robustness to dynamics**.

In addition to this, in the different solving techniques developed for *DCOPs*, the concept of **cooperation is fundamental** for the interactions between agents to reach high quality global solutions. Indeed, from the analysis of the different algorithms presented here, the following cooperation mechanisms can be identified:

- The **Value** messages communicated by agents to their descendants in *ADOPT* help agents to cooperatively solve a *DCOP*;
- In the same manner, an agent in *OptAPO* can initiate cooperative mediation sessions to solve local *DCOP* among a set of agents;
- The pheromone used by the ants in *ACO* gives relevant information about promising regions of the search space and guides the behaviour of other ants;

- The velocity and position of local and global best solutions used by the particles in *PSO* are cooperative information exchanged for better exploration phase;
- Mostly all entities agentification approaches use cooperative interactions between agents in order to equilibrate the satisfaction degree of the different agents and enable them to reach their local objective from which the global solution emerges.
- In *CBR* techniques, the problem description can be divided into several sub-problems each handled by an agent. Cooperation is required between the different agents in order to find the best combination between the different found results.

To sum up, we notice that some sort of cooperative behaviours is required to achieve efficient results. This is a reason why the *Systèmes Multi-Agent Coopératifs (SMAC)* team developed the *Adaptive Multi-Agent System (AMAS)* theory (chapter 2) based on cooperative decisions.

Unlike variables agentification and nature inspired agentification techniques where a formal framework is used to develop the solving system, techniques based on entities agentification depends on the problem definition. Thus, the choice of entities to agentify (physical or functional) is related to the way a designer understands the problem, and the developed solutions are problem specific. This limits the reuse of such systems.

In addition to this, solving problems using such techniques require a bottom-up approach where we consider the different entities of the problem and their interactions. Thus specific methodologies different from the traditional top-down methods are required. Methodologies such as *GAIA* [Zambonelli et al., 2003], *ADELFE* [Bernon et al., 2002; Picard, 2004], *INGENIAS* [Pavón and Gómez-Sanz, 2003], *PASSI* [Cossentino and Potts, 2002], *SODA* [Omicini, 2001], *ASPECS* [Cossentino et al., 2010] or *TROPOS* [Castro et al., 2001] have been developed to guide designers in such a way. Their main strengths but also limits are that such methodologies are designed to guide the development of multi-agent systems independently from the type of problem to solve and remain at a high level of abstraction. Thus, an important effort is required to design a specific instantiation of a *MAS* as no agent model ready to use is available.

The contribution of this thesis concerns the improvements of the *AMAS* theory from both the theoretical and engineering points of view. At the theoretical level, this thesis aims at redefining the *Non Cooperative Situation (NCS)* proposed by the *AMAS* theory for optimisation under constraints of complex problems and at defining the agents criticality level underlining the satisfaction degree of each agent.
From an engineering point of view, this thesis aims at defining the *AMAS4Opt* agent model that proposes agent behaviours and interactions based on cooperation mechanisms for complex problem solving under constraints. This model must:

- ▷ present a sufficient level of abstraction enabling its instantiation on different types of complex problem under constraints;
- ▷ be straightforward and intuitive so it can be easily understandable and used by different engineers without the intervention of *MAS* experts;
- ▷ match with the description of optimisation problems under constraints, so that no translation framework is required;
- ▷ match the agents definition to the problem entities so no parameter tuning is needed;
- ▷ focus on local agent behaviours and interactions using cooperation, so the adaptivity of the system is increased;
- ▷ provide agents with local goals that once accomplished enable the global function of the system to emerge. No global objective function used as a fitness evaluation is required.

2 Theory & Tools for the Study

« In theory there is no difference between theory and practice. In practice, ther is. »

Yogi Berra

Contents

The chapter in english starts page 51.

Résumé général du chapitre

Comme souligné dans le chapitre précédent, la coopération est une notion fondamentale régissant les comportements et les interactions des agents au sein des systèmes multi-agents adaptatifs et améliorant les résultats obtenus. Ainsi, le modèle d'agent générique développé au sein de cette étude utilise la coopération telle que définie par la théorie des AMAS (Adaptive Multi-Agent Systems) comme mécanisme de coordination. Ce chapitre se divise en quatre parties: la théorie des AMAS, la méthodologie ADELFE, le langage de modélisation AMAS-ML et le générateur d'API MAY (Make Agent Yourself).

*La théorie des AMAS repose sur le théorème de l'adéquation fonctionnelle: "**Pour tout système fonctionnellement adéquat, il existe au moins un système à milieu intérieur coopératif qui réalise une fonction équivalente dans le même environnement**". Ainsi, cette théorie propose d'utiliser le principe de l'auto-organisation par coopération pour des systèmes ayant des environnements fortement dynamiques ou dont la fonction est impossible à spécifier exhaustivement.*

*Le théorème de l'adéquation fonctionnelle a d'importantes répercussions sur la conception de systèmes multi-agents adaptatifs par auto-organisation. Il suffit, lors de la conception d'un système multi-agent, de se **concentrer sur la conception des agents coopératifs auto-organisateurs et de se focaliser sur leurs interactions**. Un état interne coopératif est ainsi établi permettant à la fonction globale d'émerger. Nous pouvons ainsi noter trois points clés de la théorie : **l'émergence, l'auto-organisation et les agents coopératifs**. Cette première partie du chapitre présente ces trois notions en se concentrant sur les différents modules constituant un agent coopératif notamment son module de coopération. Ce dernier définit l'**attitude coopérative** de l'agent ainsi que les règles comportementales nécessaires à la résolution des **sept situations non coopératives** (incompréhension, ambiguïté, incompétence, improductivité, conflit, concurrence, inutilité) qu'il peut rencontrer.*

*Concevoir des logiciels capables de s'adapter à un environnement dynamique impose une méthode de conception rigoureuse qui se distingue de l'approche globale-descendante habituelle. Pour répondre à ce besoin, plusieurs méthodes furent développées (GAIA, MESSAGE, DESIRE, etc.). Ces méthodes se basent sur des architectures d'agents connus (BDI ou FIPA), et n'intègrent que difficilement les notions d'agents coopératifs, d'auto-organisation et d'ouverture, primordiaux pour la théorie des AMAS. Ainsi une nouvelle **méthode ADELFE** (Atelier pour le DEveloppement de Logiciels à Fonctionnalité Emergente) a été développée. Le but de cette méthode est de guider les développeurs au cours de la conception de systèmes ouverts, complexes et distribués, en se basant sur la théorie des AMAS et sur le concept d'émergence. Elle est basée sur une méthode orientée objet, suit le RUP (Rational Unified Process) et utilise AUML (Agent-UML) afin de rester le plus proche possible des standards et outils utilisés par les ingénieurs. En se décomposant en cinq phases, ADELFE couvre le processus de développement logiciel dans son intégralité et permet aux concepteurs de revenir aux résultats précédents pour les modifier ou les compléter. Chacune de ces phases se décompose en un ensemble d'activités, chacune contenant un ensemble d'étapes. Comme cette méthode ne concerne que des applications suivant la théorie des AMAS, des activités ou étapes spécifiques telles l'identification des agents, leurs interactions et leur conception ont été ajoutées au RUP.*

*Le langage de modélisation **AMAS-ML** permet la conception des agents identifiés par la spécification des différents modules composant un agent. L'outil **Make Agent Yourself (MAY)** quant à lui est utilisé pour générer, à partir d'une architecture d'agent déduite de la modélisation AMAS-ML, une infrastructure d'agent dédiée prête à l'emploi.*

*Nous concluons ce chapitre sur le fait que la théorie des AMAS ainsi que la méthodologie ADELFE ont montré leur **adéquation** pour la conception de systèmes multi-agents capables de résoudre un large éventail de problèmes présentant différents niveaux de complexité. Cependant, ils restent à un **niveau d'abstraction assez élevé** et leur utilisation requiert la présence d'un expert AMAS. Pour cela, ce travail propose la **définition d'un modèle d'agent dédié spécifiant l'utilisation de cette théorie et guidant les ingénieurs pour la résolution de problèmes complexes sous contraintes**.*

2.1 Introduction

As stated in the previous chapter, in complex problem solving under constraints, a cooperative behaviour of the agents is required to achieve efficient results. Thus, the proposed generic agent model is based on cooperation mechanisms as presented in the *Adaptive Multi-Agent System (AMAS)* theory.

In this chapter, the *AMAS* theory (section 2.2) on which the agent model is based is presented. The *ADELFE* Methodology and two software design tools (section 2.3, 2.4, 2.5) developed for the design of adaptive multi-agent system based on this theory are also introduced. Finally, the conclude summarizes the advantages of this theory and tools and underlines why they are insufficient to be used by an engineer without the help of an *AMAS* expert, introducing by that my contribution to this toolbox which aims to facilitate their usage for optimization in complex problem solving under-constraints.

2.2 The *AMAS* Theory: Cooperative Self-Organisation

The *Adaptive Multi-Agent System (AMAS)* theory [Gleizes et al., 2008; Georgé et al., 2003; Glize, 2001; Gleizes et al., 1999] proposes to develop agents focusing on the well-discussed advantages of cooperation. It is based on the theorem of functional adequacy.

2.2.1 The Theorem of Functional Adequacy

Cooperation was extensively studied in computer science by Axelrod [Axelrod, 1984] and Huberman [Huberman, 1991] for instance. "*Everybody will agree that cooperation is in general advantageous for the group of cooperators as a whole, even though it may curb some individual's freedom*" [Heylighen, 1992]. Relevant biological inspired approaches using cooperation are for instance *Ants Algorithms* [Dorigo and Caro, 1999] which give efficient results in many domains. In order to show the theoretical improvement coming from cooperation, the *AMAS* [Gleizes et al., 1999] theory which is based upon the following theorem has been developed. This theorem describes the relation between cooperation in a

system and the resulting functional adequacy[1] of the system. A complete demonstration of this theorem is given in [Glize, 2001]

Theorem: *For any functionally adequate system, there exists at least one cooperative internal medium system that fulfills an equivalent function in the same environment.*

Definition: *A cooperative internal medium system is a system where no Non Cooperative Situation (NCS) exist.*

Definition: *An agent is in a NCS when:*

- $(\neg c_{per})$ *a perceived signal is not understood or is ambiguous;*
- $(\neg c_{dec})$ *perceived information does not produce any new decision;*
- $(\neg c_{act})$ *the consequences of its actions are not useful to others.*

Cooperation is defined as the ability agents have to work together in order to realize a common global goal. It implies that the activities of the agents are supplementary, and dependency links and solidarity exist between them. They have a cooperative attitude that satisfies four properties:

1. **Sincerity**: If an agent knows that a proposition p is true, it cannot say anything different to others;
2. **Willingness**: Agents try to satisfy a request if it is coherent with their own skills and the current state of the world, and if no prejudice results from the action, either to the acting agent or to another. If there is a resulting prejudice, refer to property three;
3. **Fairness**: They always try to satisfy, when it is possible, agents with the higher level of difficulty or criticality;
4. **Reciprocity**: Each agent of the same society knows that it and the others verify these three main properties.

The objective is to design systems where agents:

- do the best they can when they encounter difficulties. These difficulties can be viewed as exceptions in traditional programming. From an agent point of view, we call them *Non Cooperative Situations (NCSs)*. An agent locally tries to detect such failures and try to repair them. These *NCSs* can be related to the internal state of the agent or to its interactions with its environment (the other agents and the system environment);

[1] *Functional* refers to the *function* the system is producing, in a broad meaning, i.e. what the system is doing, what an observer would qualify as the behaviour of a system. And *adequate* simply means that the system is doing the *right* thing, judged by an observer or the environment. So *functional adequacy* can be seen as *having the appropriate behaviour for the task*.

- ▷ anticipate *NCSs*. The agent always chooses the actions with minimal disturbance to the other agents it knows. It tries to anticipate [Doniec et al., 2005] (for him and others) problems that can introduce *NCSs*;
- ▷ are cooperative towards the system and other agents. The first point implies that agents not reaching their own goal can be seen as generating *NCSs* that must be repaired. Being cooperative toward other agents implies that when detecting or anticipating *NCSs* agents try to always help agents with the higher level of difficulty ;

In others words, the *AMAS* theory considers that the agents, by trying to always have a cooperative attitude, act by re-organizing their acquaintances and interactions adequately with the others agents.

2.2.2 Consequence of the Functional Adequacy Theorem

This theorem means that we only have to use (and hence understand) a subset of particular systems (those with cooperative internal mediums) in order to obtain a functionally adequate system in a given environment. We concentrate on a particular class of such systems, those with the following properties [Gleizes et al., 1999]:

- ▷ the system is cooperative and functionally adequate with respect to its environment. Its parts do not 'know' the global function the system has to achieve *via* adaptation;
- ▷ the system does not have an explicitly defined goal, rather it acts using its perceptions of the environment as a feedback in order to adapt the global function to be adequate. The mechanism of adaptation results from each agent trying to maintain cooperation using their skills, representations of themselves, other agents and the environment;
- ▷ each part only evaluates whether the changes taking place are cooperative from its point of view – it does not know if these changes are dependent on its own past actions;
- ▷ Basically, the idea is that it is easier and more efficient to design systems by focusing on the agent and the cooperative self-organising mechanisms which will result in the adequate system, than trying to produce the right global system directly.

This way of engineering systems has been successfully applied on numerous applications with very different characteristics for the last ten years (autonomous mechanisms synthesis [Capera et al., 2004], flood forecast [Georgé et al., 2003], electronic commerce and profiling [Link-Pezet et al., 2000],etc.). For each, the local cooperation criterion proved to be relevant to tackle the problems without having to resort to an explicit knowledge of the goal and how to reach it.

2.2.3 Achieving Self-Adaptation and Self-Organisation

We consider that each part P_i of a system S achieves a partial function f_{P_i} of the global function f_S (figure 2.1). f_S is the result of the combination of the partial functions f_{P_i},

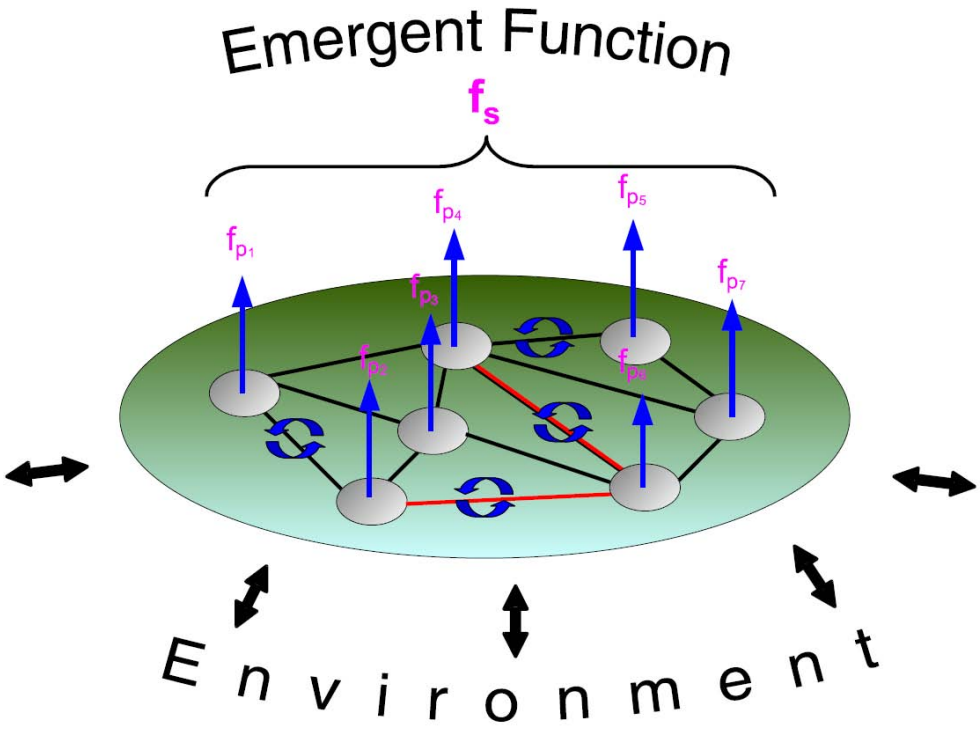

Figure 2.1 — Adaptation: changing the function of the system by changing the organisation

noted by the operator "$\circ$". The combination being determined by the current organisation of the parts, we can deduce $f_S = f_{P_1} \circ f_{P_2} \circ ... \circ f_{P_n}$. As generally $f_{P_1} \circ f_{P_2} \neq f_{P_2} \circ f_{P_1}$, by transforming the organisation, the combination of the partial functions is changed and therefore the global function f_S changes. This is a powerful way to adapt the system to its environment. A pertinent technique to build these kinds of systems is to use adaptive MAS. As in Wooldridge's definition of *multi-agent systems* [Wooldridge, 2009], we will be referring to systems constituted by several autonomous agents, plunged into a common environment and trying to solve a common task.

2.2.4 Architecture of an *AMAS* Agent

A cooperative agent in the *AMAS* theory has the four following characteristics. First, an agent is **autonomous**: an agent has the ability to decide to refuse a request or start some activity on its own. Secondly, an agent is **unaware of the global function** of the system; this global function emerges (from the agent level towards the multi-agent level) [Georgé et al., 2003]. Thirdly, an agent can **detect non cooperative situations** and acts to return to a cooperative state. And finally, a cooperative agent is not altruistic in the meaning that an altruistic agent always seeks to help the other agents. Indeed, it is **benevolent** i.e. it seeks to achieve its own goal while being cooperative[2].

Cooperative agents are equipped with several modules representing a partition of their

[2]Note that sometimes, this still means that an agent accepts to delay its own goal if it judges that the other agent is in a more critical state for instance.

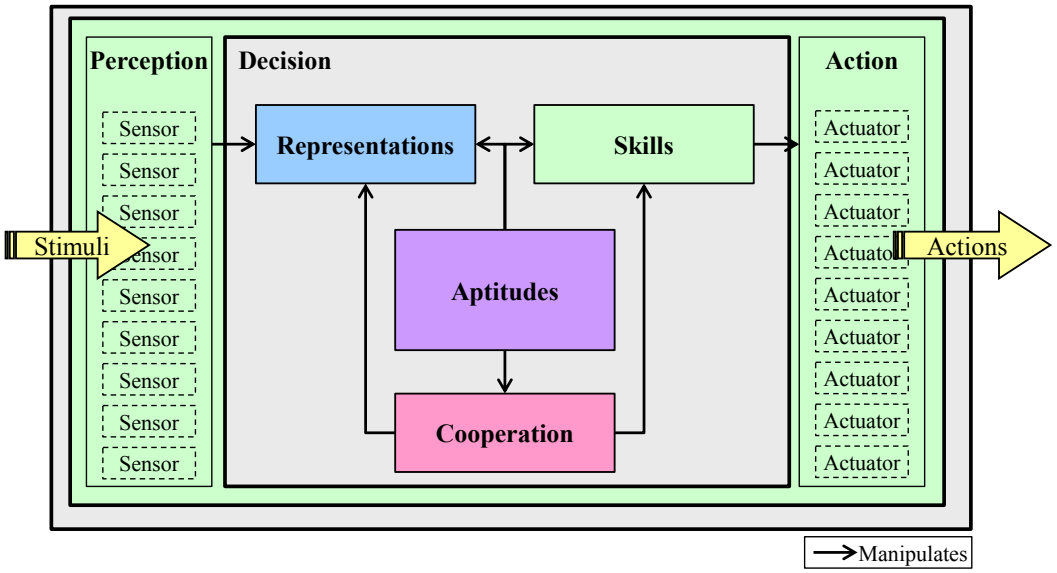

Figure 2.2 — The different modules of a cooperative agent [Bernon et al., 2004]

physical, *cognitive* and *social* capacities (figure 2.2). Each module represents a specific resource for the agent during its *perceive-decide-act* life cycle. Four of the modules are quite classical in an agent model [Wooldridge, 2009]. The novelty comes from the Cooperation Module which contains local rules to solve or anticipate *NCS*.

2.2.4.1 Interaction Module

Agent interactions are managed by two modules. The Perception Module represents the inputs the agent receives from its environment. Inputs may have different natures and types: integer or boolean for instance for simple agents, or symbolic messages in a mail box for more cognitive agents. The Action Module represents the output and the way the agent can act on its physical environment, its social environment or itself (considering learning actions for example). Similarly to the perceptions, actions may have different granularities: simple effectors activation for a robot or semantically complex messages sending for social agents.

2.2.4.2 Skill Module

Even if cooperative agents have a strong focus on trying to avoid *NCS*, they also have several tasks to complete. The ways to achieve their goals are expressed in the Skill Module. Skills are knowledge and know-hows about given knowledge fields and enable agents to realise their partial function – as a part of a *MAS* that produces a global function. Simply said, skills are needed for an agent to be able to act. No technical constraints are required to design and develop this module. For example, skills can be represented as a classical or fuzzy knowledge base of facts and rules on particular domains. It also can be decomposed into a lower level *MAS* to enable learning, as in the *ABROSE* on-line brokerage application [Gleizes and Glize, 2000], where skills were decomposed into a semantic network.

2.2.4.3 Representation Module

Agents in a *MAS* evolve in a given environment (physical or social). The representation module concerns the beliefs an agent has on this environment. It includes the agent representation of other agents (social environment) and other entities (physical environment). In addition to this, the representation module encapsulates the intrinsic characteristics of the agent itself such as its constraints or its criticality degree and the representation it has on itself. Thus, this module can be divided into two parts: the intrinsic *characteristics* of the agent and the *representations* it has on itself and on its environment (physical or social). This module is manipulated by the Perception Module during the perception phase of the agent life cycle.

2.2.4.4 Aptitude Module

The Aptitude Module contains generic tools an agent needs to accomplish its treatment. It represents the way an agent must accomplish the decision phase of its life cycle. It manipulates and activates the different modules of the agent. For instance it can be the usage of a *Monte Carlo* selection method to chose which skill to activate or an inference engine that indicates in which situation the agent is.

2.2.4.5 Cooperation Module

The cooperative attitudes of agents are implemented in the Cooperation Module. This module manipulates the Skills and Representations, in order to anticipate or detect and repair *NCSs*. Therefore , cooperative agents must possess rules to detect *NCSs*. Seven types of *NCSs* have been identified. For each *NCS* detection rule, the Cooperation Module associates one or several actions to process to avoid or to solve the current *NCS*. In the following, we present for each *NCS* how it can be detected, repaired and anticipated.

INCOMPREHENSION. This personal *NCS* is related to the interpretation of the messages in the perception life cycle step. It informs that the agent is not able to extract any understandable information from the received message. It is detected locally by the agent when messages are interpreted.

To solve this *NCS*, the agent can for instance ask the sender to modify its message, or ask other agents that may understand it for a translation/decryption.

Agents can anticipate this *NCS* for themselves by ensuring accurate presentation of themselves when meeting useful new agents and improving the comprehension of their environment. When having adequate information on other agents and when it is possible, they can anticipate and prevent it for other agents by sending them understandable information.

AMBIGUITY. Related to the update of the local representation in the perception life cycle step, this *NCS* informs the agent that different interpretations are possible, and by that, an accurate representation update is not possible. That can be due for instance to missing information.

To solve this *NCS*, the agent can ask the sender for additional information, wait until this information is available or ask for other agents' help.

Agents are evolving in a context and anticipating this *NCS* is highly dependent on their representation of this context. They can anticipate and prevent it for themselves as previously with accurate representation or by collecting information they judge useful. When having adequate information on other agents and when it is possible, they can anticipate and prevent it for other agents by sending the necessary information.

INCOMPETENCE. Related to the definition of the list of possible actions in the decision life cycle step, this *NCS* is detected when the agent does not have the competence to treat received information such as answering an agent request.

To solve this *NCS*, the agent can inform the sender of its incompetence or redirect the request to competent agents.

As previously, agents can anticipate and prevent this *NCS* for themselves by an accurate presentation of themselves. When having adequate information on other agents and when it is possible, they can anticipate and prevent it for other agents by sending requests to the adequate agents.

UNPRODUCTIVITY. Related to the definition of the list of possible actions in the decision life cycle step, this *NCS* is detected when the agent has accurately interpreted the received information but cannot use it to produce any useful information for himself because it already has this information, it is of no interest for him or the received information is incomplete (partial unproductivity).

The two first reasons can be repaired by informing the sender about the uselessness of the information sent and updating the agent profile for the sender. The last is repaired by asking for additional information.

Anticipating this *NCS* consists in informing other agents about the update in the agent profile such as the agent's main interest or the information produced by the agent and judged helpful for others.

CONFLICT. This *NCS* is detected either when considering the list of possible future actions or when detecting a conflict in the environment. In the first case, among the list of possible actions some are conflicting and such actions cannot be performed by the agent at the same time (i.e. lack of resources). In the second case, the conflict can either be due to a previous action performed by the agent or another agent, or a change in the environment not related to agent activity. This situation is also detected when the agent considers that modifying the environment (realizing some actions) can prevent other agents from reaching their goals (i.e. when using a resource required by another agent).

Solving the first *NCS* case consists in choosing to perform the most critical (higher priority) actions as well as trying to minimize the disturbances or prejudice to others. In the second case, the agent needs to find a set of actions that can resorb the detected conflicting situations.

Anticipating this *NCS* depends on the knowledge an agent has concerning the intentions of other agents and requires specific knowledge structures that an agent can access before taking an action.

CONCURRENCE. Related to the choice of the list of actual actions to perform in the decision life cycle, this *NCS* concerns the interactions between the agent and its environment. It is detected when among the list of possible actions, some can put the agent in concurrence or competition with other agents. The agent is able to perform the actions but considers that there are other agents able to perform the same actions and reach the same state in the environment.

To solve this *NCS*, the agent interacts with the other agents to decide which of them is more appropriate to realize the action.

As previously, anticipating this *NCS* depends on the knowledge an agent has concerning the intentions and competences of other agents.

USELESSNESS. Related to the choice of the list of actual actions to perform in the decision life cycle, this *NCS* is detected when the agent considers itself not useful (or not useful enough) for the system or its environment. This can be due to a lack of information or unused knowledge.

To solve this situation, the agent must choose actions to acquire new information, to profit from its knowledge or change its place in the organization by interacting with other agents.

Anticipating uselessness or partial uselessness situations depends on the interactions an agent has with its environment: is it adequately solicited by the environment? Does it have sufficient interactions with the environment? Or can it balance the load of other agents?

2.2.5 Internal Functioning of an *AMAS* Agent

The *NCS* are cooperation failures that an agent must anticipate or detect and repair. They can be assimilated to "exceptions" in traditional programming as they are unexpected events that can occur during runtime with the difference that the agent behaviours must be prepared to repair *NCSs*. Our definition of cooperation (section 2.2.1) leads to the definition for an agent of three local meta-rules the designer has to instantiate according to the problem to solve:

- Meta-rule 1 (c_{per}): Every signal perceived by an agent must be understood without ambiguity.
- Meta-rule 2 (c_{dec}): Information coming from its perceptions has to be useful to its reasoning.
- Meta-rule 3 (c_{act}): This reasoning must lead the agent to produce actions which have to be useful for other agents and the environment.

During the perception phase of the agents' life cycle, the Perception sub-Modules updates the values of the sensors. These data directly imply changes in the Skill and Representation Modules. Once the knowledge is updated, the decision phase must result in the choice of actions. During this phase, the agent uses its Aptitude, Skill and Cooperation Modules to compute its knowledge and decide which actions to perform. Once an action

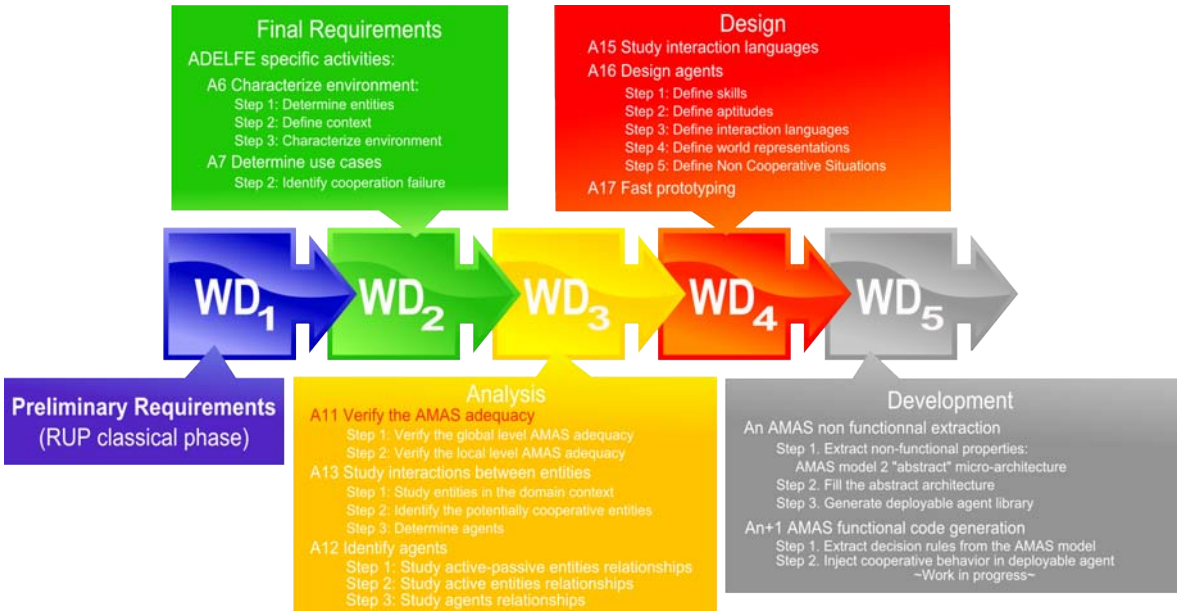

Figure 2.3 — *ADELFE* Methodology [Rougemaille et al., 2009]

is chosen, during the action phase, the agent acts by activating its effectors or changing its knowledge.

2.3 The *ADELFE* Methodology

Designing adaptive multi-agent systems require a specific methodology different from the top down traditional methods. Indeed, in such systems, we concentrate on the different parts of the system and their interactions. The global function of the system emerges from these interactions. Different methodologies [Henderson-Sellers and Giorgini, 2005; Bergenti et al., 2004] such as GAIA, DESIRE or INGENIAS have been developed. As these methods, based on well known agents architectures such as BDI or FIPA, sorely integrate cooperative agents the main principle for the *AMAS* theory, a new method was required. Thus, the *Atelier de Développement de Logiciels à Fonctionnalité Emergente (ADELFE)* methodology [Bernon et al., 2002; Picard, 2004; Bernon et al., 2005; Rougemaille et al., 2008] was developed. *ADELFE* is a toolkit to guide designers through the development phase of complex, open and distributed systems based on the *AMAS* theory and the concept of emergence. It is based on some well-known tools and notations coming from the object-oriented software engineering: *UML* (Unified Modelling Language) and *RUP* (Rational Unified Process). It uses *AUML* (Agent-UML) to express interaction protocols between agents.

The *ADELFE* methodology is divided into 5 phases, each including a set of activities (*A*) divided into different steps (*S*) (figure 2.3). Because *ADELFE* is devoted to the design of *AMAS*, specific activities have been added to the RUP:

During the final requirements study (WD2):

- A_6: Characterization of the environment,
- $A_7 - S_2$: Identification of the *cooperation failures*.

During the analysis phase (WD3):

- A_{11}: Verification of the *AMAS* adequacy,
- A_{12}: Identification of the agents that are involved in the system being built,
- $A_{13} - S_3$: Study of the relationships between agents.

During the design phase (WD4):

- A_{15}: Study if the interaction languages enabling agents to exchange information,
- A_{16}: Complete the design of these agents. An agent that intervenes in an *AMAS* is composed of different parts that produce its behaviour: skills, aptitudes, the interaction language, world representations and Non Cooperative Situations.
- A_{17}: Fast prototyping. This is often necessary to verify that the behaviour of these agents is the desired one, and observe the result of their interactions.

During the development phase (WD5), the main guiding principle is the *separation of concerns*. Indeed, this step is decomposed into two steps:

- A_n: Study the non-functional part of the agent, also called the *operating concerns*. This step describes the basic mechanisms of the agent and generates a deployable library of the designed agents,
- A_{n+1}: Study the functional part of the agent, also called the *behaviour concerns*. This step integrates the behaviour (cooperation and decision rules) of the agent to the generated deployable agent library.

Different tools have been developed to ease the usage of the *ADELFE* methodology. Among them, we detailed the *AMAS Modelling Language (AMAS-ML)* used for the design of agents (A_{16}) and *Make Agents Yourself (MAY)* used for the definition of the agent architecture and the generation of the deployable agent library (*WD5*).

2.4 *AMAS* Modelling Language

The A_{16} activity of the *ADELFE* Methodology proposes to define the *Skill*, *Aptitude*, *Interaction*, *Representation* and *Cooperation* modules. [Rougemaille et al., 2008] introduces the *AMAS Modelling Language (AMAS-ML)* based on the *AMAS Meta-Model* (figure 2.4) to facilitate the definition of those different modules. *AMAS-ML* distinguishes between the system point of view and the agent point of view.

The former considers the environment of the system. Indeed, an *AdaptiveMultiAgentSystem* evolves in a given *ExternalEnvironment* presenting useful

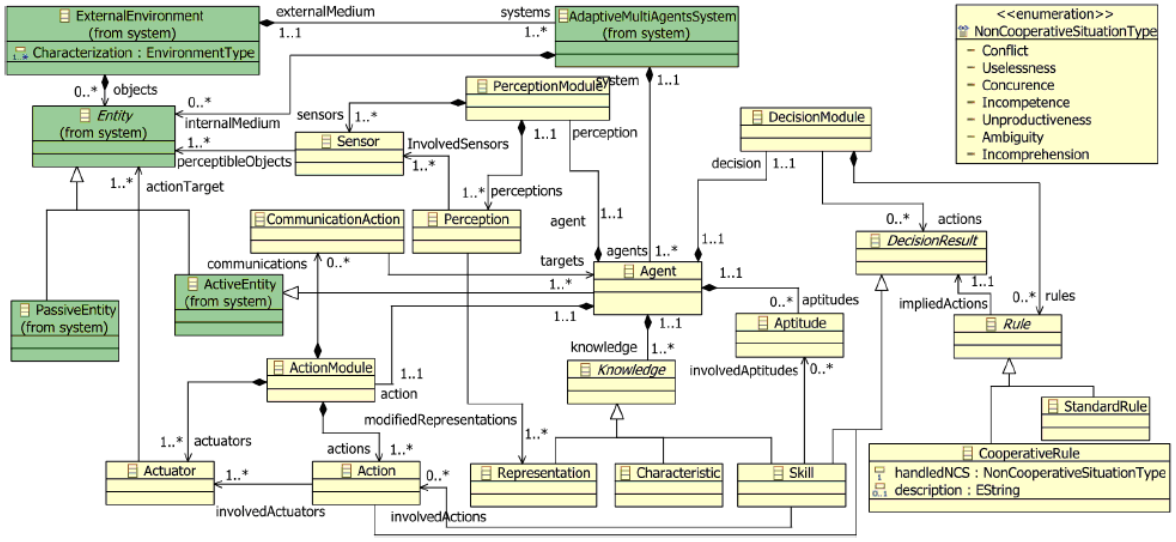

Figure 2.4 — *AMAS Meta-Model* [Rougemaille et al., 2008]

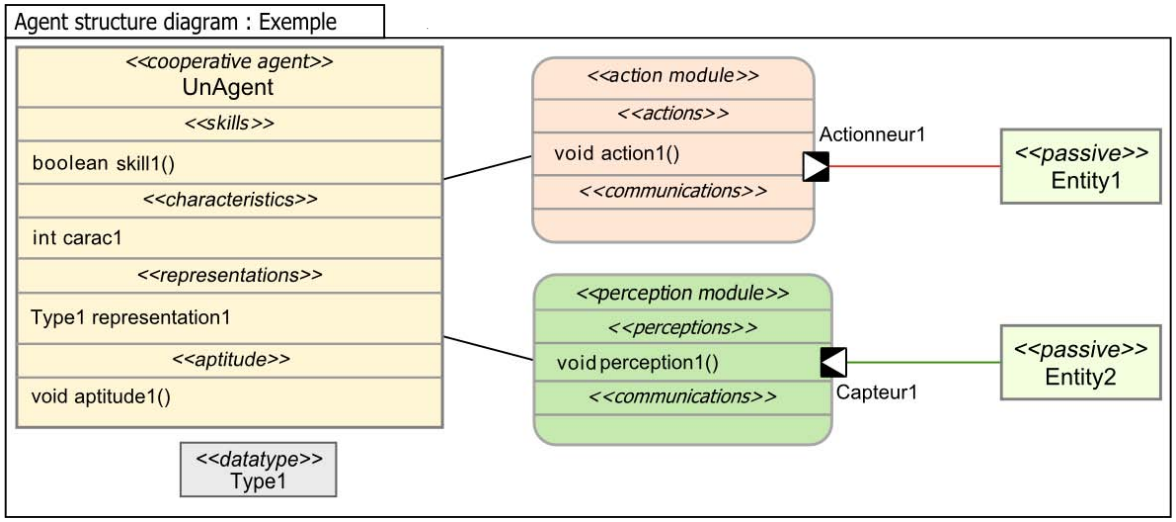

Figure 2.5 — *AMAS-ML* agent description [Rougemaille et al., 2008]

properties for the design of the system itself. In addition to this, the agents interact with the entities (*Entity*) of this environment using perceptions and actions.

The latter concerns the different modules that compose an agent. These modules handle the agent activities and its life cycle. Usually, the life cycle of a cooperative agent is divided into three steps: perception, decision and action. Given these steps and their functional requirements, five modules are defined (figure 2.5).

- *PerceptionModule & ActionModule* represent the possible interactions of the agent with its environment (*Perception, Action*) and the means the agent has to realise them (*Actuator, Sensor*). Among them we distinguish *CommunicationAction* which enables agents to exchange information and requests.
- *DecisionModule* decomposed into a set of *Rules*: *StandardRule* and *CooperativeRule* (figure

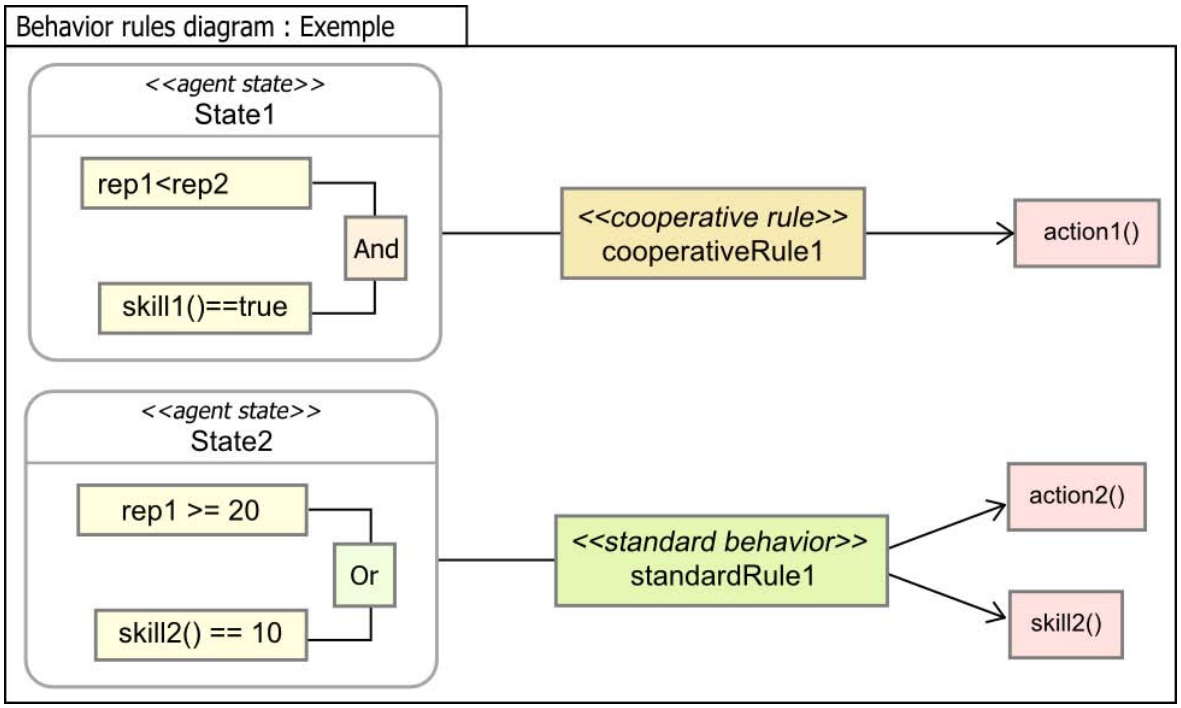

Figure 2.6 — *AMAS-ML* diagrams for the behavioural rules of an agent [Rougemaille et al., 2008]

2.6), enables the agents to decide which actions (*DecisionResult*) to perform given their knowledge. These rules[3] represent the capacity of an agent to detect and repair a *NonCooperativeSituation* or to perform its nominal behaviour.

- *Knowledge* concerns all the knowledge an agent has: the representation (*Representation & Characteristics*) of its environment and itself and its *Skills*.
- *Aptitude* concerns the tools such as a random selection or statics computation, an agent has to perform its skills and for reasoning.

