Conception et développement de la nouvelle plateforme APIC

Cédric Dufourg

Conservatoire National des Arts et Métiers

Centre Régional de Franche-Comté

Centre d'Enseignement de Belfort

Mémoire

présenté en vue d'obtenir

le diplôme d'ingénieur CNAM

Spécialité : Informatique

par Cédric DUFOURG

Conception et développement de la nouvelle plateforme APIC

JURY :

- KAMEL BARKAOUI PRESIDENT DE JURY
- ALAIN-JEROME FOUGERES
- EGON OSTROSI
- PHILIPPE DESCAMPS

Conservatoire National des Arts et Métiers

Centre Régional de Franche-Comté

Centre d'Enseignement de Belfort

Mémoire

présenté en vue d'obtenir

le diplôme d'ingénieur CNAM

Spécialité : Informatique

par Cédric DUFOURG

Conception et développement de la nouvelle plateforme APIC

(Agents for Products Integrated Configuration)

Ce projet a été mené au sein du laboratoire M3M de l'Université de Technologie Belfort-Montbéliard, sous le tutorat d'Alain-Jérôme Fougères, responsable de l'activité de recherche, sur la période de décembre 2008 à janvier 2010.

Sujet

D'une part il est demandé de concevoir et de développer une nouvelle version stable de la plateforme APIC qui pourra supporter les futures expérimentations du laboratoire, tout en respectant les orientations déjà définies : les agences déployées en réseau, les micro-outils pour l'interaction des acteurs/experts avec la plateforme, les communautés d'agents de configuration (fonctions, contraintes, spécifications, solutions), les algorithmes de configuration. D'autre part il est demandé de choisir, puis de développer un algorithme d'optimisation pour la proposition des solutions de configuration.

Résumé

La plateforme APIC (Agents for Products Integrated Configuration) aborde la configuration de famille de produits de manière innovante, basée sur l'utilisation de multiples agents flous représentant chaque fonction, contrainte, spécification, solution de l'approche proposée par le laboratoire. Chaque agent détermine alors sa valeur de part ses relations et ses échanges avec ses homologues. De cette détermination et de ses échanges émerge, après une phase d'optimisation, la solution optimale au problème de configuration posé.

Mots-clés :

Agents, Logique Floue, Optimisation, Configuration de produit

Subject :

On the one hand it's asked to design and develop a new stable version of APIC platform that will support future laboratory experiments, while respecting the existing guidelines: network deployed agencies, micro-tools for the interaction between actors / experts and the platform, the agent configuration community (functions, constraints, requirements, and solutions), configuration algorithms. On the other hand it's asked to choose, and then develop an optimization algorithm for the proposed configuration solutions.

Abstract :

The APIC platform (Agents for Products Integrated System) addresses the configuration of product family in an innovative approach based on the use of multiple fuzzy agents representing each function, constraint, specification and solution of the approach proposed by the laboratory. Each agent determines her own value from its relationships and exchanges with the others. From this determination and these exchanges emerges, after a phase of optimization, the optimal solution to the posed configuration problem.

Key Word :

Agents, Fuzzy Logic, Optimization, Product Configuration

Table des matières

REMERCIEMENTS

Je tiens tout naturellement à remercier le laboratoire M3M et toute son équipe pour m'avoir permis de réaliser ce projet.

Je tiens à remercier tout particulièrement Alain-Jérôme Fougères, responsable de l'action de recherche du projet, pour avoir accepté d'être mon tuteur mais aussi pour sa patience et ses conseils précieux qu'il a su me promulguer tout au long de ces 13 mois de recherche et de rédaction.

Enfin un remerciement particulier à Michel Ferney, directeur du laboratoire M3M et du CNAM Belfort au moment de l'initialisation du projet, pour son écoute particulière et la solution qu'il a su me proposer pour me permettre de réaliser ce projet en parallèle de mon activité professionnelle principale.

TABLE DES FIGURES

TABLE DES TABLEAUX

INTRODUCTION

La configuration de famille de produits est au centre des problématiques de productivité et de rentabilité du cycle de vie de nouveaux produits de masse configurables.

Le projet APIC (Agents for Products Integrated Configuration) est mené par le laboratoire M3M de l'UTBM depuis 2004. Il propose une approche innovante de configuration de produits basée sur une vision fonctions, contraintes, spécifications et solutions où chaque élément constituant cette vision est représenté par un agent flou.

Après plusieurs évolutions successives en termes de fonctionnalités attendues et de calculs engendrés, les versions précédentes de la plateforme de configuration de produits ne permettaient plus de mettre en place les nouvelles exigences de l'équipe.

C'est dans ce contexte qu'il m'a été confié de concevoir et de développer une nouvelle version de la plateforme APIC devant à la fois répondre aux exigences actuelles de l'équipe mais devant aussi permettre une évolution future facilitée de la plateforme.

Dans un premier temps, ce document abordera le contexte de l'étude présentant le laboratoire où a été mené le projet ainsi que les fondements du projet APIC. L'analyse et la conception de la plateforme sera ensuite abordée en détail, de l'analyse des besoins et des orientations générales choisies, au développement du noyau de la plateforme puis des interfaces utilisateurs. Enfin la mise en œuvre d'une proposition de solution optimale sera présentée, de la définition de solution optimale au choix et à la proposition de l'algorithme d'optimisation.

CONTEXTE DE L'ETUDE

Ce premier chapitre a pour objectif de détailler le contexte de l'étude. Dans un premier temps le laboratoire de recherche dans lequel a été mené ce projet sera présenté. Puis dans un second temps, les fondements du projet seront abordés. Enfin une présentation rapide des précédentes versions de la plateforme sera réalisée.

1. *Le laboratoire M3M*

1.1 L'UTBM

L'UTBM (Université de Technologie de Belfort Montbéliard), créée en 1999, est née du regroupement de l'Ecole Nationale d'Ingénieurs de Belfort fondée en 1962 et de l'Institut Polytechnique de Sévenans dont l'antenne de l'UTC a été implantée à Sévenans en 1985.

Figure 1 : Situation géographique de l'UTBM

Située en plein cœur du bassin industriel lié au secteur automobile du constructeur historique Peugeot, l'UTBM appuie sa fonction de formation sur des activités de recherches menées au sein de

sept unités de recherche autour d'une thématique commune portant sur les transports terrestres et l'énergie.

1.2 Les laboratoires

Les unités de recherche sont représentées dans le diagramme ci-dessous (Figure 2):

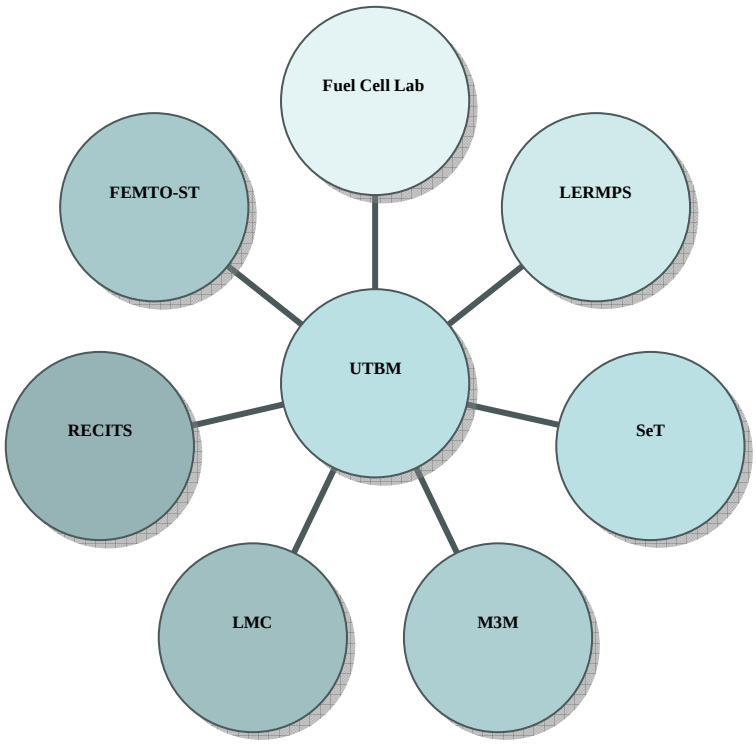


Figure 2 : Les unités de recherche de l'UTBM

La dénomination des sigles de ces sept laboratoires illustrés par le diagramme précédent (Figure 2) sont :

- Fuel Cell Lab : Laboratoire commun systèmes à piles pour les transports
- LERMPS : Laboratoire de Recherche sur les Matériaux, les Procédés et les Surfaces
- SeT : Laboratoire Systèmes et Transports
- FEMTO-ST : Franche-Comté Electronique Mécanique Thermique et Optique - Sciences et Technologies
- RECITS : Laboratoire Recherche et Etudes sur les Choix Industriels, Technologiques et Scientifiques

- LMC : Laboratoire de Métallurgies et Cultures
- M3M : Laboratoire Mécatronique 3 M

1.3 Les équipes du laboratoires M3M

Le laboratoire Mécatronique 3M (Méthodes, Modèles, Métiers) mène ses activités de recherche au sein de trois équipes, représentées par le diagramme suivant (Figure 3) :

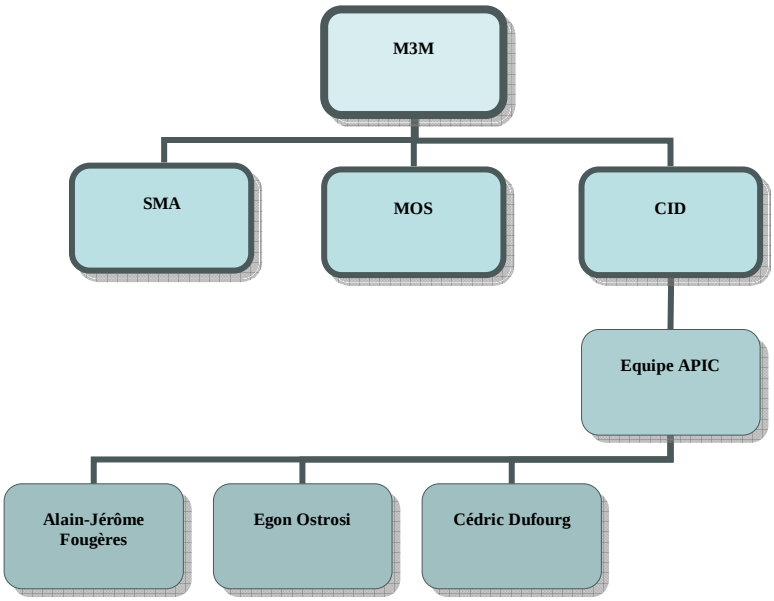


Figure 3 : Structure du laboratoire M3M

Ainsi, en complément du second niveau hiérarchique du diagramme ci-dessous (Figure 3), l'activité du laboratoire est menée par les équipes :

- Systèmes Mécaniques Adaptatifs (SMA) : Cette équipe oriente ses recherches sur la maîtrise de la conception des systèmes mécaniques adaptatifs et des conditions de la production industrielle de leurs structures mécaniques en composite ; ainsi que sur la modélisation probabiliste des systèmes mécaniques complexes
- Modélisation et Optimisation des Structures (MOS) : Cette équipe oriente ses recherches sur l'optimisation multicritère, l'analyse de sensibilités, les algorithmes génétiques ; ainsi que sur la modélisation numérique du contact
- Conception Innovante et Distribuée (CID) : Cette équipe oriente ses recherches sur le développement de la connaissance sur le processus de conception ainsi que sur la mise au point des outils et des méthodes d'aide à la conception

C'est au sein de l'équipe CID, que l'équipe APIC a été formée. Illustrée par le troisième niveau hiérarchique et les niveaux suivants du diagramme précédent (Figure 3), elle est constituée par Alain-Jérôme Fougères et Egon Ostrosi. C'est cette équipe que j'ai eu la chance d'intégrer pour mener le projet décrit ultérieurement dans ce document.

2. Les fondements du projet APIC

2.1 Configuration de produit

Les différentes étapes de la mise sur le marché d'un produit, de sa conception à sa fabrication, à sa vente et à sa réponse aux besoins des clients, sont importantes dans la détermination de la compétitivité d'une entreprise sur un marché.

De plus, de nos jours, les consommateurs souhaitent pouvoir personnaliser leurs produits parmi un choix varié d'éléments, produits devant souvent être renouvelés pour éviter une lassitude des clients.

Pour répondre à ces attentes, les entreprises ont dû proposer de complexifier et de diversifier leurs produits tout en tenant compte de la diminution du cycle de vie d'un produit et de son délai réduit de mise sur le marché.

Pour rester compétitives, les entreprises doivent réduire les temps et les coûts de conception d'un produit en mettant en œuvre une méthode de configuration de produit. Différentes approches existent dans ce domaine, certaines plus orientées consommateur, d'autres plus orientées production. Celle proposée dans le cadre du projet APIC, présentée dans le point suivant, tente de fournir une approche globale du problème.