2.5 *MAY*: Make Agents Yourself

Another tool developed by the *Systèmes Multi-Agent Coopératifs (SMAC)* team to facilitate the development of systems based on the *AMAS* theory is *Make Agents Yourself (MAY)*[4] [Noel and Arcangeli, 2011]. *MAY* is a generator of dedicated agent frameworks available as an Eclipse plugin. This tool is made to build application-specific species of agents and to generate dedicated frameworks providing them. Such frameworks are then usable to program agents compliant with the corresponding species of agents.

MAY supports the description of each type of agent as a component-based architecture and generates the JAVA code required to implement the different components. *MAY*

[3] In these diagrams, formal and semi-formal notations are used such as ! for not and @ for an agent address.
[4] www.irit.fr/MAY

generates the whole *Application Programming Interface (API)*, that is, tools to execute, create and deploy the specific agent species. The component-based architecture is divided in two levels each containing a set of μ-components:

- ▷ *container* (the how), hidden to the user of the framework, defines how the agent works. Thus, this level contains functioning components such as the life cycle of the agent or its message manager. This level is common to all agents of the same species.
- ▷ *application* (the what) concerns what the agent does and the way it uses its tools to achieve its goals, includes the perception, decision and action components. This level can be different among a set of agents from the same species.

The two levels are interrelated and components from one level can require or offer services to components from the same or the other level. As for example, in figure 2.7, the component *LifeCycle* requires the methods *perceive(), decide()* and *act()* from the components *Perceive, Decide* and *Act*. The method *receive()* of the component *Receive* enables the agent to communicate with its environment. Thus, this method is called *external*. Whenever a message is received, this method requires the method *put()* from the component *MailBox* in order to drop the message off in the mail box of the agent.

These two levels link different μ-components [5] together to form a μ-architecture [6] realising the species [Noel, 2011]. As agents from the same species can differ by their application level, the μ-architecture of a given species contains the common μ-components. Before the code generation, it must be extended to include the μ-components of the application level specific to the different agents in the species.

The usage of a component-based architecture increases the flexibility level during the design (changing the implementation of a component without touching the others) and the reuse of already designed μ-components (such as the communication components).

2.6 Conclusion and Analysis

This chapter has presented :

- ▷ the *AMAS* theory on which this thesis is based;
- ▷ the *ADELFE* methodology used to develop adaptive multi-agent systems based on this theory;
- ▷ two software design tools used to facilitate the development of such systems.

Two points are to be noted. First, the *AMAS* theory proposes a specific theoretical framework that defines the behaviour and the architecture of agents in a multi-agent system. Nevertheless, this theory remains a high level and general guide and when confronted to

[5] A component is called μ-components as it is a part of a μ-architecture (pronounced "micro")

[6] A μ-architecture represents the architecture of an agent. It is called μ-architecture in order to differentiate between the system architecture and the architecture of the agents that constitute the system.

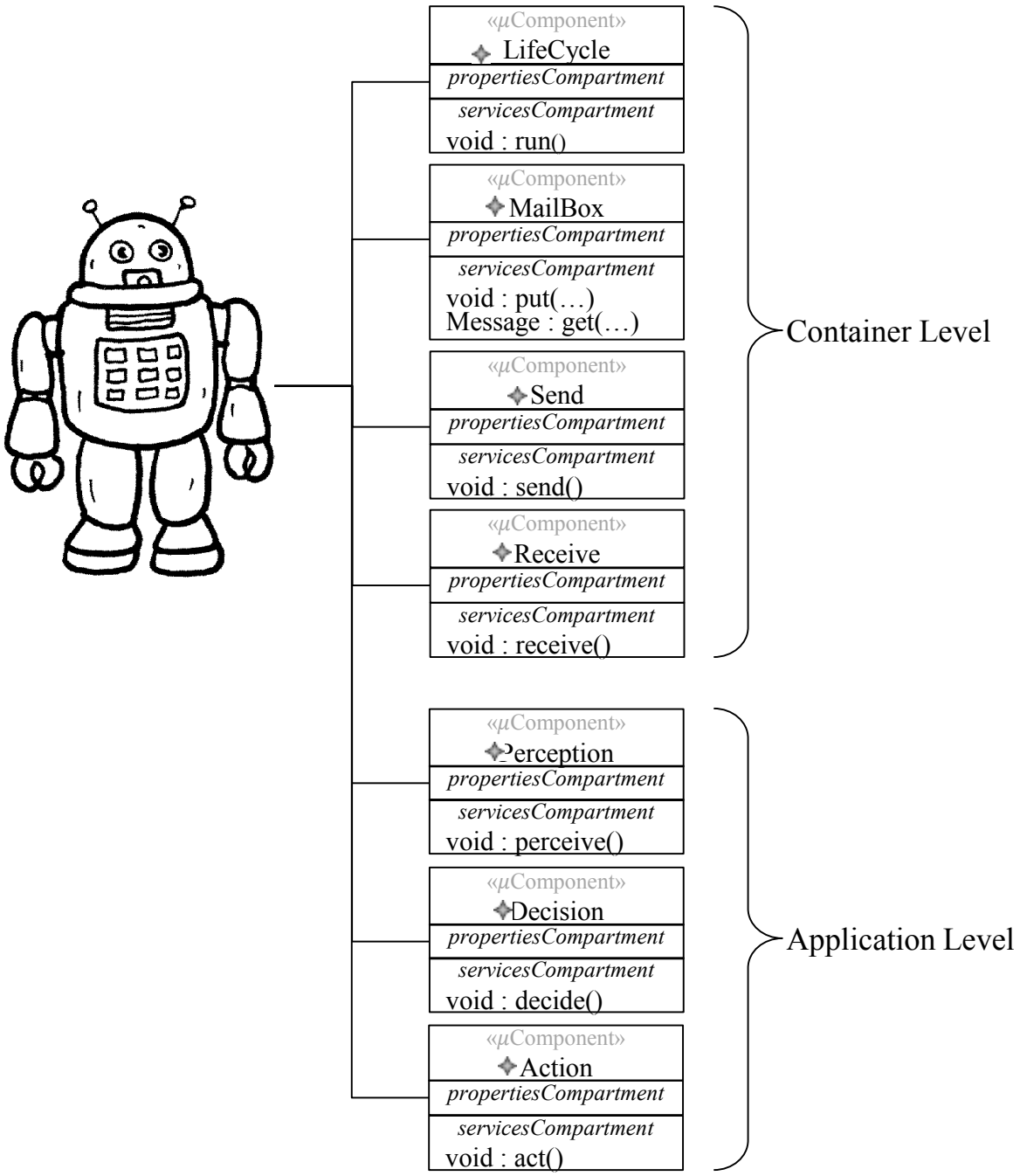

Figure 2.7 — A minimal μ-architecture of an agent

a problem, it can be difficult to correctly instantiate it to build the adaptive multi-agent system that can solve the considered problem. Second, the different steps of the *ADELFE* methodology guide the designer to design its system but remain at a certain abstraction level due to the generic nature of *ADELFE*. This is a strength as it can be used on a broad spectrum of applications, but also a drawback as it requires, when using it, the help of both and agent and application domain experts.

Still, the *AMAS* theory has shown its adequacy to solve complex problems and *ADELFE* has been applied successfully on numerous applications. Indeed, it has succeeded to solve a large variety of complex problems under constraints such as timetabling and scheduling [Picard et al., 2005; Clair et al., 2008] or conceptual aircraft design [Welcomme et al., 2009]. This motivates us to specialise its usage for such types of problems by defining semi-generic agent behaviours, interactions and architectures, ready to use by an engineer for whatever complex problem under constraint he has to tackle.

3 A Generic Agent Model for Complex Problem Solving

« We shall neither fail nor falter; we shall not weaken or tire...give us the tools and we will finish the job. »

Winston Churchill

Contents

The chapter in english starts page 69.

Résumé général du chapitre

Une introduction générale à différentes méthodes de résolution existantes a été présentée dans le premier chapitre de ce travail. Cette présentation a souligné les avantages et inconvénients de ces méthodes et leur évolution pour mieux répondre aux applications actuelles, notamment l'intégration de la distribution et de la décentralisation. Ces deux mécanismes nécessaires pour répondre à la complexité croissante exigent la mise en place de mécanismes de coordination afin de permettre au système de converger vers la solution optimale. Après une analyse des ces mécanismes, la coopération apparaît comme un point central et commun à ces différents mécanismes. La théorie des AMAS, introduite dans le deuxième chapitre, repose sur la définition de comportements coopératifs d'agent pour la construction de système auto-adaptatif. Le deuxième chapitre a souligné l'adéquation de cette théorie pour répondre efficacement aux applications actuelles ainsi que son niveau d'abstraction assez élevé limitant son utilisation à des experts du domaine.

Dans ce chapitre, la contribution principale de ce travail est présentée. Elle consiste en la définition du modèle d'agent AMAS4Opt spécialisant l'utilisation de la théorie des AMAS pour la résolution de problèmes d'optimisation sous-contraintes se caractérisant par différents degrés de complexité. Le but est d'améliorer l'utilisation de la théorie des AMAS pour les problèmes d'optimisation par la définition de comportements coopératifs d'agents et la mise en place d'une architecture agent dédiée pour ce type de problème.

Les problèmes d'optimisation sous-contraintes sont généralement décrits comme un ensemble d'entités sujettes à des contraintes. Dans les systèmes multi-agents, ces entités sont représentées par des agents interagissants. En analysant ces interactions tout au long du processus de résolution, nous pouvons distinguer les agents ayant des contraintes à résoudre et les agents possédant les compétences nécessaires pour les aider. De ce fait, deux types de rôles peuvent être identifiés: ***le rôle contraint et le rôle service****. Un agent ayant des contraintes à résoudre possède le rôle* ***contraint****. Un agent possédant les compétences nécessaires pour aider les autres agents a le rôle* ***service****. Un agent peut avoir les deux rôles et passer de l'un rôle à l'autre en fonction de son état. La première partie de ce chapitre se consacre à la définition de ces deux rôles et la mise en place pour chacun d'eux d'un algorithme générique présentant les différentes étapes constituant le comportement de l'agent.*

*Les agents de notre modèle interagissent directement par l'****utilisation de messages****. La deuxième partie de ce chapitre définit les différentes catégories auxquelles appartiennent les messages utilisés par les agents:*

- ***Les requêtes de service*** *utilisées par les agents ayant le rôle contraint pour demander les services des agents ayant le rôle service.*
- ***Les requêtes d'information*** *utilisées par les agents des deux rôles pour demander des informations requises à leur fonctionnement.*
- ***Les réponses aux requêtes*** *utilisées par les agents pour répondre aux requêtes d'information ou de service reçues.*

▷ ***Les messages d'information*** *utilisés par les agents pour échanger des informations jugées utiles.*

Les agents de notre modèle sont coopératifs et agissent d'une manière à équilibrer leur degré de satisfaction. Pour cela, ***le degré de criticité*** *d'un agent est défini comme étant la mesure de son degré d'insatisfaction. Ce degré est un point clé du comportement coopératifs des agents. En effet, il dénote la difficulté d'un agent à atteindre son but. En tant qu'agents coopératifs, les agents ayant le rôle* ***service*** *du système privilégient d'aider les agents ayant le rôle* ***contraint*** *ayant des degrés de criticité élevés afin d'équilibrer le degré de satisfaction des différents agents. Ainsi, lors de la demande d'un service, un agent ayant le rôle* ***contraint*** *doit en fonction de ses connaissances calculer de degré d'une manière bien précise.*

Un agent coopératif doit maintenir une attitude coopérative vis-à-vis du système et des autres agents. Pour cela, il doit anticiper ou détecter et réparer des situations d'échec à la coopération appelées ***situations non coopératives****. Des sept catégories définies dans la théorie des Adaptive Multi-Agent System (AMAS), cinq concernent les agents impliqués dans la résolution des problèmes d'optimisation sous contraintes:* ***l'incompétence, l'improductivité, l'inutilité, le conflit*** *et* ***la concurrence****. La section 5 du chapitre détaille chaque catégorie en spécifiant comment un agent peut l'anticiper ou la détecter et la réparer.*

Avant de conclure, les sections 6 et 7 définissent les différents modules composant l'architecture du modèle d'agent en utilisant les outils ***AMAS Modelling Language (AMAS-ML)*** *et* ***Make Agents Yourself (MAY)****.*

3.1 Introduction

Algorithms based on domain entities agentification seem to be the most suitable techniques to answer the growing complexity of today's applications. Those algorithms have shown their adequacy to solve a large variety of complex problems. Being close to a natural description of the problem and designing the agents as close as possible to the entities of the problem, allow the definition of more intuitive and rich solving algorithms which are then more robust to dynamics, flexible, open and provide a relevant level of adaptation in real-time.

The state of the art (chapter 1) discussed the evolution of the different solving techniques and pointed out the quality of such algorithms. Two major points were underlined.

First, different well-known methodologies have been developed to help the design of such algorithms. Their main advantage but also their limit is that they remain a high level guide and can be used to solve different problems but their application requires expert knowledge. They have been used to solve different problems resulting in different specific hardly reusable algorithms. Contrary of other agentification techniques where generic algorithms ready to be instantiated for optimisation under constraints such as *Ant Colony Optimization (ACO)* or *Asynchronous Distributed Constraint Optimization (ADOPT)* exist, **such methodologies do not propose agent models ready to be instantiated to a given problem**.

Second, the different agentification techniques used to solve optimisation under constraints problems mainly differ by the coordination mechanisms used by the agents

of the system to reach a solution. The state of the art chapter underlined the fact that **cooperation** is the main coordination mechanism and that a sort of cooperative behaviour is required to achieve efficient results.

Chapter 2 presented the *AMAS* theory where the agents design focuses on the well-discussed advantages of cooperation. Indeed, this theory proposes to build functional and adequate systems by concentrating on the cooperative behaviours of the agents. The *ADELFE* methodology defined to help the design of systems based on the *AMAS* theory has also been presented. Unfortunately, in the same manner as other methodologies, *ADELFE* remains at a high level of abstraction meaning that a lot of agent expertise is needed to design a specific application following the methodology.

This chapter, the main contribution of my thesis, aims at filling the gap between the methodologies and the specific solving algorithms for optimisation under constraints problems by defining a generic agent model: *AMAS for Optimisation (AMAS4Opt)*. This model proposes agent behaviours and interactions based on cooperative mechanisms as defined in the *AMAS* theory and that are dedicated for the solving of optimisation under constraints problems.

The chapter is organised as follows. Section 3.2 introduces the two roles that agents of this model can have. Sections 3.3, 3.4 and 3.5 present how agents interact and cooperate. In section 3.6, the different modules of each role are described using the *AMAS-ML*. Before the conclusion (section 3.8), the agent architecture described using *MAY* is presented.

3.2 Agent Roles

Complex problems under constraints are formalized as a set of entities submitted to a set of constraints. In multi-agent systems, the entities are mainly represented by agents. Depending on the real-time interactions the multi-agent system has with its environment, the organization between its agents emerges and constitutes the solution to the addressed problem.

When analysing the interactions between agents during the solving of optimisation under constraints problems, we can distinguish agents subjected to constraints and requiring the help of other agents (this situation can evolve during the solving process). Thus, two types of roles can be underlined: **the constrained role and the service role**. Agents submitted to constraints have the constrained role and are considered as the solving initiator. They express the problem and by solving their constraints the solution is reached. To solve their constraints, these agents request the services of agents having the service role. One agent can have one or both roles and switches at runtime between them depending on the situation it faces.

Agents designed with this model are based on the *Adaptive Multi-Agent System (AMAS)* theory. They possess a cooperative attitude. They are subject to cooperative failures called *Non Cooperative Situations (NCSs)* that they must solve by being as cooperative as possible. Thus, they are provided with criticality degrees representing their difficulties and importance (section 3.4). Given their cooperative attitude, agents always try to help the

more critical agents, thus equilibrating the satisfaction degrees of all the agents.

3.2.1 Constrained Role

The set of agents having this role possess the main constraints of the problem and by solving their constraints, they will lead to the solution of the problem. They are considered as the problem solving initiators. They each have a local goal which is to maximize their satisfaction degree by reducing their criticality degree. This criticality degree is computed using each agent's local knowledge and representations such as its dissatisfaction degree (unsolved constraints), the time spent searching for a solution, its insufficient knowledge (the agent does not know relevant agents) or the prejudice it may cause if its goal is not reached. The equilibrated maximization of their satisfaction is necessary to reach the optimized global solution. Thus, agents try to equilibrate their criticality degrees cooperatively.

Agents having this role, mainly interact with agents having the service role. For that, they must be able to compute their criticality degree using their local knowledge and representations. Their local knowledge mainly concerns the constraints they have. Their local representations concern the information on other agents and the environment. It is updated during the perception phase of the agent's life cycle.

Algorithm 3.1: The outline of the decision phase of agents having the constrained role.

```
while not satisfied do
    search for adequate service;
    if service found then
        evaluate the criticality level;
        request for service;
        if service accepted then
            search for improvements;
        end
    else
        request for information;
    end
end
```

Algorithm 3.1 presents the outline of the behaviour of an agent having the constrained role. It is divided into the following steps:

1. search for adequate service: first, the agent tries to search in its knowledge (representation on other agents) if there exist agents with the adequate service.

2. evaluate the criticality level: once an agent with the adequate service is found, the agent having the constrained role evaluates its criticality level using its local knowledge and representations. This criticality degree underlines the difficulties the agent has to find the adequate service and its importance for the agent.

3. request for service: after the evaluation of its criticality, the agent sends a request for service to the chosen agent indicating its criticality degree.

4. search for improvements: if its request is accepted, the agent having the constrained role will still search for improvements. Indeed, to reach a good solution, agents must have the best place they can find in the organisation. Thus, whenever they found an adequate agent having the service role, they will continue searching if other more suitable agents can be found.

5. request for information: during its searching process, the knowledge of the agent can be insufficient to reach its goal. Thus, it requests for information from already known agents to improve its acquaintances and discover adequate services.

During those different steps, as cooperative agents, agents having the constrained role face several *NCSs* such as *uselessness NCS* when their knowledge is insufficient. They solve these situations using specific cooperative rules introduced in section 3.4.

3.2.2 Service Role

Agents having the service role possess the required knowledge, skills and competences to help solve the constraints of agents having the constrained role. They ensure the connectivity of the resolution as they take into account the needs and difficulties of the agents having the constrained role and contribute to reach the criticality equilibrium between them. They have knowledge on other services of the system and can delegate a part of the requested services to other agents having this role or ask the assistance of other agents to realize their services.

Algorithm 3.2: The outline of the decision phase of agents having the service role.

```
get all the received requests;
while there exists an untreated request do
    if request for information then
        answer the request;
    end
    if request for service then
        register the request & its criticality;
    end
end
select the most critical request among the registered requests;
inherit criticality;
send accept request answer to selected agent having the constrained role;
send reject request answers to other agents;
```

At each life cycle, its behaviour is outlined by algorithm 3.2 where two types of requests can be underlined: information and services. An agent having the service role responds directly to the first one as it does not implicate its engagement to perform a service. For the

second one, the agent must consider the criticality degree of the requesting agents. Indeed, given its cooperative behaviour, it responds to the requests for service by equilibrating the satisfaction degree of the requesting agents. Thus, it selects the most critical agent and inherits its criticality. It faces several *NCSs* and solves them using specific cooperative rules introduced in section 3.4.

Agents having the service role may require the help of other agents to perform their service. They can ask for **service delegation** or **assistance**. The service delegation consists in dividing a service into different sub-services and delegating each sub-service to a given relevant agent. Requesting for assistance means that to perform its service, the requesting agent requires the help of another agent and both will perform the same service. In both cases, agents requiring the help of other agents possess both roles and switch to the constrained role in order to find the relevant agent.

3.3 Agent Interaction and Communication

Different types of messages can be exchanged between agents depending on their role. Among them we underline:

- **Request for service**: This request is used by agents having the constrained role to request services from agents having the service role. To specify their requests, agents use different information based on their local knowledge and representations. When requesting a service the criticality degree of the agent requesting the service is needed. Agents having the constrained role compute this degree using their local knowledge. When delegating services or asking for service assistance, agents having the service role after switching to their constrained role, transfer the criticality degree received from the agents requesting their service adding to it their own criticality representing their difficulty to obtain the service they are searching for.

- **Request for information**: such interactions exist between agents when requesting for additional information. It implicates agents from both roles.

- **Answer to requests**: used by service role agents to answer the received requests.

- **Information messages**: in addition to the service requests and answers, agents can exchange information judged to be useful for other agents of the system.

Figure 3.1 illustrates some of the possible interactions between four agents (*A1, A2, A3, A4*). *A1* and *A4* possess both constrained and service roles. *A3* requests the service of *A1* specifying its criticality degree (*criticality1*). To respond to *A3* request, *A1* requires the help of another agent. Thus, it switches to its constrained role, computes its criticality degree (*criticality2*) and requests the service of *A4*. *A4* is available to respond to *A1* request and sends it an *accept()* answer. *A2* is available but its service is not requested. Thus it informs other agents (here *A4*) about its availability.

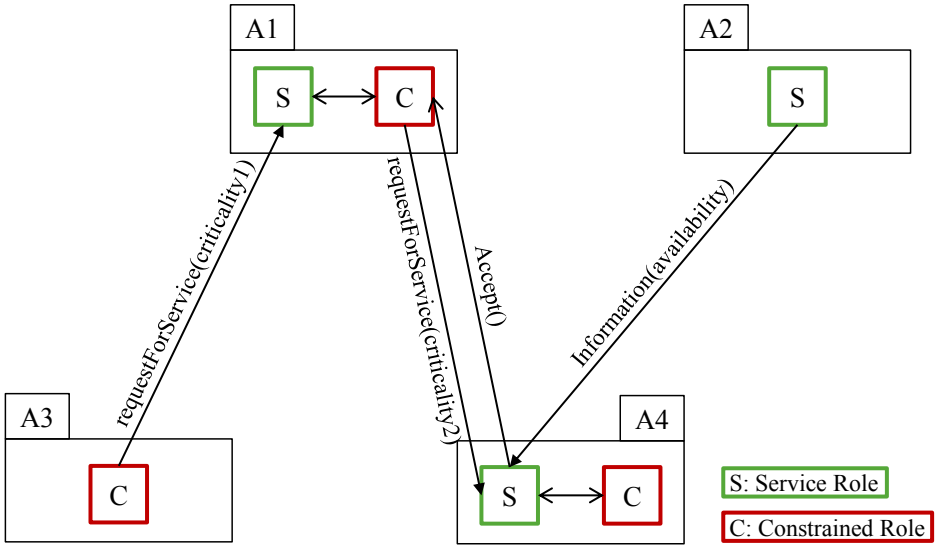

Figure 3.1 — Example of agent interactions

Note that the definition of those messages remains a guide for the agent interactions and must be adapted to a given application. They represents a subset of the *FIPA*[1] *Communicative Acts* and can be implemented by them: the request for services or information messages can be implemented using the *request* act, the answer to requests can be implemented using the *Agree* and *Refuse* acts and the information using the *Inform* act.

3.4 Agent Criticality

Agent criticality is defined as the local measure of the dissatisfaction degree of an agent. It denotes the difficulties an agent has to reach its goal and its importance for the system. In adaptive multi-agent systems, this measure is very important for the cooperative behaviour of the agents. Indeed, by their cooperative attitude, agents always try to help the most critical agents and must anticipate or detect and repair cooperation failures called *NCSs*. In such systems, this criticality measure is considered as the foundation of cooperative behaviour.

In our agent model, **two criticality degrees** are to be distinguished. First, **when requesting for a service**, an agent must be able to compute its criticality degree using its local knowledge and representations. It combines different measures such as the number of unsolved constraints, the time spent searching for a solution, the insufficiency of the agent knowledge or the prejudice it may cause if the agent does not reach its goal. Thus, this measure is relatively dependent on the situation of the agent. Second, **when an agent asks for service delegation or assistance**, it must be able to associate to each request the adequate criticality. For instance, when asking for assistance, the agent must combine the criticality of the received request with its own difficulty to find this assistance. On the contrary, when asking for service delegation it must divide the criticality of the received

[1] www.fipa.org

request in proportion to the delegated part of the service and combine it to its own local knowledge.

This criticality degree represents for each agent the difficulty to reach its goal. As cooperative entities, agents act in order to equilibrate the satisfaction degree among themselves and reduce the global criticality of the system. When detecting a *NCS*, the agent associates a criticality measure to this *NCS*, depending on its situation or the situation of other agents involved in this *NCS*. As the agent always seeks to find an equilibrium of satisfaction, the repair of detected *NCSs* is highly dependent on these criticality degrees. Thus, an agent must take them into consideration when deciding which actions to perform.

3.5 Cooperative Rules

As stated on the definition of the *AMAS* Theory, when designing an adaptive multi-agent system, designers concentrate on ensuring local cooperative interactions between agents. In this theory, agents always aim at acting and interacting cooperatively between them in order to maintain a cooperative attitude. Thus, they detect or anticipate problems that introduce cooperation failures called *NCS* and choose actions to repair or avoid them.

Depending on their roles, agents face different *NCS* categories. From the seven defined categories of *NCS*, agents considered in optimization under-constraints are only subject to *five*: **incompetence, unproductiveness, uselessness, conflict,** and **concurrence**. The **incomprehension** and **ambiguity** categories are related to communication. We consider that information exchanged between agents is understandable and non-ambiguous in these kinds of systems as agents are the result of the same design process, by the same designer, or common specifications. They are always able to interpret the messages, extract useful information and update their local representations and knowledge.

Concerning the others (**incompetence, unproductiveness, uselessness, conflict,** and **concurrence**), their description in the *AMAS* theory remains a high level guide for the definition of cooperative failures that agents can encounter. In the following, these *NCSs* are specialized for the optimisation under-constraints problems. For each *NCS*, a description of the situations where such *NCS* can occur is given. The way an agent solves this *NCS* is defined as a cooperative rule. The set of cooperative rules are to be integrated in the *ADELFE* methodology and can be used when trying to solve optimisation under-constraints problems.

3.5.1 Incompetence

This *NCS* occurs when an agent having the service role receives a request for service but does not have the required qualification to respond. If the agent knows another qualified agent, it can forward the service to this agent or informs the sender about its existence (figure 3.2). In all cases, the agent must send its incompetence to the original sender so it can update its local representations (figures 3.2, 3.3), so as not to repeat the same error.

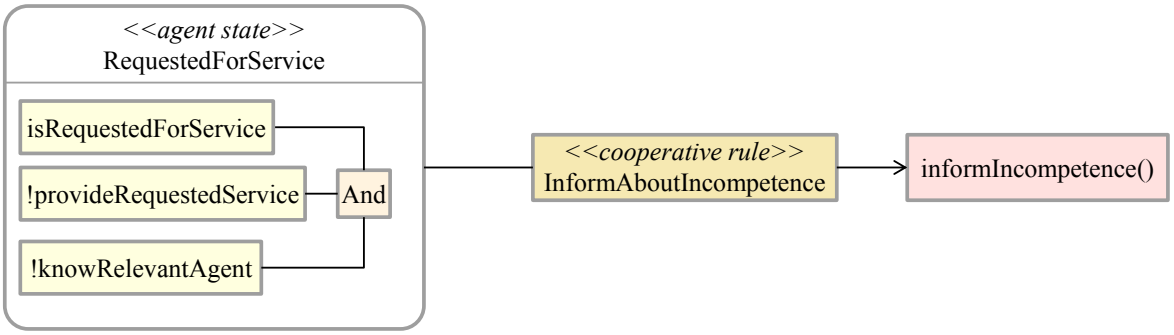

Figure 3.2 — Inform about incompetence *NCS* rule diagram

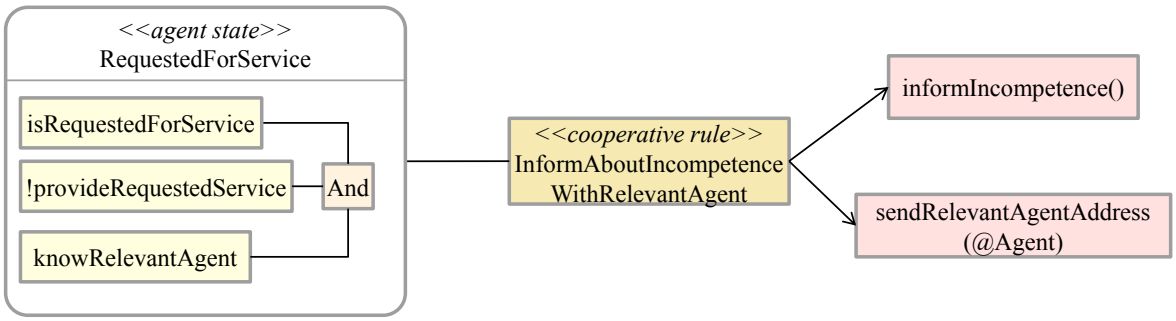

Figure 3.3 — Inform about incompetence and send the address of the qualified agent *NCS* rule diagram

3.5.2 Unproductiveness

This *NCS* is detected when the received information is incomplete, already known by the agent or has no interest for the agent. For example, the information is incomplete when an agent having the constrained role explores the environment and receives the addresses of new agents but does not have any knowledge on them. In this case, the agent must contact these agents in order to improve its acquaintances (figure 3.4). Whenever receiving already known or uninteresting information, the agent informs the sender which updates its representations (figure 3.5). The implementation of this last rule can be useful for instance when memory/network load balance is an issue. Indeed, the designer can choose to implement it so as to reduce network load since the sender after updating its representations will not send this information again or as frequently. But, the designer must also consider the memory space used by the sender which must register the interests of each agent. In case this rule is not implemented, the agent just ignores such received information. In addition to these situations, this *NCS* occurs when the information of the agent is insufficient to reach its goal (figure 3.6). In this case, the agent must contact already known agents in order to get useful information. Agents from both roles can face this *NCS*.

3.5.3 Uselessness

This *NCS* concerns both *constrained* and *service* roles. Agents having the constrained role are considered as the problem solving initiator. When entering the system, they explore the

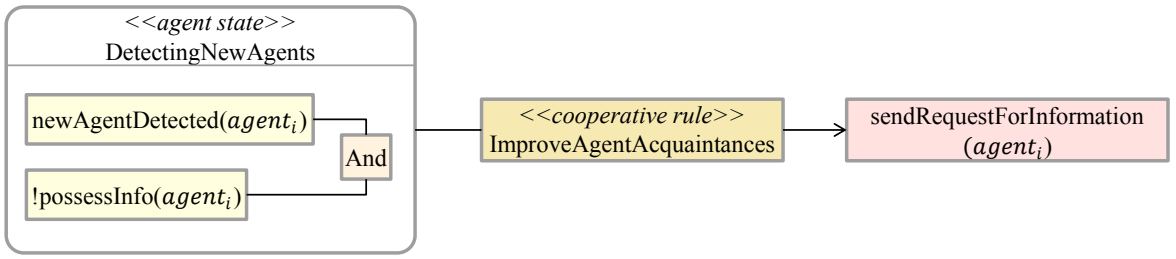

Figure 3.4 — Improving agent acquaintances *NCS* rule diagram

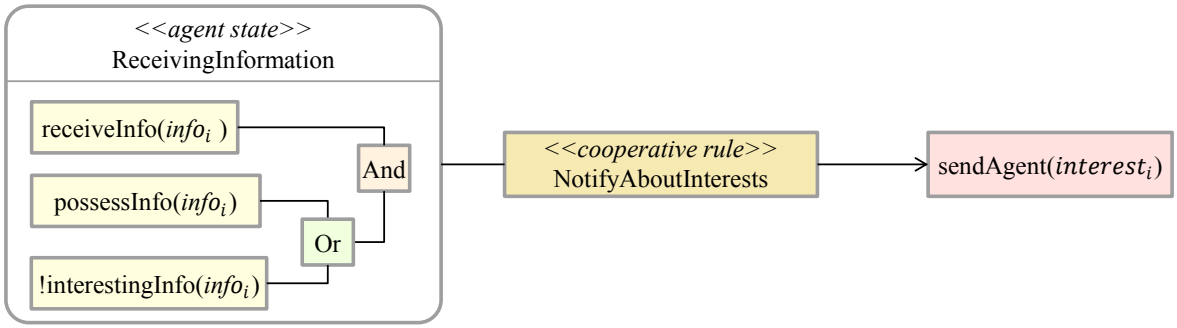

Figure 3.5 — Informing other agents of the agent interests *NCS* rule diagram

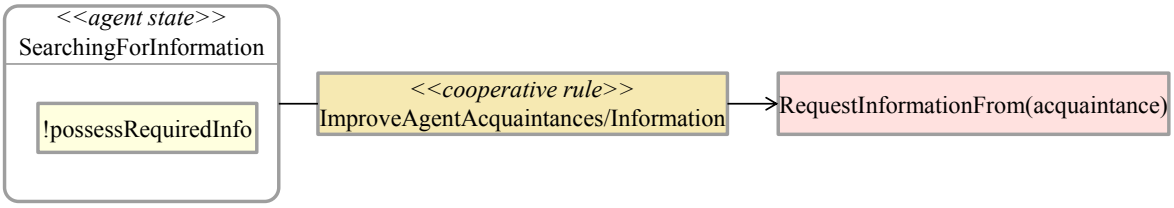

Figure 3.6 — Improving agent acquaintances or information *NCS* rule diagram as current information is insufficient to reach the agent goal

environment searching for adequate services. This *NCS* is detected during this exploration phase as agents have not found their place in the organization yet or as agents have found a place in the organisation but better places or better interactions can exist (*partial Uselessness*).

During the exploration phase, if the exploring agent knows a qualified agent but it has not contacted it, it must send it a request (figure 3.7). If no qualified agents are known, the requesting agent must search for them by asking for the neighbourhood of his acquaintances for instance (figure 3.8).

An agent having the service role detects this *NCS* when its services are not solicited by its environment. As a cooperative agent, it considers itself useful for the system and must help other agents to reach their goal (figure 3.9) by proposing its service to them when it considers it is necessary (for load balancing for instance).

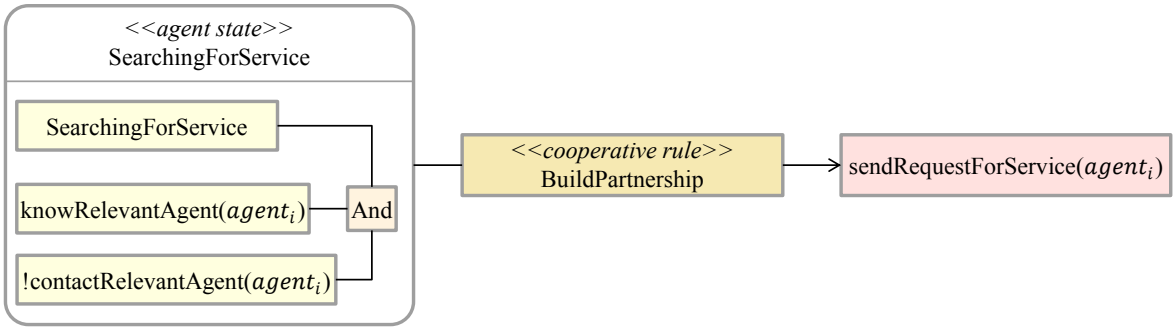

Figure 3.7 — Searching for partnership *NCS* rule diagram

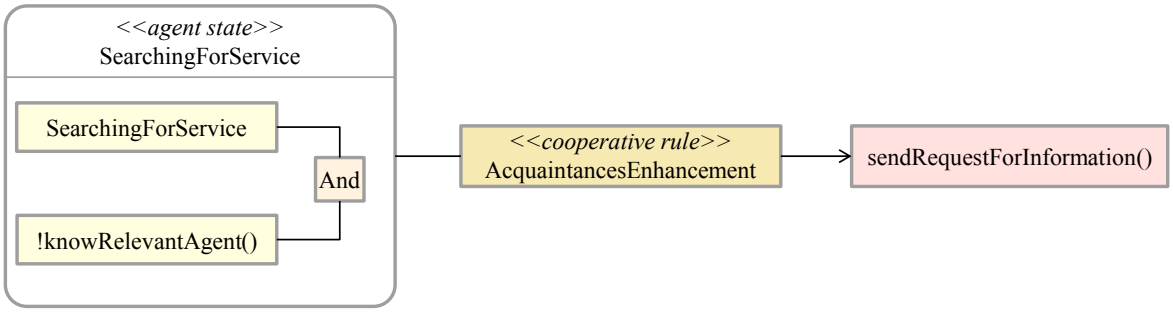

Figure 3.8 — Acquaintances enhancement *NCS* rule diagram

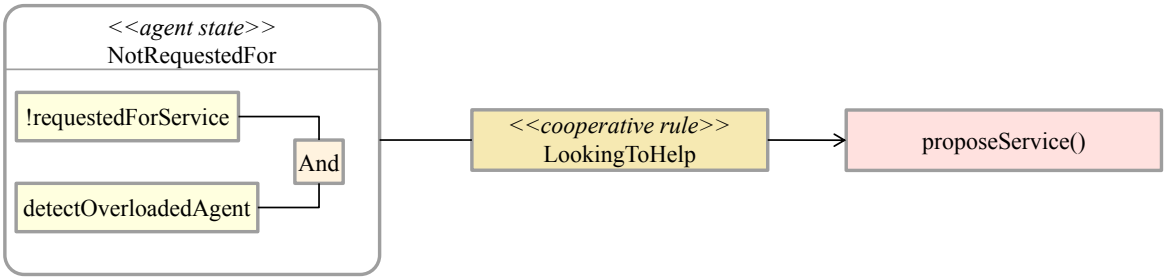

Figure 3.9 — Looking to be helpful *NCS* rule diagram

3.5.4 Conflict

This *NCS* occurs when different agents want to access the same service. It is related to services requests. This *NCS* can be detected in two manners. First, agents offering the service when receiving different requests detect the conflict and solve it using the criticality degree of agents requesting the service (figure 3.10). This criticality degree represents the difficulty of an agent to obtain the service and the importance of the service for its constraints satisfaction. Second, in some specific systems, agents requesting for service detect the conflict between them and communicate together beforehand to decide which one of them is the most appropriate to access the service. This second manner is less straightforward than the first one as it requires additional information exchange.

Figure 3.10 — Conflict *NCS* rules diagram

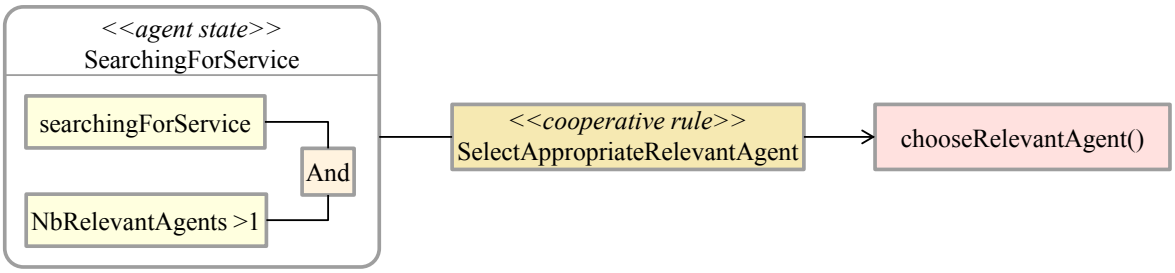

Figure 3.11 — Concurrence *NCS* rules diagram

3.5.5 Concurrence

While exploring the environment, an agent requesting a service can detect different adequate agents having the service role. As the conflict *NCS*, this *NCS* can be detected either by the agent requesting the service or by the agents offering the service. In the first case, the agent detects a concurrence *NCS* between the relevant agents and must choose the most appropriate one using its local representations and knowledge (figure 3.11). In the second case, the agents must interact between them and decide which one of them is the most appropriate to answer the request. The first manner is more intuitive and easier to implement. The choice of the relevant agent is based on different criteria such as the load of the agent, the delay, the communication security, the time to obtain a response or the easiness to obtain the service.

3.5.6 Illustration of some Non Cooperative Situations

Figure 3.12 illustrates three different *NCSs* between agents having constrained or service roles.

The first one (in the left of figure 3.12) presents a *conflict* situation between two agents (*A2 & A3*) requesting services from one agent (*A1*). *A1* decides which one to choose using the criticality degree sent by each requesting agent.

The second situation (in the middle of figure 3.12) presents a *Uselessness* situation faced by the agent *A4*. Here, *A4* has the constrained role but does not know any relevant agent having the service role. Thus, it has to send requests for information.