2.2 Approche Spécification, Fonction, Contrainte, Solution

Proposée par l'équipe CID et soutenue par Eugène Deciu dans sa thèse [Deciu, 2007], l'approche Spécification, Fonction, Contrainte, Solution propose une approche innovante de la méthode de conception connue sous le nom de « Design for x ». Cette approche est illustrée par le schéma suivant (Figure 4) :

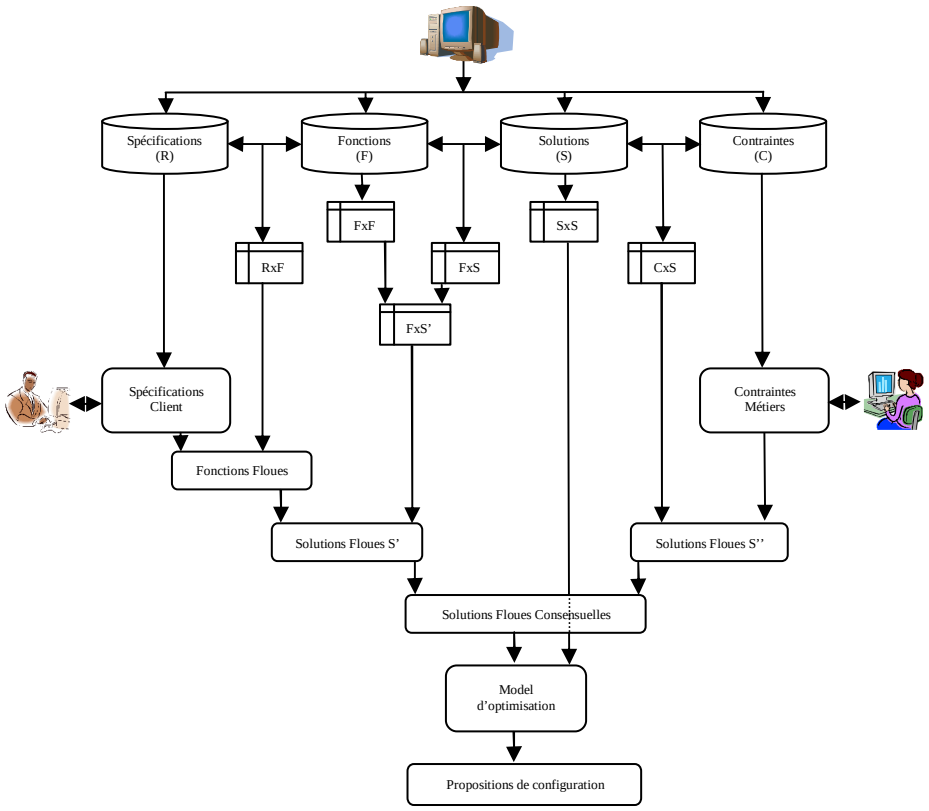


Figure 4 : Représentation de l'approche

Comme le montre ce schéma (Figure 4), cette approche est une approche de conception intégrée basée sur l'utilisation de multiples modèles flous. Ces modèles correspondent d'une part aux différents métiers (Contrainte) intervenant durant le processus de conception, puis d'autres parts aux critères fournis par le client (Spécification) et leurs correspondances avec un ensemble de fonctions induites (Fonction), puis enfin aux composants et à leurs interactions (Solution).

Cette approche tient compte du fait que chaque intervenant, qu'il soit un client ou un corps de métiers intervenant dans le cycle de vie du produit, ne dispose que d'une vue partielle du produit à concevoir.

2.3 Logique floue

La logique floue, formalisée par Lotfi Zadeh en 1965 [Zadeh, 1965], est basée sur la théorie des ensembles flous qui est une extension de la théorie des ensembles classiques. Cette logique est basée sur l'utilisation de valeurs floues définies dans l'intervalle [0,1]. Cette représentation est particulièrement utilise lorsque la définition d'une valeur ne peut pas être déterminée de façon précise et elle est souvent utilisée dans le domaine de l'intelligence artificielle.

Un exemple souvent cité pour illustrer ces valeurs est celui de la température d'une eau et de la détermination de son état (chaude ou froide). A partir de quelle température considère-t-on une eau comme chaude ou froide ? Ainsi dans le monde de la logique binaire, une eau ne peut être que froide (0) ou chaude (1), si l'on admet que l'état « chaude » est l'état validant. En revanche l'approche floue permet de nuancer les précédents états avec des états intermédiaires comme par exemple moyennement froide (0,25), tiède (0,5), moyennement chaude (O,75), etc.

Appliquée au domaine de la configuration de produits, cette approche permet par exemple de répondre à des critères d'associations, comme « Ces deux éléments se complètent-ils ? » : Un peu, moyennement, parfaitement ? ; ou bien encore à des critères d'évaluations subjectifs, comme l'évaluation du prix d'un produit : Peu couteux, très couteux,…

2.4 Optimisation

L'optimisation ou recherche de solutions optimales consiste à proposer la ou les solutions répondant au mieux au problème posé. Cette recherche revient souvent à minimiser ou maximiser une fonction, la fonction objectif, qui permet d'évaluer la qualité d'une solution. Cette recherche nécessite alors de mettre en place un algorithme permettant d'explorer ces différentes solutions.

Différentes méthodes permettent alors l'exploration de ces solutions. Parmi elles, les méthodes complètes qui explorent la totalité de l'arbre des solutions pour fournir la ou les solutions optimales mais qui posent souvent le problème de l'explosion combinatoire ; ou bien encore les méthodes incomplètes qui ne parcourent qu'une partie de l'arbre des solutions se contentant de trouver une solution suffisamment bonne (mais pas forcément la meilleure).

Le problème est alors de proposer une heuristique de recherche correspondant aux besoins de qualité des solutions trouvées (les meilleures ou les suffisamment bonnes) tout en tenant compte du rapport entre la qualité des solutions trouvées et le temps nécessaire à leurs recherches.

2.5 Monde agent

La programmation orientée agents a été proposée par Yoav Shoham en 1993 [Shodam, 1993] comme nouveau paradigme de la programmation. Cette nouvelle approche propose d'envisager la conception d'une application non plus de manière monolithique où un seul processus à la charge de réaliser la

totalité du travail, mais plutôt de la réaliser en mettant au point des entités autonomes, communicantes et coopérantes, les agents, réalisant chacune une partie du travail dans un but commun.

Les principaux intérêts de cette subdivision de la tâche à réaliser sont d'en permettre sa distribution mais aussi sa décomposition en traitements plus élémentaires.

Alain-Jérôme Fougères [Fougères, 2004] propose que le niveau de traitement des informations qu'un agent reçoit de ses confrères peut être défini selon le modèle de Rasmussen [Rasmussen, 1983]. Ce modèle définit le traitement de l'activité selon trois niveaux :

- Habilités (*Skills*): Premier niveau du modèle, le traitement de l'information est toujours identique, purement réactif
- Règles (*Rules*): Second niveau du modèle, le traitement de l'information est différencié selon des règles prédéfinies.
- Connaissances (*Knowledge*): Dernier niveau du modèle, le traitement de l'information est déterminé selon les connaissances acquises.

Les agents de la plateforme APIC sont définis par le second niveau de ce modèle, définissant le diagramme d'activité suivant (Figure 5) :

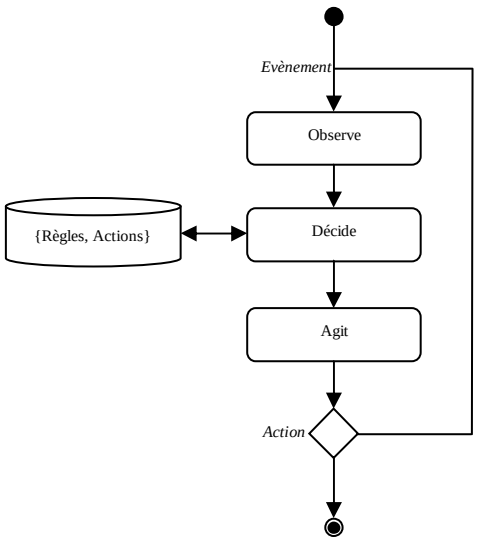


Figure 5 : Diagramme d'activité d'un agent

3. Les versions précédentes de la plateforme

La plateforme APIC a connu différentes versions, reflets de son évolution :

- **2006** : Première version réalisée en Java par David Nicoud et Jérémy Saleur dans le cadre des travaux de laboratoire réalisés durant leur formation à l'UTBM. Cette version a permis de valider l'approche Spécification, Fonction, Contrainte, Solution proposé par l'équipe CID. Les calculs n'étaient pas encore agentifiés, tout comme la plateforme qui n'offrait pas non plus ni le support réseau, ni les échanges entre acteurs, ni la proposition de solutions optimales.
- **2008** : Seconde version réalisée en C par Alain-Jérôme Fougères qui a permis de valider l'approche agent et de proposer un premier algorithme d'optimisation fournissant des solutions locales.

ANALYSE ET CONCEPTION DE LA NOUVELLE PLATEFORME

Ce second chapitre traite de l'analyse et de la conception de la nouvelle plateforme. Après une présentation de l'analyse des besoins, la conception de la plateforme et des différents composants sera détaillée.

1. *Analyse des besoins*

Première phase de réalisation du projet, l'analyse des besoins a permis de mettre en évidence les attentes du laboratoire et ainsi de définir le cahier des charges de la plateforme ainsi que les orientations générales prises par le projet.

1.1 Cahier des charges

La plateforme devra être développée sous Java et représenter l'approche Spécification, Fonction, Contraintes et Solution proposée par l'équipe CID du laboratoire M3M. Pour cela, comme l'illustre le schéma ci-après (Figure 6), elle sera constituée de communautés où chaque spécification, fonction, contrainte ou solution devra être représentée par un agent. Ces agents devront avoir une représentation floue de leur valeur ainsi que de leurs relations avec d'autres agents. Ces relations qu'un agent pourra avoir avec d'autres agents pourront être de nature intercommunautaires ainsi qu'intracommunautaires [Ostrosi et al, 2008].

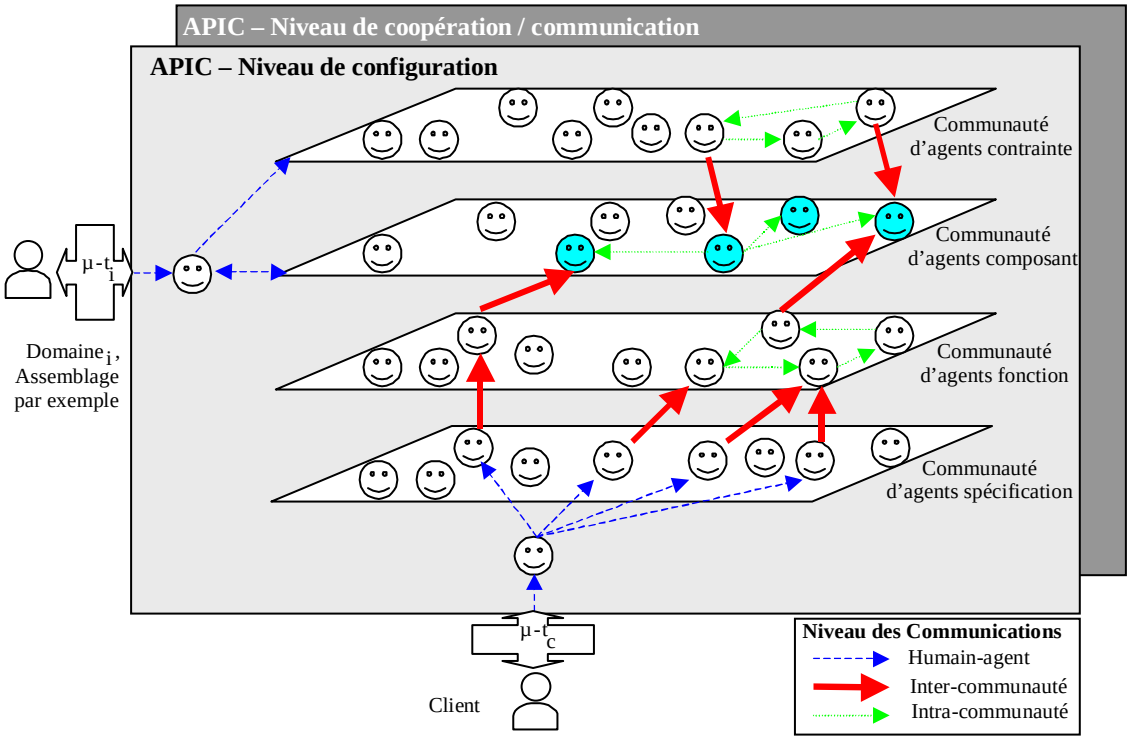


Figure 6 : Représentation agentifiée de la plateforme

Les relations intercommunautaires entre agents seront orientées et définiront les valeurs des agents influés. Les calculs engendrés par ces dernières seront de deux sortes, énumérés ci-après :

- La communauté Fonction déterminera ses valeurs en effectuant le MinMax du vecteur de valeurs des spécifications et de la matrice spécification-fonction.
- La communauté Solution déterminera ses valeurs en effectuant plusieurs calculs intermédiaires. Le premier calcul réalisera le MinMax du vecteur de valeurs des fonctions et de la matrice fonction-solution. Le second calcul déterminera le minimum des 1-MinMax des valeurs de chaque corps de métier et de leurs matrices métier-solution. Un dernier calcul réalisera alors le minimum des deux précédents calculs pour déterminer la valeur définitive de la communauté solution.

Les relations intracommunautaires moduleront les matrices des relations intercommunautaires de la communauté en réalisant le MinMax de la matrice des relations intracommunautaires avec chacune des matrices de relations intercommunautaires

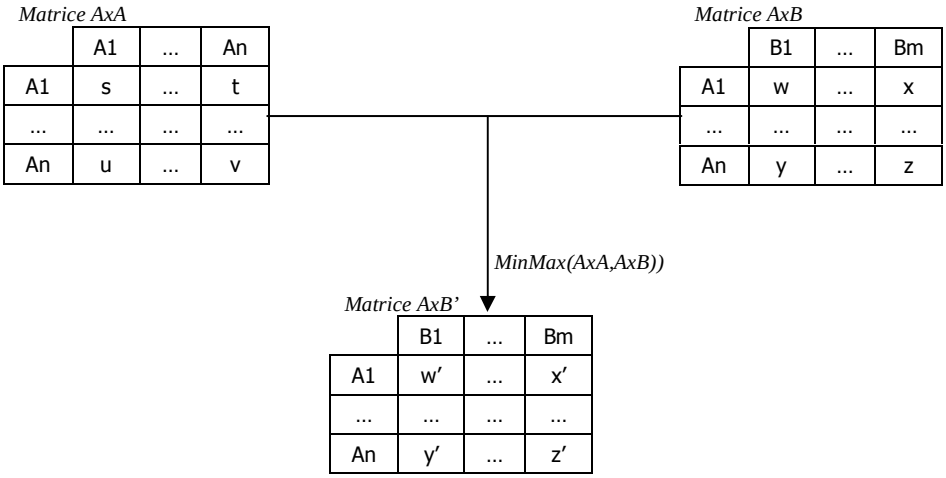


Figure 7 : Représentation des relations intercommunautaires

Une fois déterminées, les valeurs des agents seront soumises à un seuil en dessous duquel la valeur de l'agent sera définie comme nulle par exemple 0,4.