Finally, agent *A7* detects a *concurrence* situation (in the right of figure 3.12) between agents *A5*, and *A6*. Choosing the appropriate agent depends on different criteria which

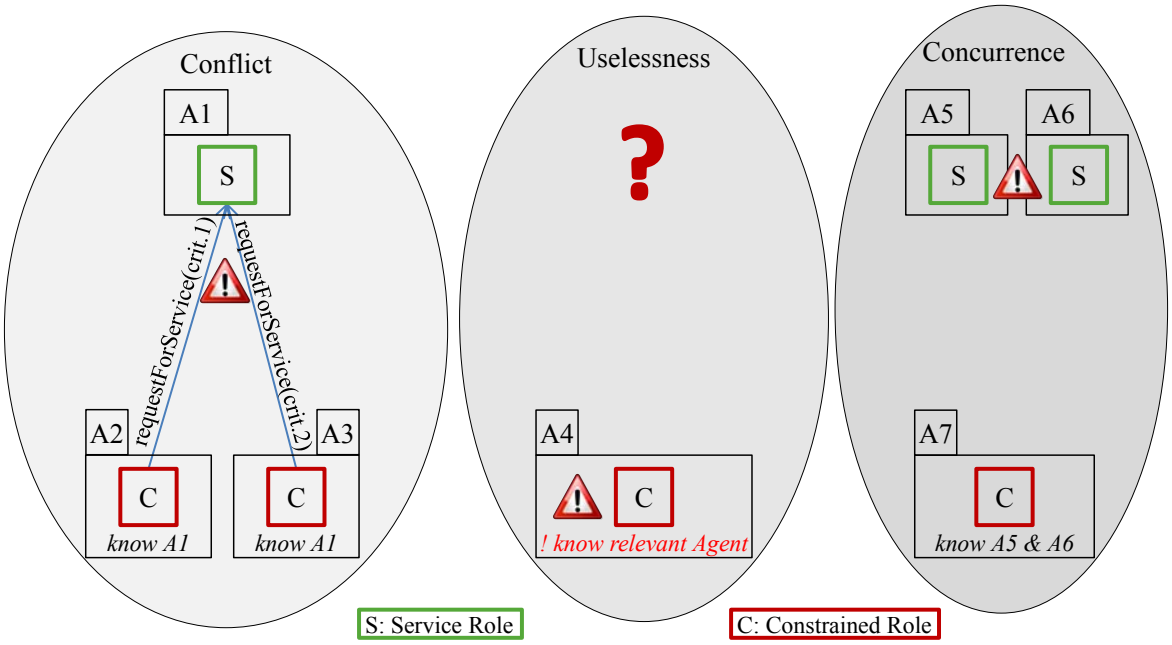

Figure 3.12 — Three *NCS* detected in scheduling in the manufacturing control problem

are application dependent (for instance, is one agent more fragile then the other? is there an associated coast?).

3.6 Specification of Agent Modules using *AMAS-ML*

In this section, a description using the *AMAS Modelling Language (AMAS-ML)* (section 2.4) of the different modules defining an agent architecture (section 2.2.4) is detailed.

Figure 3.13 (resp. figure 3.14) introduces the modules of an agent having the constrained role (resp. the service role). The architecture of an agent playing both roles can be obtained by the aggregation of both architectures.

In both figures we distinguish three main parts:

- The *perception module* enables the agent to perceive its environment.

  For the agent having the service role, this module enables the agent to perceive the requests for services (*requestForService()*) and for informations (*requestForInformation()*).

  The answers to the requests (service, delegation of service or service assistance) of agent having the constrained role such as the acceptance (*requestedServiceAccepted()*) or rejection (*requestedServiceRejected()*) of a request, are found in this module.

  Several information judged useful to the agent (both roles) by other agents can also be perceived (*information()*).

  Note that, the different methods of this module are generic and can be instantiated depending on the application by adding additional parameters specifying the requested service and the agent criticality for instance. In addition to this, in this model agents

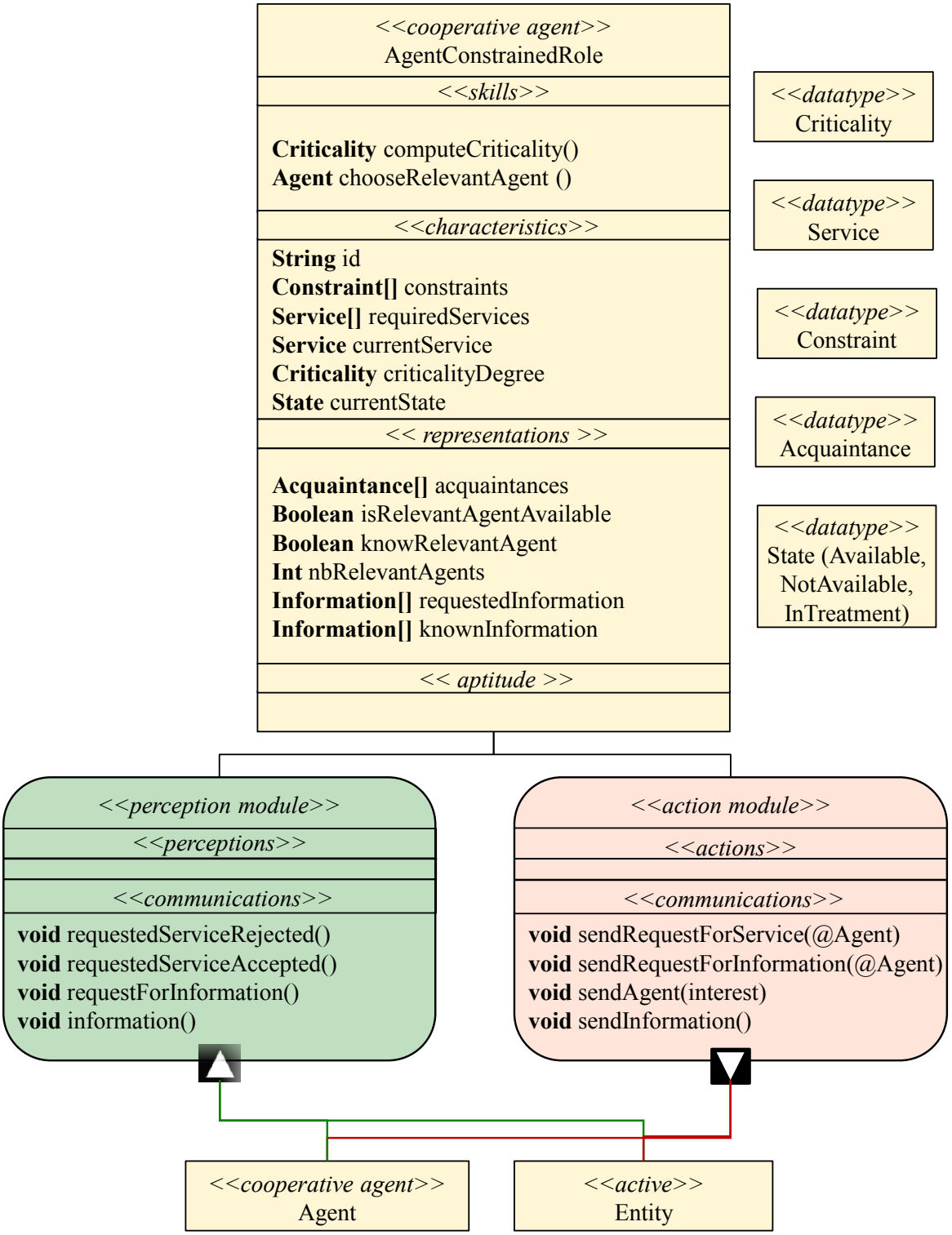

Figure 3.13 — The *AMAS-ML* description of agents having the *Constrained Role*

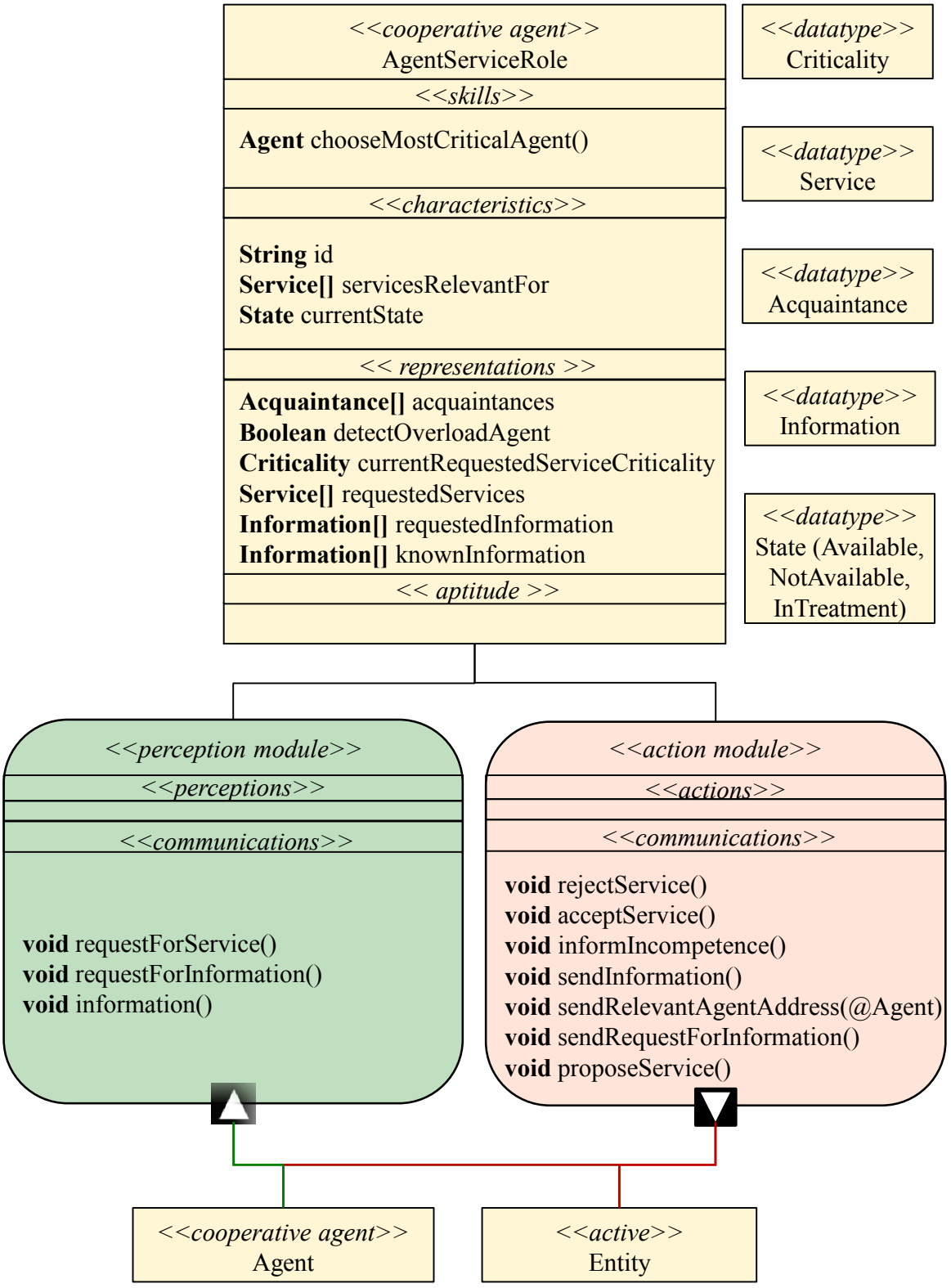

Figure 3.14 — The *AMAS-ML* description of agents having the *Service Role*

communicate using messages. Thus, the described methods are in the *communications* part of this module. In some cases when, for example, the neighbourhood of agents is fixed or agents are executed in the same environment, agents can access the information of other agents directly without requesting it. Such access can be implemented directly using *getters* methods, and these methods belongs to the *perceptions* part of this module.

- The *action module* allows the agent to act on its environment.

  The agents having the constrained role use this module to send services requests (*sendRequestForService(@Agent)*) to other agents. This module is used by agents of both roles for informations requests (*sendRequestForInformation(@Agent)*) .

  It is also used by agents having service role to answer services or information requests (*rejectService()*, *acceptService()*, etc.).

  In some cases, this module is used by the agents to solve *NCSs*. For example, when receiving the address of a new agent (unproductiveness *NCS*), the agent uses the *sendRequestforInformation(@Agent)* action to acquire information on that agent. A service role agent detecting that other agents are overloaded (*detectOverloadAgent*) uses its *proposeService()* action to try to solve its *Uselessness NCS*.

  Like the *perception module*, methods of this module are generic and must be instantiated accordingly to the developed application. Direct access methods implemented as *setters* can also be found in the *actions* part of this module.

- The *knowledge modules* of the agent regroups its *skills, characteristics, representations* and *aptitude* modules.

  The *skill* module contains methods enabling the agent to decide how to act or to perform local computation.

  The *characteristics* and *representations* modules contain the information the agent has on itself and its environment (social and physical). Such information is used by the *skills module* of the agent. They are updated locally by the agent during the perception phase.

  The *aptitude* module can be instantiated by specific tools to the studied application judged by the designer to be useful for the treatment of the agent.

  Some examples on the usage of theses modules:

  - when looking for a service, the agent computes using its *acquaintances* the number of relevant agent (*nbRelevantAgent*). If a relevant not contacted agent is known (*knownRelevantAgent*), its sends a request to it by specifying its criticality using the *computeCriticality()* skill, if not, it sends requests for informations to its acquaintances.
  - When an agent detects a *Concurrence NCS* that occurs when different relevant agents are detected, it uses its *chooseRelevantAgent()* skill to decide which agent to contact.
  - When a service role agent receives different requests (*requestedServices*), it decides which request to choose using its *chooseMostCriticalAgent()* skill.

The different fields and methods of theses modules are required by the constrained role agents to satisfy their constraints, and by the service role agents to decide how to treat the received requests. They are generic and their instantiation depends on the studied applications.

Those three defined parts, using the interaction and cooperative rules of each agent, describe the architecture of the agent. The architecture of an agent having the service role is similar to architectures defined by agent-role methodologies [Kendall, 2001]. Nevertheless, it differs by the cooperative rules that impact the behaviour of agents, and the fact that services offered by these agents are designed to help agents having the constrained role to satisfy their constraints. Thus, these two agents architectures are specific to problem solving under constraints.

3.7 MAY Agent Architecture

When analysing the different modules of both roles in order to design the agent architecture using *Make Agents Yourself (MAY)*, a large similarity is noticed. Indeed, the architecture definition using *MAY* consists in defining how the different μ-components are connected (section 2.5). As the difference between both roles concerns internal functions performed by the μ-components of the agent, agents having one or both roles are considered being from the same species. Thus, one species of agents called *Resolution Agent* is created.

This architecture contains six μ-component (*LifeCycle, MailBox, Send, Receive, Perception, Decision* and *Action*) divided into the two levels as presented in figure 2.7.

In addition to these μ-components, two μ-components are to be added : the *Characteristics* and *Representations* μ-components. Those μ-components must offer a set of methods required by the *Perception* μ-components to maintain the agent knowledge up to date, the *Decision* μ-components to access this knowledge and decide which actions to perform and the *Action* μ-components to perform the set of decided actions. As the *characteristics* and *representations* modules are different for both roles, the *Resolution Agent* species is completed by the appropriate μ-component (figures 3.15, and 3.16) before agent creation. An agent having both roles must implement the methods of both μ-components.

Considering the skills of agents, in our case, they are internally used by the *decision* module. Thus, no additional μ-component is required for them. They are implemented as private methods in the implementation of the *Decision* μ-component. Thus, both roles share the same *Decision* μ-component.

3.8 Conclusion

In this chapter, the generic agent model *AMAS for Optimisation (AMAS4Opt)* developed to specialise the usage of the *AMAS* theory for complex optimisation problems under constraints has been detailed.

By analysing the agents interactions during the solving process, two roles that guide

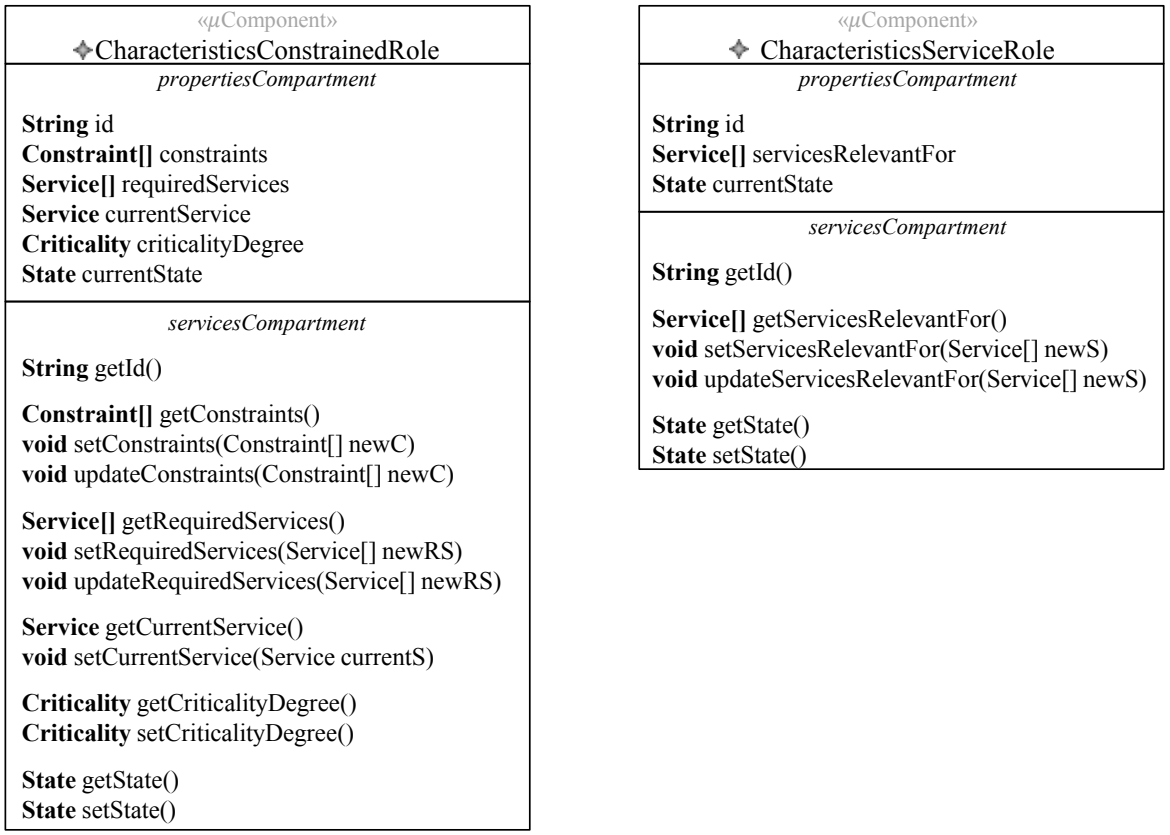

Figure 3.15 — Characteristics μ-components for *constrained* and *service* roles

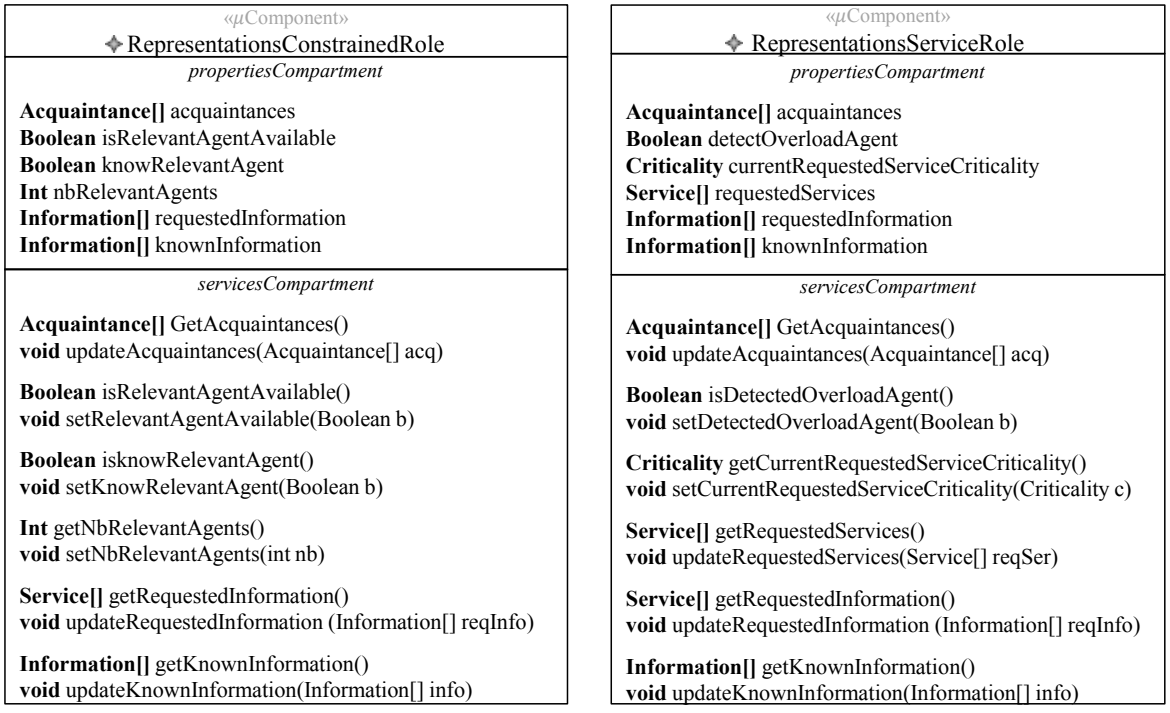

Figure 3.16 — Representations μ-components for *constrained* and *service* roles

the agents behaviours have been underlined: *constrained* and *service* roles. For each role an outline of the agent behaviour is given. In this model, agents interact using direct communications. The different required messages have been identified and discussed. In addition to this, as stated in the *AMAS* theory, agents cooperate in order to reach their local goal and to equilibrate their satisfaction degrees. Thus, they possess a cooperative attitude and always try to anticipate or detect and repair cooperation failures called *Non Cooperative Situations (NCSs)*. A list of *NCSs* that agents of this model can encounter has been addressed.

Concerning the theoretical contribution of this thesis, the *NCSs* of the *AMAS* theory have been specialised for the solving of optimisation under constraints problems. For each rule, an explicit definition including how the *NCS* can be detected by agents, to which role it belongs and how the agent can solve it is given. In addition to this, the agent criticality has been introduced and defined to guide the cooperative choices of the agents. Different measures representing the local measure of the dissatisfaction degree of the agent can be used to compute this criticality degree. Still, this measure is application dependent.

From the engineering point of view, two generic algorithms defining the behaviour of both roles are given. Each algorithm provides the engineer with the different steps to follow in order to design their agents. The definition of each *NCS* states when an agent can detect it, by that helping the engineer to identify the *NCSs* that its agents may face. The cooperative rules defined to solve each *NCS* provide the engineer with the knowledge, representations and skills required by each agent to anticipate or detect and repair *NCSs*.

In conclusion, the *AMAS4Opt* model defines the process to follow to design dedicated agents for the solving of optimisation under constraints problems. It requires the analysis of the natural description of the problem to identify the agents and their roles. Still, as the proposed roles and behaviours are close to the problem description, we feel confident that this identification is intuitive. In order to evaluate our model, chapter 5 presents two instantiations of this model for the solving of two well-known optimisation problems: the scheduling in manufacturing control problems and the design of complex products.

4 Criteria for the Evaluation of Self-Adaptive Multi-Agent Systems for Complex Problem Solving

« Not everything that can be counted counts and not everything that counts can be counted. »

Albert Einstein

Contents

The chapter in english starts page 89.

Résumé général du chapitre

La complexité croissante des logiciels favorise la conception de systèmes multi-agents auto-organisateurs[1] *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. Parmi ces propriétés, nous distinguons :*

- ***l'auto-organisation*** *: processus par lequel un système change, lors de son exécution, son organisation sans aucune intervention externe. 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;*
- ***l'auto-réparation*** *: mécanisme permettant la détection et la réparation des erreurs;*
- ***l'auto-régulation*** *: mécanisme permettant l'ajustement autonome des paramètres.*

Une forte dynamique liée à l'apparition d'évènements imprévus caractérise de nombreuses applications actuelles comme en télécommunications, économie ou bio-informatique. Les systèmes classiques supportent difficilement ces caractéristiques mais, contrairement aux systèmes self- , des preuves formelles concernant leur performance peuvent exister. De plus, le fonctionnement de systèmes classiques peut être validé avant même leurs déploiements 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 aux concepteurs de systèmes multi-agents adaptatifs 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, ce chapitre focalise 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 ou la complexité sont plus appropriés et doivent être considérés.* ***L'objectif de ce chapitre est de définir et analyser les critères essentiels d'évaluation de ces systèmes auto-adaptatifs.***

Ce chapitre présente les critères identifiés selon trois parties:

La première concerne ***l'évaluation du système en cours de fonctionnement****. Cette catégorie concerne la validation de l'adéquation fonctionnelle du système. Elle se décompose en des mesures de*

[1] Systèmes multi-agents capables d'auto-organisation en cours d'exécution

performances *(le temps, la charge de communication, la précision et la qualité, l'indéterminisme, la progression et la gestion de la mémoire) ainsi que des mesures de* ***robustesse et d'homéostasie*** *tel que le temps d'adaptation nécessaire après l'apparition d'une perturbation.*

La deuxième catégorie étudie les ***caractéristiques intrinsèques du système****. Ces critères pouvant être étudiés avant l'exécution du système, permettent de souligner la complexité algorithmique du système et la décentralisation du contrôle. En effet, dans ces systèmes, la distribution et la décentralisation rendent impossible l'évaluation de la complexité algorithmique globale du système. Ainsi les mesures portant sur la localité de la résolution telle que la complexité des algorithmes locaux à chaque agent, la localité de leur actions, le degré de décentralisation sont plus appropriés.*

La troisième catégorie considère la ***méthodologie de conception****. En effet, la conception des logiciels capables de s'adapter à un environnement dynamique impose une méthode de conception rigoureuse qui se distingue de l'approche globale-descendante habituelle. Différentes méthodes de conception ont été développées présentant des degrés de difficultés variés. Cette catégorie propose des critères d'évaluation de ces méthodologie selon trois points: l'identification des agents et leur conception, la facilité de déploiement et de distribution et la généricité. Ces critères sont plus qualitatifs que quantitatifs, mais sont utiles pour les prochains développements basés sur la même approche. Cette caractérisation souligne les points positifs et négatifs d'une méthodologie par rapport à une autre.*

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 pour souligner ses limites et ses avantages. La dernière partie de ce chapitre propose un ensemble de moyens facilitant cette comparaison et présente un panel de comportements typiques associés aux processus d'auto-adaptation. Elle souligne le fait qu'une vue* ***multi-objectif*** *est plus adaptée pour la comparaison de ces systèmes vu l'interdépendance des différents critères mesurés.*

4.1 Introduction

In the last few years, the growing complexity of current applications has led to designing self-organizing systems presenting self-* properties [Serugendo et al., 2005]. Those systems are necessarily composed of several autonomous interacting entities (called agents) acting in an environment. In general, the global behaviour of the system emerges from the local interactions between its agents. Those systems behave autonomously and must handle the dynamics coming from exogenous or endogenous changes[2], *i.e.*, they are able to self-adapt [Bernon et al., 2003; Robertson et al., 2001].

Moreover, those systems are characterized by different self-* properties such as:

- self-organization: the mechanism or the process enabling a system to change its organization without explicit external control during its execution time [Serugendo et al., 2006].

[2]Endogenous changes are perturbations caused by entities belonging to the system. Exogenous changes solely occur from entities outside the system. Handling these perturbations can lead to change agents behaviour or even the global objective of the system.

- ▷ self-stabilization or homeostasis: the system always reaches a stable state [Würtz, 2008].
- ▷ self-healing: the mechanism that allows a system to detect and recover from errors.
- ▷ self-tuning: the system is able to adjust its parameters.

Generally, those characteristics make systems more robust than classical ones and by that more adequate to solve complex real-life problems common in many domains: telecommunications, economics, bio-informatics, industrial environments, etc. Real-life problems present a lot of dynamics where events are unpredictable and can occur any time. Classical systems with no self-⋆ properties are usually inappropriate to solve such problems. The main advantages of classical systems over self-⋆ systems are that, for most, formal proofs of reaching good solutions exist and the functioning of the system can be validated before deploying it in a real environment. Classical systems are usually predictable in their functioning. The question is how can we point out the advantages of self-⋆ systems over classical ones when for instance most of them are not deterministic? Or, how can self-⋆ properties guarantee good solutions under perturbations? Or, how can we prove that they are functionally adequate (e.g they do what they are designed for)? Yet, there are no tools to formally prove the behaviour of self-⋆ systems [Edmonds and Bryson, 2004; Edmonds, 2005; Gleizes et al., 2008] and only **empirical evaluation** can help answering those questions.

As those systems are different from classical ones, new or updated characterization and evaluation criteria are required for analysing the contribution of self-⋆ properties and system performance. In this work, we aim at collecting a number of ideas and propositions about how to evaluate such systems. The idea is to propose relevant evaluation criteria that help to validate the behaviour of these systems, underline their advantages on classical approaches by showing how self-⋆ properties can increase the performances and thus, ease their acceptance for the industry.

For the evaluation of those systems, researchers commonly use a very reduced number of criteria such as time or communication load. These two criteria are important but not sufficient to highlight their specificity. Other criteria such as homeostasis, decentralization degree[3] or complexity are more appropriate and must be studied.

Generally, these systems show two kinds of behaviour: the *nominal behaviour (N)* which is the normal behaviour (i.e. the behaviour of classical systems) and the *self-⋆ behaviour (S)* which enables the system to handle unexpected dynamics. In some systems, those behaviours are distinct. In some others, they are intricate. This chapter aims at guiding the evaluation of this kind of system from the design phase to the execution phase by providing the main criteria regrouped in different sets.

In the following, the identified evaluation criteria are classified in three parts: the first concerns the system at runtime, the second discusses its intrinsic characteristics and the third presents the methodologies characterization. In each part, the main criteria to measure are defined and a way to examine them is given when possible. Finally, a proposition to compare different systems to point out their originality, advantages and weaknesses is presented.

[3] A measure of the control and decision making distribution between agents[Loor and Chevaillier, 2003]

4.2 Evaluation of the System at Runtime

Different questions are to be considered by designers to validate the well-functioning of the system and of the self-* mechanisms, such as: is the system able to solve the problem for which it is designed? Is the system able to self-adapt in an efficient way? Are the self-* properties implemented in the system sufficient to insure the needed robustness? etc.

Evaluating the system at runtime concerns the results obtained at the end of its execution but also its behaviour during it. The former part gives information on the system performances. While the latter part enables the study of how the system handles dynamics and points out the self-* characteristics. Note that the end of the execution of a self-adaptive system is not so easy to determine. This kind of systems has to continuously self-adapt to its environment and usually keeps running indefinitely. If it has to reach a given state or a solution, the end cannot be known by the system itself, mainly because of the distribution. Designers must add a global observer of the system which has to decide if the system has reached its goal and by that has to stop.

4.2.1 Performance metrics

The evaluation criteria presented in this section concern the results of different system executions for a given scenario. Six points are considered: **time, communication load, non determinism, accuracy/quality, progress** and **memory usage**. Once these measures are done, they can be used by the evaluator to compare its system with others.

Time. Measuring the time studies the computation time needed by the system to reach a solution. In real-time dynamic environments, systems must be reactive and respond quickly by proposing good solutions. In some other static environments, the quality of solution is important and thus, additional computational time can be given to the systems in order to get optimal solutions. By studying this criterion, designers can know if their system is suitable or not to a given context.

Commonly utilized, time has different definitions. In [Meisels et al., 2002], it is expressed by the number of checked constraints, or more generally the number of performed atomic operations. Another frequent way is the measurement of computing time (CPU) [Clair et al., 2008], [Hansen et al., 2008; Meignan et al., 2008]. For this, the characteristics of machines should be considered. [Dongarra, 1988] introduces some transformation ratios in order to make times of different machines comparable. Others authors [Clair et al., 2008], [Lynch, 1996] study the time as the number of steps needed by agents to reach the solution[4]. Another common way is the time needed by the most costly operations.

We consider that the number of steps needed by agents is the most appropriate and easiest way to measure time. In fact, considering the measurement of CPU needs to take into account machines and platform characteristics, while counting the number of verified constraints in distributed context must consider the asynchrony between the agents [Meisels

[4] In [Clair et al., 2008], it is calculated as the fabrication cycle number. While in [Lynch, 1996], it is the number of rounds for a synchronous algorithm.

et al., 2002] which is also the case of considering the time needed by the most costly operations.

At the beginning of distributed systems, designers worried that the systems spend too much time communicating than working. Currently in self-$\star$ systems, the dilemma has shifted to the ratio between working and self-adapting. In self-$\star$ systems, we propose to distinguish when it is possible the Nominal Time (NT) needed to perform its usual function (which could be to solve a problem) from the Self-$\star$ time (ST) needed to self-adapt to changes in the environment. For example, in a mail server representing self-healing characteristics a distinction can be done between the time spent by the server doing its nominal behaviour (sending or receiving mail messages) from the time spent on searching for help when it is overloaded. When this distinction is not possible, an evaluation of the Mean of the Time Differences (MTD) between the execution of the system in a scenario without perturbation and its execution in a scenario with perturbations can be realized. For instance, in a self-$\star$ system solving a constraint satisfaction problem, it is difficult to perform this separation. Some can consider that the time spent by an agent to find an affectation for its variable is the NT, while others consider it as the ST.

Communication Load. In self-$\star$ systems, agents usually need to communicate in order to reach the solution or the organization needed to attain the solution. Examining the communication load enables the study of network load which is an important criterion for the deployment of the system in a real environment. Indeed, systems with high communication load require network infrastructure that can handle important communications. Studying this criterion gives information on which infrastructure a system can be deployed.

A way to measure this parameter is the amount of exchanged messages. The exchanged messages can be sorted in two sets: the messages Nominal Communication Load (NCL) for the solving part and the messages Self-$\star$ Communication Load (SCL) for self-adaptation. For example, in a mail server representing self-healing characteristics we can distinguish between sent or received mail messages and messages asking for help when the server is overloaded.

But, as the number of exchanged messages is highly dependent on the centralization/decentralization degree[5] of the algorithms, it would be interesting to analyse the communication load in regard to the decentralization degree. In decentralized algorithms more messages are needed among agents while in centralized approaches agents do not have to exchange a lot of information. Unfortunately, rare are the studies that combine them.

Different other parameters are interesting to examine when studying the communication load [Brito et al., 2004]. Among them, the message *size* and *complexity* are to be underlined as they are relevant indicators of the time needed to understand and process messages. The network load is mainly calculated using this communication load in addition to transit and

[5]Calculated as the ratio between the number of agents in charge of the decisions and the total number of agents.

latency time.

Accuracy/Quality. The accuracy or quality of solution refers to functional adequacy of the designed system. A system is functionally adequate if it is useful for its environment [Gleizes et al., 2008; Parunak and Brueckner, 2004] which means that it does what it is conceived for. A convenient measure for solution quality is the distance from the optimal value ($\delta(S_{cur}, S_{opt})$). The main problem of this measure is the determination of the optimal solution. In some cases, this value is known for some instances, but in general it is unknown. That's why some authors usually use scenario belonging to well-known benchmarks and compare their solution with best known solutions. The quality of self-adaptation can be highlighted when running the system on a sufficiently high number of scenarii presenting different levels of dynamics and unpredicted events. The aim is to show that the system maintains a high quality level.

Non Determinism. In self-* systems, information are distributed among agents which behave asynchronously. These characteristics normally make the resolution non deterministic. As a consequence, for one considered scenario, the solutions obtained from one execution to another may be different [Gaillard et al., 2009]. Studying the distances between those obtained solutions gives information about the stability of the system in comparison with exact approaches. Usually authors use measures of central tendency such as mean or median and variability such as standard deviation and variance to examine the non determinism of a system [Chiarandini, 2005].

Progress. Evaluating the Nominal Progress (NP) of the system refers to the study of how the system progressively reaches a complete solution. A measure of this could be the percentage of the main goal reached at each step. To do this measure it is not necessary to know the optimal solution. It is just the measure of the activity. For example, in [Clair et al., 2008], the progress is measured as the total number of finished containers at each step. During time, this can be observed by looking at the curves representing this progress as they will have specific forms (linear, exponential, logarithmic) (section 4.5). More interestingly for self-* systems, we can analyse how self-adaptation influences this progress: does it maintain or even increase the rate (which is an ideal situation), does the progress slow down (how much?), or even worse, is there a decrease in the progress (the worst being a reset to zero of the progress). Finally, we can also evaluate Self-* Progress (SP), the progress of the self-adaptation following a perturbation in the same manner as NP.

Memory Usage. Another interesting uncommon criterion is the amount of memory used by each agent for the solving part Nominal Memory (NM) and the self-adaptation part Self-* Memory (SM). This criterion has a direct influence on the computational complexity (section 4.3.1) when for example, at each step an agent must process information from all other agents or just from a local neighbourhood. The deployment of the system in real-world application is also affected by a high usage of the memory, for example in the context of ambient intelligence where small devices are usually constrained by small

memory capacities [Weber et al., 2005].

4.2.2 Homeostasis & Robustness

As stated above, the main property of self-* systems is their ability to maintain their functioning in an environment presenting a high level of dynamics. In other words, we can say that self-* systems possess robustness, and homeostasis/self-stabilization abilities. The robustness is the property of a system that is able to maintain its behaviour under perturbations [Robertson et al., 2001]. Adaptation capabilities such as the ability of the agents to treat the perturbations are means to obtain this robustness. The homeostasis ability is what enables the system to regain its normal behaviour after being perturbed. In [Würtz, 2008, chap. 5], authors defined the homeostasis as the capacity of regaining an ideal state in which the system is operating in a maximally efficient way. Studying those two abilities enables to understand in depth how the system handles the dynamic changes of the environment, and by that validate its adequacy to solve the problem. These qualities guarantee that the system will continue to function when confronted to difficulties. It can be noticed that certain situations require either strong *homeostasis* capabilities or strong *robustness* (as these are not exactly identical).

Robustness. Considering dynamics is a very interesting point in complex problem solving. Events occur at different steps and are usually unpredictable. A robust system is a system which is able to maintain a stable behaviour even under perturbations. To measure the robustness, the following points can be considered:

- The system maintains a functional adequacy even during perturbations. In other terms, the variation of the solution quality is weak under perturbations $\boxed{\Delta Q < \epsilon}$ where ΔQ is the variation of the solution and ϵ is a very small value.
- The new solution (state) reached by the system is close to the previous one.
- The amount of changes inside the system between the state when the perturbation occurs and the new stable state, is minimal.

Time for Adaptation. The time for adaptation is the time needed by the system to regain its normal behaviour after the occurrence of a perturbation. When measuring the time (section 4.2.1), ST which represents the time needed by the system to self-adapt along its execution was introduced. While the time for adaptation studied here, is the $\boxed{\text{Self-}^\star \text{ Time for Adaptation } (STA_{lc})}$ needed to take into account one type of dynamic changes. The number of steps needed by agents to return to the normal functioning is an appropriate measure of this criterion. Two levels are to be considered to measure adaptation time: *local level* and *global level*.

In the first one, the number of steps needed by each agent is examined. Measuring this time allows the designer to examine the self-adaptability of the agents. In the second

one, the self-adaptivity of the global behaviour of the system is studied by measuring the number of steps needed by the system to regain its normal behaviour (figure 4.3).

Measures for runtime evaluation

- ▷ Time for self-adaptation ST, time for solving NT, time for adaptation STA_{lc} or the mean of the time differences MTD.
- ▷ The number of messages for solving NCL and the number of messages for self-adaptation SCL.
- ▷ The quality of the solution ($\delta(S_{cur}, S_{opt})$) and the degree of non determinism of the resolution.
- ▷ The progress of the resolution NP and the progress of the self-adaptation SP.
- ▷ The memory usage rate for the nominal behaviour NM and the self-$\star$ behaviour SM.
- ▷ The robustness level of the system ($\Delta Q < \epsilon$).

4.3 Intrinsic Characterization of the System

The main limit of the evaluation criteria presented in the previous section is that they can only be measured during or after the execution of the system. Since this can only be done on a limited number of scenarii or situations, these evaluations cannot guaranty the qualities of the system in every real-world situation and on the long run.

This section completes the previous one by defining, discussing and presenting as evaluation criteria several characteristics of self-$\star$ systems which can be used without having to actually run the systems. Given the knowledge of the nature of a system, of its functioning, its computational complexity, the **decentralization**, the **requirements of the local algorithm** or the **influence of the number of agents** and their **computation needs** can be partially studied. This enables to already evaluate and compare such systems.

4.3.1 Computational Complexity

Very early in Computer Science [Fortnow and Homer, 2003][6], the need arose to be able to measure the time or memory required by a program or an algorithm. Basically, by looking at a classic algorithm, one can ascertain how the execution would fare in a worst-case scenario and by expanding the set of input data to a real-world problem, one can predict that the algorithm would fail to give a response in a reasonable time. This problem of *scalability* has been thoroughly studied in the last 40 years and a formal theoretical framework has been given [Wilf, 1986][7]. With it comes the well-known but difficult to completely grasp notions

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

of *orders of complexity* and *NP-hardness*. The main idea to keep in mind is that we currently have problems which we do not know how to solve in a polynomial time such as scheduling problems or routing problems.

The general theory is quite out of the scope of this work but the question remains when evaluating a self-* system: can we know how efficient it is by looking at the code? If the system attains its self-* properties *via* a global control algorithm, it perfectly fits the previous framework and can thus be studied and evaluated by its tools. For evaluation purposes, this is an interesting advantage, but the major drawback is that interesting problems combine with exponential complexity thus rendering these systems basically non-functional.

4.3.2 Decentralisation and Local Algorithms

In our experience, nearly all systems exhibiting self-* properties on real-world complexity scale problems rely in one way or another on distribution of the computation, decentralization of the control and *local* mechanisms to explore the search space. When one local, even simply calculated (in terms of complexity) change can have non-linear repercussion throughout the whole system, how can we calculate the computational complexity of the whole system? There exists work [Papadimitriou et al., 1990] on complexity theories for local search algorithms, with for instance the definition of the *Polynomial Time Local Search (PLS)* class, but they fix very strict boundaries.

Local Computational Complexity, Decentralization, Action Locality, Initial and Acquired Knowledge and **Agent Number Influence** are criteria to study to validate the decentralisation level and the efficiency of local algorithms.

Local Computational Complexity. Pending on a breakthrough in Theory of Complexity, we are thus reduced to analyse the local algorithm by itself. This is already a criterion to disqualify certain local algorithms which require too much computational power.

Decentralization. We can complete the study of the Local Computational Complexity criteria with an evaluation of some sort of ratio between the decentralization and the amount of local computation needed. If the system has limited decentralization, it may fall in the *NP-complete* problem class. If the decentralization is important, but the local computation complex, then each local decision may be *NP-complete*. In short, how many, how decentralized, how simple are the agents in the system and how many other agents does it need to know or interact with?

Bear in mind that this criterion cannot be used alone as the least complex local algorithm would be for instance to always choose a random action for an agent, but would certainly produce one of the worst systems ever. On the other hand, it can be usefully applied to filter local algorithms growing too much in complexity. For instance, an algorithm where an agent has to check all the properties of all the other agents of the system, and with a prediction for n steps ahead depending on its chosen action, is a very naive approach.

Action Locality. In the same way, we can also add our understanding of the

repercussions of a local action on the rest of the system. Since self-* systems are complex systems, they are prone to the well known *butterfly effect* as we can observe in systems which use *gossip* mechanisms or propagation in neural networks [Jelasity and Babaoglu, 2006]. But we can already evaluate how probable this will be by studying the strategies of the agents in regard to this. For instance, does an agent systematically broadcast its informations and needs? Or does it proceed by selecting its targets based on an evaluation of their relevance? Another example would be the *minconflict* [Minton et al., 1992] types of heuristics which try to progress in the solving by minimizing the negative impact inside a neighbourhood, and by consequence, reduce the overall complexity.

Initial and Acquired Knowledge. Some algorithms require specific or extensive knowledge to be provided to the system before it can even start. For instance, in many meta-heuristics approaches like *Ant Colony Optimization (ACO)* or *Particle Swarm Optimization (PSO)*, a preliminary analysis of the solution space is required to set up parameters like the number of self-* agents or the probability distribution characterizing their behaviours [Dréo et al., 2005]. In the same way, some algorithms require this during their activity to be able to be efficient. This can be evaluated and used as a criterion as this kind of systems are more difficult to deploy.

Agent Number Influence. The number of agents in a system can drastically increase or decrease the efficiency of the system. This is an important factor to ascertain the optimal organization in high cost problems, for instance when deploying satellite swarms [Bonnet and Tessier, 2008]. More importantly for self-* systems, it enables to judge the scalability of a system as it evolves during its life. This can be of course evaluated during runtime, but also by analysing how the system is built: does it facilitate the introduction of new agents? Can agents easily delegate? Are extensive self-organizing abilities present? etc.

Local Complexity and Decentralization Criteria

- ▷ Calculate the computational complexity of the local decision process.
- ▷ Evaluate how decentralized the system is.
- ▷ Evaluate how much influence the action of one agent potentially has on the rest of the system.
- ▷ Evaluate the difficulty to provided required initial knowledge to the system or the acquisition of it during execution.
- ▷ Analyse the scalability of the system.

4.4 Development Methodologies Characterization

Another perspective to analyse before developing a given distributed self-* system is the development methodology, i.e. the process to follow and the concepts to manipulate

during the development phases (analysis, design, implementation, etc.). In fact, all the approaches one can consider to develop a self-* system are not equal from this point of view. However, even if it is possible to try to define criteria concerning this perspective, highlighted points are more qualitative than quantitative. Some analysis on how a self-* systems has been developed, are useful for next developments. Different studies has been realized for classical methodologies. Our aim is to underline the criteria dedicated to the self-* properties such as: **identification of autonomous agents, distribution, genericity and flexibility**.

Analysis and Identification of Autonomous Agents. During the analysis of a problem to solve, many factors may make the task tough. In some cases the development is straightforward while in some others it asks for some expert knowledge. One difficult task for the development of self-* system is agents identification. In some methodologies, problem specifications lead in a straightforward manner to this identification. While in others, some analysis phases must be done. For example, in problem solving (using the *Constraint Optimization Problem (COP)* framework), developing a *Distributed Constraint Optimization Problem (DCOP)* solver like *Distributed Breakout Algorithm (DBA)* [Hirayama and Yokoo, 2005], *Environment, Reactive rules, and Agents (ERA)* [Liu et al., 2002] or approaches like *ACO* [Dorigo and Stützle, 2004] or *PSO* [Kennedy and Eberhart, 2001], requires transforming the problem to solve into the constraint satisfaction problem framework. This can be a difficult task especially considering that choices made during this phase may strongly impact on the performances of the system and some parameters requiring expertise knowledge must be fixed.

Another kind of approach may not directly match a problem into a given formal or semi-formal framework. For instance, the *Adaptive Multi-Agent System (AMAS)* theory [Picard and Glize, 2006] focuses much on the properties or behaviour an agent may have (e.g. cooperativeness) and therefore the designer will have to find what are the agents within the problem that can be agentified as to implement a self-* solver. This tack can also be difficult in this kind of approach and a good methodology has to guide the engineer for this identification.

Distribution and Deployment Ease. All the developed systems are not easily distributable depending on the paradigms they rely on such as communication. We distinguish between two types of communication: stigmergic ones and direct ones.

Stigmergic and swarming self-* approaches, like *ACO*, require the development or the use of specific environmental frameworks, such in *tuple spaces middleware* like the SwarmLinda [Charles et al., 2004] framework or the *Tuples On The Air (TOTA)* middleware [Mamei and Zambonelli, 2005] that can support the pheromone deposits, updates and monitoring.

Direct communication approaches require dedicated frameworks like *Java Agent DEvelopment Framework (JADE)*[8] to be deployed. This also requires to develop the entities as agents and programming them using dedicated libraries.

[8] `http://jade.tilab.com/`

Additional effort must be done to deploy such systems. The evaluation of these efforts underline the fact that the methodology is easily used by any designer or requires specific knowledge.

Genericity. Self-* approaches may also differ concerning their level of genericity, i.e. the provided effort to adapt a method to a given problem. For instance, *DCOP* approaches like *DBA* or *ACO* are completely generic, in the sense that they provide distributed solvers for problems as soon as they are expressed as *COP*. However they still require some adjustment by setting parameters, like the time limit for *DBA*, the probability distribution of behaviours for *ERA*, or the entire set of pheromones and collective parameters for *ACO*. This criteria enables the disqualification of ad-hoc methodologies (designed for a specific problem) compared to generic methodologies used for different types of problem.

Development Methodology Evaluation Criteria

- How much effort is required to adapt the general concepts of the method to a specific problem?
- How much effort is required to distribute a problem among a collective of self-* entities?
- How much effort is required to deploy the developed system?
- How large is the scope of problems for which the method is relevant or applicable?
- How difficult is it to add new constraints and problems to an existing system?

4.5 Comparative Evaluation

The characterization and evaluation of a system provide quantitative and qualitative values which enable to judge it and compare it to other systems in order to highlight its strengths and its weaknesses. This section proposes means for comparing different systems and a panel of typical behaviours associated to their self-adaptation process.

Evaluation of performance enables to show that a system is better or worse than another. The criteria to compare two systems can be a subset chosen among the previous presented measures (sections 4.2, 4.3, 4.4). Usually, systems behave as seen in figure 4.1 where one system is better for a criterion but not for another one.

Those criteria are highly inter-independent and a multi-objective view is more appropriate. This multi-objective view cannot be reduced into an unique dimension combining all these criteria because they are very context sensitive, depending for instance on the application domain or initial requirements. Consequently, a *radar view* is proposed as a relevant way to judge the functional adequacy of a self-* system. An "ideal system" is on the radar center because it outperforms the others in all the dimensions.

Still, each criterion, when analysed in detail, can bring valuable information for

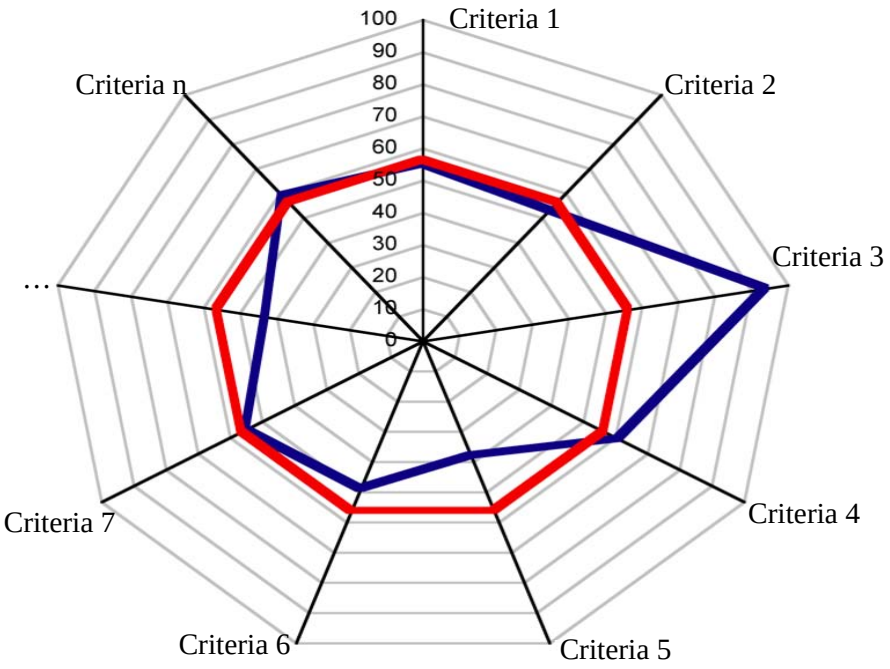

Figure 4.1 — A radar chart to compare self-⋆ systems

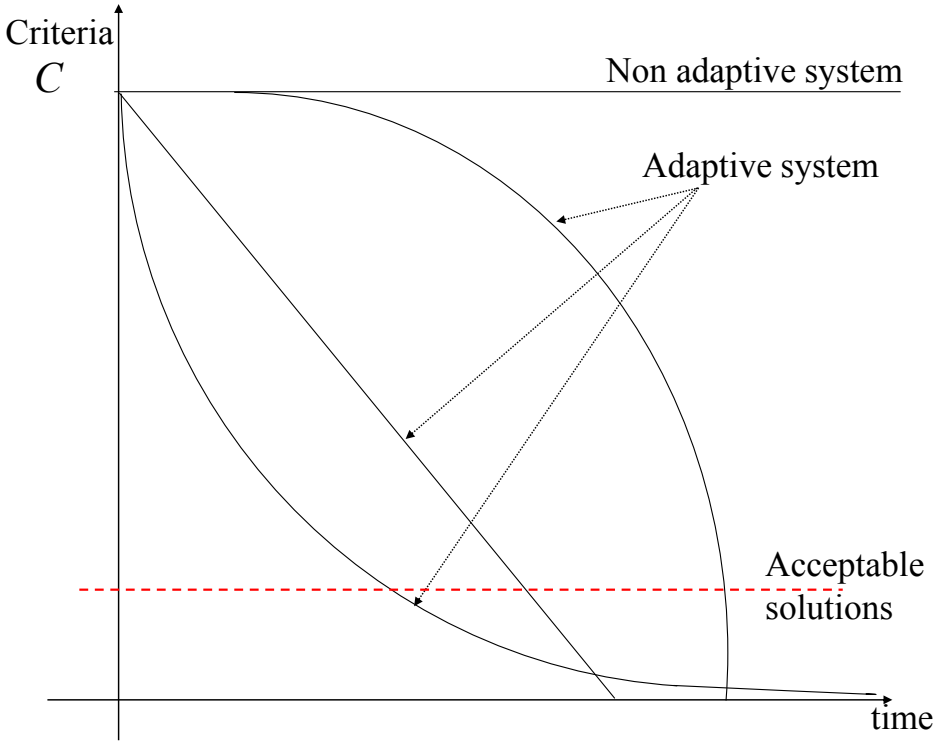

Figure 4.2 — How does the system reach a solution when confronted to a new problem?

comparing self-⋆ systems. In the following, several illustrative curves representing a comparison between different systems are given. Each curve can be seen as representing an archetypal behaviour for a given criterion or another.

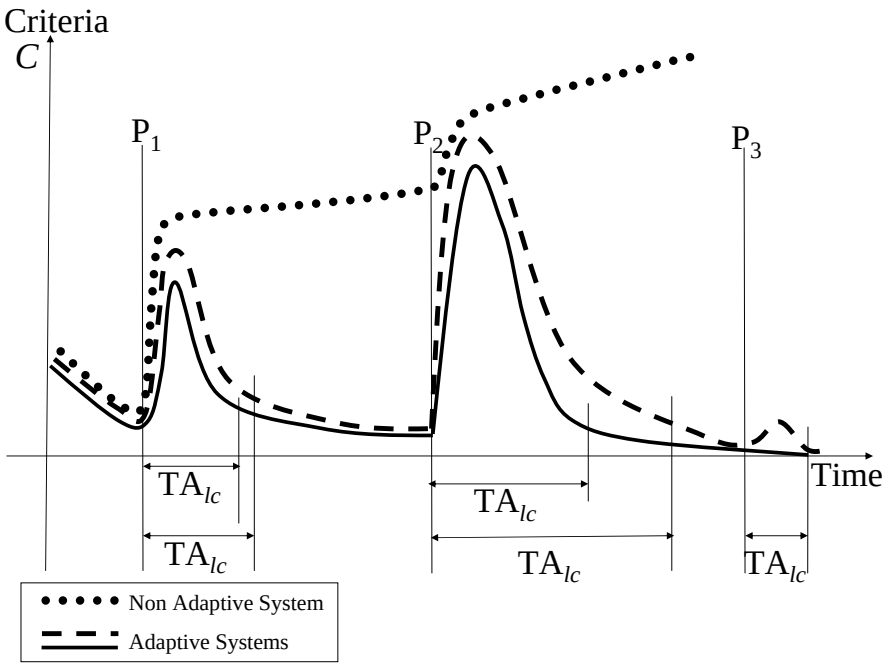

Figure 4.3 — How does the system adapt to change?

A first criterion that can be studied is the dynamic of adaptation of the system when a perturbation occurs during time, $C{=}\delta(S_{cur}, S_{opt})$ (section 4.2.2). A distinction is done between the **non-adaptive** systems and the **self-adaptive** systems. The former is not able to adapt its behaviour under dynamic changes and by that cannot improve its solution. For the latter, three types of response are differentiated as seen on figure 4.2:

- ▷ **diagonal line** when the system improves its solution in proportion to time.
- ▷ **lower curve** which is usually interesting for critical real-world applications as the system is able to rapidly adapt itself and proposes acceptable solution rapidly.
- ▷ **upper curve** which represents a system showing a certain inertia to self-adapt and starting to propose acceptable solution late.

In the same manner, this figure can be used to illustrate the response type of the system to one dynamic change (the event occurs at time $t = 0$).

Figure 4.3 is used to illustrate the occurrence of different types of perturbations separately. If we consider the quality of solutions as *Criterion C*, we can see that at the beginning of the curves, the three represented systems improve their solutions. When the first perturbation P_1 occurs, the distance to the optimal solution greatly increases for all of them. We can clearly see that a system with no adaptation then worsens its quality more and more with each new perturbation. An adaptive system recovers from the event.

Another information we can read on these curves, is the **time for adaptation** STA_{lc} (section 4.2.2). A self-adaptive system is able to regain an optimal state more or less quickly (see full line and dashed line on the figure).

Another criterion that we can consider using the same figure, is the **robustness** of the system. For example here, the system represented by the full curve is more robust than the one represented by the dashed curve, as its reaction to perturbation is less important especially for perturbation P_3. Moreover, this can also illustrate that some adaptive systems are able to *self-learn* from their adaptation to a type of perturbations so that whenever such a perturbation happens again, the system is ready to respond efficiently.

Finally, the **number of messages** SCL used for adaptation can also be studied in this same figure. We can see that a non-adaptive system maintains the same number of messages when perturbations occur (In figure 4.3, it is equal to zero and it is represented by the x-axis), while the self-adaptive systems increase SCL when perturbations occur and decrease it after adapting their behaviour. The fact that the system represented by the full curve needs less messages than the one represented by the dashed curve could be explained by the decentralization degree in the system or the efficiency of the interaction mechanisms (section 4.3.2).

4.6 Main Difference between Self-* and Classical Systems

An important limitation when trying to ascertain that a self-* system behaves well, comes from its main difference with classic software: its ability to change, adapt, evolve. Both the computational complexity and decentralization might completely change when a system is subject to important modifications in its structure, the way it works, its communication and so on. The only objective measure would be to freeze the system or make a snapshot and calculate its computational complexity and distribution at that moment. Furthermore, by regularly taking these snapshots, this can become another criterion: does the system grow like a cancer or is it fully functional? Can the system simplify itself when needed?

But this ability is potentially also their greatest strength. Where a problem would need a higher complexity class algorithm, the self-* system can cut through the search space to reach a specific state of itself. Ideally, this would represent an optimal system for given criteria. This is for instance the case when designing self-adaptive applications which can autonomously build relevant models of a complex systems (natural phenomena, human activity) [Georgé et al., 2009].

4.7 Conclusion

In this chapter, a set of comparison criteria for self-* systems has been presented. I propose to observe and analyse them from three points of view:

1. Evaluate the system at runtime by studying its behaviours when subject to endogenous and exogenous dynamics,

2. Evaluate the intrinsic properties of a system by focusing on the decentralization and the locality properties of the system, from its specifications,

3. Evaluate the methodology to develop the system from the modelling to the deployment phases.

Once collected, these criteria must be analysed in global viewpoint. However, aggregating such disparate criteria (quantitative or qualitative) is not an easy task since they are highly inter-independent and a multi-objective view is more appropriate. This point is discussed by illustrating some criteria with theoretical charts.

This work is a first step for evaluating self-* systems and highlighting the consequences of their specific behaviours devoted to their adaptation which is missing in classical systems. Nevertheless, formal validation is fundamental to enable the promotion of these systems in industries. Currently, as it is impossible to prove all the behaviours of these systems, partial proofs can be done and a promising lead is mixing simulation and formal classical proofs.

This work [Kaddoum et al., 2009] is currently being improved in collaboration with Claudia Raibulet from the Università degli Studi di Milano-Bicocca [Kaddoum et al., 2010]. In this work, the criteria for the evaluation of self-* systems in general have been addressed including architectural criteria and underlining how adding adaptivity can influence the nominal part of a system.

5 Application, Experimentation & Validation

« Anyone who has never made a mistake has never tried anything new. »
Albert Einstein

Contents

The chapter in english starts page 107.

Résumé général du chapitre

Lors des précédents chapitres, nous avons défini le modèle d'agent AMAS for Optimisation (AMAS4Opt) pour la résolution de problèmes d'optimisation sous contraintes ainsi qu'un ensemble de critères pour l'évaluation des systèmes adaptatifs. Afin de valider ces contributions, deux applications clés d'optimisation connues pour leur complexité ont été choisies: ***le contrôle manufacturier*** *et* ***la conception de produits complexes****.*

Dans notre étude, le contrôle manufacturier concerne la planification des tâches dans une usine. En d'autres termes, nous nous intéressons à la gestion des ressources de production afin de mieux répondre aux besoins des clients (respect des délais) tout en respectant un ensemble de contraintes telles que la disponibilité des ressources et l'ordre des traitements. Ainsi, les commandes des clients sont représentées par des containers. Chaque container se décompose en une liste ordonnée de traitements devant être traités par des stations de travail adéquates de l'usine. Cette application se caractérise par la combinatoire élevée produisant un espace de recherche de taille considérable ainsi que la nécessité de la prise en compte en temps réel d'évènements dynamiques tels que les pannes des stations de travail ou l'arrivée de nouvelles commandes dans le système.

Notre deuxième application concerne la conception de produits complexes. D'une manière générale, la conception de produits complexes est une application multi-disciplinaire et multi-objectif où un ensemble de disciplines, reliées par des interdépendances non-linaires, doivent travailler ensemble afin de concevoir un produit tout en satisfaisant leurs objectifs. Par exemple, concevoir une voiture implique des ingénieurs provenant de divers domaines tels que la mécanique, l'électronique, l'informatique ou même l'écologie. Vouloir une voiture puissante implique un moteur plus puissant donc une consommation plus importante mais respectant des contraintes écologiques. Atteindre un équilibre entre ces différentes disciplines n'est pas toujours une tâche évidente. Jusqu'à maintenant, les différentes études dans ce domaine se sont intéressées à la modélisation des disciplines et leurs interactions. Dans ce travail, nous proposons de traiter ce problème en considérant un autre point de vue. En effet, si l'on considère un domaine de conception bien déterminé, nous constatons qu'un ensemble de produits testés et validés existe. Chacun de ces produits est décrit par un ensemble de caractéristiques déjà définies. C'est en se basant sur cet ensemble de produits existants que nous proposons la conception de nouveaux produits pour lesquels uniquement un sous-ensemble de caractéristiques est défini. La complexité de cette application est due à la diversité et au volume des données manipulées ainsi qu'à l'existence d'interdépendances non connues et non linéaires entre les caractéristiques du problème.

Dans ce chapitre, nous présentons ces deux problèmes ainsi que l'initialisation du modèle d'agent sur chacun des deux systèmes multi-agents: ***Self-Adaptive Flexible scheduling (SAFlex)*** *pour la gestion du contrôle manufacturier et* ***Self-Adaptive Population Based Reasoning (SAPBR)*** *pour la conception de produits complexes. Les performances des deux systèmes sont étudiées en utilisant les différents critères d'évaluation proposés dans le chapitre 4. De plus, SAFlex a été comparé à deux approches basées sur l'apprentissage par renforcement.*

Les résultats obtenus sont encourageants tant au niveau du modèle défini qu'au niveau des systèmes développés.

En effet, les agents du modèle AMAS4Opt ont été conçus de façon à être le plus proche possible du fonctionnement des entités réelles du problème. Ceci a permis de partir de la description naturelle des problèmes posés et en déduire les agents des systèmes, leur comportement ainsi que les règles coopératives guidant leur comportement. De ce fait, la généricité du modèle AMAS4Opt par son adaptation à deux applications présentant des caractéristiques différentes et son aspect intuitif par la représentation des entités du domaine ont été validés.

Concernant les systèmes multi-agents développés, les résultats obtenus confirment l'adéquation de la théorie des ***Adaptive Multi-Agent System (AMAS)*** *pour la résolution de ce type de problèmes présentant des caractéristiques variées.*

5.1 Introduction

In order to validate the *AMAS4Opt* agent model that was developed and the evaluation criteria that were defined, two applications known for their complexity have been chosen. In the first one, the *Manufacturing Control Scheduling Problem*, besides the combinatorial complexity, is of interest due to the need to manage dynamics such as station breakdowns or the arrival of new containers. In the second one, *Design of Complex Products*, the complexity is due to the volume of manipulated data that is produced by unknown (or at least not precisely known) and often complex interrelated non-linear functions. Those two applications have been chosen because of their very different characteristics so as to verify that the agent model is sufficiently generic.

In this chapter, both problems are described and formalized. Then, the adaptive multi-agent systems instantiating the generic agent model for solving the problems are presented. Each multi-agent system is then evaluated accordingly to several of the previously defined evaluation criteria. A comparison with two other approaches is also provided for the *Manufacturing Control Scheduling Problem*.

The chapter is organised as follows:

- section 5.2 is dedicated to the *Manufacturing Control Scheduling Problem* application;
- in section 5.3 the *Complex Problem Design* application is presented;
- section 5.4 presents an evaluation of the *AMAS4Opt* agent model;
- section 5.5 concludes the chapter by discussing the perspectives for both applications and underlines the advantages of the agent model that was defined.

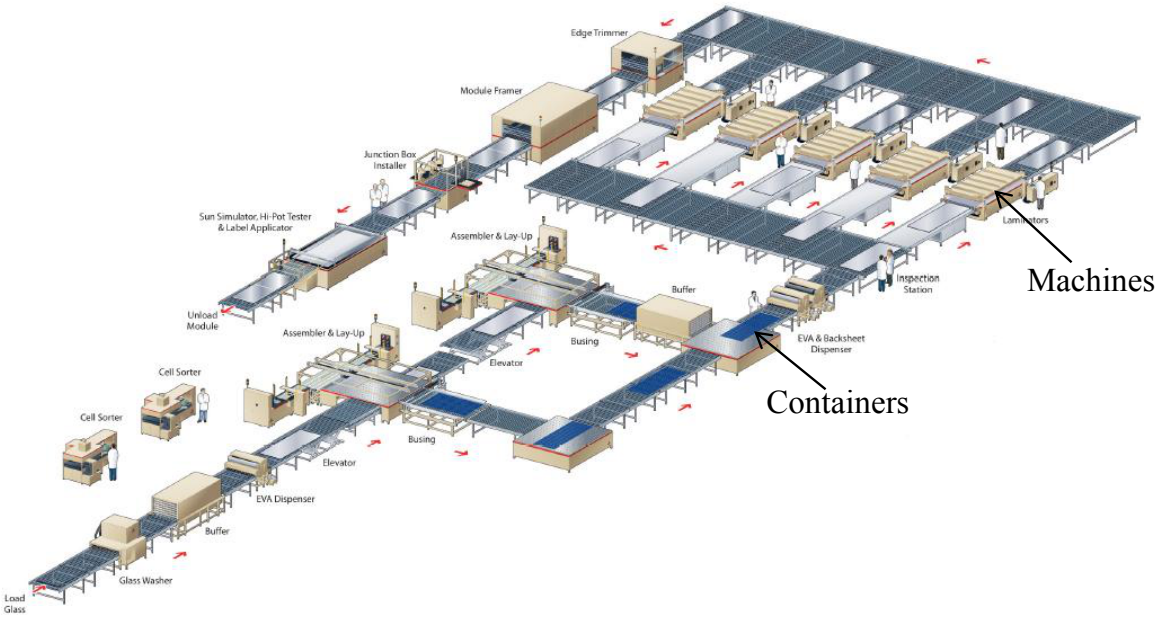

Figure 5.1 — A factory representation

5.2 Manufacturing Control Scheduling Problem

Production planning and control is concerned with the application of scientific methodologies to problems faced by production management, where materials and/or other items, flowing within a manufacturing system, are combined and converted in an organised manner to add value in accordance with the production control policies of management [Wu, 1994]. Production planning and control can be decomposed into several elements. Our study concerns one of them, the scheduling or production activity control, defined by [Duggan and Browne, 1991] as *Principles and techniques used by the management to plan in the short term, control and evaluate the production activities of the manufacturing organisation*. To sum up, the manufacturing control scheduling problem is a mean to plan the use of resources for production by respecting predefined constraints (delivery delays, operators schedule, etc.). Thus, it concerns the scheduling of the operations of different containers representing the client orders on the factory stations (figure 5.1).

Unexpected events occur in most of the real-world manufacturing systems and lead to a new type of scheduling problems called *Dynamic Flexible Job Shop Problems (DFJSPs)*. In such problems, there is a theoretically infinite set of containers representing client's orders that arrive continually over time and must be treated respecting delays. In addition, other events can occur such as station failures, processing time changes, etc. Different solving methods have been developed. They are categorized in three main categories: **off-line scheduling, rolling-time scheduling** and **completely real-time scheduling** [Ouelhadj and Petrovic, 2009].

The **off-line scheduling** methods are solving strategies where everything is known in advance. They build robust predictive scheduling based on predictability measures. In these methods, whenever a non-considered real-time event occurs, a complete rescheduling

is necessary, generating a new schedule from scratch to be used henceforth. In principle, such methods are better to maintain optimal solutions but are rarely achievable in practice especially in environments with high levels of dynamics where the determination of predictability measures is difficult.

Rolling-time scheduling techniques are methods that generate a first schedule given the currently known information and adjust it to maintain shop efficiency and performances during execution. The adjustment techniques can be periodic, event driven or hybrid. In the periodic adjustment techniques, schedules are repaired at regular intervals. While in the event driven techniques, schedules are repaired whenever an unexpected real-time event occurs. Such methods usually perform well as they build complete schedules using the optimization of global objective functions, but in highly dynamic environments, they require additional CPU time as they have to often repeat the rescheduling procedure.

In the **completely real-time scheduling**, the schedule is constructed step by step and decisions are made locally in real-time. Nearly all the solving methods of this category are based on decentralized and distributed multi-agent systems where agents represent manufacturing entities, have the ability to generate their local schedules, react locally to real-time events and interact together to generate global robust schedules. As decisions are made locally, global optimality is not guaranteed but the obtained systems are robust and can naturally adapt to real-time events.

In this first part of the chapter, the *SAFlex*, a completely real-time scheduling technique developed in this PhD is detailed. *SAFlex* is an adaptive multi-agent system based on the *AMAS* Theory constructed using the agent model defined in chapter 3. The presentation is organized as follows. First, the *DFJSP* (section 5.2.1) is formalised. Next, the designed *Multi-Agent System (MAS)*, the agents and their behaviour are described (section 5.2.2) and the obtained results (section 5.2.3) regarding self-adaptivity are discussed. Then, the comparison established between *SAFlex* and two learning methods during my 3-months visit to the Vakgroep Informatiechnologie, Departement Industrieel Ingenieur at KaHo Sint-Lieven, Gent, Belgium, is presented (section 5.2.4).

5.2.1 Dynamic Flexible Job Shop Problem

The *DFJSP* is described by the following hypothesis:

- A set of stations $S = S_1; S_2; ...; S_m$, each being able to process a set of operations.
- The operation sets on different stations may be different, and the processing time of each operation is station-dependent.
- Pre-emption is not allowed (an operation being processed cannot be interrupted to be replaced by another) and each station can perform only one operation at a time.
- A set of independent containers $C = C_1; C_2; ...; C_n$.
- Each container is decomposed in a sequence of ordered operations $O_{1;j}; O_{2;j}; ...; O_{l;j}$ (where l is the number of operations for C_j).

- Operations of the same container have to respect the finish-start precedence constraints given by the operation sequence.
- Containers are released at different arrival times and their treatments must be ended before a given due date.
- During processing, there are some unpredictable events such as station breakdowns, station repair, due date changes and changing of the processing time.

The *DFJSP* consists in scheduling each operation on an appropriate station, and to sequence the operations on adequate stations respecting the finish-start precedence constraints and the delays. We denote d_j the *due date* of container C_j, $s_{i;j}$ the *start time* of an operation $O_{i;j}$ and $cT_{i;j}$ its *completion time*. The completion time CT_j of a container C_j is equal to the completion time of its latest operation $O_{l;j}$. The *tardiness* of a container C_j is $T_j = \max(CT_j - d_j; 0)$. If $T_j > 0$, then the container C_j is tardy and the *tardiness state* $ST_j = 1$ else $ST_j = 0$. The obtained schedule must minimize:

- $T_n = \sum_{j=1}^{n} ST_j$: the number of tardy containers;
- $\overline{T} = \frac{1}{n} \sum_{j=1}^{n} T_j$: mean Tardiness;
- $T_{max} = \max\{T_j | 1 \leq j \leq n\}$: maximum Tardiness;
- $CT_{max} = \max\{CT_j | 1 \leq j \leq n\}$: the makespan or the completion time of the last container that leaves the system;

For simulation purpose, we considered that time is divided into steps called *fabrication cycles*. Each fabrication cycle corresponds to a certain time t.

5.2.2 The Adaptive Multi-Agent System

SAFlex is a self-adaptive multi-agent system inspired by the *Distributed AMAS Constraint Optimisation Problem (DAMASCOP)* approach [Clair et al., 2008] designed during my master 2 research. *SAFlex* has been designed using the agent model presented in chapter 3.

Given the problem description, two types of cooperative agents have been developed: *Container* agents and *Station* agents. The Container agents represent the containers entering the system and must schedule their operations on the different stations in the factory. If no containers enter the system, the stations can stay idle and no schedule is to be established. Thus, it is the Container agents that possess the constraints of the problem and that must interact with the Station agents in order to construct the schedule and solve the problem. Given the defined agent model, Container agents possess the *constrained role* and Station agents possess the *service role*. The behaviours of both agents are described in the following sections.

5.2.2.1 Agent Interactions & Communications

In this system, agents communicate using messages. Table 5.1 summarizes the different messages sent by each type of agent. As those messages are an instantiation of the agents interactions defined in section 3.3, they are divided into the same four categories presented in that section. We have changed the names of the functions so that they correspond to the treated application.

Agents	Request for service	Request for Information	Answer to requests	Information
Container	*askToUse()*	*askForNeighbourhood()* *askForProcessingTime()* *askForFutureUsage()*		*cancelFutureUse()* *cancelUse()*
Station			*usageAccepted()* *usageRejected()* *futureUsageAccepted()* *futureUsageRejected()* *neighbourhood()* *processingTime()*	*newStationCriticality()* *stationInPerturbation()*

Table 5.1 — Messages in *SAFlex*

The *askToUse()* message is used by the Container agent to ask the immediate use of the station that can accept (*usageAccepted()*) or refuse (*usageRejected()*).

The *askForFutureUse()* is used to inform the station about a future reservation of the station that can also accept (*futureUseAccepted()*) or refuse (*futureUseRejected()*).

The Container agent can cancel an immediate reservation or a future use notification using *cancelUse()* and *cancelFutureUse()* messages.

The Container agent can ask a given station for its neighbourhood or its processing time for a specific operation using *askForNeighbourhood()* or *askForProcessingTime()* messages. The station answers to these requests using *neighbourhood()* and *processingTime()* messages.

The Station agent can inform Container agents requesting its service or being treated by it about its current state if it is in perturbation (*stationInPerturbation()*) or about updated information concerning the higher criticality level of the received requests (*newStationCriticality()*).

5.2.2.2 Agent Criticality

An important notion when defining agents using our model is the *agent criticality*. As defined in section 3.4, when requesting a service, agents must compute their criticality and associate it to their requests. In the manufacturing control scheduling problem as defined above, the Container agents are the agents that request the services of Station agents. Thus, they are the agents that compute their criticality degree which must reflect their difficulties to be treated by an adequate station.

Four values define the criticality of a Container agent:

1. The **station optimality rank**: this first value indicates the preference the Container agent attributes to each qualified station it has in its local representations. As the processing time of each operation is station dependent, the Container agent will order

the different known and qualified Station agents from the lower processing time to the higher processing time in an *optimality ranks list*. This list is then used by the Container agent as the order in which it will send its requests. For the Station agent, this value is interpreted as the difficulty the Container agent has to get accepted by its preferred station. Thus, the higher this value is, the higher the criticality of the Container agent is, as it means that the agent cannot be accepted at a better ranking station.

2. The **prejudice**: when requesting the service of a Station agent, the Container agent knows that other qualified stations can exist. The prejudice value is used by the Container agent to inform the requested Station agent about the prejudice that it may cause if it does not accept to treat the operation of the Container agent. This value is computed using the residual time considering the processing time of the next Station agent in the *optimality ranks list*. As the processing time of the next station is usually higher than the current one, this value indicates the loss of time caused to the Container agent if it is not accepted by its preferred Station agent. The Station agent will put a higher criticality on the Container agent to which it causes the higher prejudice.

3. The **residual time**: this value indicates the time left (remainingProcessingTime) to the Container agent to treat its operations before its dead-line. As the processing time of each operation is station dependent, Container agents are provided with the highest known processing time for each operation. When computing the residual time of the current operation the Container agent uses the effective processing time of the requested station. The less residual time a Container agent has the more it is critical.

Residual time = (deadLine - releaseDate) - remainingProcessingTime

4. The **number of qualified stations** known by the Container agent: this last value indicates the number of qualified stations known by the agent. Thus, a Container agent knowing several qualified stations is less critical than an agent knowing only one qualified Station.

When receiving several usage requests, the Station agent orders them beginning from the first value and in case of equality, the next value is considered.

5.2.2.3 Data Types

Different *dataTypes* manipulated by both agents have been defined: **Acquaintance**, **Operation**, **Work**, **Criticality** and **Information**.

1. **Acquaintance**: an acquaintance is another known station that is physically close to the current station. For a Station agent, the list of acquaintances defines its *neighbourhood*. The Station agent sends this *neighbourhood* to a Container agent when rejecting a request for service (*usageRejected(), futureUsageRejected()*) or when asked for its neighbourhood (*neighbourhood()*). The Container agent registers in these structures the different *explored stations*.

2. **Operation**: for the Station agent, it defines the processing time of each operation for which the station is qualified. When released, the Container agent is provided with the maximum known processing time for each operation it needs. This structure is updated by the Container agent after it receives, from the Station agent, the processing time of a given operation (*processingTime()*).

3. **Work**: it is used by agents to make, accept or reject requests. It contains the Container agent requesting the service, the requested Station agent, the Operation needed by the Container agent with the real processing time of the requested station, its criticality, the expected time to begin its treatment[1] and the current position of the Container agent. This last information is updated accordingly to the position of the station where the treatments of the Container agent are processed.

4. **Criticality**: contains the four values defined in section 5.2.2.2.

5. **Information**: this structure is used by the Container agent to store information judged useful for its reasoning concerning Station agents. Such information is sent by the Station agent when rejecting the service request received from a Container agent. It contains:

   - ▷ the Station agent concerned by the registered information;
   - ▷ the possible beginning time of the next processing operation that can be accepted by the Station agent[2];
   - ▷ the maximum criticality of the requests perceived by the Station agent. This information is used by the Container agent when trying to renew its request to this station. If its criticality is less than the received value, it does not renew its requests. The Station agent updates this information (*newStationCriticality()*) whenever a change is detected.

5.2.2.4 Container Agent

The Container agent (CA_j) represents a given container C_j and has to explore the factory searching for qualified stations for its operations. As containers arrive at different times, Container agents are created when the corresponding container is released in the factory. At the beginning, each CA_j is provided with the address of a station chosen randomly. This station can be qualified or not for its first operation.

Figure 5.2 describes the skills, characteristics, representations and the interactions of the CA_j with other agents of the system and its environment. The description is provided using *AMAS Modelling Language (AMAS-ML)*.

The behaviour of the CA_j is described considering its *Status*(figure 5.3): AVAILABLE, IN TREATMENT and IN PERTURBATION.

[1] This time is equal to 0 for an immediate use request (*asktoUse()*) or to a given computed time by the Container agent for a future usage request (*askForFutureUsage()*).

[2] In case the Station agent is *Booked* by a Container agent or *in Treatment*, this value corresponds to the end of its current treatment. In case it is *in Perturbation*, the Station agent does not provide this information.