Les différents micro-outils devront permettre à la fois la saisie et la modification des valeurs des différentes relations ainsi que le paramétrage des valeurs des spécifications clientes et des contraintes métiers. Ils devront aussi permettre l'interaction des différents acteurs de la configuration de produits en leur proposant un service de messagerie instantanée.

1.2 Cycle de développement

Le cycle de développement suivi pour mener le projet est le cycle en spirale illustré par le schéma ci-dessous (Figure 8):

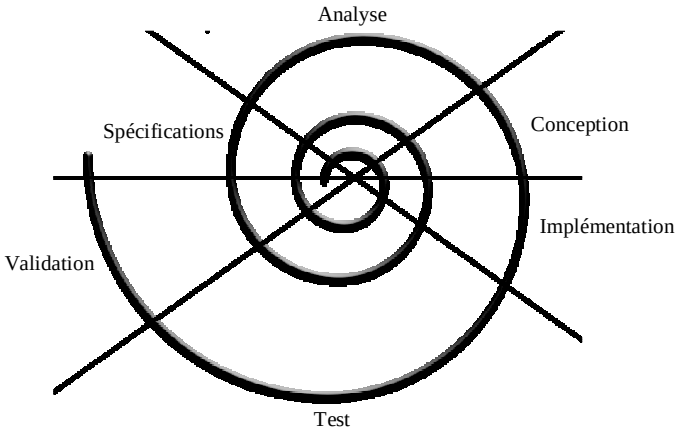


Figure 8 : Cycle de développement en spirale

Comme l'illustre ce schéma (Figure 8), le cycle en spirale propose l'approche de conception d'un logiciel selon un cycle récursif composé de six étapes : spécifications, analyse, conception, implémentation, test et validation.

Ce cycle a été suivi de manière successive pour chacun des grands points du projet : réalisation du noyau en premier lieu, suivi de la mise en place des micro-outils et enfin l'optimisation avec ses propositions de solutions optimales.

A chaque stade, la validation de la solution proposée a été réalisée suite à ensemble de tests réalisés à l'aide du jeu de test présenté dans le point suivant.

1.3 Jeu de test

Afin de valider de manière progressive selon le cycle proposé la plateforme réalisée, un jeu de test fourni par l'équipe du laboratoire a été utilisé tout au long de la réalisation du projet.

Ce jeu de test est basé sur la configuration d'une chaise de bureau composée de quatre éléments (classes de composants) admettant chacun quatre variantes. Illustré par le schéma ci-après (Figure 9), cette configuration se veut simple de manière à fournir une approche pédagogique aux problèmes abordés et un suivi plus aisé des résultats obtenus.

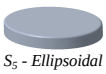
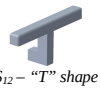
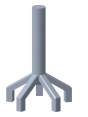
Chair components	**Physical solutions – S_k**				
	1	*2*	*3*	*4*	*5*
Seat	 S_1 - *Square*	 S_2 – *Half round*	 S_3 - *Rectangle*	 S_4 - *Round*	 S_5 - *Ellipsoidal*
Back	 S_6 - *Square*	 S_7– *Trapezoidal_a*	 S_8 – *Trapezoidal_b*	 S_9 - *Round*	 S_{10} - *Ellipsoidal*
Armrest	 S_{11} – *"L" shape*	 S_{12} – *"T" shape*	 S_{13} – *"J" shape*	 S_{14} - *Ellipsoidal*	 S_{15} - *"U" shape*
Stand	 S_{16} – *Straight_a*	 S_{17} – *Straight_b*	 S_{18} - *Round*	 S_{19} – *Slant_a*	 S_{20} – *Slant_b*

Figure 9 : Composants du jeu de test

Les spécifications clientes associées à ce jeu de test au nombre de onze sont représentées dans le tableau suivant (Tableau 1) :

Spécification	Description
r1	Taille
r2	Poids
r3	Prix
r4	Type Bureau
r5	Type Bar
r6	Type salle de classe
r7	Type classique
r8	Confort
r9	Praticité
R10	Durabilité
R11	Stabilité

Tableau 1 : Spécifications clientes du jeu de test

Les fonctions associées à ce jeu de test au nombre de quatre sont représentées dans le tableau suivant (Tableau 2) :

Fonction	Description
f1	Support du poids de la partie basse du corps d'une personne dans la position assise
f2	Support du dos d'une personne dans une position assise
f3	Support des bras d'une personne dans une position assise
f4	Offre une capacité de mouvement des pieds d'une personne dans une position assise

Tableau 2 : Fonctions du jeu de test

Les contraintes métiers associées à ce jeu de test sont regroupées au sein de quatre corps de métier (production, assemblage, maintenance et recyclage) regroupant respectivement sept, six, six et quatre contraintes. Cette répartition est représentée dans le tableau suivant (Tableau 3) :

Assemblage		Production	
Contrainte	Description	Contrainte	Description
C1-1	Permet une inspection visuelle	c2-1	Constitue des formes simples
C1-2	Permet un montage sur un axe uniforme	c2-2	Évite les différences de section transversale
c1-3	Permet une standardisation	c2-3	Évite les grandes courbures
c1-4	Permet un accès aux outils d'assemblage	c2-4	Fournit un soutien adéquat aux surfaces
c1-5	Permet un accès pour l'ajustement sans désassemblage des autres composants	c2-5	Évite les usinages inutiles
c1-6	Permet un blocage des éléments à assembler facile	c2-6	Évite les sections excessivement minces
c1-7	Permet une manipulation aisée		
Maintenance		Recyclage	
Contrainte	Description	Contrainte	Description
c3-1	Les composants peuvent être échangés facilement	c4-1	Facile à démonter et à remonter en utilisant un assemblage adapté et des techniques de fixation
c3-2	Les composants peuvent être similaires aux attentes de la vie	c4-2	Possibilité de réutilisation
c3-3	Accessibilité	c4-3	Facilité de tri
c3-4	Sureté	c4-4	Accès pour le nettoyage
c3-5	Facilité à assembler et à désassembler		
c3-6	Vise la simplicité		

Tableau 3 : Contraintes du jeu de test

1.4 Orientations générales

Lors de la phase de lancement du projet, différentes alternatives pour les différents aspects de mise en place de la nouvelle plateforme APIC ont été analysées et discutées avec l'équipe du laboratoire de

manière à définir les principales orientations suivies par le projet en terme de gestion des aspects agent, réseau, sauvegarde et évolutivité de la plateforme. Ces orientations ainsi que leurs principales alternatives sont succinctement décrites tour à tour dans les paragraphes suivants.

1.4.1 Conception des agents

La plateforme APIC reposant sur l'utilisation d'agents, il a été primordial de définir les attentes de l'équipe afin de déterminer quelle approche mettre en place. De la discussion avec l'équipe du laboratoire est ressorti un besoin de fonctionnalités assez limité des agents. Ces derniers, bien que dépassant le stade d'agents réactifs, ne sont pas entièrement cognitifs. Des outils de développement de systèmes multi-agents sont disponibles sur le marché et permettent la modélisation d'architecture orientées multi-agents respectant les différentes normes définies en terme d'échange, de langage et de structure. Cependant ces outils sont pour la plupart orientés vers la modélisation d'agents cognitifs complexes et ne permettent que très rarement d'implémenter les architectures modélisées. Dès lors l'utilisation de ces outils aurait risqué d'alourdir inutilement la plateforme en termes de fonctionnalités qui ne seraient pas utilisées par la suite par cette dernière. C'est pourquoi, en accord avec l'équipe du laboratoire, le choix de concevoir et d'implémenter les agents spécifiques sans utiliser d'outils particuliers a été retenu.

1.4.2 Support réseau

De manière à offrir à APIC le support réseau attendu, différentes approches ont été étudiées :

- La première approche permettant de distribuer APIC en réseau consiste à utiliser les sockets en vue de réaliser la communication des différents éléments à travers le réseau. Ce modèle de communication est illustré par le schéma ci-dessous (Figure 10) :

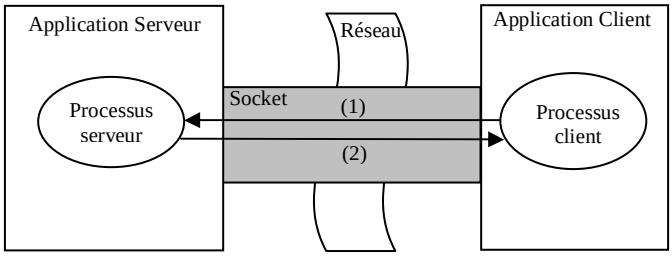


Figure 10 : Approche Socket

Comme le montre le schéma ci-dessus (Figure 10), un tube de communication bidirectionnel est créé entre l'application serveur et celle client. Le client envoie *(1)* une requête au serveur, lequel y répond ensuite *(2)*. La solution socket nécessite cependant la définition exhaustive de la structure des requêtes échangées entre client et serveur, seuls les types primaires étant transmissibles par le biais des sockets.

- La seconde approche permettant d'offrir à APIC le support réseau attendu repose sur l'utilisation des appels de procédures distantes. Cette technique permet de traiter un objet distant comme si ce dernier était présent localement sur la machine. Parmi les solutions utilisant ce mécanisme, peuvent être citées Rmi et Corba, présentées succinctement ci-après :
  - Rmi (Remote Method Invocation) est une solution d'appel de procédure distante propriétaire proposée par Sun pour son environnement de développement Java. Elle repose sur l'utilisation d'un bus d'accès réseau (Bus Rmi) permettant à un serveur de partager un objet instancié qui pourra être ensuite accédé et utilisé par un client depuis n'importe quel point du réseau. Ce processus est illustré par le schéma ci-dessous (Figure 11) :

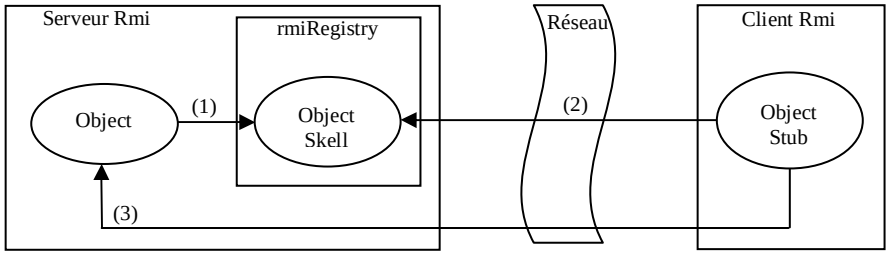


Figure 11 : Approche RMI

Comme l'illustre ce schéma (Figure 11), le serveur souhaitant partager un objet instancié met à disposition (*1*) au sein de la *rmiRegistry* le squelette de son objet (*Object Skell*). Le client enregistre alors une référence à l'objet distant (*2*) grâce à un talon de l'objet (*Object Stub*). Le client est alors en mesure d'accéder aux méthodes de l'objet distant de manière totalement transparente.

  - Corba (Common Object Request Broker Architecture) est une solution d'appel de procédure distante née d'un consortium créé en 1989 de plus de 800 membres en vue de favoriser l'interopérabilité et la portabilité d'applications réparties. Tout comme

Rmi, Corba repose sur l'utilisation d'un bus d'accès réseau (*ORB - Object Request Broker*) permettant à un serveur de partager un objet instancié qui pourra être ensuite accédé et utilisé par un client depuis n'importe quel point du réseau. Ce mécanisme est illustré par le schéma ci-dessous (Figure 12) :

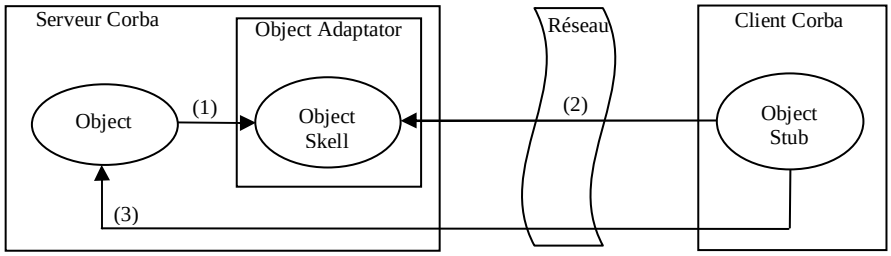


Figure 12 : Approche Corba

Comme l'illustre ce schéma (Figure 12), le serveur souhaitant partager un objet instancié met à disposition (1) au sein de l'*Object Adaptator* (*OA*) le squelette de son objet (*Object Skell*). Le client enregistre alors une référence à l'objet distant (2) grâce à un talon de l'objet (*Object Stub*). Le client est alors en mesure d'accéder aux méthodes de l'objet distant de manière totalement transparente. Contrairement à Rmi, Corba propose une interopérabilité entre les différents langages de programmation et plateformes. Pour cela les objets Corba sont compilés de manière autonome et utilisés ensuite par les différents systèmes à l'aide d'une interface commune (*IDL - Interface Description Language*).

Après discussion avec l'équipe du laboratoire, la solution Rmi a été retenue, la plateforme ne nécessitant pas à l'heure actuelle de permettre un accès depuis d'autres plateformes que Java. De plus Rmi offre un support plus vaste sur la possibilité de rendre un objet complexe distant accessible, du moment que ses composants soient sérialisables, par rapport à la technologie Corba qui est plus restrictive sur les types composant l'objet partagé.