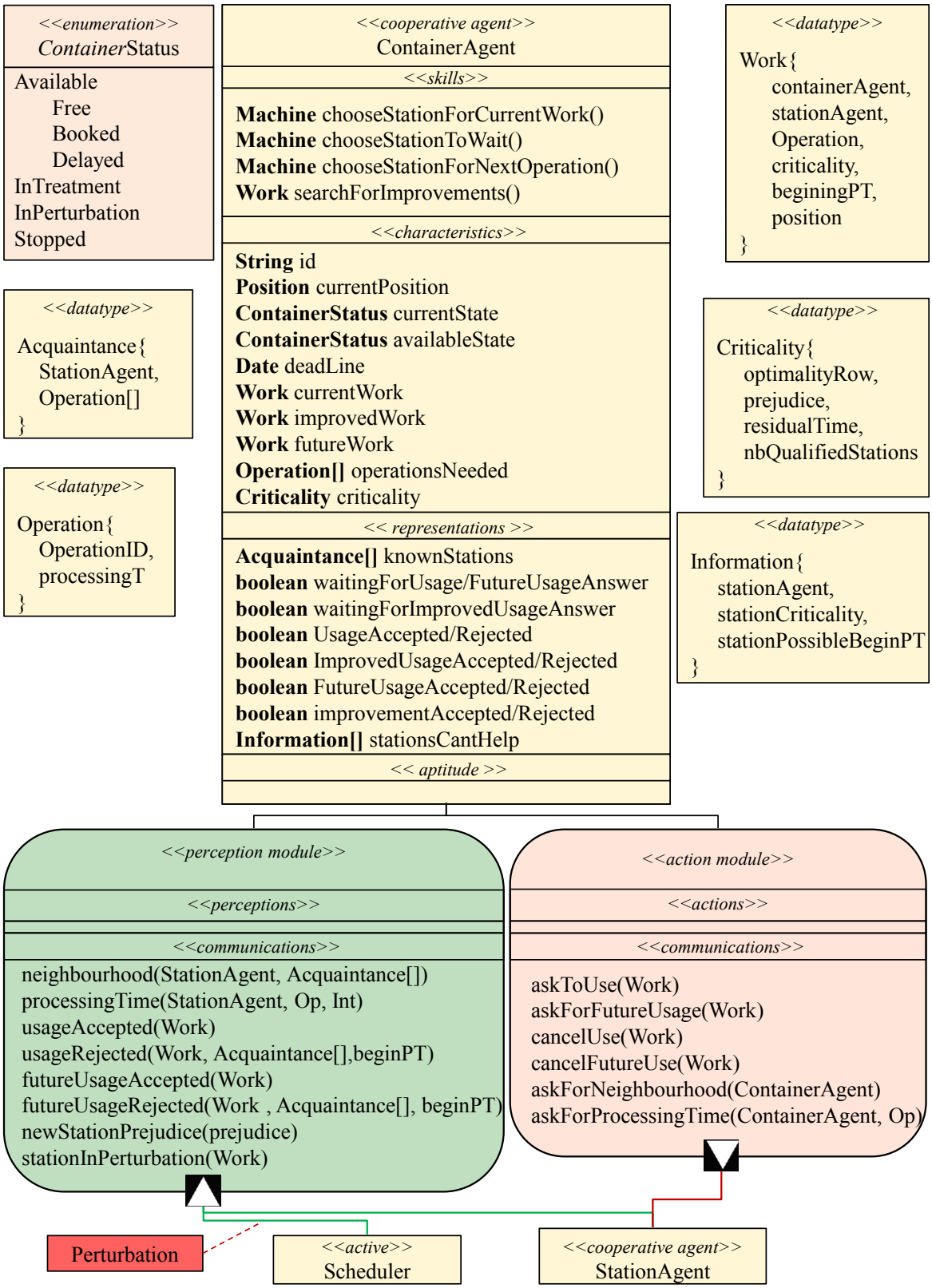

Figure 5.2 — Container agent description using *AMAS-ML*

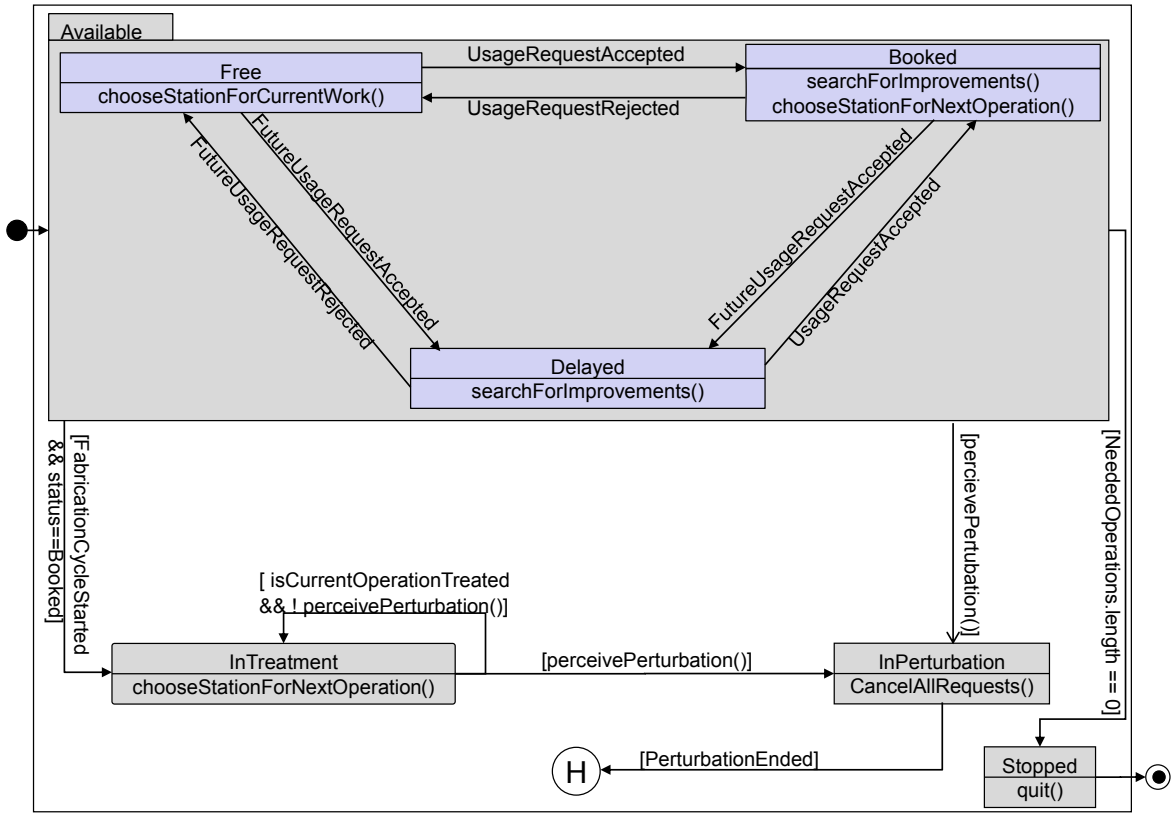

Figure 5.3 — Container agent states

AVAILABLE. This status corresponds to the exploration phase of the CA_j. Three different sub-statuses are distinguished: *Free, Booked* and *Delayed*.

In the *Free* state, the CA_j seeks a station for its current operation. For that, it selects from its knowledge the qualified stations and sends them *processing time* requests (*askForProcessingTime()*). After receiving the stations responses (*processingTime()*), the CA_j computes its *optimality ranks list* and chooses the most appropriate station depending on the lower possible processing time. If several stations have the same processing time it chooses the nearer station to its current position. Once a station is chosen, the CA_j sends it a *usage* request (*askToUse()*) that includes information about the requested operation and its criticality. If the station accepts the request, the CA_j becomes *Booked*. If not, the CA_j has two possibilities: it delays its treatment on the station that refused the request and sends it a *future usage* request (*askForFutureUsage()*) or selects a new station and sends it a *usage* request (*askToUse()*). In the first case, if the station informed by the *future usage* request accepts it, the Container agent becomes *Delayed*.

In the *Booked* state, the CA_j continues the exploration of the factory for improvements. Such improvements occur when the CA_j discovers new stations that may treat its operation faster that the current booked one. To try to discover such stations, the CA_j sends *neighbourhood* requests (*askForNeighbourhoud()*) to already known stations. When this neighbourhood is received, the CA_j selects the new not known qualified stations and asks them for their *processing time*. After that, if better stations are discovered, the CA_j sends them

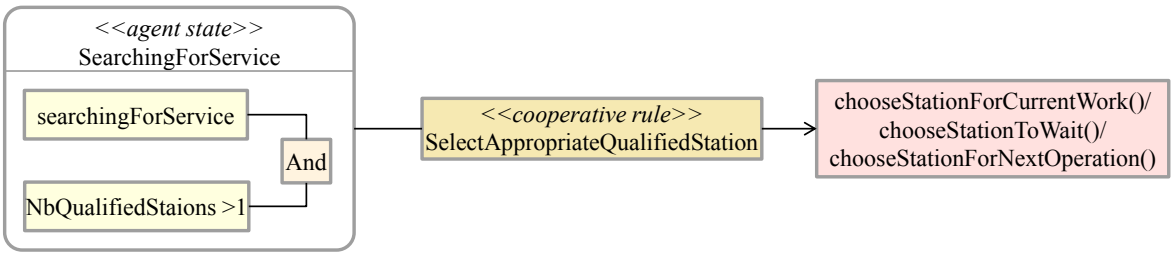

Figure 5.4 — The *select appropriate qualified station* Cooperative Rule for solving the stations concurrence *NCS*

either *usage* requests or *future usage* requests. If a more adequate station accepts to treat the current operation, the CA_j cancels the already booked station and changes its status correspondingly to the accepted request (*usage* request: it stays in the *Booked* state; *future usage* request: it switches to the *Delayed* state). In addition to this, the CA_j sends *future usage* requests to qualified stations for its next operation. It uses the same process as if in the *Free* state.

In the *Delayed* state, the CA_j behaves in the same way as in the *Booked* state. It continues the exploration of the factory sending *usage/future usage* requests for its current operation and switches states according to the accepted requests. As in this state the agent is not sure about the treatment of its current operation it will not send any *future usage* requests concerning its next operation.

Finally, the CA_j can easily switch between the three states *Free, Booked and Delayed* depending on the information it perceives.

5

Four *Non Cooperative Situations (NCSs)* which have been instantiated from the generic *NCSs* found in section 3.5 for *DFJSP*, can be underlined in this state:

- **Stations concurrence** (figure 5.4): several qualified stations can be detected by the Container agent when searching to treat its operations. The Container agent chooses the most appropriate station using the station processing time, its position, the possible beginning time of its processing and the perceived criticality of the station. This cooperative rule is an instantiation of the *Concurrence* rule (section 3.5.5).
- **Uselessness after usage/future usage request** (figure 5.5): this situation, an instantiation of the *Uselessness* rule (section 3.5.3), occurs when the requested station cannot treat the current operation (*Free* state) or next operation (*Booked* state) of the CA_j. To solve this *NCS*, the CA_j must request the service of another station. Thus, it selects an appropriate station and sends it the adequate request (*askToUse(), askForFutureUsage()*).
- **Suboptimal affectation** (figure 5.6): in a *Booked* or *Delayed* state, the CA_j continues the exploration of the factory searching for improvements. Indeed, as a cooperative agent, the CA_j must find the best possible place in the organisation (*partial Uselessness NCS* section 3.5.3). Thus, whenever an improvement is detected the CA_j sends adequate requests and tries to obtain the service.
- **Knowledge Unproductiveness** (figure 5.7): the CA_j does not know any other adequate

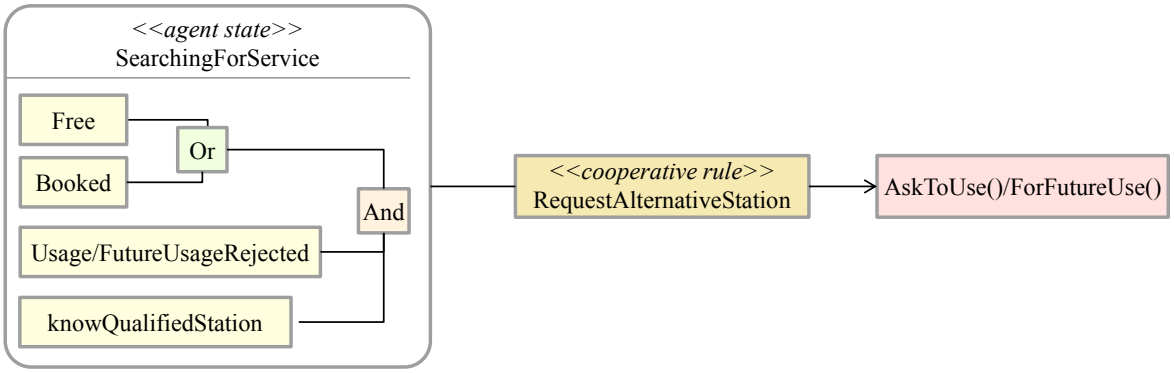

Figure 5.5 — The *request alternative station* Cooperative Rule for solving the uselessness of a usage/future usage request *NCS*

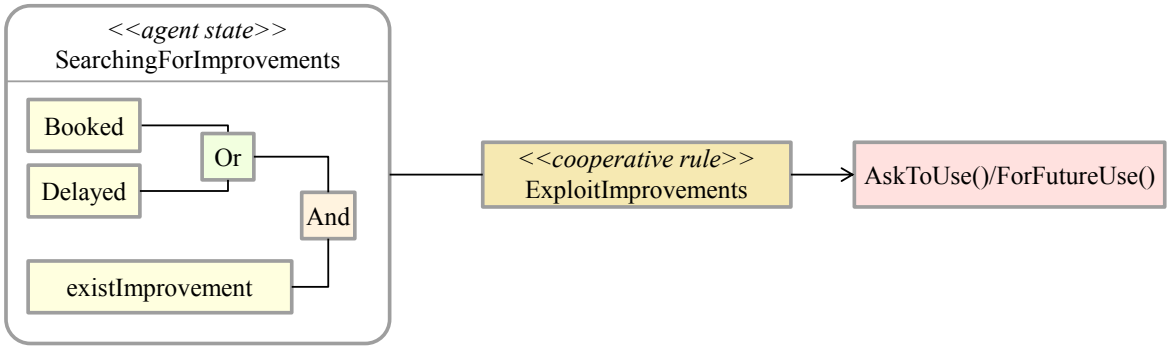

Figure 5.6 — The *exploit improvements* Cooperative Rule for solving the suboptimal affectation *NCS*

available station. Thus, it asks stations for their neighbourhood. If no adequate station is found, the CA_j waits for the next fabrication cycle. This situation is an instantiation of the Unproductiveness cooperative rule (section 3.5.2).

In addition to these cooperative rules, the Container agent possesses *useful and important informations concerning its next qualification* for the Station agents. Given its cooperative attitude, it forwards these informations by sending *future usage* requests in order to anticipate future *NCS*.

IN TREATMENT. The current operation of the CA_j is treated by a station. The CA_j knows when its current treatment can be finished, and it sends *future usage* requests to qualified stations for its next operation. If one of its requests is accepted, the CA_j will automatically ask the usage of the station accepting the request as soon as its current treatment is finished. During its treatment, the CA_j always searches for improving the treatment of its next operation as described in the *Available* state. In this state, the CA_j can be informed by the Station agent about a perturbation that has occurred on the station. In this case, the CA_j switches its state to *In Perturbation* and waits until the perturbations are finished. Moreover, given its cooperative attitude, it cancels and stops sending information about its next operation.

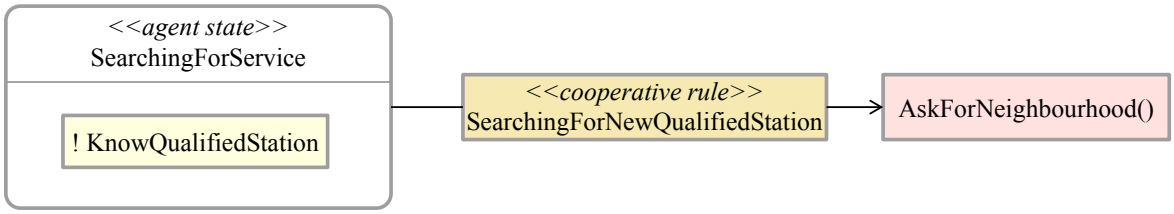

Figure 5.7 — The *searching for new qualified station* Cooperative Rule for solving the knowledge unproductiveness *NCS*

The **Stations concurrence**, **Uselessness after future usage request**, **Suboptimal affectation** and **Knowledge Unproductiveness** non cooperative situations can also occur in this state. The CA_j repairs them in the same way as in the *Available* status.

IN PERTURBATION. The CA_j has problems (blocked on a station, is being modified, etc.) and by that cannot provide any information about its current or next operations. As a cooperative agent, it cancels any reservation or sent request and stays idle until the perturbation is finished.

5.2.2.5 Station Agent

The Station agent (SA_i) represents a given station S_i. It is qualified for the treatment of a set of operations required by the Container agents. It is the agent with the service role. Its role in the factory is to help the Container agents to find their schedules. Thus, it receives the requests of the Container agents and tries to respond to them while being as cooperative as possible. Figure 5.8 describes its skills, characteristics, representations and its interactions with other agents of the system and its environment. The description is provided using the *AMAS-ML* modelling language.

The behaviour of the Station agent SA_i consists in selecting between the Container agents *usage/future usage* requests which operation to treat. It receives different types of requests:

- Processing time requests for given operations;
- Neighbourhood requests from Container agents exploring the factory;
- Usage requests;
- Future Usage requests about upcoming operations (regroups current and next operation treatments).

The SA_i responds to the two first types of requests independently of its status using *processingTime()* and *neighbourhood()* messages. For the *usage* and *future usage* requests, the SA_i responds accordingly to its different states as described in figure 5.9: AVAILABLE, IN TREATMENT and IN PERTURBATION.

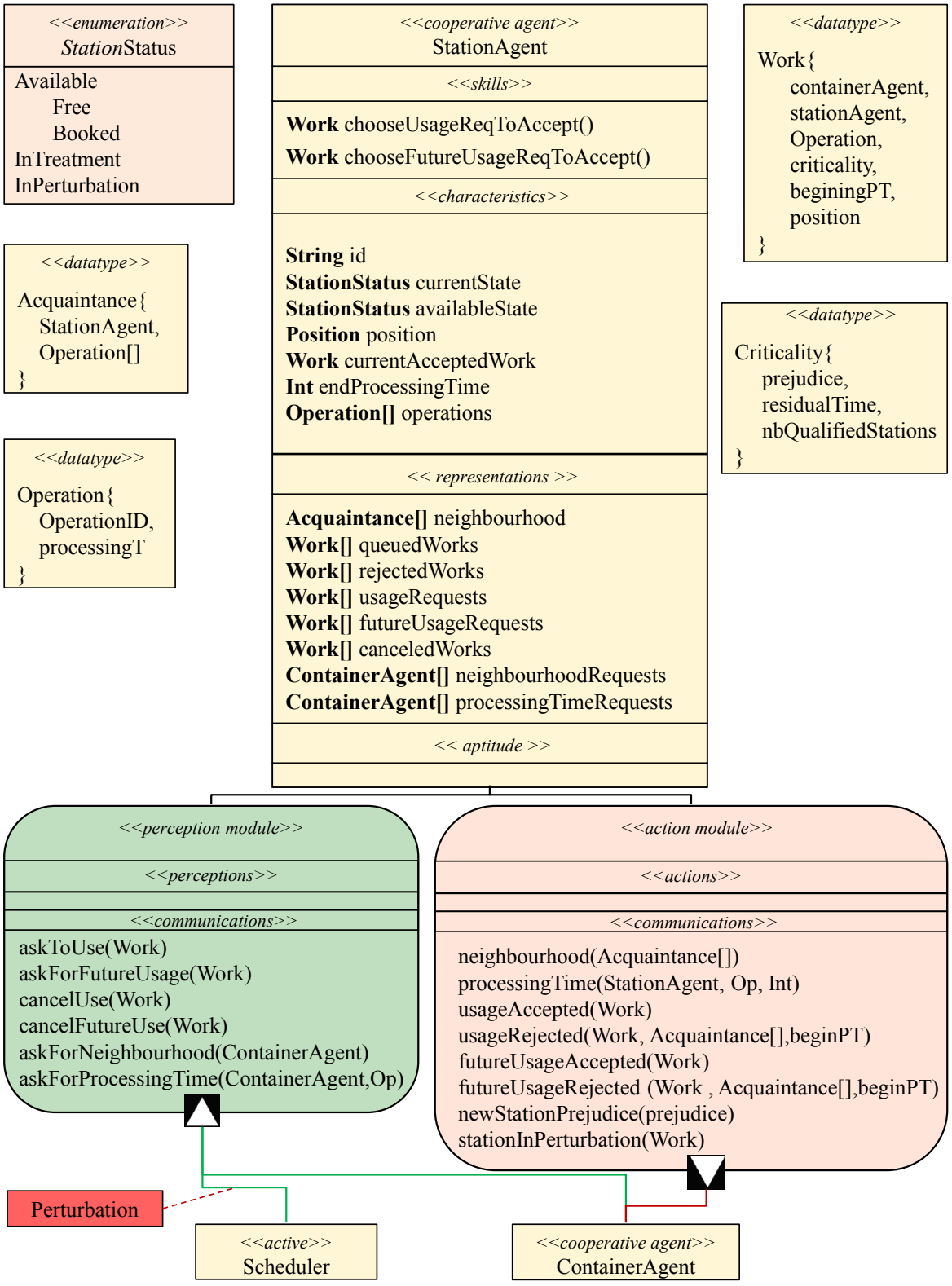

Figure 5.8 — Station agent description using *AMAS-ML*

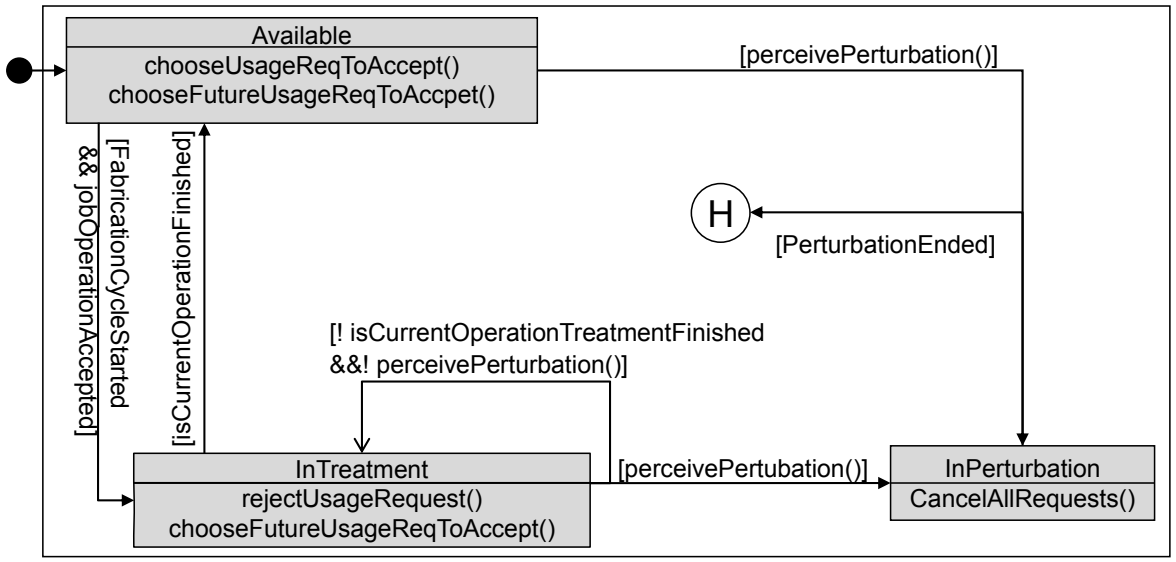

Figure 5.9 — Station agent states

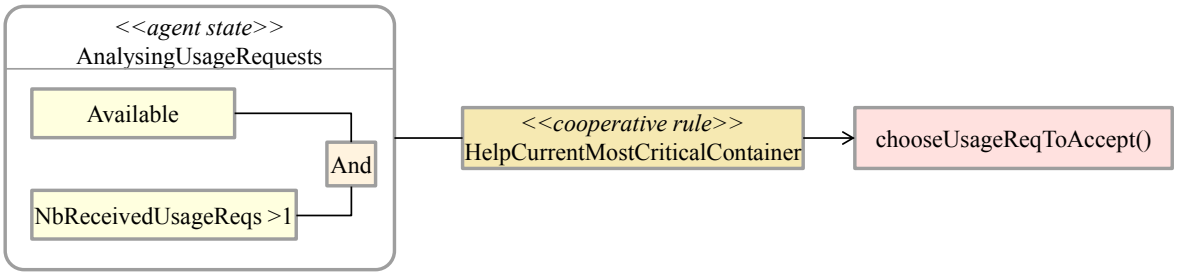

Figure 5.10 — The *help current most critical Container* Cooperative Rule for solving the conflict *NCS*

5

AVAILABLE. The Station agent has no operation in treatment. It decides using an evaluation of the criticality level of the perceived *usage/future usage* requests which ones to keep and which ones to reject. Indeed, a Station agent can accept several *future usage* requests depending on the beginning time of their processing and the duration of their treatment. Those requests will be treated by the station after its current treatment is finished (note that for the future requests it is only the Container agent that is delayed and has to wait until the Station agent finishes its current treatment). The SA_i can only accept one usage request, which it does depending on the level of criticality of the container and the processing duration of the requested operation.

In this state, the Station agent is confronted to the *Conflict NCS* (figures 5.10, 5.11). Indeed, by receiving different *usage/future usage* requests at the same time, the SA_i detects the conflict between the Container agents. It solves this situation using the criticality level and the processing time of each requested operation.

IN TREATMENT. The SA_i is treating an operation, it cannot accept another one. By that, it rejects all the current *usage* requests by notifying the requesting agents adding the ending processing time of its current treatment and its neighbourhood. Concerning the perceived

Figure 5.11 — The *help future most critical Container* Cooperative Rule for solving the conflict *NCS*

future usage requests, it keeps the more urgent ones that begin after its current treatment is finished. As in the *Available* state, the SA_i must handle *Conflict NCS* for *future usage* requests (figure 5.11).

IN PERTURBATION. The SA_i cannot treat any operation and cannot provide any information about the ending of the perturbation. As a cooperative agent, it rejects all the requests it receives and notifies the requesting agents about its state (*stationInPerturbation()*).

At each fabrication cycle, agents cooperate locally by using the updated information about their states to decide how to act. Without the use of any global information, the schedule of the current available operations on the available stations is obtained after a certain number N of agent life cycles (perceive, decide and act).

5.2.3 SAFlex Results & Discussions

SAFlex is tested using the *Multi-Agent for Supply Chain* (MASC) platform developed by André Machonin and all in the context of the multi-team working group *CollInE*[3]. The code counts more than 90 classes, with 20 classes generated using *Make Agents Yourself (MAY)*, implemented in Java.

For the evaluation of *SAFlex*, 90 instances (45 without perturbations affecting stations and 45 with perturbations) were generated using different parameters of the problem. Their generation is described in section 5.2.4.3. In order to explain obtained results, one simple representative instance with and without stations perturbations is presented first. It is characterized by:

- 6 stations;
- 10 containers: 2 having 5 operations and 8 having 6 operations;
- 7 containers are released at time $t = 0$, 1 at time $t = 1$, 1 at time $t = 7$ and 1 at time $t = 17$. Due Dates range between time $t = 25$ and $t = 44$.

[3] http://www.irit.fr/COLLINE/

This instance was also transformed in an instance with perturbations by adding 25 station breakdowns affecting the 6 stations at different times. *SAFlex* runs on both instances for 100 times and average results are computed (table 5.2).

	Without pertubations	With perturbations
T_n	0,86	4,12
$\overline{T}$	0,14	1,7
T_{max}	0,99	7,34
CT_{max}	39,87	48,27
Calculation time (ms)	0,17	0,25

Table 5.2 — Average results for SAFlex

Figure 5.12, shows the obtained schedules for the studied instances, using the average completion time CT_{max}. The blue lines (first 3 vertical lines) represent the release dates, the red ones (five last vertical lines) the due dates and perturbations are represented by rectangle shapes coloured with yellow and orange stripes. Stations are represented using the letter M and each container is represented with a different color using the letter J. The evaluation of *SAFlex* is done regarding several criteria that underline self-adaptivity (chapter 4).

A first point to underline is the increase of the CT_{max}. The station breakdowns occurring at the beginning impose some delays on the container treatments which increase the complete *Makespan*. Thus, the calculation time also increases.

To understand how *SAFlex* reacts to these perturbations, the calculation time needed by agents at each Fabrication Cycle is studied. Figure 5.13, presenting the evolution of the computation time under perturbations per fabrication cycle, shows that whenever new containers are released, the calculation time is increased than reduced as the perturbation is handled by the system. Another point to underline is that the most unstable period of the resolution is between the fabrication cycle 7 and 20, which corresponds to the arrival of different station breakdowns. Even under these perturbations, we can see that the locality of agents actions prevents a drastic increase in calculation time.

	Without pertubations	With perturbations
T_n deviation	0,7	1
$\overline{T}$ deviation	0,14	0,6
T_{max} deviation	0,8	1,4
C_{max} deviation	1,236	1,99
Calculation time (ms) deviation	0,04	0,09

Table 5.3 — Standard deviation over 100 executions

The decentralized and distributed nature of multi-agent systems makes such systems non deterministic. Thus, an interesting criterion to study is the stability of the obtained solutions from one execution to another. Here, the standard deviation ratio over 100 executions (table 5.3) is computed. The low values obtained demonstrate the stability of the system, and the concentration of the different solutions around the average. It can be interpreted as a consistent behaviour of the system (as opposed to more chaotic/random

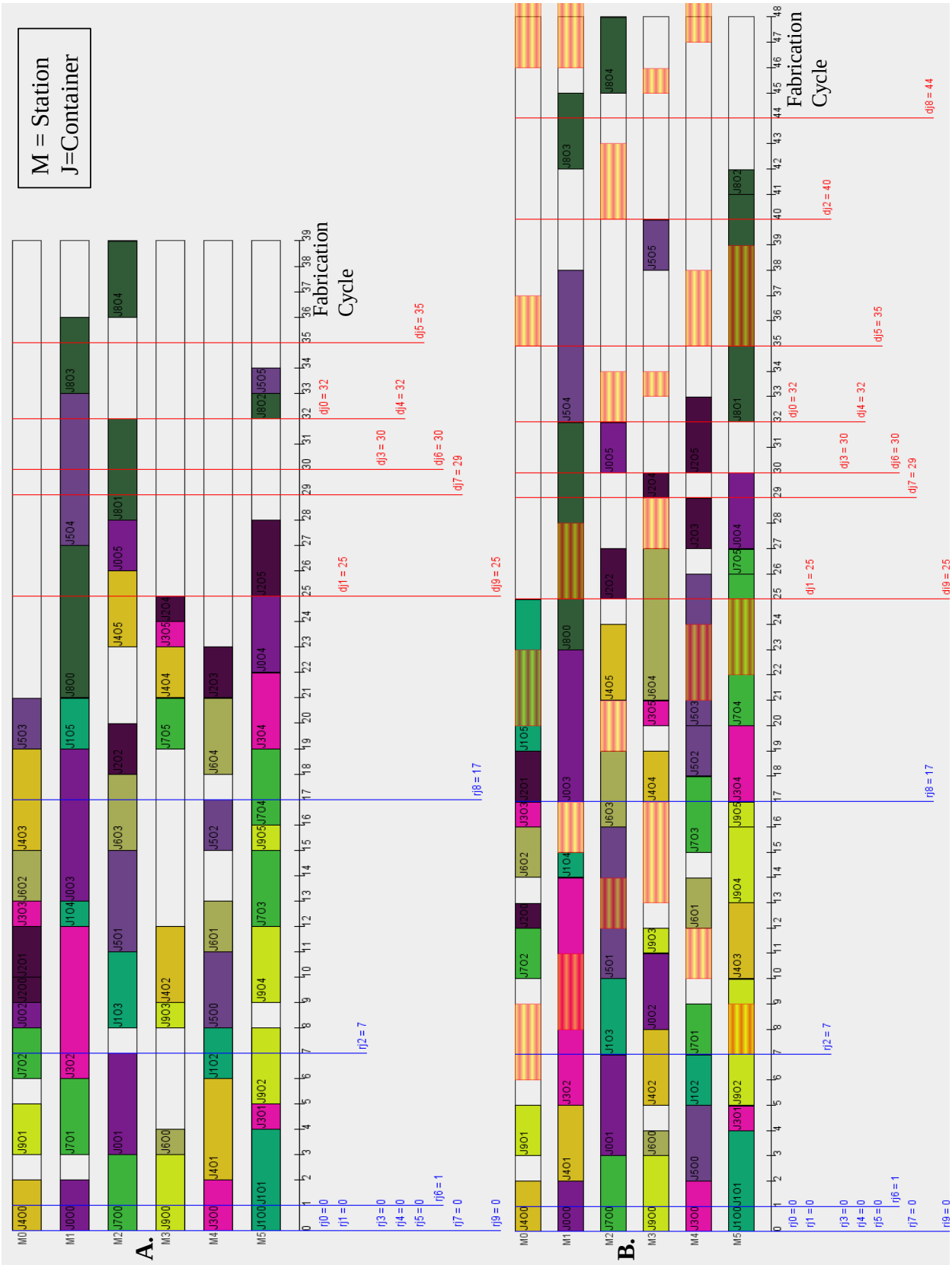

Figure 5.12 — The schedules computed by *SAFlex* for the two studied instances without (A.) and with (B.) perturbations) (IHM designed by Tony Wauters from the Vakgroep Informatietechnologie, Departement Industrieel Ingenieur, KaHo Sint-Lieven, Gent - Belgium.

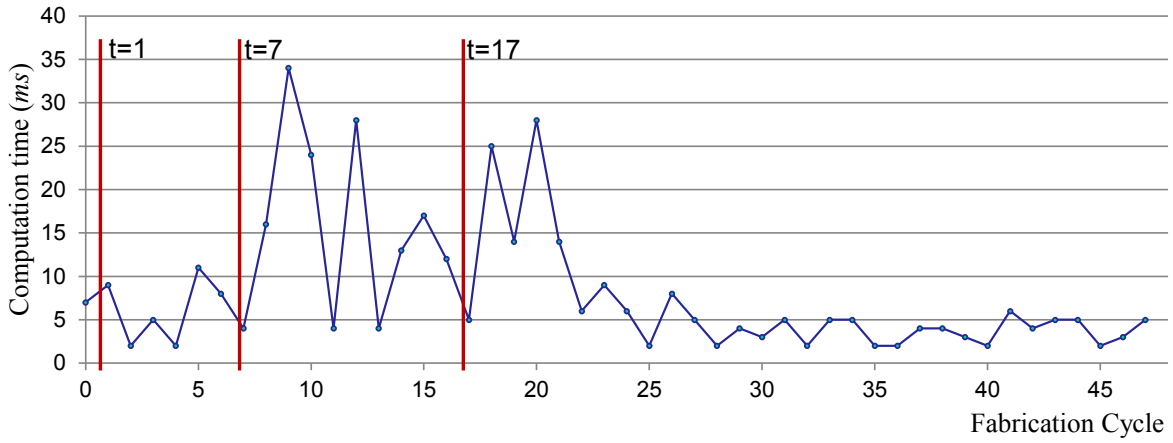

Figure 5.13 — Evolution of the computation Time (*ms*) per fabrication cycle for the instance with perturbations

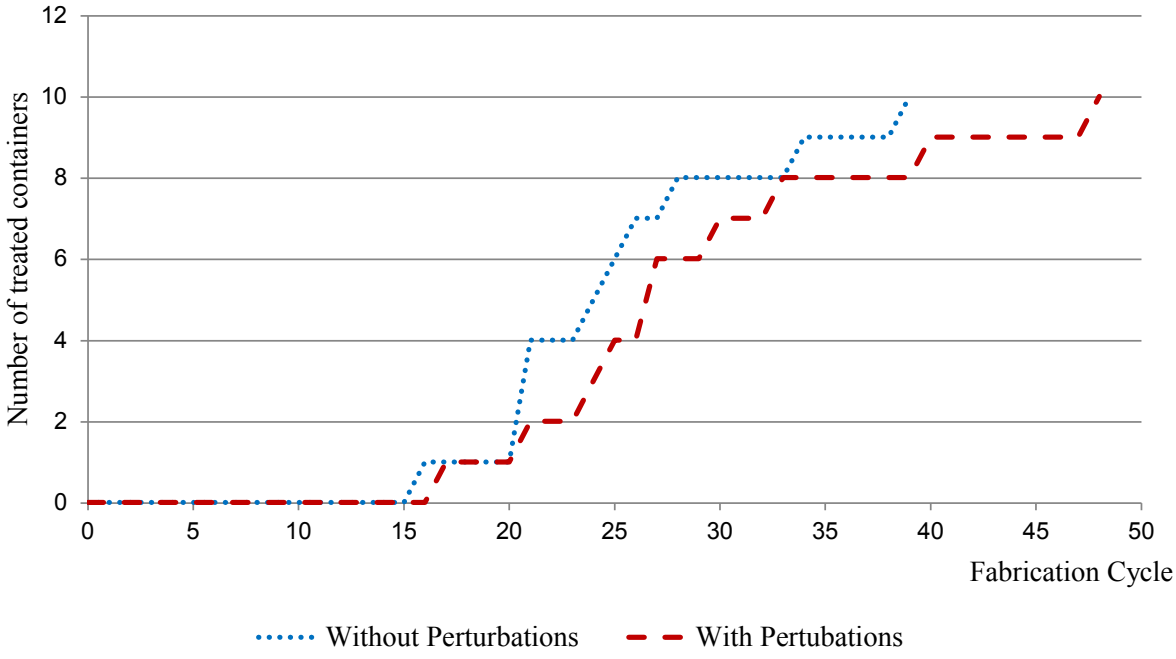

Figure 5.14 — The progress of treated container throughout the system execution

behaviours).

Another interesting criterion is the *Progress* which refers to the study of how the system progressively reaches a complete solution under perturbations. In *SAFlex*, the evolution of finished container treatments during simulation is studied. Figure 5.14 summarizes the results. Note that the curves have the same progression, which means that, when the perturbations occur, the system is slowed down and after the agents self-adapt, it regains its stable behaviour. Another point which is studied using larger instances, concerns the reaction of the system when a group of urgent containers arrives. The progress showed us that the evolution of finished containers is slowed down when these urgent containers arrive and the evolution increases after these containers are treated. Thus, we deduce that

the priority measure computed locally by *Container agents* and used by the *Station agents* is adequate and important for the agents decisions.

Finally, the simplicity of the agents behaviour can be underlined. Indeed, the agents adapt their behaviour locally to the perceived events. This action locality reduces the effect of perturbations, increases the adaptivity and thus, improves the robustness of the system. In addition to this, the simple behaviours reduce the complexity of local algorithms which help the system to keep stable computation time even with larger instances.

Other larger instances including a large number of stations and containers was also tested. For example, one of the larger instances defined with perturbations is characterized by:

- 15 stations;
- 20 containers: 2 having 10 operations, 7 having 11 operations, 3 having 12 operations, 5 having 13 operations and 3 having 14 operations;
- Release dates range between $t = 0$ and $t = 190$. Due Dates range between time $t = 177$ and $t = 363$;
- 76 perturbations occurring at different intervals of time.

	simple instance	large instance
T_n	4,12	0,4
$\overline{T}$	1,7	0,125
T_{max}	7,34	2,4
CT_{max}	48,27	341,2
Calculation time (ms)	0,25	3,3

Table 5.4 — Average results for SAFlex applied to two instances (simple and large)

We note, given the differences presented in table 5.4, that *SAFlex* is scalable. It is able to handle the dynamic events keeping reasonable calculation time and providing good results regarding the defined objectives.

5.2.4 Comparative Study & Discussion

In this work, a comparison between the *Reinforcement Learning* technique and the *AMAS* Theory has been established. *SAFlex* is compared to two different approaches. Each was built using *Reinforcement Learning* techniques in multi-agent system. The first one, Learning/Optimization approach is an off-line scheduling technique. The second, On-line Forward Optimization is a rolling time scheduling technique. This study was done during my 3-months visit in Belgium in cooperation with Yailen Martinez from the Computational Modeling Lab, Vrije Universiteit Brussel - Belgium and Tony Wauters from the Vakgroep Informatietechnologie, Departement Industrieel Ingenieur, KaHo Sint-Lieven, Gent - Belgium [Kaddoum et al., 2010].

Evaluation Criterion	Considered	Not Considered	Remarks
Resolution Time	✓		
Number of messages		χ	The number of messages was not studied as in this study, the agent behaviour and their criticality management were our first interest. Still, this criterion will be studied in order to deploy the system in real environments.
Quality of solution	✓		
Progress	✓		
Memory usage		χ	This approach is not memory intensive as agents do not require to register a lot of information. Still, this criterion is important(i.e. when deployed on embedded systems) and will be studied to compare this approach to other self-organised multi-agent systems.
Robustness	✓		
Local computational complexity	✓		
Decentralisation	✓		
Action locality	✓		
Initial and acquired knowledge	✓		
Agent number influence	✓		The system tested on large instances does not seem to suffer from scalability issues. Still further studies are required.
Analysis and identification of autonomous agent		χ	Those criteria depend on the *AMAS4Opt* agent model used in this study. See section 5.4 at the end of the chapter for additional information.
Distribution & deployment ease		χ	
Genericity		χ	

Table 5.5 — Summary of the considered and not considered evaluation criteria introduced in chapter 4 for the evaluation of *SAFlex*.

5.2.4.1 Learning/Optimization Approach

The *Learning/Optimization method* is an off-line scheduling approach divided in two phases. First, a two-stage learning method is applied to obtain feasible schedules, which are then used as initial data for the mode optimization procedure [Van Peteghem and Vanhoucke, 2008] developed during the second phase.

The implemented learning method decomposes the problem following the assign-then-sequence approach [Pezzella et al., 2008]. Therefore we have two learning phases, during the first phase operations learn which is the most suitable station (one agent per operation which chooses a station from the given set) and during the second phase stations learn in which order to execute the operations in order to minimize the makespan (one agent per station choosing which operation to process next from the queue of operations waiting at the corresponding resource). Once a feasible schedule is obtained, the mode optimization procedure is executed. This procedure has the following steps:

1. Step 1: Order the operations according to their end times (the time when they were ended in the schedule received as input).
2. Step 2: Taking into account the previous ordering, for each operation, choose the station that will finish it first (shortest end time, not shortest processing time). The result is a backward schedule.
3. Repeat steps 1 and 2 to obtain a forward schedule.

Once the mode optimization is executed, the quality of the solution is taken into account to give feedback to the agents of the learning phase.

5.2.4.2 On-line Forward Optimization

The *On-line Forward Optimization method (OFO)* uses an event-driven re-scheduling technique. A first schedule given all the available information is generated. At each fabrication cycle, if any new information is available, a new schedule is generated without changing the already scheduled and started operations.

For scheduling the operations, a serial schedule generation method with a station choice optimization procedure is used [Van Peteghem and Vanhoucke, 2008]. The *OFO* method works as follows. At fabrication cycle 0 an empty schedule S is initialized. If at fabrication cycle t a container is released or a station is in perturbation, an optimization step is applied. After Q iterations, the optimization step returns a good partial schedule. At each optimization iteration, a random but feasible operations list is constructed. This operations list contains all new operations, and all already scheduled operations from the previous best partial schedule that have not been started yet at fabrication cycle t. This operations list is used in a serial schedule generation method that takes into account the station perturbations to generate a new partial schedule. The best partial schedule during the Q optimization iterations is kept. The quality of a partial schedule is determined using an objective function combining the different problem objectives to optimize (section 5.2.1). This method uses the learning automata and at each iteration of the Q iterations, a feedback

is sent to the agent that learns the best actions to perform [Wauters et al., 2011, 2010]. After the optimization step, the scheduled operations are added to the schedule S. If the schedule S already contained some operations, it will use the new values. This optimization step is executed at each fabrication cycle where perturbation events occur.

5.2.4.3 Experimental Setup

For the comparison of the different approaches we generate a representative set of instances of the dynamic flexible Job shop scheduling problem. For that, classical flexible job shop scheduling instances were used as a base problem [Brandimarte, 1993] to which release-dates, due-dates and perturbations were added. From the base problem instances, 9 problems were chosen. For each problem, 5 instances without station breakdowns perturbations using a uniform distribution between given limits were randomly generated: the number of containers ranges from 10 to 20, the number of stations ranges from 4 to 15 and the number of operations for each container ranges from 5 to 15. Then, we added perturbations to each instance (station breakdowns). Thus, the final test set contains 90 scenario instances (45 without perturbation and 45 with perturbations) with widely varying characteristics.