1.4.3 Sauvegarde

De manière à permettre à la plateforme de sauvegarder les différents paramètres saisis, deux approches ont été étudiées :

- La première approche consiste à utiliser une base de données dans laquelle seraient stockées l'ensemble des paramètres saisis de la plateforme. Cette approche nécessite la mise en place d'un système de gestion de base de données ainsi que sa localisation dans la plateforme.
- La seconde approche consiste à utiliser des fichiers xml en vue de stocker les paramètres saisis de la plateforme. Cette approche, déjà utilisée dans les versions précédentes de la plateforme, permet une gestion plus aisée de l'emplacement du stockage des paramètres et de son éventuel déplacement ainsi qu'une modification manuelle directe plus facile.

Après discussion avec l'équipe du laboratoire, la solution xml a été retenue.

1.4.4 Evolutivité

Afin de répondre aux besoins d'évolutivité de la plateforme mise en œuvre aussi bien en nombre d'agents gérés qu'en nombre de communautés utilisées ou bien encore de type de calcul, un soin particulier a été apporté à l'analyse des résultats attendus de manière à généraliser au maximum les composants de la plateforme et leurs comportements.

Ainsi l'analyse des calculs engendrés par les relations intercommunautaires montre qu'il est possible de généraliser les calculs réalisés selon le schéma suivant (Figure 13):

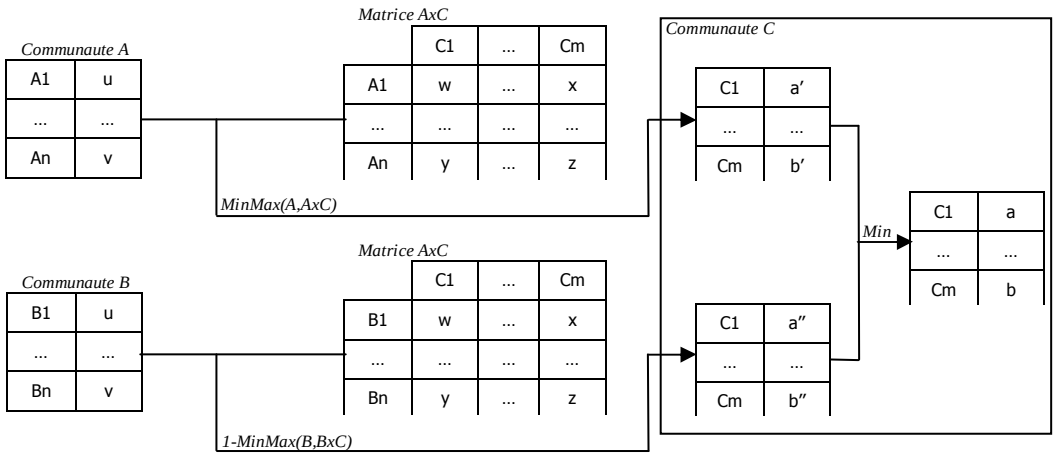


Figure 13 : Représentation des relations intercommunautaires

En effet une communauté peut avoir de 1 à n relations influées qui déterminent ses propres valeurs. En réalisant en premier lieu le type de calcul associé à chaque communauté (*MinMax* ou *1-MinMax*)

pour chacune de ses relations influées, puis en réalisant en second lieu le minimum de ces premiers, le résultat obtenu correspond alors aux attentes spécifiées.

Le cas particulier de la communauté contrainte, où les différents métiers interviennent de manière autonome sur leurs propres contraintes, entre dans ce cadre en impliquant des calculs intermédiaires supplémentaires représentés par le schéma ci-dessous (Figure 14):

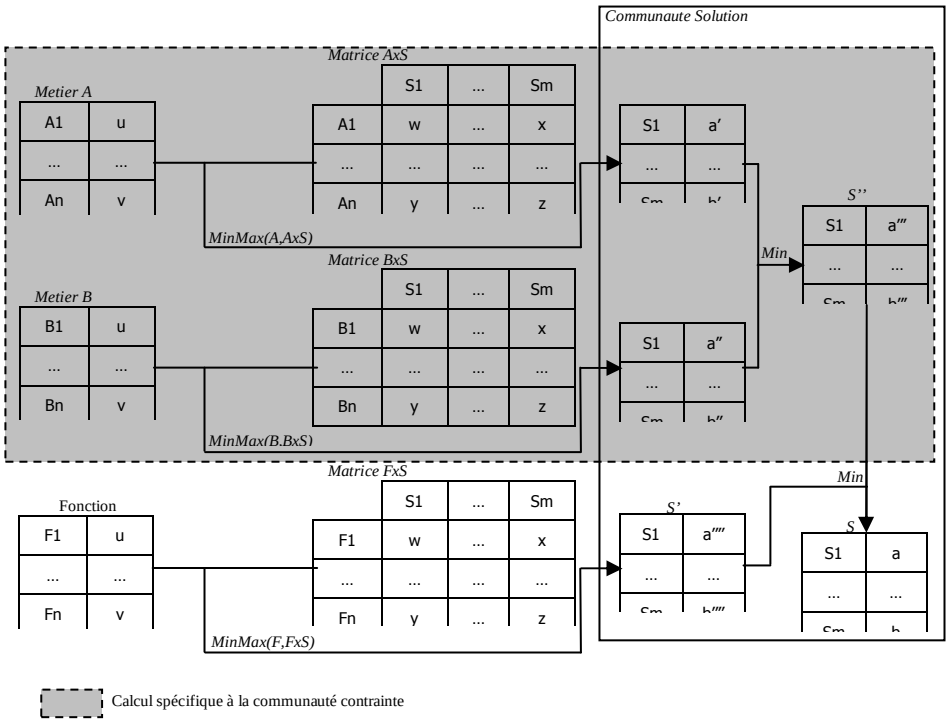


Figure 14 : Cas particulier des contraintes

L'analyse de ces calculs intermédiaires montrent qu'en segmentant physiquement en n communautés la communauté logique contrainte, où chaque nouvelle communauté sera associée à l'un des n corps de métier intervenants dans le processus de configuration de produit, le résultat final obtenu est identique. En effet le minimum d'un minimum d'un ensemble de valeurs avec d'autres valeurs est égal au minimum de l'ensemble de ces valeurs.

$$Min(Min(val1,val2,\ldots),valn)=Min(val1,val2,\ldots,valn)$$

L'application de ce processus permet une approche globale du fonctionnement des communautés de la plateforme et permet une modularité de la plateforme en termes du nombre de communautés constituant l'univers de fonctionnement de cette dernière.

De plus, comme le montre les travaux du laboratoire [Fougères, 2004], les agents au sein d'une même communauté sont identiques. Ceci permet donc ensuite de pouvoir généraliser à leur tour les agents de la plateforme ainsi que leurs fonctionnalités.

Enfin, la généralisation des composants de la plateforme permet un entretien du code plus aisé aussi bien pour la modification des calculs et algorithmes mis en œuvre que pour le rajout de nouvelles fonctionnalités tant au niveau communauté qu'au niveau agent.

2. *Le noyau*

Ce point s'intéresse à la conception et au développement du noyau de la nouvelle plateforme, seconde phase de réalisation du projet mené. Après une présentation des outils permettant l'utilisation et la manipulation des fichiers de configurations liées à la plateforme, les principaux composants constituant ce noyau seront abordés.

2.1 La manipulation des fichiers de configuration

La nouvelle plateforme devait permettre un paramétrage dynamique ainsi qu'une saisie des données et leur sauvegarde. Comme présenté dans les orientations générales (1.4 Orientations générales, p.21), le choix de l'utilisation de fichiers Xml a été réalisé. De manière à pouvoir utiliser et manipuler ces fichiers une classe *ChargeurXML* a été implémentée. Le diagramme de classe ci-dessous (Figure 15) illustre cette classe :

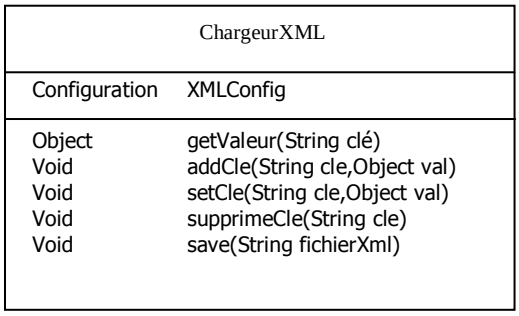


Figure 15 : Diagramme de classe simplifié ChargeurXML

Comme le modèle ce diagramme (Figure 15), cette classe permet l'accès (*getValeur(…)*), la modification (*setCle(…)*) ainsi que la création de clés (*addCle(…)*) au sein du fichier Xml manipulé. La sauvegarde des modifications apportées au fichier est réalisée par la méthode *save(…)* qui admet en paramètre le chemin et le nom du fichier Xml à sauvegarder.

2.2 Plateforme

Premier élément constituant le noyau, l'élément Plateforme sera présenté dans les points suivants. Après avoir détaillé les éléments de paramétrage issus du fichier de configuration Xml, les principaux rôles de cet élément seront décrits.

2.2.1 Paramétrage

La configuration des paramètres globaux de la plateforme est réalisée à l'aide du fichier Xml nommé configuration.xml dont le contenu est présenté par la capture suivante (Figure 16) :

```
<Plateforme>
        <Seuil>0.4</Seuil>
</Plateforme>
```

Figure 16 : Capture du fichier de configuration de la plateforme

Comme le montre cette capture (Figure 16), le fichier Xml de configuration lié à la plateforme ne comporte à l'état actuel qu'une seule clé (*Seuil*) liée à la définition globale du seuil en dessous duquel la valeur propre d'un agent sera mise à 0.

Ce fichier pourra acquérir dans les évolutions futures de la plateforme l'ensemble des paramètres communs aux différents éléments composant la plateforme.

2.2.2 Création des communautés

La charge de créer les différentes communautés formant l'univers de travail est attribuée à la plateforme. Cette création se réalise en trois principales phases, illustrées par le diagramme suivant (Figure 17) :

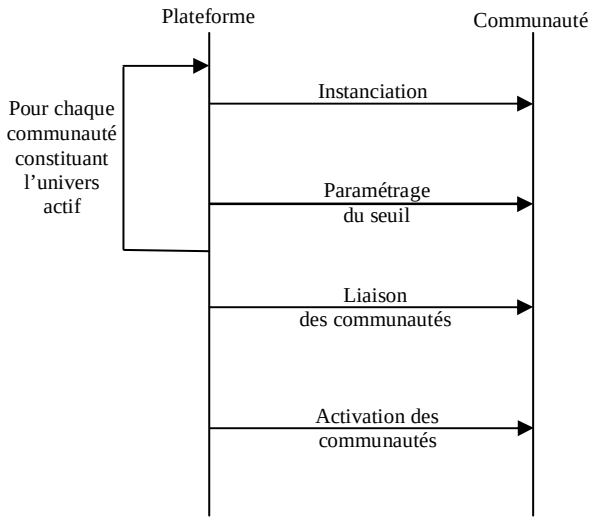


Figure 17 : Création des communautés

Comme le montre ce diagramme (Figure 17), la plateforme instancie en premier lieu chaque communauté. Durant cette phase le nom ainsi que le chargeurXml associés à la communauté lui sont transmis ainsi que le seuil qui sera utilisé ultérieurement par les agents.

En second, la plateforme crée pour chacune des communautés qu'elle héberge, les liaisons influentes qu'elle possède avec les autres communautés, selon le paramétrage stocké dans les fichiers xml de configuration des communautés. En parallèle, les communautés concernées initialisent les relations influées avec ces communautés.

Enfin la plateforme active les différentes communautés.

2.2.3 Détection des phases de configuration

Les principales phases de la réalisation de propositions de configurations optimales de produit se succèdent avec en premier lieu la définition des spécifications clientes, puis la saisie des contraintes métiers associées à ces derniers; le calcul des solutions floues consensuelles est alors réalisé suivi du calcul des propositions de configurations optimales.

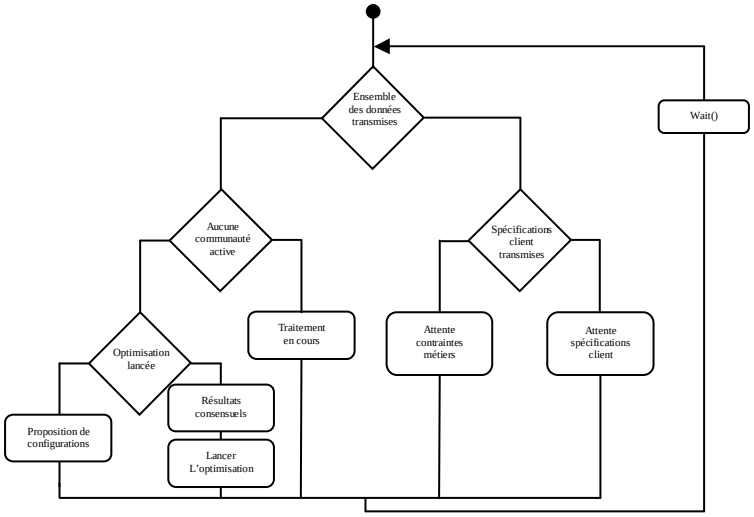


Figure 18 : Détection des phases de configuration

Comme le montre le diagramme ci-dessus (Figure 18), de par son rôle de supervision des différentes communautés la constituant, la plateforme détecte ces différentes phases en utilisant leurs propriétés *donnéesTransmises* et *statutCommunauté* qu'elle stocke et met à jour au sein des vecteurs *donnéesTransmises* et *statusCommunautes*. L'association de l'index de position dans ces deux vecteurs avec sa communauté respective est stockée grâce à deux tables de hachage, *htDonnéesTransmises* et *htStatutsCommunautes*, admettant pour clé la référence de la communauté et comme valeur la position au sein du vecteur respectif. La mise à jour de ces informations est représentée par le diagramme suivant (Figure 19) :

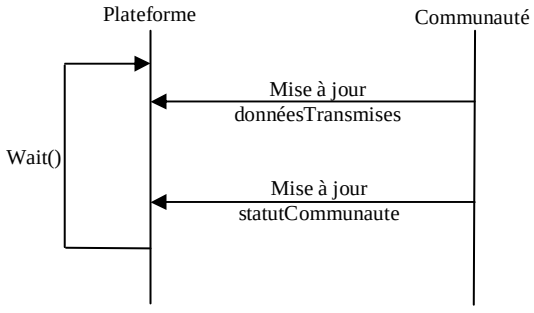


Figure 19 : Supervision des communautés

2.2.4 Support du déploiement réseau

La plateforme instancie un objet ServeurRmi qui a pour rôle de permettre le partage et l'accès distant des objets gérés par la plateforme. Lors de son instanciation, le *serveurRmi* initialise une RmiRegistry si aucune RmiRegistry n'est disponible sur le poste hébergeant le *serveurRmi*.