Due-dates are generated uniformly between a lower bound LB_r and an upper bound UB_r. The lower bound is the best lower bound found in literature for the base problem [Pezzella et al., 2008]. For the upper bound value, we use a greedy first available station heuristic to compute a Makespan value for each instance. This heuristic assigns operations one by one to the first available station.

Release-dates are also uniformly generated between 0 and $max(0; d_{i,j} - p_{max})$. Where $p_{max} = \sum_{i=1}^{n} max_k(p_{i;j;k})$ is the maximum total processing time of a container C_j considering the maximum processing time of each of its operations $max_k(p_{i;j;k})$.

For the generation of the perturbations, we use two distributions: *Poisson* and *Erlang* [Winston, 2003]. The *Poisson* distribution with mean $\frac{UB_r}{\theta}$ determines the inter-arrival time between perturbations. Higher θ values result in a lower mean inter-arrival time, and thus in more perturbations. The Erlang distribution with rate parameter R and shape parameter k computes the duration of the perturbations, with a mean duration of $\frac{k}{R}$.

For the instances with perturbations, $\theta = 5$, $R = 2$, and $k = 6$ were used, so a mean perturbation time of 3 fabrication cycles is obtained. Settings for the *OFO* approach are $Q = 50$. For *SAFlex*, the number of agents life cycle for each fabrication cycle N is fixed to 15.

In this study, our interest is focused on the 4 objectives (the number of tardy containers T_n, mean Tardiness $\overline{T}$, maximum Tardiness T_{max} and makespan CT_{max}) defined in section 5.2.1 and the computation time. As each objective provides precise information on the approach behaviour, it is impossible to reduce them into one unique dimension. To compare the three studied approaches, a multi-objective presentation providing a global view on those objectives is more appropriate. Consequently, we present the results using a radar view having 5 directions : four for the objectives ($T_n, \overline{T}, T_{max}, CT_{max}$) and one for

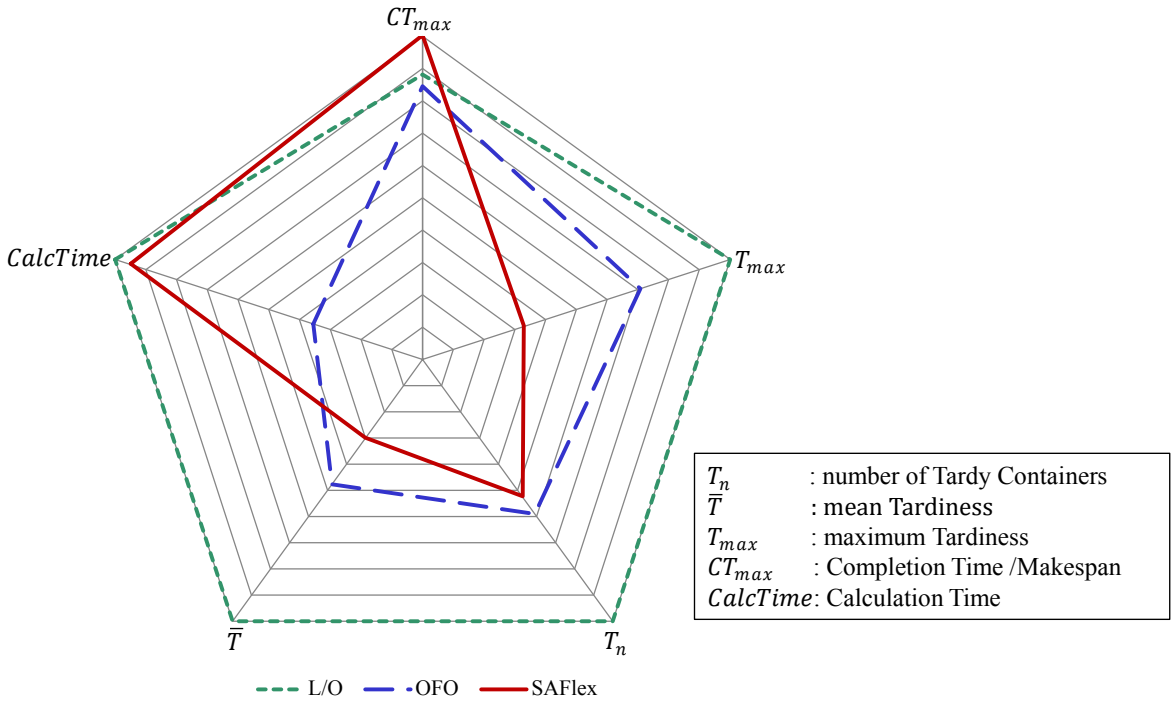

Figure 5.15 — Approaches Comparison

the computing time. On each direction, the scale represents a percentage, 100% being the worst behaving method, 0% representing the value 0 for a given measure. Thus, we obtain a normalized view for each criterion, facilitating the comparison. The *best system* is on the radar center because it outperforms the others in all the dimensions.

For each instance, we performed 10 runs and represented the average results using a radar view. We notice a certain variability between the results of different instances which means that the characteristics of the studied instances impact the obtained results. As stated in the introduction, we are interested in real-life problems with a high level of dynamics. In such context, it is difficult to characterize the instances that can occur. By that, a global view considering average results of all the given instances is more adequate to compare the functioning of the three approaches, as it represents the mathematical expectation of the behaviour in a real-world situation. This global view is presented in figure 5.15.

5.2.4.4 Discussion

In this section, the results regarding the four objectives defined in section 5.2.1 (the number of tardy containers T_n, mean Tardiness $\overline{T}$, maximum Tardiness T_{max} and makespan CT_{max}) to which the calculation time is added, are studied on the three different solving methods, representing different approaches to the Dynamic Flexible Job Shop Problem. The advantages and limits of the presented approaches regarding the 5 directions of the radar view are discussed (figure 5.15).

Considering the Makespan (CT_{max}) objective, we notice in one hand, that *OFO* is performing best. *OFO* computes a global function of weighted objectives. An increased

weight on this objective ensures good results but the tardiness objectives are not fully optimized. On the other hand, *SAFlex* has the worst CT_{max}. This is due to the fact that, in this approach, we focus on the tardiness of containers. In a context of a real-time environment, we decided to provide the best satisfaction for the client orders by reducing the delays and respecting the due dates. That enables *SAFlex* to be better for T_n, $\overline{T}$ and T_{max}, but still having a reasonable CT_{max}.

Considering the tardiness objectives T_n, we notice that *OFO* and *SAFlex* both compute a relatively good T_n. An interesting point to see is that *SAFlex* optimizes $\overline{T}$ and T_{max} better. This is a consequence of the priority level used by the *Station agents* to choose between the *Container agents*. This priority level can be considered as the *Container Agent* satisfaction degree, and by always choosing the agent with the highest criticality, we try to equilibrate this satisfaction degree. That results in an equilibrated tardiness between containers and reduces T_{max}.

Regarding the calculation time, the simulations were done on different types of computers which increases the difficulty to study this criterion. Nevertheless, we notice that *OFO* is the best approach in that direction. This fact is in contradiction with what we underlined in the introduction (section 5.2) about the rolling-time scheduling methods. Indeed, its stated that such methods perform well but increase their computation time in a dynamic environment, but this is hidden because of the averaging. To show that effect, we studied the difference between the average calculation time for the instances without perturbations and instances with perturbations. The results, summarized in table 5.6, show that when adding perturbations, *OFO* increases significantly the calculation time while in *SAFlex*, the time increases slowly. The reason resides in the fact that in *SAFLex*, complexity is distributed between the agents which reason at a local level, continuously taking into account dynamic events in their reasoning, whereas *OFO* needs complete rescheduling.

	Without perturbations	With perturbations	Increase
OFO	0,24 *ms*	1,02 *ms*	+325%
SAFlex	1,18 *ms*	1,32 *ms*	+12%

Table 5.6 — Average Calculation Time for instances without and with perturbations

From a general point of view, rolling and real time scheduling techniques are more appropriate for DFJSP solving than off-line scheduling techniques. Here, *SAFlex* and *OFO* are adequate systems, each having its limits and advantages. An interesting study would be to try to combine their advantages in order to improve the results.

5.3 Design of Complex Product

A complex product is generally a system composed of numerous interconnected parts representing a specific discipline and is developed using associated expertise. Those parts are interdependent and the function of the system is provided by their interactions. The design of such complex product involves the coordination between different correlated disciplines and must satisfy a set of performances and objectives before it can be manufactured and sold. By that, the complex system design problem is a multidisciplinary and multi-objective optimization problem.

Different optimization approaches have been developed for solving design problems such as *Multidisciplinary Design Optimization (MDO)* or *Multi-Objective Optimization (MOO)* [Brochtrup and Herrmann, 2006]. [Welcomme et al., 2009] introduces a distributed self-adaptive multi-agent approach for the *Aircraft Preliminary Design problem* where agents have disciplines knowledge and act cooperatively to solve the problem.

In this work, a new original approach based on the generic agent model is proposed to solve the problem. It does not require explicit model knowledge. Indeed, for each design domain, a generally huge set of real already designed elements exists. Considering, for example, the *Aircraft Preliminary Design problem*, the set of already defined populations of aircraft such as $A380$, $A320$, etc. is large. Such aeroplanes have been tested and validated. Thus, when constructing a new aeroplane, it is interesting to profit from the already known and acquired knowledge from the already existing aeroplanes, especially considering that this knowledge does not only contain the disciplines information but also the engineers experience. Another problem where it is interesting to profit from the already existing knowledge is the *seed problem*. Large data bases of plant phenotypes exist. When searching for crossing different plants in order to obtain specific phenotypes, it is useful to profit from such existing data bases. Unfortunately, given the huge volumes of data bases, the existing knowledge is not well exploited. This approach can also be useful in different other domains such as the design of vehicle engines or for the invention of new medicine.

The *SAPBR* approach, given this set of already existing elements and a new element for which some of the characteristics are known and the others are unknown, aims at finding the values of the unknown characteristics. In this approach, disciplines are represented by their characteristics. Their interdependencies exist but are unknown.

5.3.1 Problem Formalization

In a design domain D (table 5.7), a set E of real already designed elements exists E=$\{E_1; E_2; ...; E_n\}$. Each element E_j is described by a set of characteristics C=$\{C_1; C_2; ...; C_m\}$ each having a known value $V_{(i;j)}$ (where i represents the number of the characteristic of the element j). So, an element E_j can be represented as a vector of values V=$\left\{V_{(1;j)}; V_{(2;j)}; ...; V_{(m;j)}\right\}$. The correlation between the different characteristics are given by a set of formulas (expressed as in formula 1) expressing each characteristics as a function $f_{(i;j)}$ of the others characteristics. Those different $f_{(i;j)}$ are unknown in our case.

Characteristic \ Element	E_1	E_2	...	E_j	...	E_n
C_1	$V_{(1;1)}$	$V_{(1;2)}$	...	$V_{(1;j)}$	...	$V_{(1;n)}$
C_2	$V_{(2;1)}$	$V_{(2;2)}$	...	$V_{(2;j)}$	...	$V_{(2;n)}$
.....						
C_m	$V_{(m;1)}$	$V_{(m;2)}$	...	$V_{(m;j)}$	...	$V_{(m;n)}$

Table 5.7 — Design Domain

Characteristic \ Element	E_1	E_2	E_3	E_x
C_1	$V_{(1;1)}$	$V_{(1;2)}$	$V_{(1;3)}$	$V_{(1;x)}$
C_2	$V_{(2;1)}$	$V_{(2;2)}$	$V_{(2;3)}$	$V_{(2;x)}$
C_3	$V_{(3;1)}$	$V_{(3;2)}$	$V_{(3;3)}$	?
C_4	$V_{(4;1)}$	$V_{(4;2)}$	$V_{(4;3)}$	?

Table 5.8 — Example of a Design Domain with the definition of a new Element

Formula 1. $V_{(i;j)} = f_{(i;j)}(V_{(1;j)}, V_{(2;j)}, ..., V_{(k;j)}, ..., V_{(m;j)})(\forall k \neq i)$

The aim of *SAPBR* is, given a data base representing the known elements of domain D and a new element $E_x = \left\{V_{(1;x)}; V_{(2;x)}; ...; V_{(m;x)}\right\}$ where only a subset of the characteristic values $V_{(i;x)}$ is defined by the engineer, to provide estimated plausible values to the non-defined subset of characteristic values $V_{(j;x)}$ ($j \neq i$). The characteristics with defined values are called *Known Characteristics*, the others are called *Sought Characteristics*.

5

5.3.2 The Adaptive Multi-Agent System

In order to design the adaptive multi-agent system and define the cooperative agents and their behaviour for solving this problem, an analysis of the problem is required. The first point to study is **what is required to obtain the value of one *Sought Characteristic***. For that, let us consider a small instance of the problem (table 5.8).

In this instance three known elements E=$\{E_1; E_2; E_3\}$ are described using four characteristics C=$\{C_1; C_2; C_3; C_4\}$. Two characteristics (C_1 and C_2) are known for the new element E_x. C_3 and C_4 are unknown. If we suppose that C_3 depends on C_1 and C_2 while C_4 depends on C_2 and C_3, we can, considering the dependency between C_3, C_1 and C_2, find an *indirect dependency* between C_4, C_1 and C_2. Thus, knowing the value of C_3 is not required for the computation of the value of C_4.

The consequence of the generalization of the concept of *indirect dependencies* that exist between the characteristics of the elements is that it is not required to consider the different *Sought Characteristics* of the new element when trying to define the value of one *Sought Characteristic*. In other words, it is sufficient to only consider the known knowledge on the *Known Characteristics* to find a value for one *Sought Characteristic*. Thus, the complete

Characteristic \ Element	E_1	E_2	E_3	E_x
C_1	$V_{(1;1)}$	$V_{(1;2)}$	$V_{(1;3)}$	$V_{(1;x)}$
C_2	$V_{(2;1)}$	$V_{(2;2)}$	$V_{(2;3)}$	$V_{(2;x)}$
C_3	$V_{(3;1)}$	$V_{(3;2)}$	$V_{(3;3)}$	?

Table 5.9 — First sub-problem of the Design Domain Example

Characteristic \ Element	E_1	E_2	E_3	E_x
C_1	$V_{(1;1)}$	$V_{(1;2)}$	$V_{(1;3)}$	$V_{(1;x)}$
C_2	$V_{(2;1)}$	$V_{(2;2)}$	$V_{(2;3)}$	$V_{(2;x)}$
C_4	$V_{(4;1)}$	$V_{(4;2)}$	$V_{(4;3)}$	?

Table 5.10 — Second sub-problem of the Design Domain Example

problem can be divided into sub-problems each containing the different known elements of the domain with the subset of their characteristics that are defined in the new element and their values for one *Sought Characteristic*. For instance, the small instance considered above is divided into two sub-problems. The first one contains the three elements, the first two characteristics (C_1; C_2) and C_3 (table 5.9). The second one contains the three elements, the first two characteristics (C_1; C_2) and C_4 (table 5.10).

The second point is **how can the value of the *Sought Characteristic* be estimated** for a given sub-problem. To answer this question, the interdependency of the characteristics must be analysed. As stated in the problem formalization (section 5.3.1) the different characteristics are correlated and each characteristic can be defined as a function of the others. In complex problem design, such functions are usually complex non-linear functions which are difficult to approximate on large intervals. But if considered in very small intervals, such functions can be approximated linearly as shown in figure 5.16. Thus, when a new value is given for a characteristic, the value of the second characteristic can be found using the linear function of the adequate interval.

Thus, for a considered element E_j, each characteristic can be expressed as a weighted sum of the other characteristics (formula 2) where $CW_{(k;j)}$ is the weight associated to the value of characteristic C_k of the element E_j.

Formula 2. $V_{(i;j)} = CW_{(1;j)} * V_{(1;j)} + CW_{(2;j)} * V_{(2;j)} + ... + CW_{(m;j)} * V_{(m;j)})(\forall m \neq i)$

Given this formula, estimating the value of the *Sought Characteristic* consists in finding the weights $CW_{(i;x)}$ associated to the values of the *Known Characteristics* $V_{(k;x)}$.

Another consequence of this linear approximation considering the relation between the different values of a given characteristic and their associated weights (figure 5.17), is that it leads to the ability to estimate the weight $CW_{(i;x)}$ of the *Known Characteristic*.

To summarize, estimating the value of a *Sought Characteristic* consists in computing

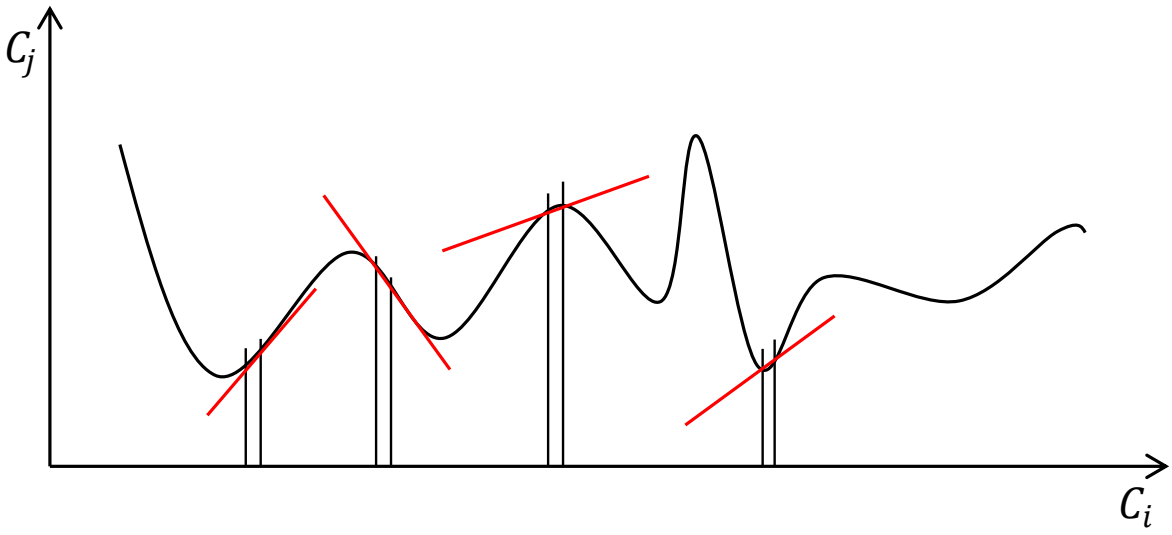

Figure 5.16 — Linear approximation on small intervals of non-linear complex functions

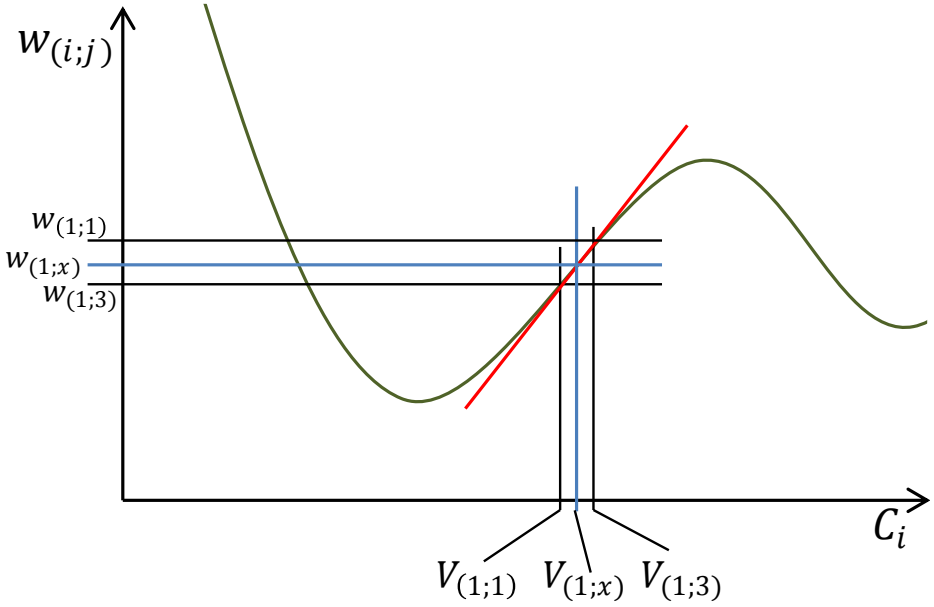

Figure 5.17 — Linear approximation of the weight of a given characteristic C_k

the different weights $CW_{(i;j)}$ of the characteristics that are known for the new element. Once these weights are estimated, the second step consists in framing the new element as tightly as possible by known elements from the domain. Then, the weights $CW_{(i;x)}$ of the *Known Characteristics* can be calculated leading to the estimation of the value of the *Sought characteristic*.

Finally, now that the solving process is defined, **the interacting entities must be identified** in order to **select among them the cooperative agents** of the adaptive multi-agent system. In our case, four entities are distinguished (figure 5.18):

- The *Known Characteristic* entity: it represents a known characteristic of the new element.

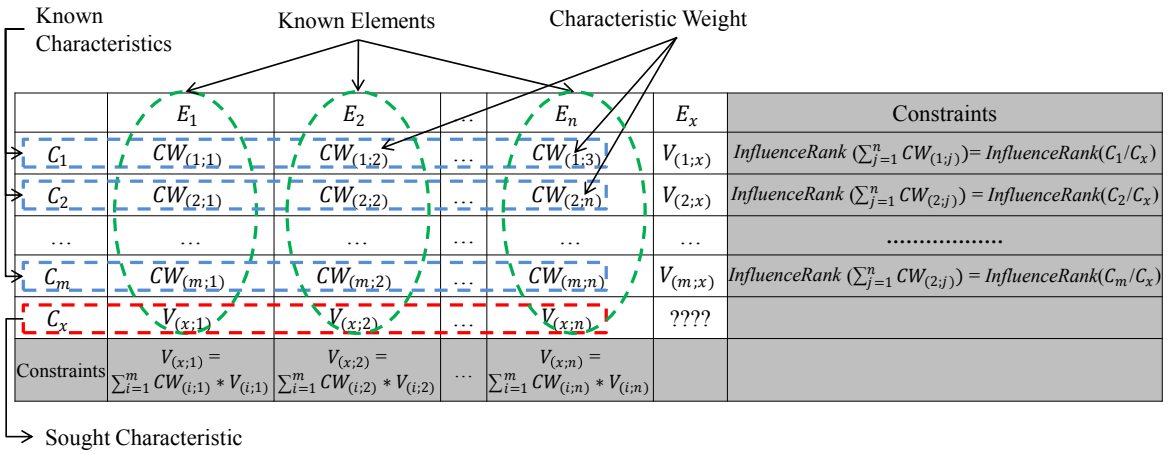

Figure 5.18 — The different interacting entities in *SAPBR*

It possesses all the known knowledge on this characteristic for each element (C_i, a line in the table of the figure 5.18). Using this knowledge and the known values of the *Sought Characteristic*, this entity computes its *Influence Rank* (section 5.3.2.3) on the *Sought Characteristic*. This influence is the constraint used in order to compute the weight associated to the value for each element so that the influence rank of the sum of the different weights (*Weight Influence*, section 5.3.2.3) is equal to the influence rank. In addition to this constraint, the *continuity* or *coherence* of the linear approximations in successive small intervals should also be respected. Thus, the weight of a given value is bounded by the weights of the values bounding this value. This constraint is called the *linearity constraint* and can be seen as a way to prevent a chaotic succession of linear approximations (a sort of *smoothing*).

- The *Known Element* entity: it represents a known element of the domain (E_j, a column in the table of the figure 5.18). It knows its values for the known characteristics and the sought characteristic of the new element. Thus, it can help for the estimation of the different weights of the values of its known characteristics so that the formula 2 can be verified among its characteristics.

- The *Characteristic Weight* entity: it represents the weight associated to a value of a given characteristic in a given element (a cell in the table of the figure 5.18). It must adjust its value in order to satisfy the known elements and the known characteristics constraints.

- The *Sought Characteristic* entity : it represents one of the unknown characteristics of the new element (C_x in the figure 5.18). This entity does not have sufficient local information in order to estimate its value. The known information on this characteristic are its values for the known elements and the influence deduced from these values that a known characteristic can have on it. The first information is used by the *Known Element* to verify the formula 2. The second one is used by the *Known Characteristic* to verify its influence rank.

Three of the four defined entities can be considered as cooperative agents: the *Known Characteristic*, the *Known Element* and the *Characteristic Weight*. Indeed they are the entities

that must cooperatively interact in order to estimate the right weights $CW_{(i;x)}$ for the values of the known characteristics in the new element in order to estimate the value of the *Sought Characteristic*. The *Sought Characteristic* is considered as an active entity that can provide useful information for other agents but does not participate in the solving process.

The *Known Element* agents and the *Known Characteristic* agents possess the constraints of the problem. Thus, they are the agents having the *constrained role*. Given their constraints, they guide the *Characteristic Weight* agents to find their values, by requesting adjustment directions that would lead to the satisfaction of their constraints. The *Characteristic Weight* agents will respond to theses requests by taking into account their criticality degrees. Thus they are the agents having the *service role*.

In brief, each sought characteristic in the new element requires its own multi-agent system for its solving. Each multi-agent system contains three different types of agents: Known Element agents, Known Characteristic agents and Characteristic Weight agents. The number of Known Element agents in one multi-agent system is equal the number of known elements in the data base. The number of Known Characteristic agents is equal to the number of known characteristics in the data base. The number of the Characteristic Weight agents is equal to the number of cells in the data base.

5.3.2.1 Agent Interactions

In this system, agents communicate using messages. Table 5.11 summarizes the different messages sent by each type of agent. They are divided into the four categories presented in section 3.3. The different messages are an instantiation of the messages defined in our generic agent model. We modified their names so that they correspond to the treated application.

Agents	Request for service	Request for Information	Answer to requests	Information
Known Characteristic	*decreaseKCWeight()* *increaseKCWeight()*	*influenceRank()* *weightsInfluence()*		*setSCDisorder()* *setSCWeightsSum()*
Known Element	*decreaseKEWeight()* *increaseKEWeight()*			
Characteristic Weight		*getNeighbourValue()*	*KEKCWeight()*	

Table 5.11 — Messages in *SAPBR*

The *decreaseKCWeight()*, *increaseKCWeight()*, *decreaseKEWeight()* and *increaseKEWeight()* messages are used by the *Known Characteristic* and the *Known Element* agents to request the modification of their weights from the *Characteristic Weight* agents that respond using the *KEKCWeight()*.

The *setSCDisorder()* and *setSCWeightsSum()* are used by the *Known Characteristic* agent to inform the *Sought Characteristic* entity about its influence. The *Sought Characteristic* entity, once having the different influences, associates a rank to each known characteristic. *Known Characteristic* agents get this rank using *influenceRank()* and *weightsInfluence()*. Those methods are implemented as getters and setters as the *Sought Characteristic* entity does not change and can be visible by other agents.

When looking to satisfy its *Linearity Constraint*, the *Characteristic Weight* agent uses the

getNeighbourValue() to access the values of its neighbours and compare them with it is current value. As this neighbourhood do not change during the execution, this method is implemented as a getter.

5.3.2.2 Agent Criticality

As stated in the description of the entities, a *Characteristic Weight* entity is associated to a specific value of a given *Known Characteristic* for a *Known Element*. Thus, each *Characteristic Weight* agent can encounter at each life cycle two requests: one coming from the *Known Characteristic* agent and one from the *Known Element* agent which can be contradictory. In addition to this, the *Characteristic Weight* agent must satisfy the *linearity constraint* with its neighbours. Adjusting its weight to satisfy this constraint can also be contradictory with the first two requests.

The *Known Characteristic* agent request is the most critical request as it is related to the influence rank computed using the disorder that the *Known Characteristic* imposes on the *Sought Characteristic*. This disorder is related to the evolution of the two characteristics which is a good indicator on how the value of the *Sought Characteristic* must evolve given the value of the *Known characteristic* (section 5.3.2.3).

The *Known Element* agent request is related to the satisfaction of formula 2. This request comes after the first one as once the influence rank of the characteristic is found, this formula guarantees a good approximation of the known values of the *Sought Characteristic*. Finally, the adjustment of the weight given the *linearity constraint* is the less critical request as it is only used to guarantee the coherence of the estimated weights.

The *Characteristic Weight* agent adjusts its step using an adaptive value tracker (section 5.3.2.3) with an adaptive step. This step adapts given the received requests and the local representations of the agent (*linearity constraint*). Whenever no contradiction exists for the value adjustment, the *Characteristic Weight* agent adjusts its value using the adapted step. Whenever contradictions exist, the *Characteristic Weight* agent tries first to satisfy the *Known Characteristic* agent. If no request from that agent exist, the *Characteristic Weight* agent tries to satisfy the *Known Element* request. Finally, if no request exists from the other agents, it tries to satisfy the *Linearity Constraint* given its representations on its neighbours.

5.3.2.3 Data Types & Tools

Six data types (**Influence Rank**, **Weight Influence**, **Characteristic Value/Weight** and **Element Value/Weight**) and one tool (**adaptive Value Tracker** are used by the different identified agents.

1. **Influence Rank**: this data type is an *Integer* computed by the *Known Characteristic* agent only at the start of the resolution. It measures the disorder of the *Known Characteristic* in the *Sought Characteristic*. The **Higher the disorder is, less the *Known Characteristic* influences the *Sought Characteristic***. To compute this disorder, the elements of each characteristic are completely ordered by their values. The *computeDisorder()* function (algorithm 5.1) is then used to compute the differences between both orders. When

the characteristic values are constant, this order is random and the function cannot be used to compute the disorder. Thus, the constant values of both characteristics must be considered separately. Table 5.12 resumes the different possible cases. When one characteristic is constant different from zero and the second is not, the algorithm 5.2 where n is the number of the *Known Elements* is used. It is based on a permutation of the ordered elements of the constant characteristic so that the computed disorder is maximised. Once computed, this value is sent to the *Sought characteristic* entity that compares it with the disorder values of the different *Known characteristic* agents and deduces the *Influence Rank* of each *Known Characteristic* agent.

Algorithm 5.1: computeDisorder()

Input: KnownCharacOrderedElem, SoughtCharacOrderedElem
Output: disorder
$disorder \leftarrow 0$
for $i \leftarrow 0$ **to** *(nbElements-1)* **do**
  $currentElemId \leftarrow KnownCharacOrderedElem.get(i).getElemID$
  $currentElemOrder \leftarrow getElemOrder(SoughtCharacOrderedElem, currentElemId)$
  $nextElemId \leftarrow KnownCharacOrderedElem.get(i+1).getElementID$
  $nextElemOrder \leftarrow getElemOrder(SoughtCharacOrderedElem, nextElemId)$
  $disorder \leftarrow disorder + |currentElemOrder - nextElemOrder|$
end

Algorithm 5.2: Disorder computation when one characteristic is constant (<>0) and the second is not

if *(n%2==0)* **then**
  $\left(\frac{n-2}{2}\right)^2 * 2 + (n-2)$
else
  $\left(\left(\frac{n-3}{2}\right) * \left(\frac{n-3}{2}+1\right)\right) * 2 + (n-2)$
end

2. **Weight Influence**: this data type is an *Integer* used by the *Known Characteristic* agent to compare the influence rank of its computed weights for each element to its *Influence Rank*. Each *Known Characteristic* computes the sum of the absolute value of the weights associated to its different values for each known element and sends this value to the *Sought Characteristic* entity that compares it with the values received from the different *Known characteristic* agents and deduces the *Weight Influence* rank of each *Known Characteristic* agent.

3. **Characteristic Value/Weight**: this data type is used by the *Known Element* agent to store the value/weight of its characteristics. It contains the characteristic ID and its value/weight for the considered known Element.

Known Characteristic	Sought Characteristic	Disorder
is constant equal to zero	*is constant = 0*	0
	is constant <> 0	Maximum disorder
	is not a constant	Maximum disorder
is constant <> 0	*is constant = 0*	Maximum disorder
	is constant <> 0	0
	is not a constant	algorithm 5.2
is not a constant	*is constant = 0*	Maximum disorder
	is constant <> 0	algorithm 5.2
	is not a constant	computeDisorder() (algorithm 5.1)

Table 5.12 — The different possible cases to compute the disorder of a *Known Characteristic* in a *Sought Characteristic*

4. **Element Value/Weight**: the Element Value data type is used by the *Known Characteristic* agent and the *Sought Characteristic* entity to store their values for the different *Known Element*. The Element Weight data type is used by the *Known Characteristic* agent to store its weights for the different *Known Element*. It contains the element ID and its value/weight for the considered characteristic.

In addition to these data types, the *Characteristic Weight* agent uses an **Adaptive Value Tracker** tool [Lemouzy, 2011] to adjust its value given the *increase/decrease* requests of the *Known Characteristic* and *Known Element* agents. The objective of this adaptive value tracker is to track a given value that is bounded in a given interval. For that, it possesses a step Δ adjusted depending on the received requests. When the successive received requests are in the same direction (increase or decrease), the step increases using, at the beginning, a geometric progression where the current step is multiplied by 2, until a maximum step (Δ_{max}) is reached. After that, the step continues to increase using a linear progression that consists in adding to the current step the Δ_{max}. Whenever two successive received requests are not in the same direction, the step is divided by 3. This adaptive value tracker manipulates the following elements :

- the variation interval $[v_{min}; v_{max}]$ in which the tracked value can vary;
- a minimum step (Δ_{min}) computed using the width of the variation interval and the wanted precision on the tracked value;
- a Δ_{max} computed using the Δ_{min} and the number of steps required to reach this value using geometric progression before switching to linear progression. In our case, this number was studied in order to divide the variation interval into two portions and the number of steps used to cover the first portion using a geometric progression starting from the Δ_{min} is equal to the number of steps used to cover the second portion using a linear progression starting from the Δ_{max}. This guarantees an optimal exploration of the interval;
- the current step Δ that starts from a random value between Δ_{min} and Δ_{max}, and adapts its value given the different successive requests.

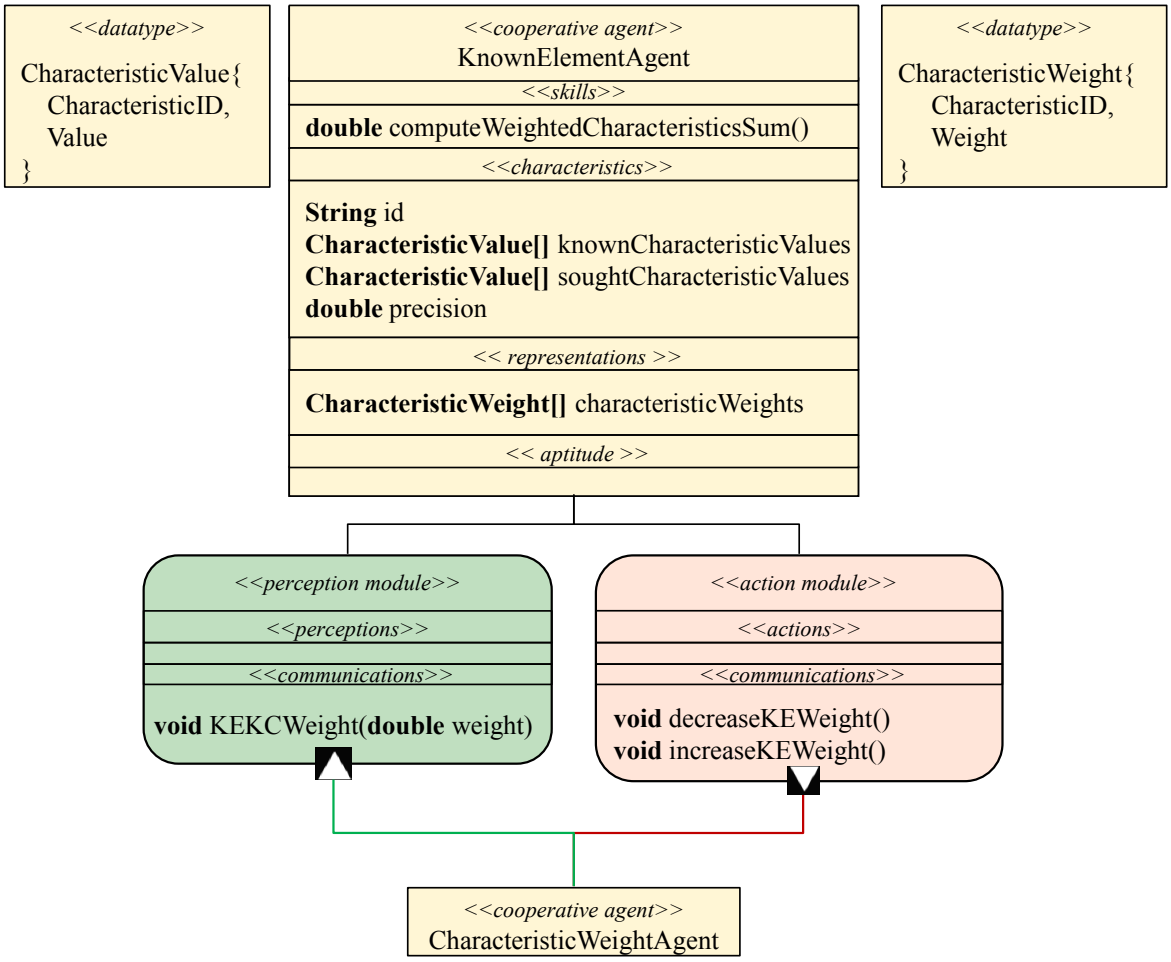

Figure 5.19 — Known Element Agent description using *AMAS-ML*

5

5.3.2.4 Known Element Agent

The *Known Element* agent represents a known element E_j of the Domain. Figure 5.19 describes its skills, characteristics, representations and interactions with other agents of the system. The description is provided using *AMAS-ML*.

The goal of this agent is to find the weights of its characteristics so as to satisfy formula 2. At each life cycle, it receives the adjusted weights $CW_{(i;j)}$ value from the *Characteristic Weight* agent (*KEKCWeight()*) corresponding to its known characteristics KC_i. Then, it computes the sum of the weighted characteristics ($weightedSum = CW_{(1;j)} * V_{(1;j)} + CW_{(2;j)} * V_{(2;j)} + ... + CW_{(m;j)} * V_{(m;j)}$) and compares it to its value for the sought characteristic.

If this *weightedSum* is higher than the value of the sought characteristic, the *Known Element* agent asks the concerned *Characteristic Weight* agents to decrease their values (*decreaseKEWeight()*).

If it is lower, it asks to increase the weight values (*increaseKEWeight()*).

When it is equal, the *Known Element* agent is satisfied and no requests are formulated. As the equality is hard to reach, the *Known Element* agent stops requesting modification

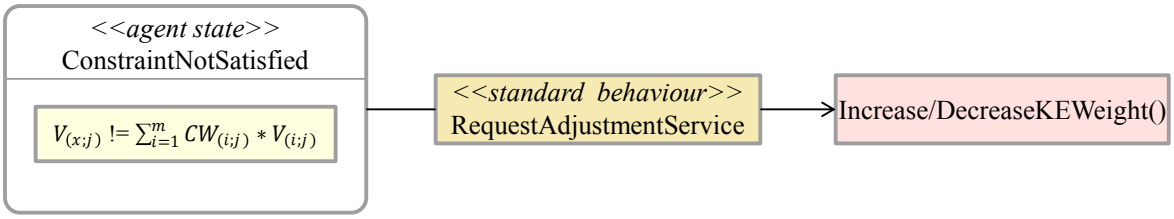

Figure 5.20 — The *request adjustment service* Cooperative Rule for solving the *better interactions partial Uselessness NCS*

whenever the difference between the *weightedSum* and its sought characteristic value is less than a precision value provided by the designer.

This agent encounters a **better interactions partial Uselessness***NCS* (section 3.5.3). Indeed, until its constraint is not satisfied the *Known Element* agent must request adjustments from the *Characteristic Weight* agents, so that a better interaction is established between them. Figure 5.20 represents this cooperative rule.

5.3.2.5 Known Characteristic Agent

The *Known Characteristic* agent represents a known characteristic entity. Figure 5.21 describes its skills, characteristics, representations and interactions with other agents of the system and its environment. The description is provided using *AMAS-ML*.

Once created, this agent computes its influence rank (section 5.3.2.3). Then, its goal is to associate a weight value to each of its values for the different elements, so that its weight influence (section 5.3.2.3) is equal to its influence rank. Thus, at each life cycle, after receiving the weight values associated to each of its elements from the *Characteristic Weight* agent (*KEKCWeight()*), it computes its weight influence and compares it to the influence rank. If both influences are equal, the *Known Characteristic* agent is satisfied, and does not formulate any request. If not, it asks the *Characteristic Weight* agents to increase (weight influence less than rank influence) their values (*increaseKCWeight()*) or to decrease (weight influence higher than rank influence) them (*decreaseKCWeight()*).

Until its weight influence is equal to its influence rank, this agent is considered in a *uselessness NCS* (section 3.5.3) as its place in the organisation is not reached yet. Thus, it sends requests to the *Characteristic Weight* agents (figure 5.22).

5.3.2.6 Characteristic Weight Agent

The *Characteristic Weight* agent represents a characteristic weight entity. Its goal is to adjust the weight associated to a given characteristic value for a specific element. Thus, it receives requests from the Known Characteristic agent and *Known Element* agent corresponding to this value. Its skills, characteristics, representations and the interactions with other agents of the system and its environment are described in figure 5.23. The description is provided using *AMAS-ML*.

When receiving the different requests from both *Known Characteristic* and *Known Element*

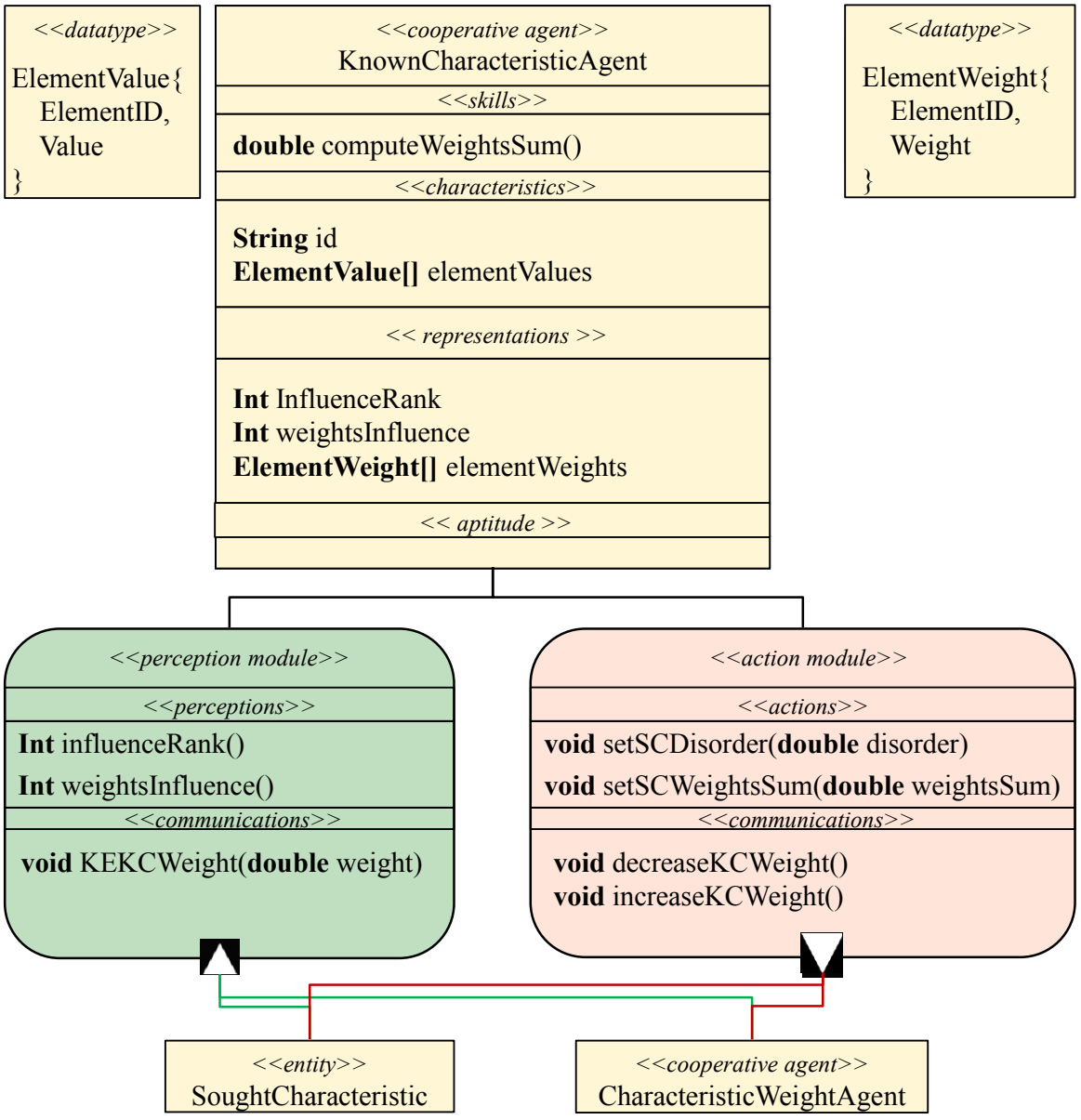

Figure 5.21 — Known Characteristic Agent description using *AMAS-ML*

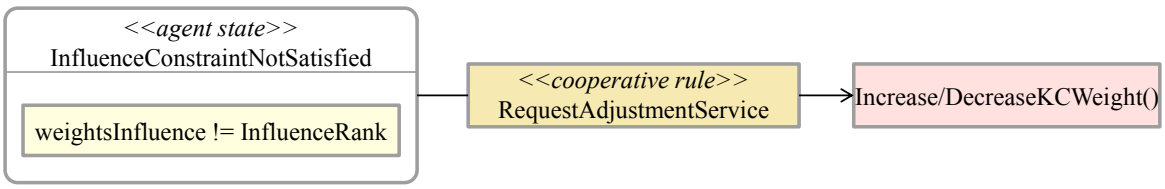

Figure 5.22 — The *request adjustment service* Cooperative Rule for solving the *Uselessness NCS*

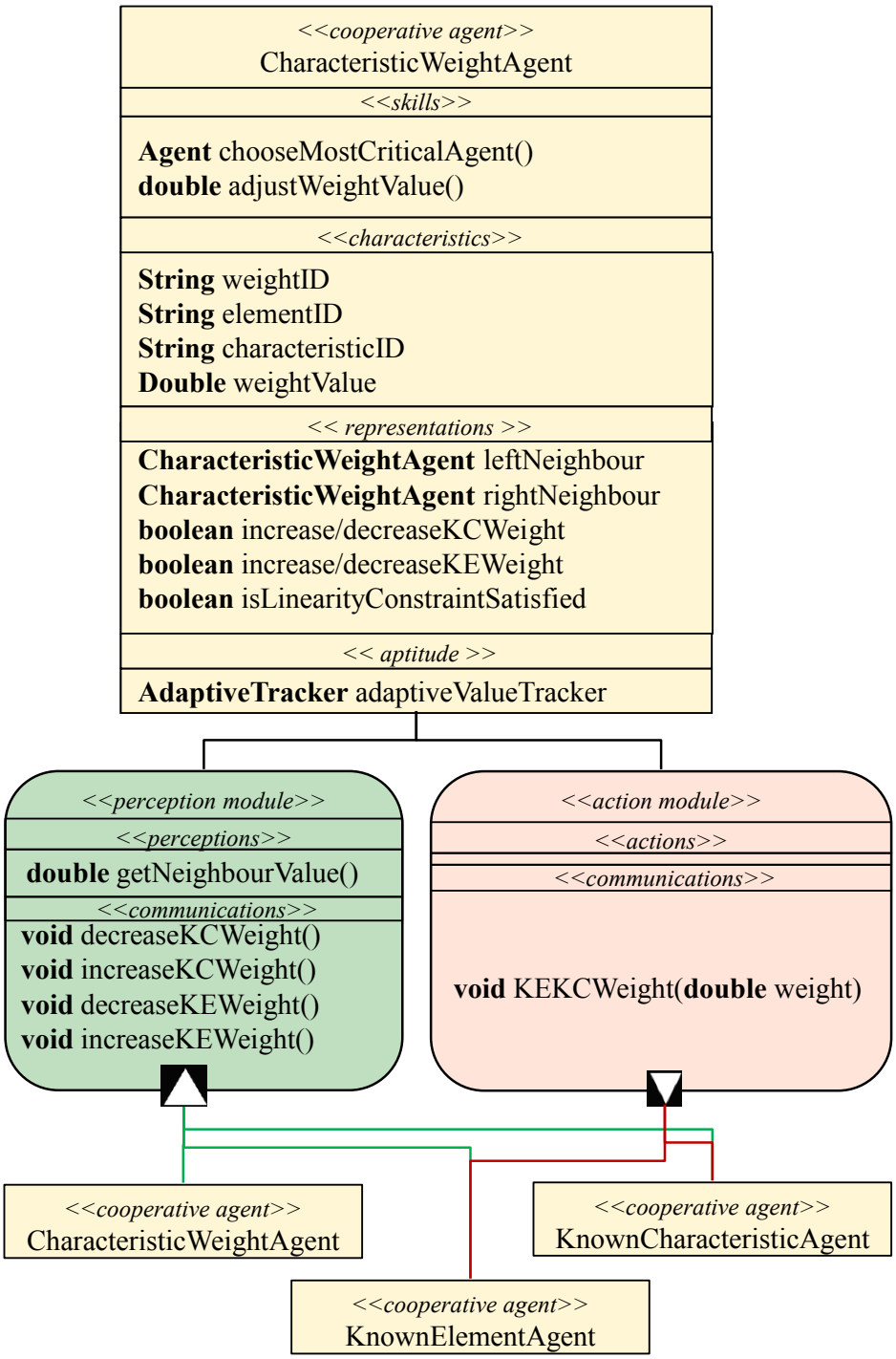

Figure 5.23 — Characteristic Weight Agent description using *AMAS-ML*

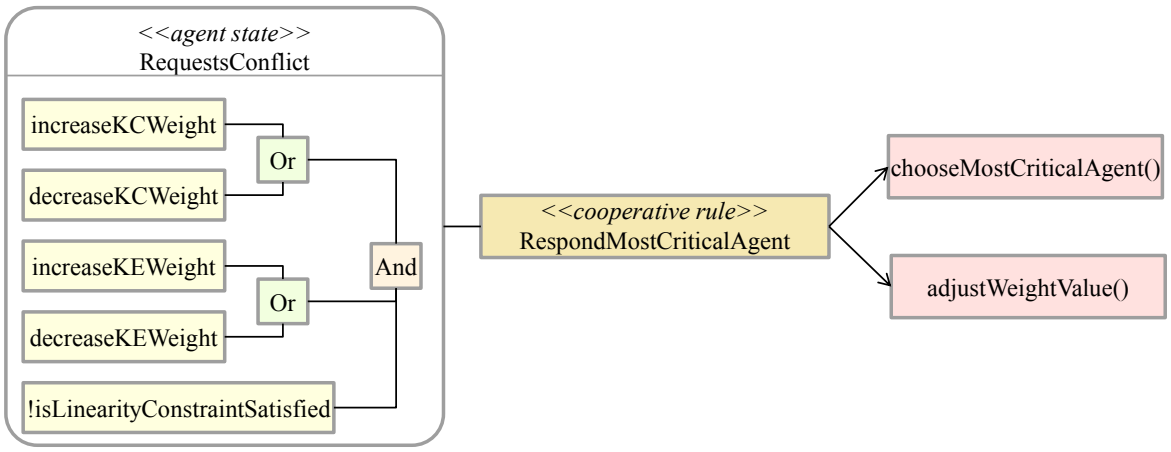

Figure 5.24 — The *respond to the most critical agent* Cooperative Rule for solving the *requests conflict NCS*

agents, it considers the most critical one given the criticality measure defined in section 5.3.2.2. Then, it adjusts its value using its adaptive value tracker defined in section 5.3.2.3 and sends the adjusted value to its *Known Characteristic* and *Known Element* agents. If no requests are received from the agents, the *Characteristics Weight* agent tries to satisfy its linearity constraint. Thus, it perceives the values of its left and right neighbours, which are the weights associated to the closed values of its known characteristic value but for different known elements, and uses the adaptive value tracker to increase or decrease its value.

This agent encounters two different *NCS*: *requests conflict* and *partial uselessness of the estimated value*.

5

The first one, an instantiation of the *Conflict NCS* (section 3.5.4), occurs when the *Characteristic Weight* agent receives requests from both *Known Characteristic* and *Known Element* agents. It solves this *NCS* by choosing the most critical one given the criticality measure defined in section 5.3.2.2 (figure 5.24).

The second one, an instantiation of the *Uselessness NCS* (section 3.5.3), occurs when the linearity constraint is not satisfied as the agent considers that it has not found the best place in the organisation regarding the representations it has of its neighbours (figure 5.25). It solves this *NCS* by adjusting its value.

5.3.3 SAPBR Results & Discussions

The *SAPBR* approach is currently applied to the *Aircraft Preliminary Design problem*. The data base involves 129 elements each described using 163 characteristics. Five test cases have been provided by the domain expert (Thierry Druot, R&D engineer at Airbus). Only one characteristic is known for the first case, five for the second case, fifteen for the third case, twenty-one for the forth case and thirty-six for the fifth case. For each case three different test files have been constructed with different values for the known characteristics. Each file has been constructed by extracting one element from the data base and keeping only for that element the values of the known characteristics defined by the corresponding test case.

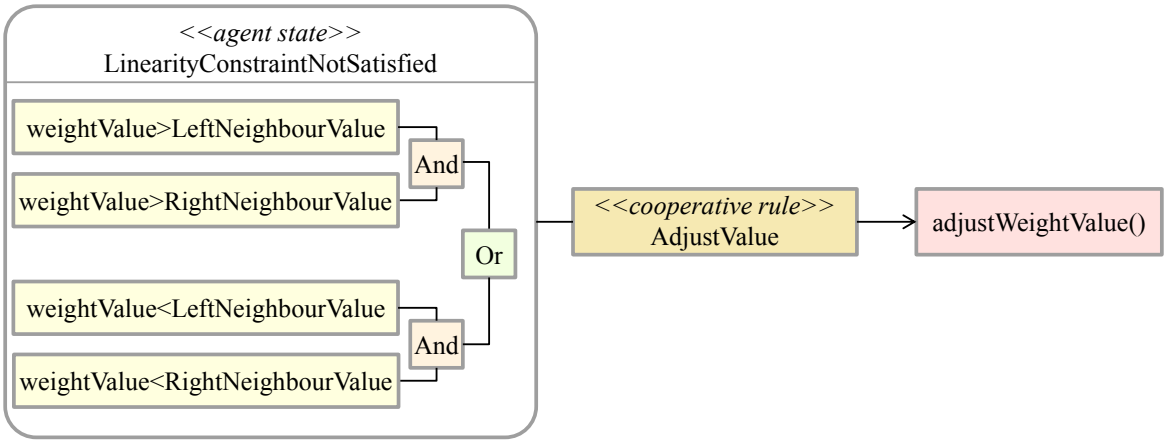

Figure 5.25 — The *adjust value* Cooperative Rule for solving the *partial uselessness of the estimated value NCS*

Then, this element is considered as the new element to estimate. Once *SAPBR* estimates the new element, the results are compared with the original element. The simulation was launched on an Intel Core 2 Duo 2.4GHz RAM:4.0Go. The code was implemented using Java and includes more than 80 classes with 24 classes generated using *MAY*.

Table 5.13 summarizes for each file (launched one time), the number of estimated sought characteristics divided into three classes depending on the distance between the estimated value *estimatedValue* and the original value *realValue* and the percentage of values estimated with a distance less than 13%. The distance has been computed using two different formulas.

First, according to the expert, estimating a value with a distance ($\frac{|realValue - estimatedValue|}{realValue} * 100$) less than 13% of the real value is an acceptable result. In our case, more than 75% of the characteristics are estimated with a distance less than 13%. Thus, the obtained results present a good *Accuracy/Quality* (section 4.2.1). In addition to this, for some characteristics, having a higher percentage is normal and can be considered as a normal result (i.e for characteristics not related or lightly related to the known characteristics given test case). But as we do not have sufficient information to detect such characteristics, additional work with the expert is required to completely validate the behaviour of the agents.

As this first formula does not provide good estimation when the *realValue* is close to zero (due to the division by the *realValue*), the distance between the *estimatedValue* and the *realValue* relatively to width of the values interval (*valueIntervalWidth*) of the characteristic ($\frac{|realValue - estimatedValue|}{valueIntervalWidth} * 100$) has been studied. Here also, the results show a good *Accuracy/Quality* as more than 88% of the characteristics are estimated with a distance less than 13%.

As stated in the description of the *Known Element* agent (section 5.3.2.4), estimating the weight so that the exact value of a sought characteristic is retrieved is difficult and a precision value defining the allowed difference between the estimated value and the real one have been introduced. In our case, we fixed this precision to 10^{-4}. The stopping criterion of the system is the number of life cycles of the agents. For this study, the number is fixed to 100 life cycles. Other stopping criteria have been identified such as the satisfaction of *Known Element* and *Known Characteristic* agents or the non-existence of possible adjustments for the

	$\frac{\lvert realValue - estimatedValue \rvert}{realValue} * 100$				$\frac{\lvert realValue - estimatedValue \rvert}{valueIntevalWidth} * 100$			
Files Names	**0-13%**	**13% - 50%**	**50% - 100%**	% under 13%	**0-13%**	**13% - 50%**	**50% - 100%**	% under 13%
File1.1KC	106	55	1	65,5	147	14	1	90,7
File2.1KC	108	52	2	66,6	157	4	1	96,9
File3.1KC	118	44	0	72,8	157	3	2	96,9
File4.5KC	122	28	8	77,2	141	15	2	89,2
File5.5KC	129	16	13	81,6	135	19	4	85,4
File6.5KC	121	34	3	76,6	111	43	4	70,3
File7.15KC	107	33	8	72,3	137	11	0	92,5
File8.15KC	74	71	3	50	119	26	3	80,4
File9.15KC	120	26	2	81	121	26	1	81,75
File10.21KC	116	21	5	81,7	124	17	1	87,3
File11.21KC	103	35	4	72,5	109	33	0	76,7
File12.21KC	113	28	1	79.5	106	34	2	74,7
File13.36KC	98	28	1	77	115	12	0	90,5
File14.36KC	109	17	1	85,8	118	9	0	92,9
File15.36KC	110	15	2	86,6	112	14	1	88,2

Table 5.13 — The number of estimated sought characteristics for two computation formulas of the distance. The number is divided into three classes depending on the distance between the estimated value and the original value and the percentage of values estimated with a distance less than 13%

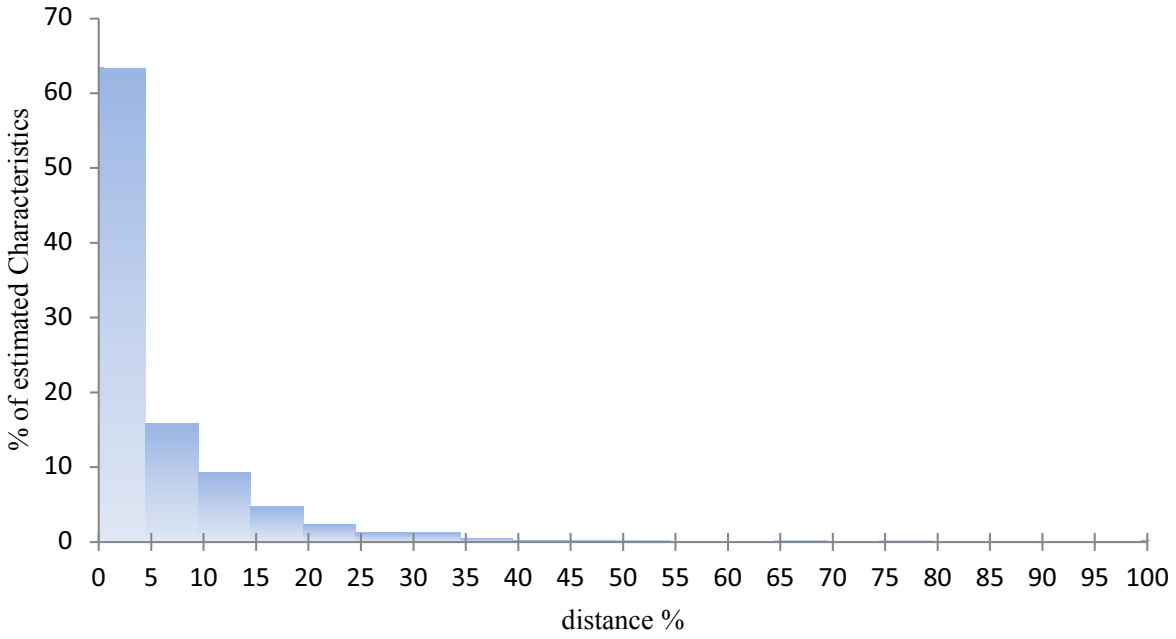

Figure 5.26 — The percentage of estimated sought characteristics classified accordingly to the distance ($\frac{|realValue - estimatedValue|}{valueIntervalWidth} * 100$) between the estimated value and the original value

Characteristic Weight without deteriorating the solution. Those criteria will be implemented in the next version of the system.

Figure 5.26 shows us that given this precision and this stopping criterion, more that 63% of the sought characteristics are estimated with a distance less than 5%. The percentage of the sought characteristics estimated with distances higher than 15% is less than 5%. These results demonstrate the adequacy of the system and validate the agents' behaviours.

In table 5.14, the average time needed by a multi-agent system to estimate the value of a *Sought Characteristic* is given, in addition to the total computation time for each file. We note that when additional characteristics are known, additional time is required before the system reaches a complete solution. This can be explained by the increase in the number of agents as each known characteristic is represented by a *Known Characteristic* agent and each of its values is represented by a *Characteristic Weight* agent. For example, from the second case (5 known characteristics) to the fourth case (21 known characteristics) the number of agents increases from 123.082 agents ((129 known element agents + 5 known characteristic agents + 5*129 characteristic weight agents) * 158 sought characteristic) to 405.978 agents ((129 known element agents + 21 known characteristic agents + 21*129 characteristic weight agents) * sought characteristic). Thus, additional time is required to exchange information between the different agents and stabilize the system. Another point that can impact the time is the stopping criterion. The system stops after a given number of agent life cycles, even if a solution is reached before. Further studies are required to improve and implement other more appropriate stopping criterion.

However, as each *Sought Characteristic* can be estimated by its own multi-agent system, the resolution can be distributed among different computers and only the average time

Files Names	**Average Time By Sought Characteristic (*ms*)**	**Total time (*ms*)**	**Number of agents**
File1.1KC	42	6.806	41.958
File2.1KC	43	7.058	41.958
File3.1KC	42	6.839	41.958
File4.5KC	172	27.204	123.082
File5.5KC	173	27.351	123.082
File6.5KC	172	27.298	123.082
File7.15KC	620	$91.808 \approx 1{,}5min$	307.692
File8.15KC	617	$91.403 \approx 1{,}5min$	307.692
File9.15KC	619	$91.725 \approx 1{,}5min$	307.692
File10.21KC	930	$132.117 \approx 2{,}2min$	405.978
File11.21KC	921	$130.833 \approx 2{,}1min$	405.978
File12.21KC	919	$130.543 \approx 2{,}1min$	405.978
File13.36KC	3.739	$474.953 \approx 8min$	610.743
File14.36KC	3.749	$476.159 \approx 8min$	610.743
File15.36KC	3.764	$478.108 \approx 8min$	610.743

Table 5.14 — The average time needed by a multi-agent system to estimate the value of a *Sought Characteristic*, the total computation time and number of agents for each file.

needed to estimate the value of one sought characteristic is significant. Regarding this time, only a few seconds (4*s* for the fifth case) are needed to propose a value to the designer. Thus, it is possible to transform the *SAPBR* software in an interactive tool, that can then profit from the feedbacks of the designer.

Concerning the decentralization criterion (section 4.3.2), the **action locality** of the agents can be underlined as only local information is used by the agents to decide and act. As each cell of the data base is represented by a *Characteristic Weight* agent, increasing the size of the data base will increase the number of cooperative interacting entities. But, considering the action locality of the agents and the **low computational complexity level** of their behaviour, this increase of the agent number should not impact the results exponentially or cause system breakdowns. We can note that distribution of the information among the agents is done such that each agent is provided with an **initial small amount of knowledge** required for its reasoning. The *Known Characteristic* and the *Known Element* agents do not acquire any additional information. The *Characteristic Weight* agent only requires to register the last received adjustment direction so they can adapt their value.

Table 5.15 summarises for the evaluation of *SAPBR* which of the defined evaluation criteria from chapter 4 have been considered and justifies why the other criteria were not considered.

Evaluation Criterion	Considered	Not Considered	Remarks
Resolution Time	✓		
Number of messages		χ	As no external events occur, the number of message exchanged by the agents at each life cycle is constant. Still, this criterion must be studied before the deployment of the system and before adding the interactions with the engineer.
Quality of solution	✓		
Progress		χ	This study is the first time the AMAS theory was confronted to such new original class of problems. That is why our aim was to show the adequacy of our model to solve such application before looking to these criteria.
Memory usage		χ	
Robustness		χ	
Local computational complexity	✓		
Decentralisation	✓		
Action locality	✓		
Initial and acquired knowledge	✓		
Agent number influence	✓		The system does not seem to suffer from any scalability issue as it was tested with more than 400000 agents. Still further studies are required.
Analysis and identification of autonomous agent		χ	Those criteria depend on the *AMAS4Opt* agent model used in this study. See section 5.4 at the end of the chapter for additional information.
Distribution & deployment ease		χ	
Genericity		χ	

Table 5.15 — Summary of the considered and not considered evaluation criteria introduced in chapter 4 for the evaluation of *SAPBR*.

5.4 AMAS4Opt Evaluation

In this chapter, the *AMAS4Opt* agent model has been instantiated for the solving of two well-known optimisation problems. Both instantiations underlined different advantages of the model:

- the defined **roles** are conceptually close to the behaviour of entities involved in optimisation under constraints problems. The constrained role matches the entities submitted to constraints whose solving lead to the global solution of the problem. The service role matches the behaviour of entities that are in the system to help other entities solve their constraints. In *SAFlex* this identification is straightforward as it is identical to the actual functioning of the entities. Indeed, in one hand, we have the containers that require to be treated on specific stations, and on the other hand, the stations that are qualified for specific treatments and can help the containers to reach their goals. In *SAPBR*, this identification is less straightforward but still intuitive. Indeed, given the definition of the problem, we know that the already known knowledge (*the known values of the different characteristics*) is required to help estimate the values of the *sought characteristics*. A further analysis of the problem was required to define the resolution process. The aim of this analysis was the find how the different entities of the problem can interact to solve the problem. From these interactions, the agent roles have been identified.

- the agents of this model are provided with **local goals, behaviours and interactions**. They locally perceive their environment and update their representations and knowledge. Thus, they are each able to handle the dynamics they perceive and rapidly adapt to new situations allowing a better handling of the problem complexity. This advantage is easily underlined by *SAFlex* as agents faces concrete dynamic events such as stations breakdowns and the obtained results show the system's robustness. In *SAPBR*, the complexity is due to the non-linear interdependencies that exist between the different characteristics. Considering these interdependencies from the local agent point of view enables a better handling of this complexity. Indeed, the agents only considers local approximations of these interdependencies which in very small intervals can be considered as linear. Thus, no complete estimation of them is required.

- the defined **agent criticality** measure guides the cooperative behaviour of the agents increasing the equilibration of the satisfaction degree of agents. The consequence is an efficient exploration of the search space. For instance, in *SAFlex*, the measure of the container criticality used by the stations to decide which containers to treat enable the system to equilibrate the agents' dissatisfaction degrees resulting in less tardiness. In *SAPBR*, the defined agent criticality helps the *Characteristic Weight Agent* to better estimate their value by replying to the more relevant agent.

- the **cooperative rules** guide the agents behaviours to reach good solutions by always seeking to be cooperative towards their environments. In comparison to the *Distributed Constraint Optimization Problem (DCOP)* based approach where a translation of the problem into the *DCOP* formalism is required, in our case, an identification of the

NCS is required. Depending on the problem this translation can be difficult. The identification of the *NCS* can also be difficult. Still, as the agent behaviour is close to the actual functioning of the entities and as their behaviours are more intuitive to define, our agent model provides for each *NCS*, specific states in which the agents can be confronted to it. This allows a more intuitive identification of these *NCSs*.

- the agent model presents a sufficient level of **genericity**. Indeed, the two applications have been chosen because of their very different characteristics. The complexity of the *Manufacturing Control Scheduling Problem* is due to the combinatorial size of the search space and to the need to manage dynamics such as stations breakdowns or the arrival of new containers. In the second one, *Design of Complex Products*, the complexity is due to the volume of manipulated data that is produced by unknown (or at least not precisely known) and often complex interrelated non-linear functions. Both developed *MAS* using our *AMAS4Opt* agent model showed encouraging results.

To conclude, we feel confident that the *AMAS4Opt* model is intuitive, and comprehensive enough so that a non *MAS* expert engineer should be able to understand, instantiate and extend to a large variety of complex optimisation under constraints problems, as it matches the problems natural description and the real-world entities functioning.

The next step of this study is to make non *MAS* expert engineers use our model and validate the advantages underlined by our experience.

5.5 Conclusion & Perspectives

In this chapter, two adaptive multi-agent systems, *SAFlex* for the Manufacturing Control Scheduling Problem and *SAPBR* for Complex Problem Design, have been implemented using the generic agent model defined in chapter 3 and evaluated using a set of defined criteria in chapter 4.

Concerning *SAFlex*, two cooperative agents have been identified: the Container agent having the constrained role and the Station agent having the service role. *SAFlex* evaluation was done on repeated runs on a set of 45 different scenarii, representative of the variety one can find in real-world manufacturing situations. We feel confident in the efficiency and robustness of this approach. In particular, we aimed at emphasising the importance of taking into account dynamic events for the purpose of realism, and showed how *SAFlex* seamlessly takes these dynamic events into consideration during the reasoning and decision making of the agents. As this decision making is done continuously and locally, the complexity of the algorithms stays the same and the calculation time only increases slightly. Scaling-up to problems containing thousands of containers and stations seems quite realistic and further work is planned to demonstrate this.

Comparing *SAFlex* with two very different state-of-the art methods, the first one, *L/O* (Learning/Optimization) being an off-line scheduling technique, the second, *OFO* (On-line Forward Optimization), a rolling time scheduling technique, ensures that advantages and drawbacks are evaluated. Indeed, these resource allocation problems are typically

multi-objective problems where optimizing a given objective usually penalizes another. 5 objectives were used in our experimentation: the number of tardy containers, mean Tardiness, maximum Tardiness, makespan and calculation time. *SAFlex* showed that it ensured an efficient and satisfactory equilibration of the different objectives as it does not worsen one objective in order to improve another one. Indeed, no explicit priority is associated to each objective as in global objective functions.

As seen in section 5.2.4.4, *SAFlex* & *OFO* both compute interesting results. In addition to comparing these approaches on larger instances with a higher level of dynamics and complexity, a further work is planned to combine their advantages in order to improve their results.

Concerning *SAPBR*, three cooperative agents have been identified: the Known characteristic and Known Element agents having the constrained role and the Characteristic Weight agent having the service role. *SAPBR* is a new approach to solve the complex product design problem. Indeed, the most common approaches use explicit models representing the disciplines of a given domain. The *SAPBR* approach on the other hand simply uses the already known elements of the domain and proposes plausible values for new products being designed. An interesting part of the development of this application was the problem analysis phase, before the identification of the cooperative agents. This problem analysis consists in describing the problem bearing in mind the generic agent model and the agents role. For that, three questions have been answered: **what is required to solve the problem**, **what is the solving process** and finally, **what are the entities participating in the solving and what are their respective local roles**.

The evaluation of *SAPBR* was done in the *Aircraft Preliminary Design* domain on 15 test files with different number of known characteristics, with a realistic base of 129 elements and 163 characteristics each. This study was a first attempt to solve such complex problems with a new original approach. This approach showed very encouraging results where other approaches struggle. In addition to this, it underlines the fact that by focusing on relatively simple cooperative agents behaviours and interactions, complex problems can be solved.

Further studies and evaluation measures such as the non determinism and the scalability are needed in order to completely validate the agents behaviour and improve the criticality measure. Indeed, currently in the Characteristic Weight agent decision process the criticality degrees of the requests are strictly ranked depending on the type of the requesting agents (first the Known Characteristic agent, second the Known Element and third the Linearity Constraint). In our point of view, a better cooperative criticality measure can be defined which would equilibrate more cooperatively the requesting agents.

The next step of this study is to improve our agents behaviour to be able to detect errors or uncertainties in the data base (the expert explained that due to human errors the values are only partially trustworthy).

Then, the reasonable calculation time required by the system to find solutions encourage us to transform *SAPBR* in an interactive software that can be used on-line by the designer. For that, the agents behaviour must also be improved to handle the dynamic events such as the introduction of new elements or known characteristics, the deletion of already defined elements or characteristics, or the transformation of a sought characteristic in a known

characteristic.

To summarize, both designed systems have shown the adequacy of our generic model and the associated *AMAS* theory compliant behaviours to solve complex problems under constraints. In addition to this, it can be noted that the *AMAS4Opt* agent model has been instantiated on two very different complex problems. This instantiation has been straightforward and underlines the importance of this specialization of the *AMAS* theory for optimization under-constraints of complex problem solving. Indeed, until now this theory remained at a high abstraction level and specializing it will help us to promote its usage for engineers that are not agent experts which is the next step of our study. Currently, this agent model is being instantiated on other applications by PhD students in the team in order to validate its usage.

Conclusion Générale

The conclusion in english starts page 159

Nous sommes quotidiennement confrontés à la résolution de problèmes d'optimisation sous contraintes. Ces problèmes se caractérisent par un degré élevé de complexité rendant cette résolution difficile. A l'origine de cette complexité croissante, ce travail souligne quatre points importants:

1. *la dynamique de l'environnement et du système lui-même rendant impossible la mise en place d'un contrôle global du système;*
2. *le volume et la diversité des données manipulées;*
3. *l'existence d'interdépendances insuffisamment définies entre les paramètres du problème;*
4. *les dimensions multi-objectif et multidisciplinaire des problèmes étudiés.*

Dans ce travail, le modèle d'agents AMAS4Opt pour la conception de systèmes multi-agents pour la résolution des problèmes d'optimisation sous contraintes a été défini et décrit. Ce modèle spécialise la théorie des AMAS pour la résolution de ce type de problème. Ce modèle se caractérise par sa généricité, sa proximité à la description naturelle de ces problèmes, sa décentralisation du contrôle et distribution du calcul. Ceci permet une instanciation directe du modèle pour la résolution d'une large variété de problèmes d'optimisation sous contraintes.

De plus, ce travail s'est intéressé à la définition d'un ensemble de critères pour l'évaluation des approches auto-adaptatives. Ces approches se différencient des approches classiques par leur capacité à résoudre des problèmes présentant une complexité élevée par l'utilisation de mécanismes self- permettant d'améliorer les performances et la robustesse des systèmes. Les critères définis permettent de souligner ces avantages afin d'améliorer l'acceptation de ces systèmes au niveau industriel.*

Cette conclusion est organisée comme suit: premièrement, les contributions au niveau applicatif et scientifique sont présentées. Par la suite, les perspectives et les améliorations possibles de ce travail sont discutées.

Contributions au niveau applicatif

Au niveau applicatif, les principales contributions de ce travail sont:

• ***La définition d'une architecture d'agent générique*** *utilisant l'outil* Make Agents Yourself. *Cette architecture a été validée lors de son utilisation pour la conception des agents des deux systèmes développés.*

• ***L'instanciation des règles coopératives*** *définies dans le modèle d'agents. Tout au long de ce travail, l'importance de la coopération comme guide du comportement des agents a été soulignée. Le modèle d'agents proposé définit un ensemble de règles coopératives permettant aux agents d'anticiper, de détecter et de réparer des situations d'échecs à la coopération. La définition de ces règles a été déduite du comportement réel des entités impliquées dans des problèmes d'optimisation sous-contraintes. Leur identification et instanciation pour la résolution des deux problèmes considérés, nous a permis de valider leurs aspects intuitif et direct.*

• ***La résolution de deux problèmes clés de l'optimisation sous contraintes****. En effet, deux systèmes multi-agents adaptatifs:* **Self-Adaptive Flexible Scheduling (SAFLEX)** *pour la planification des tâches dans une usine et* **Self-Adaptive Population Based Reasoning (SAPBR)** *pour la conception de produits complexes. Ces deux systèmes ont été conçus en utilisant le modèle d'agents AMAS4Opt. Cette conception a souligné l'aspect intuitif de notre modèle et l'aide supplémentaire qu'il offre aux concepteurs par la proposition d'une architecture d'agent prête à instancier.*

• ***La comparaison de SAFLEX*** *avec deux approches basées sur l'apprentissage par renforcement. De cette comparaison, l'importance de la criticité des agents définie comme guide de la coopération et le moyen pour atteindre de bonnes solutions a été soulignée.*

• ***L'utilisation des critères d'évaluation.*** *Les performances des deux systèmes développés ont été analysées en utilisant les critères d'évaluations définis. Ceci a permis de souligner les avantages et limites de notre approche distribuée surtout concernant la gestion de la dynamique.*

• ***L'adéquation du raisonnement local pour la résolution de problèmes complexes.*** *Les résultats obtenus avec SAPBR ont confirmé l'efficacité du comportement simple et local des agents pour la résolution de problèmes complexe et difficile où une résolution globale est impossible.*

Contributions au niveau scientifique

Au niveau scientifique, les contributions majeures de ce travail sont le modèle d'agents, la définition de la criticité des agents ainsi que la définition des critères d'évaluation.

• ***Le modèle d'agent AMAS4Opt*** *défini comme une importante spécialisation de la théorie des AMAS, a permis d'enrichir cette théorie et faciliter son usage pour la conception de systèmes multi-agents pour la résolution de problèmes d'optimisation sous-contraintes. Ce modèle présente un niveau d'abstraction bien choisi lui permettant d'être instancié d'une manière directe pour la résolution d'une large variété d'applications du domaine. Dans ce modèle, deux rôles d'agents ont été identifiés: le rôle contraint et le rôle service. Le premier concerne les agents soumis à des contraintes et nécessitant pour leur résolution l'aide des agents ayant le rôle service. Les agents de ce modèle possèdent des buts locaux, des comportements locaux, et interagissent d'une manière coopérative. Ils sont conçus de manière à suivre du mieux possible le comportement des entités du problème. La localité de leurs actions et leur autonomie permettent une meilleure gestion de la dynamique et accroissent la robustesse du système.*

• ***La définition de la criticité des agents.*** *Ce travail a souligné l'importance de la coopération pour guider le comportement des agents. La mesure de criticité a été définie comme élément central pour le processus de décision coopératif des agents. Nous avons souligné l'importance de cette mesure pour permettre aux agents d'atteindre de bonnes solutions.*

• ***Les critères d'évaluation*** *ont été définis pour rendre compte de l'importance et de l'efficacité des approches auto-adaptatives par rapport aux approches classiques. Contrairement aux approches classiques où des preuves formelles peuvent exister sous certaines conditions, l'évaluation des approches auto-adaptatives reste une tâche difficile. Dans ce travail, une première étude concernant l'évaluation de ces systèmes, la validation de leur comportement et la mise en évidence de leurs avantages par rapport aux approches classiques, a été établie.*

Perspectives au niveau applicatif

Au niveau applicatif, les perspectives concernent:

• ***Le développement d'outils de conception*** *implémentant notre modèle d'agent et guidant les ingénieurs lors de la conception d'agents basés sur ce modèle. Ces outils sont à intégrer à la méthodologie ADELFE pour faciliter la conception de systèmes multi-agents adaptatifs pour la résolution de problèmes d'optimisation sous-contraintes.*

• ***La validation du modèle d'agent*** *par son instanciation sur d'autres applications du domaine.*

• ***La validation des caractéristiques de notre modèle*** *par son utilisation par des ingénieurs non-experts de la théorie des AMAS. Le but étant de souligner les limites de ce modèle et ainsi pouvoir l'améliorer.*

• ***La combinaison des avantages de SAFlex et des approches basées sur l'apprentissage par renforcement.*** *En effet, la comparaison de ces différentes techniques de résolution a souligné pour chacune d'elles ses limites et ses avantages. Une étude intéressante que nous souhaitons continuer en collaboration avec l'équipe Vakgroep Informatietechnologie est l'évaluation des ces approches sur des cas présentant des degrés élevés de complexité afin de pouvoir combiner leurs avantages et améliorer les résultats.*

• ***La Comparaison de SAFLex et SAPBR avec d'autres approches distribuées et auto-organisées*** *afin d'analyser et d'étudier des critères liés à la décentralisation du contrôle et la distribution du calcul.*

• ***La comparaison de SAPBR avec des approches basées sur le raisonnement par cas ou des méthodes statistiques****. En effet, par leur définition, les approches basées sur le raisonnement par cas ou les méthodes statistiques semblent être pertinentes pour la résolution du problème traité par SAPBR.*

• ***Le développement d'outils pour l'évaluation*** *des systèmes auto-adaptatifs utilisant les critères d'évaluation définis dans ce travail. Ces outils devront guider l'évaluation de ces systèmes et intégrer des mesures génériques permettant de valider la robustesse et l'adaptation des ces systèmes à des environnements fortement dynamiques.*

Perspectives au niveau scientifique

Au niveau scientifique, les perspectives concernent:

• ***L'amélioration de la criticité des agents**. En effet, cette mesure est pour l'instant dépendante de l'application traitée. Étant donnée son importance pour le comportement coopératif des agents et son influence sur les résultats, cette mesure est parfois difficile à définir. Ainsi, un processus générique permettant aux agents de découvrir cette mesure et de la normaliser entre eux doit être développé.*

• ***L'amélioration des critères d'évaluation**. Dans ce travail, une première étude concernant comment des critères d'évaluation déjà existants peuvent être utilisés pour souligner l'adaptation et la robustesse des approches distribuées et décentralisées a été établie. Nous souhaitons améliorer la définition de ces critères par l'introduction de mesures prenant en compte la dynamique et les étendre par l'exploration d'autres critères.*

• ***La définition de modèles d'agents basés sur la théorie des AMAS pour d'autres types de problèmes**. Le succès de l'instanciation de notre modèle pour la résolution de deux applications présentant des caractéristiques variées, et la facilité amenée par ce modèle à la théorie des AMAS et à la méthodologie ADELFE, nous encouragent à définir d'autres modèles d'agents spécialisant la théorie des AMAS pour d'autres types de problèmes.*

Conclusion & Perspectives

« The most exciting phrase to hear in science, the one that heralds discoveries, is not 'Eureka!' but 'Now that's funny?' »

Isaac Asimov

THE research presented in this document started from the observation that optimisation under-constraints problems are omnipresent nowadays and showing a growing complexity which makes their solving really challenging. We identified four important reasons for this increasing complexity:

1. the dynamics of the environment and the system itself making a global control of the system hard to achieve or even impossible;
2. the volume and diversity of interrelated data;
3. the existence of insufficiently defined non-linear interdependencies between the parameters of the problem;
4. the existence of multi-objective and multidisciplinary dimensions.