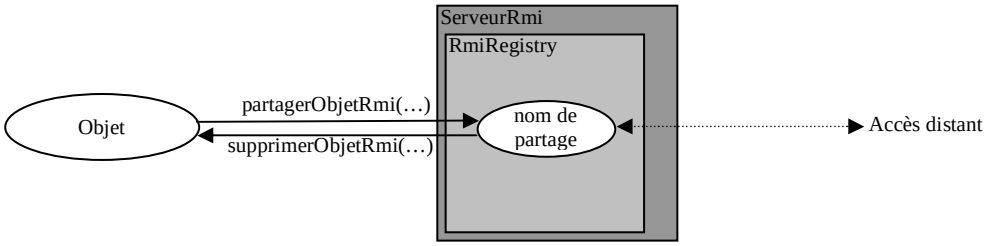


Figure 20 : Le serveurRmi

Comme l'illustre le schéma ci-dessus (Figure 20) le partage d'un objet s'effectue en faisant appel à la méthode *partagerObjetRmi* qui admet en paramètres l'objet à partager ainsi que son nom de partage. L'objet ainsi partagé est ensuite accessible par son nom de partage depuis les autres stations. Dans la situation où le nom de partage est déjà utilisé, l'ancien partage est alors écrasé par le nouveau. La suppression d'un partage s'effectue quand à elle par l'appel de la méthode *supprimerObjetRmi* qui admet comme paramètre le nom de partage de l'objet dont il faut supprimer l'accès distant.

2.3 Communautés

Second élément constituant le noyau, l'élément Communauté sera présenté dans les points suivants. Après avoir détaillé les éléments de paramétrage issus du fichier de configuration Xml, les principaux rôles de cet élément seront décrits.

2.3.1 Paramétrage

La configuration des paramètres d'une communauté lambda est réalisée à l'aide du fichier Xml nommé lambda.xml dont le contenu est présenté par la capture suivante (Figure 21):

```
<Communaute>
        <nom>nomCommunaute</nom>
        <relationsIntraCommunaute value="false" />
        <relationsCommunaute>
                <Communaute nom="nomCommunauteLiee"
                                id_type_comput="1" />
                … Pour chaque association avec une communauté …
        </relationsCommunaute>
        <Agent nom="r1" description="detaille r1">
                <relationsAgent>
                        <Agent nom="f1">0.7</Agent>
                        <Agent nom="f2">0.7</Agent>
                        … Pour chaque association avec un agent …
                </relationsAgent>
        </Agent>
```

Figure 21 : Capture du fichier de configuration d'une communauté

Comme le montre cette capture (Figure 21), le fichier Xml de configuration lié à une communauté comporte différentes clés qui permettent de définir :

- Le nom de la communauté (*nom*)
- L'existence de relations intracommunautaire (*relationsIntraCommunautaires*)
- Les relations intercommunautaires avec d'autres communautés (*relationsCommunaute*) identifiées par une sous clé (*Communauté*) qui détermine la communauté influée (*nom*) et le type des calculs engendrés (*id_type_comput*)
- Les agents (*Agent*) qui constituent la communauté sont définis par leur nom (nom) et leur description (*description*). Une sous-clé définit les relations qu'un agent possède (*relationsAgent*) et sera transmise ultérieurement à l'agent intéressé pour effectuer son paramétrage.

2.3.2 Création des agents

La communauté instancie chaque agent la formant selon la liste des agents gérés par la communauté dans le fichier de configuration xml de cette dernière. Chaque agent créé est ajouté au vecteur

agentsGeres. En parallèle la position de cet agent dans ce dernier est utilisée comme valeur au sein de la table de hachage *htAgentsParRef* qui admet comme clé l'agent ajouté. De la même manière l'association entre agent et nom d'agent est insérée dans la table de hachage *htAgentsParNom* utilisant pour clé le nom de l'agent et comme valeur la référence de l'agent. Enfin la communauté paramètre le seuil de l'agent. L'ensemble de ces opérations est schématisé par le diagramme ci-dessous (Figure 22) :

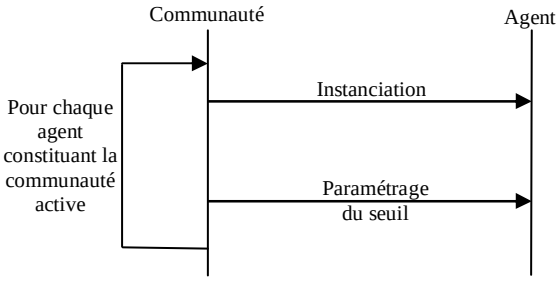


Figure 22 : Création des agents

2.3.3 Association des agents

A la demande de la plateforme, la communauté a en charge d'associer chacun de ses agents selon le paramétrage issu du fichier de configuration selon le diagramme suivant (Figure 23) :

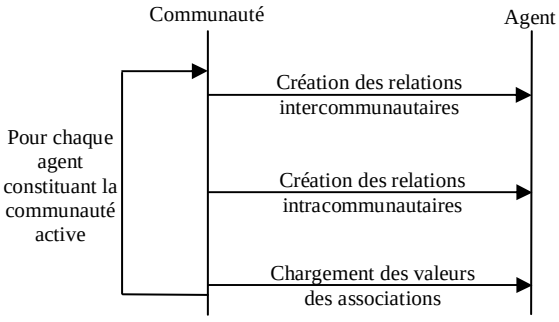


Figure 23 : Association des agents

Comme le montre ce diagramme (Figure 23), la communauté crée en premier lieu les relations intercommunautaires. Pour chacun des agents de chacune des communautés influées, issues du vecteur *communautesInfluees*, la communauté initialise alors une nouvelle connaissance pour chacun

de ces agents. En parallèle, la liaison inverse de l'agent de la communauté liée vers l'agent de la communauté est elle aussi initialisée.

En second lieu, si elles ont été définies, la communauté crée les relations intracommunautaires. La communauté initialise alors pour chacun de ses agents une nouvelle connaissance avec chacun des autres agents formant la communauté.

Enfin les valeurs d'associations, issues du fichier de configuration xml de la communauté, sont transmises aux agents.

2.3.4 Supervision des agents

La communauté a pour autre rôle de superviser les agents qui la composent. Cette supervision se réalise en deux aspects distincts.

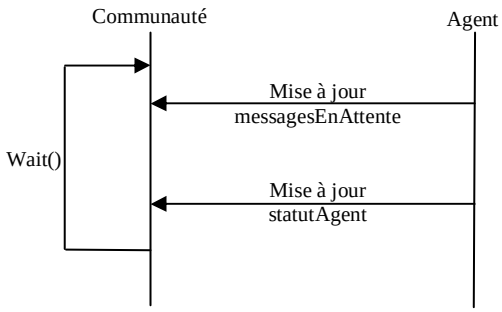


Figure 24 : Supervision des agents

Le premier aspect, comme l'illustre le diagramme ci-dessus (Figure 24), permet à la communauté de superviser l'état de ses agents et d'en déduire son propre état afin d'en informer la plateforme pour que cette dernière puisse déterminer l'avancement de la configuration de produits.

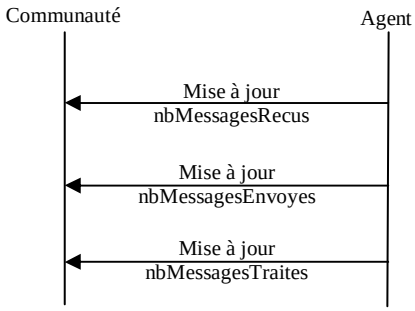


Figure 25 : Supervisions des quantités de messages échangés

Le second aspect, comme le montre le schéma ci-dessus (Figure 25), observe un rôle de comptabilisation du nombre de messages envoyés, reçus et traités par ses agents, leurs additions et leurs transmissions aux micros outils.

2.3.5 Point d'accès des micros outils

La communauté a pour autre rôle de proposer le point d'accès des différents micros outils. Cet accès est généré par le biais de l'objet *AccesMicroOutil* qui propose un ensemble de méthodes permettant l'accès aux communautés et à leurs agents.

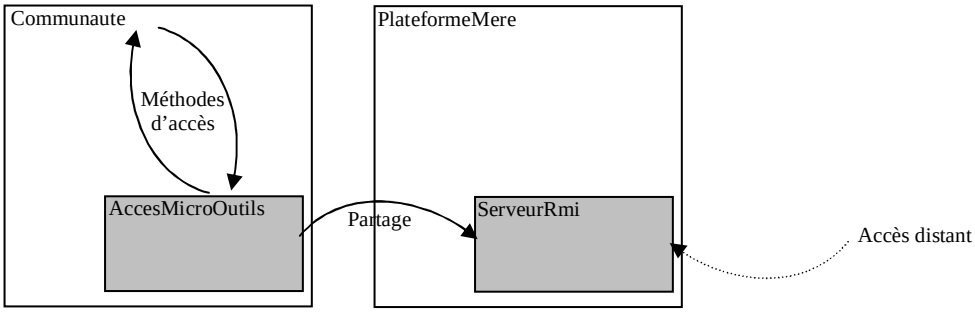


Figure 26 : Schéma du AccèsMicroOutils

Chaque communauté dispose ainsi de son propre *accesMicroOutil* qu'elle partage sur le réseau par le biais du *serveurRmi* de la plateforme qui la contient sous le nom de partage correspondant à son nom de communauté. L'accès distant se réalise par le biais de l'interface *interfaceAccèsMicroOutil*.

2.4 Agents

Dernier élément constituant le noyau, l'élément Agent sera présenté dans les points suivants. Après avoir détaillé les éléments de paramétrage issus du fichier de configuration Xml, les principaux rôles de cet élément seront décrits.

2.4.1 Paramétrage

La configuration des paramètres d'un agent lambda est réalisée à l'aide d'une sous partie du fichier de configuration de sa communauté présenté par la capture suivante (Figure 27) :

```
<Agent nom="r1" description="detaille r1">
        <relationsAgent>
                <Agent nom="f1">0.7</Agent>
                <Agent nom="f2">0.7</Agent>
                … Pour chaque association avec un agent …
        </relationsAgent>
</Agent>
```

Figure 27 : Paramètres de configuration des agents

Comme le montre cette capture (Figure 27), le contenu de la sous-clé *relationsAgent* est transmis par la communauté à l'agent. Ce contenu est composé de clés qui correspondent aux relations de l'agent avec les autres agents (*Agent*) qui sont identifiés par leurs noms (*nom*) et dont la valeur d'association définit la valeur des clés.

2.4.2 Modélisation des agents de la plateforme

La modélisation des agents conduit au diagramme de classe simplifié suivant (Figure 28) :

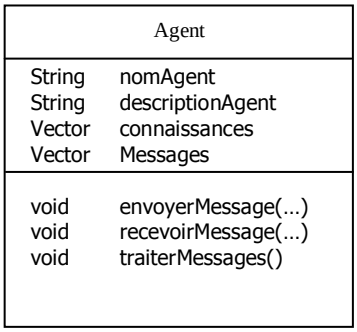


Figure 28 : Diagramme de classe simplifié Agent

Comme le montre ce diagramme (Figure 28), un agent possède parmi ces propriétés un nom (*nomAgent*), une description (*descriptionAgent*), un vecteur regroupant ses connaissances (*connaissances*, présentées dans le point suivant), ses messages non traités (*messages*, présentés dans le point suivant). Afin de lui permettre de communiquer avec ses homologues, un agent dispose parmi ses méthodes, celles qui lui permettent d'envoyer (*envoyerMessage(…)*), de recevoir (*recevoirMessage(…)*) et de traiter (*traiterMessage()*) des messages.

2.4.3 Validation du modèle agent

De manière à valider le modèle des agents proposés par rapport au concept agent usuel, il est nécessaire de valider les points suivants :

- Connaissances propres

L'analyse des besoins a montré que les connaissances d'un agent de la plateforme portent sur les relations qu'il possède avec les agents des autres communautés ou de sa propre communauté ainsi que les valeurs propres de ces agents. Ce sont ces connaissances qui permettront à l'agent de déterminer sa propre valeur. Ces connaissances sont locales et donc propres à chaque agent qui les maintient à jour grâce aux messages qu'il échange avec ses connaissances.

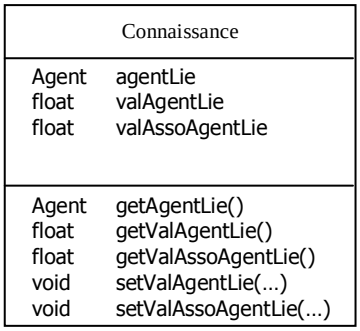


Figure 29 : Diagramme de classe simplifié Connaissance

Comme l'illustre le diagramme de classe ci-dessus (Figure 29), les connaissances de l'agent ont donc été implémentées à l'aide de la classe Connaissance. Cette classe a pour propriétés la référence (*agentLie*), la valeur propre (*valAgentLie*) et la valeur d'association de l'agent lié (*valAssoAgentLie*). Les différentes valeurs sont de type float afin de représenter leur aspect flou. Cette classe dispose aussi de méthodes qui permettent l'accès et la modification à ces propriétés respectives.

- Communiquant

L'analyse des besoins en terme de communication d'un agent montre que ce dernier a besoin de pouvoir communiquer aux autres agents, d'une part sa valeur propre, et d'autre part sa valeur d'association. Dans le cadre des échanges de valeurs d'associations, un accusé de réception (ack) est

attendu par l'agent émetteur de la proposition. Ce protocole est représente par le schéma suivant (Figure 30) :

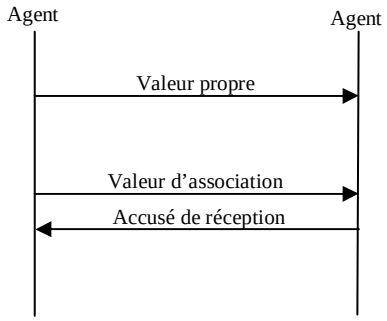


Figure 30 : Echanges des valeurs d'associations

A terme, dans les évolutions futures de la plateforme, un agent devrait être en mesure de discuter ses valeurs d'associations avec les agents concernés en cas de désaccord sur une liaison. Ces échanges futurs sont illustrés par le diagramme ci-dessous (Figure 31):

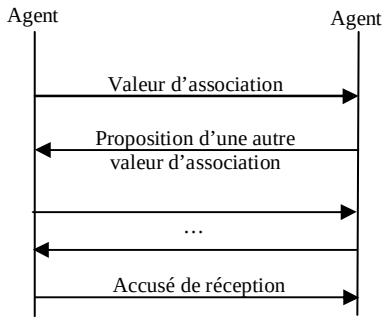


Figure 31 : Discussion des valeurs d'associations

Pour répondre à ces besoin, la communication entre deux agents est implémentée au niveau agent par les méthodes *envoyerMessage* et *recevoirMessage*. A la réception d'un message, l'agent concerné ajoute le message reçu au vecteur *messages*. Afin d'éviter des inter-blocages entre deux agents s'envoyant simultanément des messages ou deux agents envoyant simultanément un message au même agent, la méthode *recevoirMessage* est synchronisée, bloquant ainsi l'accès aux méthodes et propriétés de l'agent recevant le message durant l'appel à cette méthode.

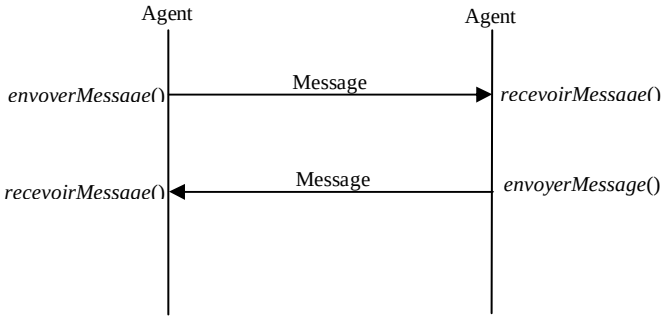


Figure 32 : Echanges entre agents

Les messages échangés par un agent ont été implémentés par la classe Message selon le diagramme de classe ci-dessous (Figure 33):

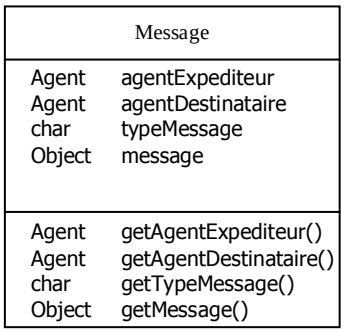


Figure 33 : Diagramme de classe simplifié Message

Comme le montre ce schéma, un message échangé entre agents comporte donc la référence des agents émetteur (*agentExpediteur*) et récepteur (*agentDestinataire*), une propriété distinguant le type de message échangé (*typeMessage*) et enfin une autre propriété contenant le corps propre du message (*message*).

Les différents types de messages échangés forment le langage représenté par le tableau suivant :

Type de message	Rôle	Contenu du message
a	Définition de la valeur d'association	Valeur d'association entre l'agent expéditeur et celui destinataire du message
A	Confirmation de la prise en compte de la valeur d'association	Valeur d'association entre l'agent expéditeur et celui destinataire du message

v	Diffusion de la valeur de l'agent	Valeur propre de l'agent expéditeur du message
o	Recherche de solutions optimales	Vecteur contenant la valeur de solution optimale et le vecteur des solutions optimales

Tableau 4 : Types des messages échangés

- Autonome

L'autonomie des agents de la plateforme est assurée en threadant les objets de type Agent. Le thread obtenu boucle alors sur le traitement des messages qu'il reçoit jusqu'à n'avoir plus aucun message à traiter dans le vecteur *messages*. Une fois l'ensemble des messages en attente traités, il se met en veille jusqu'à réception de nouveaux messages à traiter. Ce fonctionnement est illustré par le diagramme suivant (Figure 34) :

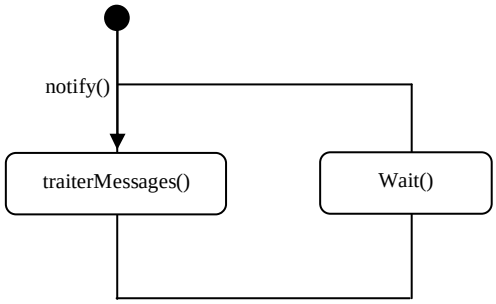


Figure 34 : Autonomie des agents

A chaque mise en vielle du thread l'agent communique à sa communauté sa suspension, puis à son réveil il lui communique sa réactivation.

Le traitement des messages s'effectue selon l'algorithme représenté par le schéma ci-dessous (Figure 35):

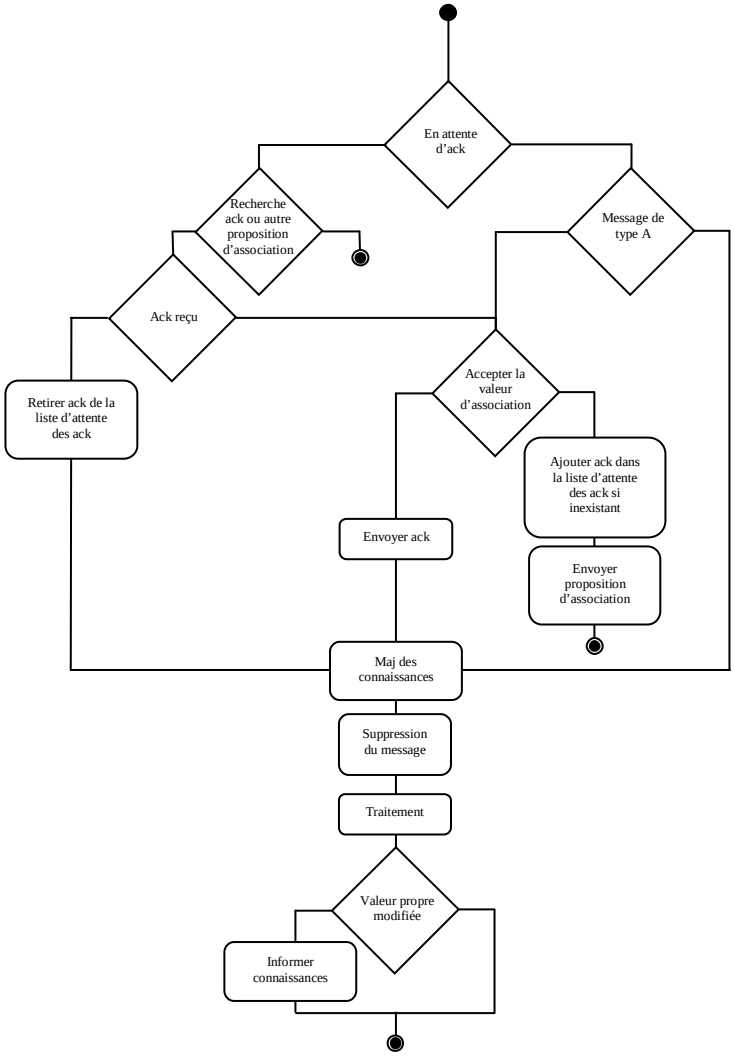


Figure 35 : Traitement des messages

Comme le montre ce diagramme (Figure 35), l'algorithme ainsi mis en place tient compte de la nécessité pour un agent de recevoir la totalité des accusés de réception des associations proposées avant de traiter les autres messages. La gestion des accusés en attente se réalise à l'aide du vecteur *enAttenteAckDe* dans lequel sont stockées les références des agents pour lesquels l'agent attend un accusé.

L'implémentation du mécanisme de traitement des messages offrant la possibilité de négocier ses associations a été réalisée. Ainsi lorsqu'un agent se trouve en attente d'un accusé d'un autre agent, s'il reçoit une autre proposition d'association de cet agent, il détermine à l'aide de la procédure

accepterAssociation s'il accepte ou non cette nouvelle valeur. Dans le cas d'une acceptation, il envoie un accusé à l'agent émetteur de la nouvelle proposition acceptée, supprime la référence à cet agent dans le vecteur *enAttenteAckDe* et met ensuite à jour ses connaissances sur l'agent émetteur. Dans le cas d'un refus, il doit alors proposer à l'agent émetteur une nouvelle proposition d'association et maintenir ou ajouter la référence à l'agent émetteur dans le vecteur *enAttenteAckDe*. L'agent émetteur suivra alors le même raisonnement à la réception de l'acceptation ou du refus. Cependant la procédure *accepterAssociation* accepte actuellement par défaut toutes les propositions, cette gestion des échanges figurant parmi les évolutions futures de la plateforme.

2.4.4 Relations intercommunautaires

L'adaptation au niveau agent des calculs générés par les relations intercommunautaires définies dans le cahier des charges se schématisent de la manière suivante (Figure 36):

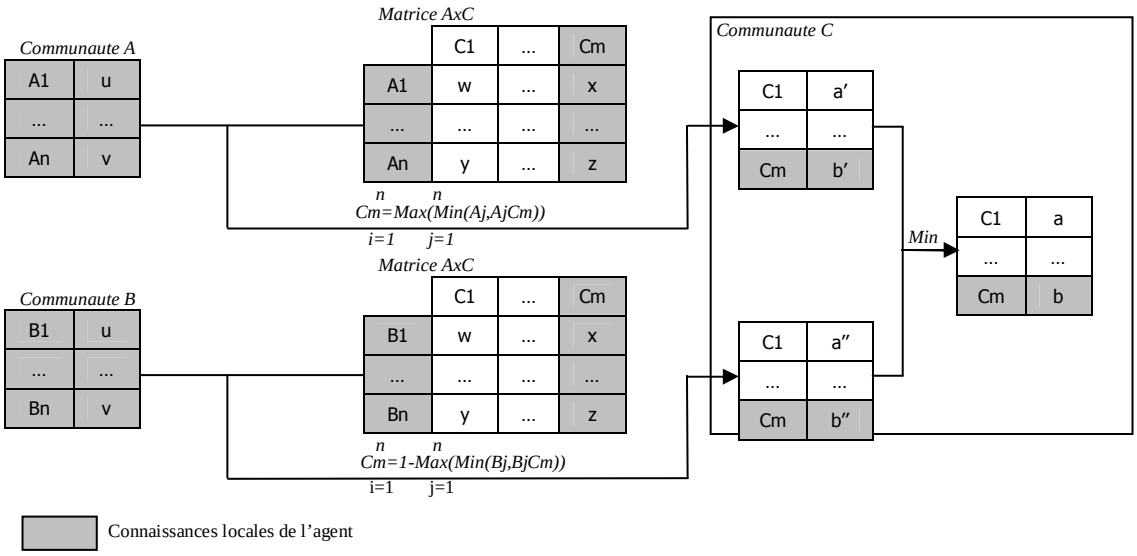


Figure 36 : Adaptation agent des relations intercommunautaires

De manière à mettre en place ces calculs, les connaissances de chaque agent sont organisées par communauté source dans le vecteur *connaissancesParCommunaute*. Chaque index de ce vecteur correspond à une communauté dont sa relation influe sur la valeur de l'agent. L'association entre la

communauté et sa position dans le vecteur est mémorisée dans la table de hachage *htConnaissancesParCommunaute*. Cette table admet pour clé la communauté concernée et pour valeur la position dans le vecteur. Dans chaque index du vecteur est situé un second vecteur regroupant les connaissances de l'agent pour la communauté indexée. En parallèle, et pour répondre au besoin de pouvoir réaliser des calculs différents selon les différentes communautés concernées, une seconde table de hachage, *htTypeCompuParCommunaute*, comporte comme clé la communauté concernée et pour valeur le type de calcul attendu.

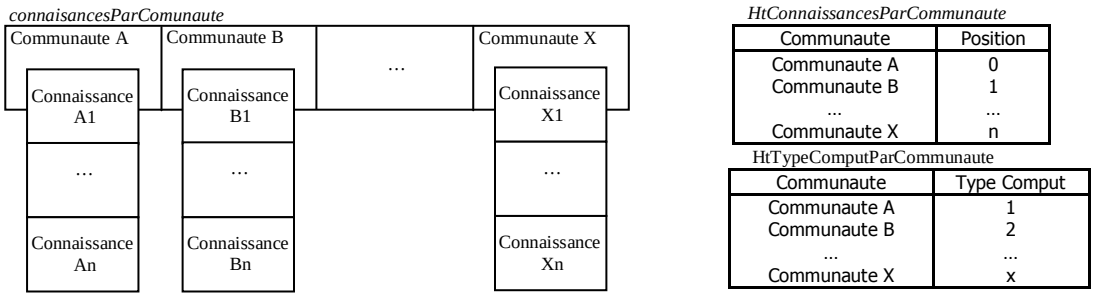


Figure 37 : Organisation des connaissances

A chaque modification de connaissance provenant d'une communauté influant la valeur de l'agent, valeur propre ou valeur d'association d'un agent, le calcul issu de la table de hachage *htTypeComputComput* est alors mené sur l'index du vecteur *connaissancesParCommunaute* déterminé à l'aide de la table de hachage *htConnaissanceParCommunaute* correspondant à la communauté concernée par la modification de connaissance.