In this work, we presented **a generic approach to design multi-agent systems for optimisation under-constraints complex problem solving**. This generic approach is an important specialisation of the *Adaptive Multi-Agent System (AMAS)* theory for this type of problems. Key aspects of this approach are genericity, simplicity, decentralization of control and distribution of computation. These aspects enable a straightforward instantiation of the approach for a large variety of optimisation under-constraints problems. Decentralisation and distribution are essential as they provide flexibility and robustness to the designed system.

Another important subject covered in this work is the definition of criteria for the evaluation of self-adaptive approaches. Such approaches differ from classical ones by their ability to treat problems in their whole complexity by proposing self-* mechanisms that increase performance and robustness. The evaluation criteria we propose point out these advantages in order to ease their acceptance for the industry.

This conclusion is organised as follows: first, we summarize the *main applicative and scientific contributions* of this work. Next, we present several suggestions for *improving this work and further research*. It concludes with final words on my personal experience.

Applicative Contributions

At the applicative level, the concrete contributions of our research are:

• **The definition of a generic agent architecture** called **Resolution Agent** using *Make Agents Yourself (MAY)*. We validate this architecture by using it to design the agents of two developed systems.

• **The instantiation of the defined cooperative rules**. During this work, we underlined the importance of **cooperation** to guide the behaviour of agents. For that, we defined a set of cooperative rules that helps agents to anticipate or detect and repair cooperation failures. We designed two multi-agent systems where those rules are instantiated. As the defined cooperative rules have been deduced from the actual functioning of the problem entities, their identification and instantiation on both applications was verified as being intuitive and straightforward.

• **The solving of two key optimisation under-constraints problems** by the development of two self-adaptive multi-agent systems: *Self-Adaptive Flexible scheduling (SAFlex)* for the *Manufacturing Control Scheduling* problem and *Self-Adaptive Population Based Reasoning (SAPBR)* for the *Complex Product Design* problem. We designed both systems using the *AMAS for Optimisation (AMAS4Opt)* agent model. The design was straightforward and we underlined the ease this model brings to the *ADELFE* methodology by the proposition of a complete agent architecture ready to be instantiated.

• **The comparison of *SAFlex*** with two reinforcement learning approaches. From this comparison, we notice the importance of the agent criticality measure for the cooperation.

• **The usage of the evaluation criteria**. Both developed applications have been evaluated using the evaluation criteria we addressed. This evaluation contributed to a better understanding of the strengths and weaknesses of our decentralized approaches especially regarding the handling of dynamics.

• **The adequacy of local treatments to solve challenging problems**. The results we obtained with *SAPBR* have underlined how simple and local agent behaviours can solve very challenging and complex problems where currently no models of the problem are known.

Scientific Contributions

The main scientific contribution of this work is the *AMAS4Opt* agent model defined as a specialisation of the *AMAS* theory. This theory had already shown its adequacy to solve a large variety of problems but required specific agent expert knowledge.

The study on the state of the art has addressed several **limits of existing approaches**.

First, the **translation of the problem in a given framework** such as the *Distributed Constraint Optimization Problem (DCOP)* framework generally requires a certain level of understanding of the global problem to be able to transform it in the given framework. Second, the **handling of dynamics** is hardly taken into account as it implies modifications on the structure of the problem and thus makes a global solving of the problem impossible. Third, **parameter tuning** is still strongly application-dependent and a difficult task as no guide exists for the parameters definition that can impact the performance.

This state of the art has also underlined the importance of the **distribution of computation** and **control decentralisation** so that the flexibility and robustness of the systems increase. It also shows how being **close to the problem definition** and **concentrating on the local behaviours and interactions of the entities** can lead to a **better understanding and solving** of the problem.

We have also underlined the fact that **cooperation** is a central and fundamental coordination mechanism in the different decentralised approaches that enables the system to converge and reach good global solutions.

The *AMAS* theory takes advantage of a specific definition of cooperation by proposing to build adaptive multi-agent systems while concentrating on local and cooperative interacting agents. The main limit of this theory is that it remains a **high level and general guide** which renders its usage by non *AMAS* experts difficult.

The concrete contributions of our research are:

• **The *AMAS4Opt* agent model** we have defined as an important specialisation of the *AMAS* theory so that its usage is facilitated for optimisation under-constraints problem. The proposed model presents a **carefully chosen level of abstraction** so as to be easily understandable, instantiated and extended to a large variety of optimisation under-constraints problems. Two agent roles have been identified. The *Constrained Role* possessed by agents submitted to constraints and considered as the problem initiators. The *Service Role* possessed by agents that can help agents having the *constrained role* to solve their constraints and reach their goals. The agents of this model are provided with **local goals, behaviours, cooperative rules and interactions** that match seamlessly with the behaviours of the problem entities. The locality and autonomy provided to the agents enable to **better handle the dynamics** and **rapidly adapt**.

• **The definition of criticality for agents in the *AMAS4Opt* model.** During this work, we underlined the importance of **cooperation** to guide the behaviour of agents through *cooperative rules*. We defined the **agent criticality** measure that helps agents in their cooperative decision process and we underlined the importance of this measure and its impact on the obtained results.

• **The evaluation criteria** we have addressed to underline the importance and the efficiency of self-adaptivity through self-* properties and the advantages of self-adaptive systems over-classical approaches. Contrarily to classical systems where formal proofs may exist under particular conditions, adaptive multi-agent system approaches **lack a relevant evaluation framework**. We conducted a first study on how such systems can be evaluated in order to **validate their behaviour**, **underline their advantages** over classical approaches by

showing how self-* properties can increase the performance and thus **ease their acceptance for industry**. We propose **evaluation criteria aiming at guiding the evaluation** of adaptive multi-agent systems from the design phase to the execution phase.

Contributions Evaluation

This work contributed to fill a gap in the existing solving techniques based on the agentification of the domain entities and to improve the *AMAS* theory from a theoretical and engineering point of view. Indeed, concerning the existing solving techniques, the state of the art underlines the existence of different algorithms to solve specific optimisation under constraints problems. These algorithms are designed using given methodologies. Such methodologies remain at a high level of abstraction enabling their usage in different domains but requiring huge effort to get instantiated to specific applications. Thus, we proposed the *AMAS4Opt* agent model as a ready to use framework. The agents of this model are provided with **local goals, behaviours, cooperative rules and interactions** that match seamlessly with the behaviours of the problem entities.

This model has been first instantiated on the manufacturing control scheduling problem. The satisfactory results helped us to validate our model. In addition to this, the usage of this model instead of applying the different steps of the *ADELFE* methodology allowed us to gain time. By that, we were able to confront the *AMAS* theory and the *AMAS4Opt* model to the design of complex products using a new original approach. The obtained result was very encouraging.

To conclude, the applicative contributions of this work point out the importance and the efficiency of the scientific contributions. Still, now that this work is completed, we feel that by addressing some challenging problems several perspectives have arisen.

Applicative Perspectives

Our perspectives at the applicative level are:

• **The development of design tools** that can be integrated to the *ADELFE* methodology. Those tools must implement in a generic manner the agent architecture we defined and guide the designers through the different design phases of our model. Code generation also needs to be proposed.

• **The validation of our model** by instantiating it to other complex applications involving **multi-criteria and multi-objective optimization**. This is the subject of a PhD work, currently done in the team and consisting in developing a multi-agent system used for integrative design[4], where different design levels, each involving different interrelated disciplines that interact also from one level to another, must be considered.

• **The validation of the intuitive aspect of our model** by making non *AMAS* experts using it. This will help to underline remaining challenges and will help improving our

[4]For an explanation of the idea of integrative design, please refer to the *ID4CS* project: www.irit.fr/id4cs

model.

• When comparing *SAFlex* to the On-line Forward Optimisation approach, we underlined the fact that both systems are adequate, each having its limits and advantages. An interesting study we would like to continue in collaboration with the Vakgroep Informatietechnologie is the evaluation of both systems in more complex scenarii, and eventually to try to **combine their advantages** in order to improve the results.

• **Comparing *SAFlex* and *SAPBR* to other distributed self-organised *MASs***, in order to validate evaluation criteria such as the number of messages, the decentralisation and the action locality.

• In the state of the art chapter, we introduced a brief description on the *Case-Based Reasoning (CBR)* techniques. Given their description, these techniques are relevant for solving the problem addressed by *SAPBR*. In addition to this, different statistical mathematical methods such as the study of correlation or the analysis of variance have been used to solve such problem. That is why we would like to **compare *SAPBR* to *CBR* techniques and to statistical mathematical methods**.

• **The development of an evaluation framework**. Given the importance of the evaluation criteria that were defined, we intend to contribute to the **development of an evaluation framework** integrating tools that can help to understand the functioning and performance of adaptive multi-agent systems, and implementing the different evaluation criteria in a generic manner so that the evaluation of such systems becomes straightforward. This framework could then be used as a general evaluation guide.

Scientific Perspectives

Our perspectives at the scientific level are:

• **Improve the criticality measure**. This measure is currently **application dependent** and **requires a certain level of understanding of the problem** to be well defined. In some problems, this can be a difficult task especially considering that this measure is a guide for the cooperative behaviour of the agent and thus highly impacts the obtained results. Different other researches in the team have **underlined the importance of this measure** and we agree on the fact that a **generic process to define this measure** must be defined. At the applicative level, we could even imagine mechanisms that can be used by agents to discover this measure and learn to normalize it for a given problem.

• **Improve existing and explore new evaluation criteria**. Further studies are required to improve the definition of the criteria we propose and to explore additional ones. We especially would like to continue the work we have started with Claudia Raibulet consisting in extending those criteria for adaptive systems in general and adding architectural criteria. This work will fit in the scope of new researches that are currently being done in the team aiming at validating and verifying the behaviour of complex systems presenting self-* properties or mechanisms.

• **(Semi-) generic agent model for other types of problems or domains**. The success of the instantiation of our model on two applications and the facilitation it brings for the

AMAS theory and the *ADELFE* methodology encourages us to define other (semi-)generic agent models specialising the *AMAS* theory for other types of complex problems such as parameter learning, complex process control or simulation.

Thoughts on the Emergence Aspect

What do we call emergence in our system? How can we call our system emergent while we know what our agents do? and different similar questions were asked during conferences or presentations. My personal opinion is: "It is all in interactions". Indeed, we design our agents providing them with local goals and behaviours. There is no control telling to those agents what to do. They are only provided with local rules that, depending on their states and their perception of their neighbours, guide the agent actions in its environment and by that its interactions with other agents. From this local agent point of view, we have no visibility on the global functioning of the system. While considering this global system point of view, we can see that given their interactions, an organisation is constructed between the agents enabling the system to converge. Thus, in my point of view, the emergence in our systems concerns the manner the agents locally explore in a deterministic way a very small part of the whole search space enabling the whole system to converge. How this exploration is done can not be predicted and is not controlled by the designer, but the efficiency of these kind of systems is to be found there.

Final Words

Reaching the end of this study, I feel that my work could be a useful contribution to complex problem solving. Interested in optimisation problems under constraints for several years now, I found in the *SMAC* team a new way to solve such problems. The *AMAS* theory proposes to address these problems in their whole complexity and solving them through simple and local mechanisms.

When I arrived in the team four years ago, I found that applying the theory and defining cooperative behaviours for an application to be a challenging task. Indeed, this approach differs from classical ones by its manner to treat the problem. Instead of thinking globally how a problem can be solved, with this theory we learn to think locally and to design efficient and simple agent behaviours that by their interactions enable the emergence of good solutions. Since I have developed different systems using this theory, I am convinced of its potential and the ability of cooperative and adaptive multi-agent systems to address complex problems with a realistic approach. For this reason, I wanted to participate in its promotion and ease its usage for non *AMAS* experts. I think and I hope that my contribution will motivate designers to use it for the solving of more realistic problems such as scheduling in a real factory, in the domotics field for energy efficient resource management or for ambient intelligence for instance.

Before concluding, I would like to mention the important experience my 3-months visit to the Vakgroep Informatietechnologie has offered to me. In particular, this visit enabled me to compare my system with other approaches and thus being able to improve my agents'

behaviours.

In addition to this, a great reward this work has given to me is the Amelia Earhart Fellowship from the Zonta International organisation. Having my work rewarded by this important institute is for me, maybe not a comfort that my work is really good but that I knew how to stand up to a great challenge. This motivated me even more to go ahead.

Finally, as this work would have not been possible without the encouragements of my supervisors and every cooperative member of the team, I would like to conclude this thesis with this quote from Virginia Burden:

"*Cooperation is the thorough conviction that nobody can get there unless everybody gets there*".

Personal Bibliography

1. Gaêl Clair, Elsy Kaddoum, Marie Pierre Gleizes, and Gauthier Picard. *Self-regulation in self-organising multi-agent systems for adaptive and intelligent manufacturing control.* In EEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pages 107 - 116, 2008.

2. Elsy Kaddoum, Marie-Pierre Gleizes, Jean-Pierre Georgé. *Designing Complex Systems: Population Based Emergence of Parameters* In the 9th European Workshop on Multi-agent Systems (EUMAS 2011), 2011.

3. Gaêl Clair, Elsy Kaddoum, Marie-Pierre Gleizes, Gauthier Picard. *Approches multi-agents auto-organisatrices pour un contrôle manufacturier intelligent et adaptatif.* In Journés Francophones sur les Systèmes Multi-Agents (JFSMA), Cépaduès, p. 191-200, 2008.

4. Elsy Kaddoum, Marie-Pierre Gleizes, Jean-Pierre Georgé, and Gauthier Picard. *Characterizing and evaluating problem solving self-* systems.* In The First International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE), 9 pages (electronic medium). CPS Production - IEEE Computer Society, 2009.

5. Elsy Kaddoum, Claudia Raibulet , Jean-Pierre Georgé, Gauthier Picard, Marie-Pierre Gleizes. *Criteria for the evaluation of self-* systems*. In Workshop on Software Engineering for Adaptive and Self-Managing Systems (at ICSE), ACM, p. 29-38, 2010.

6. Elsy Kaddoum, Marie-Pierre Gleizes, Jean-Pierre Georgé, Pierre Glize, Gauthier Picard. *Analyse des critères d'évaluation de systèmes multi-agents adaptatifs*. In Journés Francophones sur les Systèmes Multi-Agents (JFSMA), Cépaduès, p. 123-132, 2009.

7. Elsy Kaddoum, Yailen Martinez, Tony Wauters, Katja Verbeeck, Ann Nowé, Patrick De Causmaecker, Greet Vanden Berghe, Marie-Pierre Gleizes, and Jean-Pierre Georgé. *Adaptive methods for flexible job shop scheduling with due-dates, release-dates and machine perturbations.* (extended abstract) Workshop on Self-tuning, self-configuring and self-generating search heuristics (Self*) in the 11th International Conference on Parallel Problem Solving From Nature (PPSN), 4 pages, 2010.

Under Review

Elsy Kaddoum, Jean-Pierre Georgé, Marie-Pierre Gleizes, Pierre Glize. *SAFlex: Self-Adaptive Multi-Agent System for Dynamic Flexible Job Shop Problem*. In Journal of Systems and Software, 2010.

Bibliography

AAMODT, A. AND PLAZA, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Commun. 7,* 1, 39–59.

AHUJA, R. K., ORLIN, J. B., AND TIWARI, A. 2000. A greedy genetic algorithm for the quadratic assignment problem. *Computers & OR 27,* 10, 917–934.

APPLEGATE, D., BIXBY, R., CHVATAL, V., AND COOK, W. 1998. On the solution of traveling salesman problems.

APPLEGATE, D. L., BIXBY, R. E., CHVATAL, V., AND COOK, W. J. 2006. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press.

AXELROD, R. 1984. *The Evolution of Cooperation*. Basic Books.

AYDIN, M. E. AND ÖZTEMEL, E. 2000. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems 33,* 2-3, 169–178.

BAKER, A. D. 1991. Manufacturing control with a market-driven contract net. Ph.D. thesis, Rensselaer Polytechnic Institute, Troy, NY, USA.

BERGENTI, F., GLEIZES, M.-P., AND ZAMBONELLI, F. 2004. *Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering handbook*. Kluwer Publishing.

BERNON, C., CAMPS, V., GLEIZES, M. P., AND PICARD, G. 2004. Designing agents' behaviors and interactions within the framework of adelfe methodology. In *ESAW*. Lecture Notes in Computer Science, vol. 3071. Springer, 311–327.

BERNON, C., CAMPS, V., GLEIZES, M.-P., AND PICARD, G. 2005. Engineering Self-Adaptive Multi-Agent Systems: the ADELFE Methodology. In *Agent-Oriented Methodologies*. Idea Group Publishing, Chapter 7, 172–202.

BERNON, C., GLEIZES, M. P., PEYRUQUEOU, S., AND PICARD, G. 2003. Adelfe: A methodology for adaptive multi-agent systems engineering. In *ESAW*. Lecture Notes in Computer Science, vol. 2577. Springer, 156–169.

BERNON, C., GLEIZES, M.-P., PICARD, G., AND GLIZE, P. 2002. The adelfe methodology for an intranet system design. In *International Bi-Conferenystems (AOIS-2002) at*

CAice Workshop on Agent-Oriented Information SSE'02 (AOIS - SSE). CEUR Workshop Proceedings, (on line).

BLUM, C., PUCHINGER, J., RAIDL, G. R., AND ROLI, A. 2011. Hybrid metaheuristics in combinatorial optimization: A survey. *Appl. Soft Comput. 11,* 6, 4135–4151.

BLUM, C. AND ROLI, A. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv. 35,* 3, 268–308.

BONABEAU, E., DORIGO, M., AND THERAULAZ, G. 1999. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Inc.

BONNET, G. AND TESSIER, C. 2008. Multi-agent collaboration: A satellite constellation case. In *STAIRS*. Frontiers in Artificial Intelligence and Applications, vol. 179. IOS Press, 24–35.

BONZANO, A., CUNNINGHAM, P., AND MECKIFF, C. 1996. Isac: A cbr system for decision support in air traffic control. In *EWCBR*. 44–57.

BRANDIMARTE, P. 1993. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research 41,* 3, 157–183.

BRITO, I., HERRERO, F., AND MESEGUER, P. 2004. On the evaluation of discsp algorithms. *The Fifth International Workshop on Distributed Constraint Reasoning (DCR04) Toronto, Canada September 27, 2004 held in conjunction with Tenth International Conference on Principles and Practice of Constraint Programming (CP 2004)* 5, 142–151.

BROCHTRUP, B. M. AND HERRMANN, J. W. 2006. A classification framework for product design optimization. In *Proceedings of IDETC/CIE 2006 conference*.

CAPERA, D., GLEIZES, M. P., AND GLIZE, P. 2004. Mechanism type synthesis based on self-assembling agents. *Applied Artificial Intelligence 18,* 9-10, 921–936.

CASTRO, J., KOLP, M., AND MYLOPOULOS, J. 2001. A requirements-driven development methodology. In *CAiSE*. Lecture Notes in Computer Science, vol. 2068. Springer, 108–123.

CHARLES, A., MENEZES, R., AND TOLKSDORF, R. 2004. On the implementation of swarmlinda. In *ACM Southeast Regional Conference*. 297–298.

CHIARANDINI, M. 2005. Stochastic local search methods for highly constrained combinatorial optimisation problems. Ph.D. thesis, Udine, Italy.

CLAIR, G., KADDOUM, E., GLEIZES, M. P., AND PICARD, G. 2008. Self-regulation in self-organising multi-agent systems for adaptive and intelligent manufacturing control. In *SASO*. 107–116.

COSSENTINO, M., GAUD, N., HILAIRE, V., GALLAND, S., AND KOUKAM, A. 2010. Aspecs: an agent-oriented software process for engineering complex systems. *Autonomous Agents and Multi-Agent Systems 20,* 2, 260–304.

COSSENTINO, M. AND POTTS, C. 2002. A case tool supported methodology for the design of multi-agent systems.

DARWIN, C. 1859. *On the Origin of Species by Means of Natural Selection*. John Murray, London.

DAVIN, J. AND MODI, P. J. 2005. Impact of problem centralization in distributed constraint optimization algorithms. In *AAMAS*. 1057–1063.

DONGARRA, J. 1988. Performance of various computers using standard linear equations software in a fortran environment. *SIGARCH Comput. Archit. News 16*, 1, 47–69.

DONIEC, A., ESPIÉ, S., MANDIAU, R., AND PIECHOWIAK, S. 2005. Dealing with multi-agent coordination by anticipation: Application to the traffic simulation at junctions. In *EUMAS*. 478–479.

DONIEC, A., MANDIAU, R., PIECHOWIAK, S., AND ESPIÉ, S. 2008. A behavioral multi-agent model for road traffic simulation. *Eng. Appl. of AI 21*, 8, 1443–1454.

DORIGO, M. AND CARO, G. D. 1999. The Ant Colony Optimization Meta-Heuristic. In *New Ideas in Optimization*. McGraw-Hill, 11–32.

DORIGO, M., CARO, G. D., AND STÜTZLE, T. 2000. Ant algorithms. *Future Generation Comp. Syst. 16*, 8.

DORIGO, M. AND STÜTZLE, T. 2004. *Ant colony optimization*. MIT Press.

DRÉO, J., PÉTROWSKI, A., SIARRY, P., AND TAILLARD, E. 2005. *Metaheuristics for Hard Optimization: Methods and Case Studies*. Springer.

DUGGAN, J. AND BROWNE, J. 1991. Production activity control: A practical approach to scheduling. *International Journal of Flexible Manufacturing Systems 4*, 1, 79–103.

EDMONDS, B. 2005. Using the experimental method to produce reliable self-organised systems. In *Engineering Self-Organising Systems*. Lecture Notes in Computer Science, vol. 3464. Springer, 84–99.

EDMONDS, B. AND BRYSON, J. 2004. The insufficiency of formal design methods - the necessity of an experimental approach - for the understanding and control of complex mas. In *AAMAS*. 938–945.

FERBER, J. 1995. *Les Systèmes multi-agents: Vers une intelligence collective*. InterEditions, Paris.

FORTNOW, L. AND HOMER, S. 2003. A short history of computational complexity. *Bulletin of the EATCS 80*, 95–133.

GAILLARD, F., KUBERA, Y., MATHIEU, P., AND PICAULT, S. 2009. A reverse engineering form for multi agent systems. In *ESAW*. Lecture Notes in Computer Science, vol. 5485. Springer, 137–153.

GEORGÉ, J.-P., GLEIZES, M. P., AND GLIZE, P. 2003. Conception de systèmes adaptatifs à fonctionnalité émergente : la théorie amas. *Revue d'Intelligence Artificielle 17*, 4, 591–626.

GEORGÉ, J.-P., GLEIZES, M.-P., GLIZE, P., AND RÉGIS, C. 2003. Real-time simulation for flood forecast: an adaptive multi-agent system staff. In *AISB'03 Symposium on Adaptive Agents and Multi-Agent Systems (AAMAS'03)*. 109–114.

GEORGÉ, J.-P., PEYRUQUEOU, S., RÉGIS, C., AND GLIZE, P. 2009. Experiencing self-adaptive mas for real-time decision support systems. In *PAAMS*. Advances in Intelligent and Soft Computing, vol. 55. Springer, 302–309.

GERKEY, B. P. AND MATARIC, M. J. 2002. Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics 18,* 5, 758–768.

GLEIZES, M. P., CAMPS, V., GEORGÉ, J.-P., AND CAPERA, D. 2008. Engineering systems which generate emergent functionalities. In *EEMMAS*. Lecture Notes in Computer Science, vol. 5049. Springer, 58–75.

GLEIZES, M.-P., CAMPS, V., AND GLIZE, P. 1999. A theory of emergent computation based on cooperative self-oganization for adaptive artificial systems. In *Fourth European Congress of Systems Science , Valencia Spain*.

GLEIZES, M.-P. AND GLIZE, P. 2000. Abrose : Des systèmes multi-agents pour le courtage adaptatif. In *8ièmes Journées Francophones d'Intelligence Artificielle Distribuée et des Systèmes Multi-Agents*. hermès, 117 – 132.

GLIZE, P. L'adaptation des systèmes à fonctionnalité emergente par auto-organisation coopérative, hdr, Université Paul Sabatier, Toulouse, France.

GLIZE, P. AND PICARD, G. 2012. Self-organisation in constraint problem solving. In *Self-organising Software: From Natural to Artificial Adaptation*. Springer. to be published.

GLOVER, F., KELLY, J. P., AND LAGUNA, M. 1995. Genetic algorithms and tabu search: Hybrids for optimization. *Computers & OR 22,* 1, 111–134.

GLOVER, F. AND LAGUNA, M. 1993. Tabu search. In *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc., Chapter 3, 70–150.

GLOVER, F. W. AND KOCHENBERGER, G. A. 2003. *Handbook of Metaheuristics (International Series in Operations Research & Management Science)*. Springer.

HAGEMAN, J. A., WEHRENS, R., VAN SPRANG, H. A., AND BUYDENS, L. M. C. 2003. Hybrid genetic algorithm-tabu search approach for optimising multilayer optical coatings. *Analytica Chimica Acta 490,* 1-2, 211 – 222.

HANSEN, K. M., ZHANG, W., AND INGSTRUP, M. 2008. Towards self-managed executable petri nets. In *SASO*. 287–296.

HENDERSON-SELLERS, B. AND GIORGINI, P. 2005. *Agent-oriented methodologies*. Information Science Reference.

HEYLIGHEN, F. 1992. Evolution, selfishness and cooperation; selfish memes and the evolution of cooperation. *Journal of Ideas 2,* 4, 70–84.

HIRAYAMA, K. AND YOKOO, M. 2005. The distributed breakout algorithms. *Artificial Intelligence 161,* 1-2, 89–115.

HUBERMAN, B. A. 1991. The performance of cooperative processes. In *Emergent computation: Self-organizing, Collective, and cooperative phenomena in Natural and Artificial Computing networks, Special issue of Physica D*. MIT Press, 38–47.

JAMES, T., REGO, C., AND GLOVER, F. 2009. A cooperative parallel tabu search algorithm for the quadratic assignment problem. *European Journal of Operational Research 195,* 3, 810–826.

JAT, S. AND YANG, S. 2010. A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling. *Journal of Scheduling*, 1–21.

JELASITY, M. AND BABAOGLU, Ö. 2006. T-man: Gossip-based overlay topology management. In *Engineering Self-Organising Systems*. Lecture Notes in Computer Science, vol. 3910. Springer, 1–15.

KADDOUM, E., GLEIZES, M.-P., GEORGÉ, J.-P., AND PICARD, G. 2009. Characterizing and evaluating problem solving self-* systems. In *The First International Conference on Adaptive and Self-adaptive Systems and Applications (ADAPTIVE 2009)*. CPS Production - IEEE Computer Society, (electronic medium).

KADDOUM, E., MARTINEZ, Y., WAUTERS, T., VERBEECK, K., NOWÉ, A., CAUSMAECKER, P. D., BERGHE, G. V., GLEIZES, M.-P., AND GEORGÉ, J.-P. 2010. Adaptive methods for flexible job shop scheduling with due-dates, release-dates and machine perturbations.

KADDOUM, E., RAIBULET, C., GEORGÉ, J.-P., PICARD, G., AND GLEIZES, M.-P. 2010. Criteria for the evaluation of self-* systems. In *Workshop on Software Engineering for Adaptive and Self-Managing Systems (at ICSE 2010)*. ACM, 29–38.

KAELBLING, L. P., LITTMAN, M. L., AND MOORE, A. W. 1996. Reinforcement learning: A survey. *CoRR cs.AI/9605103*.

KENDALL, E. A. 2001. Agent software engineering with role modelling. In *AOSE*. Lecture Notes in Computer Science, vol. 1957. Springer, 163–170.

KENNEDY, J. 2010. Particle swarm optimization. In *Encyclopedia of Machine Learning*. 760–766.

KENNEDY, J. AND EBERHART, R. C. 2001. *Swarm Intelligence*. Morgan Kaufmann.

KIRKPATRICK, S., JR., D. G., AND VECCHI, M. P. 1983. Optimization by simmulated annealing. *Science 220,* 4598, 671–680.

LACOUTURE, J., NOËL, V., ARCANGELI, J.-P., AND GLEIZES, M. P. 2011. Engineering agent frameworks: An application in multi-robot systems. In *PAAMS*. Advances in Intelligent and Soft Computing, vol. 88. Springer, 79–85.

LEMOUZY, S. 2011. Systèmes interactifs auto-adaptatifs par systèmes multi-agents auto-organisateurs : application à la personnalisation de l'accés à l'information. Ph.D. thesis, Université de Toulouse (Paul Sabatier), France.

LINK-PEZET, J., GLIZE, P., AND GLEIZES, M. P. 2000. Abrose: An adaptive multi-agent tool for electronic commerce. In *WETICE*. 59–66.

LIU, J., JING, H., AND TANG, Y. Y. 2002. Multi-agent oriented constraint satisfaction. *Artificial Intelligence 136,* 1, 101–144.

LOOR, P. D. AND CHEVAILLIER, P. 2003. Solving distributed and dynamic contraints using an emotional metaphor : Application to the timetabling problem. In *5th EURO/INFORM international conference, new opportunities for operations research*. 144.

LYNCH, N. A. 1996. *Distributed Algorithms*. Morgan Kaufmann.

MADUREIRA, A., SANTOS, J., AND PEREIRA, I. 2009. A hybrid intelligent system for distributed dynamic scheduling. In *Natural Intelligence for Scheduling, Planning and Packing Problems*. 295–324.

MAHER, M. L. AND DE SILVA GARZA, A. G. 1997. Case-based reasoning in design. *IEEE Expert 12,* 2, 34–41.

MAILLER, R. AND LESSER, V. R. 2006. Asynchronous partial overlay: A new algorithm for solving distributed constraint satisfaction problems. *J. Artif. Intell. Res. (JAIR) 25*, 529–576.

MAMEI, M. AND ZAMBONELLI, F. 2005. Programming stigmergic coordination with the tota middleware. In *AAMAS*. 415–422.

MARTINEZ, Y., WAUTERS, T., CAUSMAECKER, P. D., NOWÉ, A., VERBEECK, K., BELLO, R., AND SUAREZ, J. 2010. Reinforcement learning approaches for the parallel machines job shop scheduling problem. In *Proceedings of the Cuba-Flanders Workshop on Machine Learning and Knowledge Discovery*.

MATSUI, T., MATSUO, H., SILAGHI, M., HIRAYAMA, K., AND YOKOO, M. 2008. Resource constrained distributed constraint optimization with virtual variables. In *AAAI*. 120–125.

MCGINTY, L. AND SMYTH, B. 2001. Collaborative case-based reasoning: Applications in personalised route planning. In *ICCBR*. Lecture Notes in Computer Science, vol. 2080. Springer, 362–376.

MEIGNAN, D. 2008. Une approche organisationnelle et multi-agent pour la modélisation et l'implantation de métaheuristiques: Application aux problèmes d'optimisation de réseaux de transports. Ph.D. thesis, Laboratoire Systèmes et Transport - Université de Technologie de Belfort-Montbéliard.

MEIGNAN, D., CRÉPUT, J.-C., AND KOUKAM, A. 2008. A coalition-based metaheuristic for the vehicle routing problem. In *IEEE Congress on Evolutionary Computation*. 1176–1182.

MEISELS, A., KAPLANSKY, E., RAZGON, I., AND ZIVAN, R. 2002. Comparing performance of distributed constraints processing algorithms. In *AAMAS-02 Workshop on Distributed Constraint Reasoning*. Number 5. 86–93.

MINTON, S., JOHNSTON, M. D., PHILIPS, A. B., AND LAIRD, P. 1992. Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems. *Artif. Intell. 58,* 1-3, 161–205.

NOEL, V. 2011. Component-based software architectures and multi-agent systems: Mutual and complementary contributions for supporting software development (to appear). Ph.D. thesis, Université de Toulouse (Paul Sabatier), France.

NOEL, V. AND ARCANGELI, J.-P. 2011. Frameworks, architectures et composants: revisiter le développement de systèmes multi- agents. In *Conférence Francophone sur les Architectures Logicielles (CAL)*. Laboratoire d'Informatique Fondamentale de Lille, 23–32.

NOWOSTAWSKI, M. AND POLI, R. 1999. Parallel genetic algorithm taxonomy. In *KES*. 88–92.

OMICINI, A. 2001. Soda: Societies and infrastructures in the analysis and design of agent-based systems. In *AOSE*. Lecture Notes in Computer Science, vol. 1957. Springer, 185–193.

OUELHADJ, D. AND PETROVIC, S. 2009. A survey of dynamic scheduling in manufacturing systems. *J. Scheduling 12,* 4, 417–431.

PAPADIMITRIOU, C. H., SCHÄFFER, A. A., AND YANNAKAKIS, M. 1990. On the complexity of local search (extended abstract). In *STOC*. 438–445.

PARUNAK, H. V. D. AND BRUECKNER, S. 2004. Engineering swarming systems. *Methodologies and Software Engineering for Agent Systems 11*, 341–376.

PARUNAK, V. D. 1987. Manufacturing Experience with the Contract Net. In *Distributed Artificial Intelligence*. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 285–310.

PAVÓN, J. AND GÓMEZ-SANZ, J. J. 2003. Agent oriented software engineering with ingenias. In *CEEMAS*. Lecture Notes in Computer Science, vol. 2691. Springer, 394–403.

PEZZELLA, F., MORGANTI, G., AND CIASCHETTI, G. 2008. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & OR 35,* 10, 3202–3212.

PICARD, G. 2004. Méthodologie de développement de systèmes multi-agents adaptatifs et conception de logiciels à fonctionnalité émergente. Ph.D. thesis, Université Paul Sabatier, Toulouse, France.

PICARD, G., BERNON, C., AND GLEIZES, M. P. 2005. Emergent timetabling organization. In *CEEMAS*. Lecture Notes in Computer Science, vol. 3690. Springer, 440–449.

PICARD, G. AND GLIZE, P. 2006. Model and analysis of local decision based on cooperative self-organization for problem solving. *Multiagent and Grid Systems 2,* 3, 253–265.

PLAZA, E., ARCOS, J. L., AND MARTIN, F. 1997. *Cooperative Case-Based Reasoning*. Vol. 1221. Spriger-Verlag, 180–201.

PLAZA, E. AND MCGINTY, L. 2005. Distributed case-based reasoning. *Knowledge Eng. Review 20,* 3, 261–265.

PRASAD, M. V. N. 2000. Distributed case-based learning. In *ICMAS*. 222–230.

PRASAD, M. V. N., LESSER, V. R., AND LANDER, S. 1996. Retrieval and reasoning in distributed case bases. *Journal of Visual Communication and Image Representation, Special Issue on Digital Libraries 7*, 1, 74–87.

RAJABINASAB, A. AND MANSOUR, S. 2011. Dynamic flexible job shop scheduling with alternative process plans: an agent-based approach. *The International Journal of Advanced Manufacturing Technology 54*, 1091–1107.

ROBERTSON, P., LADDAGA, R., AND SHROBE, H. E. 2001. Introduction: The first international workshop on self-adaptive software. In *IWSAS*. Lecture Notes in Computer Science, vol. 1936. Springer, 1–10.

ROUGEMAILLE, S., ARCANGELI, J.-P., GLEIZES, M. P., AND MIGEON, F. 2009. Adelfe design, amas-ml in action. In *ESAW*. Lecture Notes in Computer Science, vol. 5485. Springer, 105–120.

ROUGEMAILLE, S., MIGEON, F., MAUREL, C., AND GLEIZES, M. P. 2008. Model driven engineering for designing adaptive multi-agents systems. In *ESAW*. Lecture Notes in Computer Science, vol. 4995. Springer, 318–332.

RUSSELL, S. AND NORVIG, P. 2003. *Artificial Intelligence: A Modern Approach*, 2nd edition ed. Prentice-Hall, Englewood Cliffs, NJ.

SERUGENDO, G. D. M., GLEIZES, M. P., AND KARAGEORGOS, A. 2005. Self-organization in multi-agent systems. *Knowledge Eng. Review 20*, 2, 165–189.

SERUGENDO, G. D. M., GLEIZES, M. P., AND KARAGEORGOS, A. 2006. Self-organisation and emergence in mas: An overview. *Informatica (Slovenia) 30*, 1, 45–54.

SHEN, W., HAO, Q., YOON, H. J., AND NORRIE, D. H. 2006. Applications of agent-based systems in intelligent manufacturing: An updated review. *Advanced Engineering Informatics 20*, 4, 415–431.

SILAGHI, M.-C. AND YOKOO, M. 2009. Adopt-ing: unifying asynchronous distributed optimization with asynchronous backtracking. *Autonomous Agents and Multi-Agent Systems 19*, 2, 89–123.

SMITH, R. G. 1980. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on C-29*, 12, 1104 –1113.

SUTTON, R. S. 1999. Reinforcement learning: Past, present and future. In *SEAL*. Lecture Notes in Computer Science, vol. 1585. Springer, 195–197.

TALBI, E.-G. 2009. *Metaheuristics - From Design to Implementation*. Wiley.

TAMILARASI, A. AND KUMAR, T. A. 2010. An enhanced genetic algorithm with simulated annealing for job-shop scheduling. *International Journal of Engineering, Science and Technology 2*, 1, 144–151.

VALCKENAERS, P., HADELI, K., GERMAIN, B. S., VERSTRAETE, P., AND BRUSSEL, H. V. 2006. Emergent short-term forecasting through ant colony engineering in coordination and control systems. *Advanced Engineering Informatics 20,* 3, 261–278.

VAN PETEGHEM, V. AND VANHOUCKE, M. 2008. A genetic algorithm for the multi-mode resource-constrained project scheduling problem. Working papers of faculty of economics and business administration, ghent university, belgium, Ghent University, Faculty of Economics and Business Administration.

VERBEECK, K. 2004. Coordinated exploration in multi-agent reinforcement learning. Ph.D. thesis, COMO, Vrije Universiteit Brussel, Belgium.

WANG, Y.-C. AND USHER, J. M. 2005. Application of reinforcement learning for agent-based production scheduling. *Eng. Appl. of AI 18,* 1, 73–82.

WAUTERS, T., MARTINEZ, Y., CAUSMAECKER, P. D., NOWÉ, A., AND VERBEECK, K. 2010. Reinforcement learning approaches for the parallel machines job shop scheduling problem. International conference on interdisciplinary research on technology, education and communication, Kortrijk, 25-27 May 2010.

WAUTERS, T., VERBEECK, K., BERGHE, G. V., AND CAUSMAECKER, P. D. 2011. Learning agents for the multi-mode project scheduling problem. *JORS 62,* 2, 281–290.

WEBER, W., RABAEY, J. M., AND AARTS, E. 2005. *Ambient Intelligence*. Springer.

WEISS, G. 1999. *Multiagent Systems, A modern Approach to Distributed Artificial Systems*. MIT Press.

WELCOMME, J.-B., GLEIZES, M. P., AND REDON, R. 2006. Self-regulating multi-agent system for multi-disciplinary optimisation process. In *EUMAS*. CEUR Workshop Proceedings, vol. 223. CEUR-WS.org.

WELCOMME, J.-B., GLEIZES, M.-P., AND REDON, R. 2009. A self-organising multi-agent system managing complex system design application to conceptual aircraft design. *International Transactions on Systems Science and Applications, Self-organized Networked Systems 5,* 3, 208–221.

WEYNS, D., BOUCKÉ, N., HOLVOET, T., AND DEMARSIN, B. 2007. Dyncnet: A protocol for dynamic task assignment in multiagent systems. In *SASO*. 281–284.

WEYNS, D., HOLVOET, T., SCHELFTHOUT, K., AND WIELEMANS, J. 2008. Decentralized control of automatic guided vehicles: applying multi-agent systems in practice. In *OOPSLA Companion*. 663–674.

WILF, H. S. 1986. *Algorithms and complexity*. Prentice Hall.

WINSTON, W. L. 2003. *Operations Research: Applications and Algorithms*, 4 ed. Duxbury Press.

WOOLDRIDGE, M. J. 2009. *An Introduction to MultiAgent Systems (2. ed.)*. Wiley.

WU, B. 1994. *Manufacturing Systems Design & Analysis: Context & Techniques*. Springer.

WÜRTZ, R. P. 2008. *Organic Computing*. Springer.

XIANG, W. AND LEE, H. P. 2008. Ant colony intelligence in multi-agent dynamic manufacturing scheduling. *Engineering Applications of Artificial Intelligence 21*, 1, 73–85.

ZAMBONELLI, F., JENNINGS, N., AND WOOLDRIDGE, M. 2003. Developing multiagent systems: The gaia methodology. *ACM Trans. Softw. Eng. Methodol. 12*, 3, 317–370.

List of Figures

List of Tables