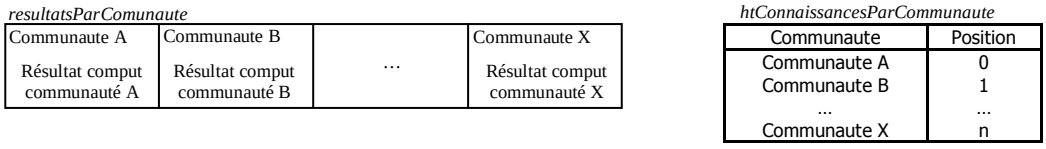


Figure 38 : Mémorisation des résultats intermédiaires

Le résultat de ce calcul est alors stocké dans l'index du vecteur *resultatsParCommunaute* déterminé à l'aide de la table de hachage *htResultatsParCommunaute* correspondant à la communauté ayant entrainé le calcul.

La valeur de l'agent résulte alors de la recherche du minimum du vecteur *resultatsParCommunaute*.

2.4.5 Relations intracommunautaires

Les relations intracommunautaires définies par le cahier des charges se proposent d'influer sur les valeurs d'associations entre la communauté qui les possèdent et les communautés en relation influées avec cette dernière.

L'adaptation au niveau agent des calculs générés par ces relations intracommunautaires se schématisent de la manière suivante :

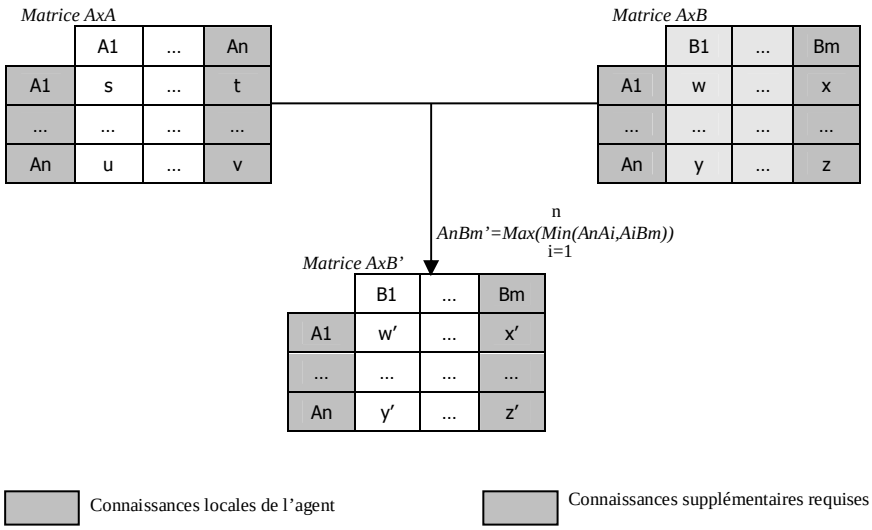


Figure 39 : Adaptation agent des relations intracommunautaires

Comme le montre ce schéma, l'application de ces relations intracommunautaires au niveau agent nécessiterait d'augmenter les connaissances de l'agent à l'ensemble des valeurs d'associations des agents de sa communauté avec la communauté influée.

Pour parer ce problème d'augmentation des connaissances, une seconde approche dans laquelle les relations intracommunautaires influeraient directement la valeur propre de l'agent de la communauté concernée par ces relations a été proposée à l'équipe du laboratoire. Cette seconde approche est représentée par le schéma ci-dessous (Figure 40) :

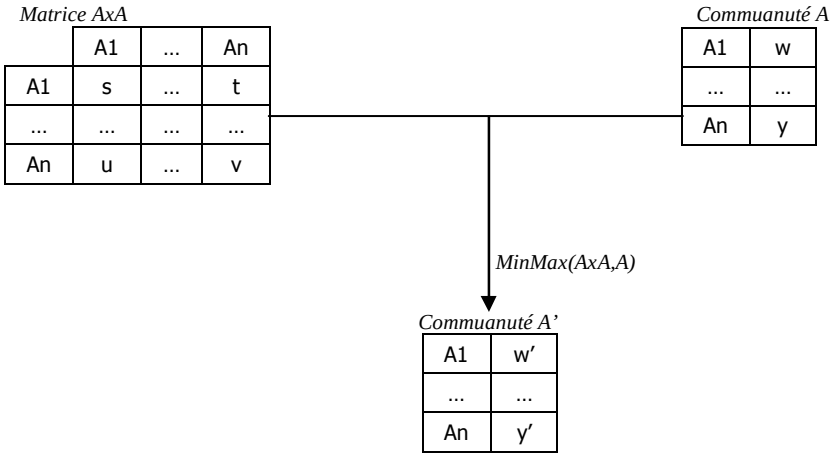


Figure 40 : Modification de l'impact des relations intercommunautaires

L'adaptation de cette nouvelle approche au niveau agent peut-être représentée par le schéma suivant (Figure 41) :

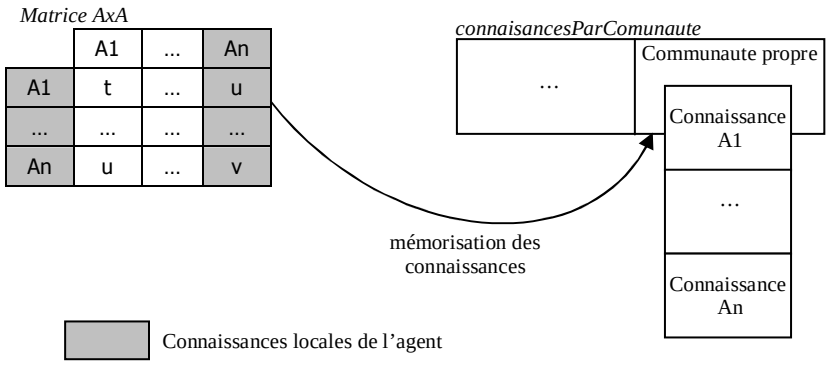


Figure 41 : Adaptation agent de la modification

Comme l'illustre ce schéma (Figure 41), cette nouvelle approche propose de traiter les relations intracommunautaires de manière identique à l'approche intercommunautaire. Les connaissances engendrées par ces relations intracommunautaires sont alors stockées dans le vecteur *connaissancesParCommunaute*. La référence à la communauté ainsi que sa position dans le précédent vecteur sont aussi stockées dans la table de hachage *htConnaissancesParCommunaute*.

Le calcul alors mené est représenté par le schéma ci-dessous (Figure 42):

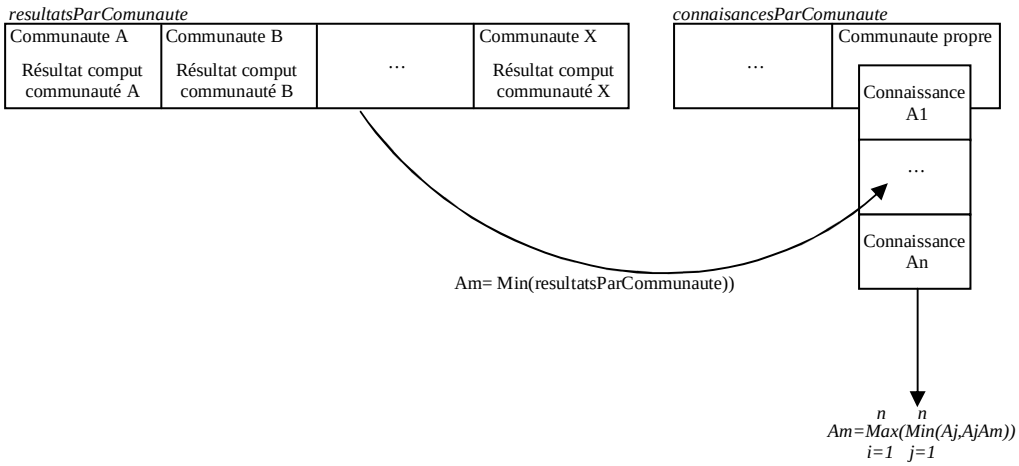


Figure 42 : Adaptation des modifications aux agents

Comme le montre ce schéma (Figure 42), la valeur du calcul résultant des relations intercommunautaires n'est plus directement communiquée aux connaissances de l'agent mais est alors stockée dans la connaissance propre de l'agent sur lui-même dans le vecteur *connaissancesParCommunaute*. Le MinMax des connaissances de la communauté propre à l'agent est alors effectué, déterminant la nouvelle propre de l'agent et sa communication aux connaissances de l'agent.

Les résultats des calculs menés par les communautés influées restent identiques avec cette nouvelle approche. En effet les agents des communautés influées utilisent leurs relations intracommunautaires pour déterminer leurs valeurs selon le mode suivant

$$Cm = \max_{j=1}^{n}(\min(Aj, AjCm))$$

La première approche qui propose d'influer les valeurs d'associations intercommunautaires substitue la valeur *AjCm* par *AjCm'* déterminée de la manière suivante :

$$AjCm' = \max_{k=1}^{n}(\min(AjAk, AkCm))$$

L'agent influé détermine alors sa valeur de la manière indirecte suivante :

$$Cm=\max_{j=1}^{n}(\min(Aj, \max_{k=1}^{n}(\min(AjAk,AkCm))))$$

$$Cm=\max_{j=1}^{n}(\min_{k=1}(Aj, AjAk,AkCm))$$

La seconde approche qui propose d'influer les valeurs des agents substitue la valeur *Aj* par *Aj'* déterminée de la manière suivante :

$$Aj'=\max_{k=1}^{n}(\min(Ak,AkAj))$$

L'agent influé détermine alors sa valeur de la manière indirecte suivante :

$$Cm=\max_{j=1}^{n}(\min(\max_{k=1}^{n}(\min(Ak,AkAj)),AjCm))$$

$$Cm=\max_{j=1}^{n}(\min_{k=1}(Ak,AkAj,AjCm))$$

Les matrices sur lesquelles se réalise le Max de détermination des valeurs des agents liés peuvent être représentées de la manière suivante :

1ère approche

	k=1	…	k=n
j=1	*Min(A1, A1A1,A1Cm)*	…	*Min(A1, A1An,AnCm)*
…	…	…	…
j=n	*Min(An, AnA1,A1Cm)*	…	*Min(An, AjAn,AnCm)*

2nde approche

	k=1	…	k=n
j=1	*Min(A1,A1A1,A1Cm)*	…	*Min(An,AnA1,A1Cm)*
…	…	…	…
j=n	*Min(A1,A1An,AnCm)*	…	*Min(An,AnAn,AnCm)*

Ce schéma montre que chaque colonne de la matrice issue de la première approche est égale à la ligne respective de la matrice issue de la seconde approche proposée.

Le Max étant réalisé sur l'ensemble de la matrice, les résultats sont donc identiques, la seconde approche permettant de rester dans un domaine de connaissances locales des agents limitées aux seuls agents avec lesquels il est en relation directe.

3. Les micro-outils

Ce point présente le troisième volet de la réalisation du projet qui s'intéresse à la conception et au développement de l'interface graphique de la plateforme et de ses modules.

3.1 Le concept de micro-outils

Le concept de micro-outils [Fougères, 2006], consiste à décomposer une interface logicielle en un ensemble de modules adaptés à la réalisation de tâches élémentaires. Plusieurs de ces micro-outils peuvent être associés pour accomplir des tâches plus complexes.

3.2 Le micro-outil de gestion de la plateforme

Ce micro-outil est destiné à deux principaux modes d'utilisation. Le premier mode est orienté paramétrage et acquisition de données de configuration pour la génération de jeux de configuration, le second mode permet la réutilisation de données de configuration déjà acquises et sauvegardées ainsi que leur modification dans un contexte d'élaboration des produits avec les différents acteurs.

3.2.1 Initialisation de la plateforme

Afin de permettre à l'utilisateur de suivre l'état d'avancement de la phase d'initialisation de la plateforme, une fenêtre (Figure 43) reprenant les principales étapes de cette initialisation en y associant une barre de progression a été mise en place.

Figure 43 : Fenêtre d'initialisation

Les principales phases d'initialisation de la plateforme sont l'instanciation du serveur Rmi et du Tchat, le partage de ce dernier au sein du serveur Rmi, le choix de l'univers de travail, la création des

communautés et de leurs agents, leurs associations et enfin le démarrage du micro-outil de gestion de la plateforme.

3.2.2 Micro-outil de gestion des univers

Durant la phase de choix de l'univers de travail, le micro-outil de gestion des univers est instancié proposant à l'utilisateur, à l'aide d'une première fenêtre de dialogue (Figure 44), soit de créer un

Figure 44 : Choix de l'univers

nouvel univers et d'en réaliser son paramétrage, soit de charger un univers existant en l'état ou bien d'en modifier son paramétrage.

Le paramétrage d'un univers proposé par ce micro-outil porte sur la configuration de son seuil (Figure 45), ainsi que pour chaque communauté qui le compose (Figure 46), son nom (zone *Nom*), le nombre d'agents la constituant (zone *Agents*) et leur nommage (bouton *Nommer les agents*), le type de calcul réalisé (liste déroulante *Type de comput*), la définition ou non de relations intracommunautaires (case à cocher *Relations Intracommunautaires*).

Figure 45 : Paramétrage du seuil

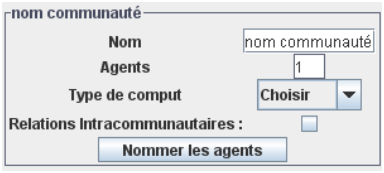


Figure 46 : Paramétrage de la communauté

Les zones *Nom* et *Type de comput* peuvent être cachées ou affichées en fonction des besoins de paramétrage propres à chaque communauté grâce aux méthodes respectives *setNomCommVisible* et *setTypeComputVisible* qui admettent en paramètre un booléen déterminant l'affichage ou le masquage de ces zones.

La fenêtre de paramétrage obtenue (Figure 47) reprend donc ces éléments en y ajoutant la possibilité de modifier le nombre de corps de métiers (zone *Nombre de métiers*) qui interviennent dans la configuration de produits de l'univers paramétré :

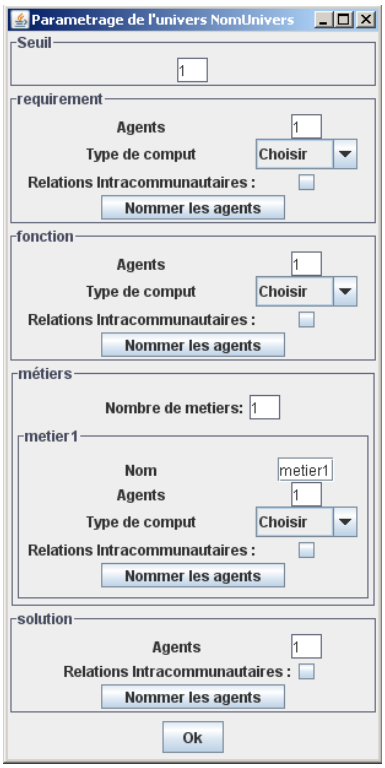


Figure 47 : Vision globale du micro-outil de paramétrage

Le nommage des agents s'effectue à l'aide d'une sous-fenêtre spécifique (Figure 48) qui permet de définir la description de chaque agent associée à son nom qui sera utilisée ensuite dans les différents micro-outils client et métiers

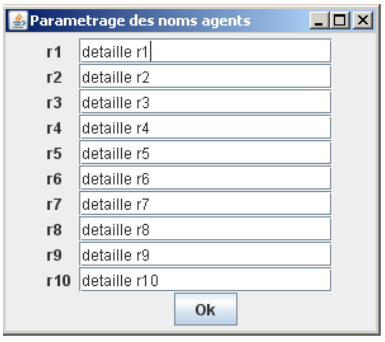


Figure 48 : Paramétrage des descriptions agents

De manière générale le comportement du micro-outil de gestion des univers peut être représenté par le diagramme ci-dessous (Figure 49) :

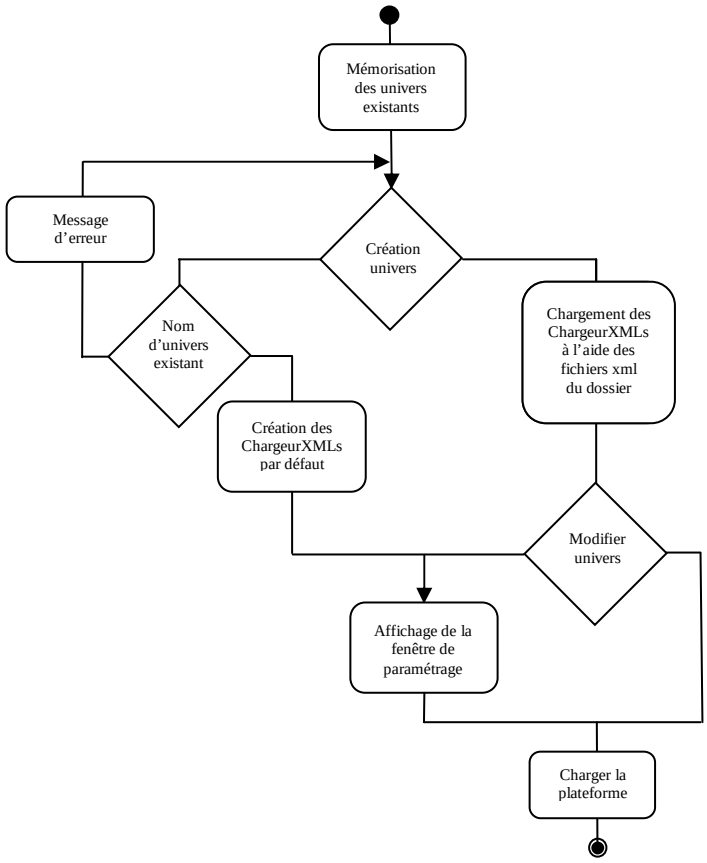


Figure 49 : Algorithme du micro-outil de gestion des univers

Lors de son instanciation, le micro-outil de gestion des univers mémorise les univers existants en effectuant la liste des sous-répertoires du dossier data. Cette liste est alors placée dans la liste déroulante de chargement d'univers (Figure 44). Chaque univers possède ainsi son dossier dans lequel sont stockés les fichiers de configuration xml de l'univers (configuration.xml pour le fichier général de configuration de l'univers ainsi que communautéX.xml pour chaque communauté constituant l'univers).

Dans le cas d'une création d'un nouvel univers et de manière à éviter l'écrasement d'un univers existant par la création d'un nouvel univers portant le même nom, cette création est soumise à la non-existence d'un univers portant déjà le même nom, faute de quoi la création est stoppée suivi d'un message d'erreur. Si la procédure est menée à terme, le micro-outil instancie les chargeurXmls

associés à l'univers et aux communautés par défaut (spécification, fonction, solution et un corps de métiers) contenant chacune un seul agent. Il les transmet alors à la fenêtre de paramétrage de l'univers (Figure 47).Il revient ensuite à l'utilisateur de modifier ce paramétrage par défaut en fonction de ses propres besoins. Une fois les modifications validées (Figure 47 bouton *Ok*), les *chargeurXmls* modifiés sont alors transmis à la plateforme.

Dans le cas du chargement d'un univers existant, le micro-outil instancie, pour chacune des communautés existantes dans l'univers à charger ainsi que pour le paramétrage général de l'univers à charger, un *chargeurXml* à l'aide du fichier de configuration Xml associé. Si l'univers est chargé sans modification de son paramétrage (Figure 44 bouton *Ok*), chaque *chargeurXml* précédemment instancié est alors transmis en l'état à la plateforme. En revanche, si la modification du paramétrage d'un univers existant est sélectionnée (Figure 44 bouton *Modifier*), les *chargeurXmls* précédemment instanciés sont alors transmis en premier lieu à la fenêtre de paramétrage de l'univers (Figure 47) et modifiés par cette dernière. Une fois les modifications validées (Figure 47 bouton *Ok*), les *chargeurXmls* modifiés sont alors transmis en second lieu à la plateforme.

La plateforme charge alors l'univers selon le protocole détaillé en 2. Le noyau (p. 27) à l'aide des *chargeurXmls* reçus du micro-outil de gestion des univers.

3.2.3 Présentation générale

Les autres micro-outils associés à la gestion et au paramétrage de la plateforme ont été regroupés au sein de trois onglets (Supervision, Relations communautés et Journal) illustrés par la capture ci-dessous (Figure 50) :

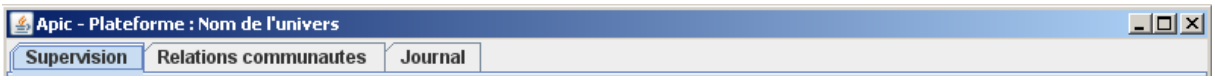


Figure 50 : Les onglets généraux

3.2.4 Onglet supervision

De manière à réaliser la fonction de supervision de la plateforme et de ses communautés, l'onglet *Supervision* est constitué de deux zones distinctes. La première (Figure 51) supervise l'état général de la plateforme et situe l'avancement de la configuration de produit (En attente de requête client, En

attente de contraintes métiers, Recherche de solution optimale, Calcul réalisé) grâce à l'algorithme présenté en 2.2.3 Détection des phases de configuration (p. 29)

Figure 51 : Supervision globale de la plateforme

La seconde (Figure 52) supervise chaque communauté constituant l'univers de travail de manière autonome selon les orientations définies en 2.3.4 Supervision des agents (p. 34) et porte sur le statut de la communauté (*Statut*), le nombre de messages envoyés, reçus et traités par cette dernière (*Messages*) et enfin la connexion du micro-outil client/métiers (*µ-outil*)

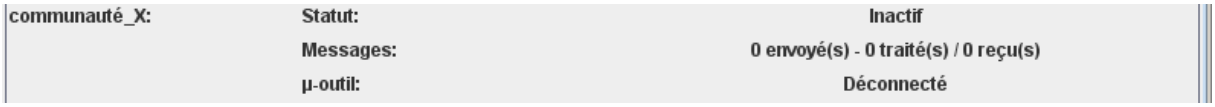


Figure 52 : Supervision par communauté

Pour assurer cette supervision autonome de chaque communauté, chaque zone de supervision utilise son propre *interfaceAccesMicroOutils* (Figure 53), interface d'accès RMI à l'*accèsMicroOutils* propre de chaque communauté :

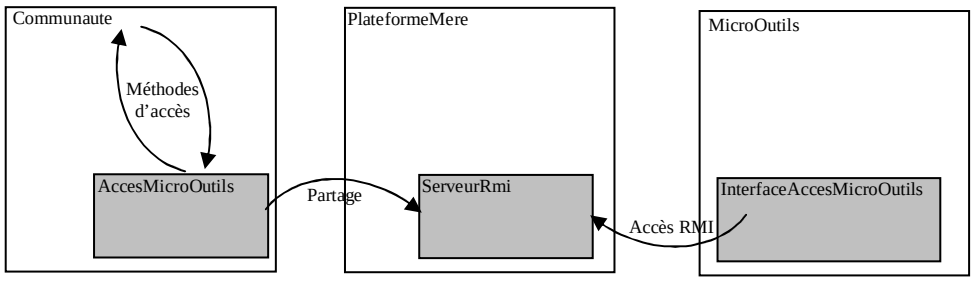


Figure 53 : Interactions avec la plateforme

Cependant de manière à permettre une communication bidirectionnelle entre le micro-outil et la communauté concernée, accès aux méthodes d'une part et transmission d'évènements d'autre part, et ainsi pallier le problème de transmission de référence Swing à travers RMI, une classe *IhmPlateformeListener* et son interface *InterfaceIhmPlateformeListener* a été implémentée et mise en place de manière inverse au niveau des *AccèsMicroOutils*. Ce processus est décrit par le schéma ci-dessous (Figure 54) :

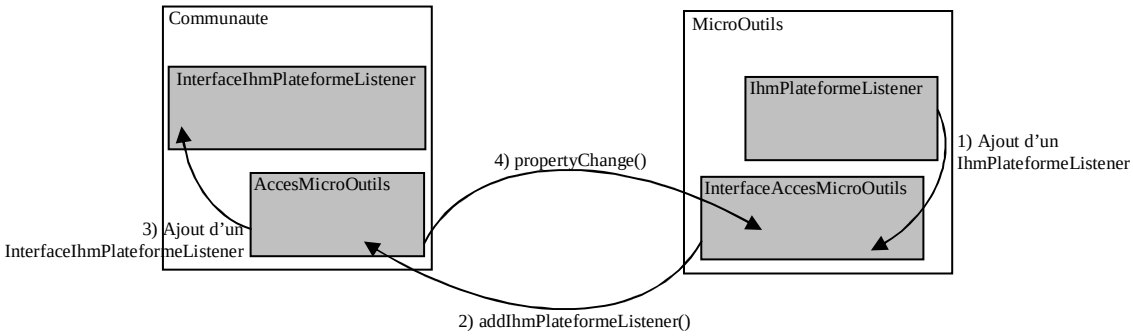


Figure 54 : Interactions avec le micro-outil

Comme l'illustre ce schéma (Figure 54), le micro-outil, après s'être connecté à son *accesMicroOutils* via RMI par le bais de son *interfaceAccesMicroOutils*, instancie un objet *IhmPlateformeListener* (1) puis l'enregistre auprès de ce dernier grâce à la méthode *addIhmPlateformeListener()* (2). Celui-ci stocke alors sa référence dans un vecteur d'*InterfaceIhmPlateformeListener* (3). La communauté est alors en mesure de transmettre des événements (changement de statut, du nombre de messages envoyés, traités et reçus) aux micro-outils enregistrés dans ce vecteur grâce à la méthode *propertyChange() (*4). A la réception d'un événement, les micro-outils traitent ensuite de manière autonome ce dernier.

L'interface globale obtenue pour cet onglet Supervision est représentée par la capture suivante (Figure 55) réalisée lors de l'utilisation du jeu de test de la chaise présenté précédemment :

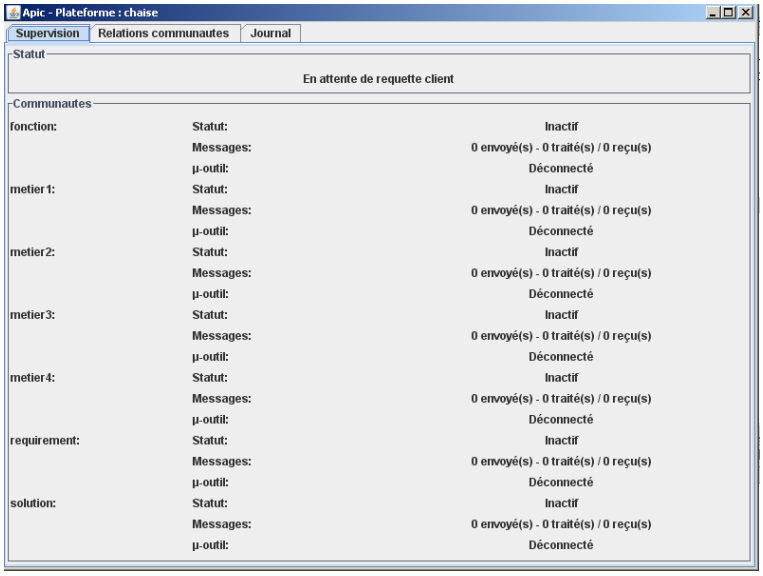


Figure 55 : Vision globale de l'onglet de supervision

3.2.5 Onglet paramétrage

En vue de permettre la modification des valeurs d'associations intra et intercommunautaires définies à l'aide du micro-outil de gestion des univers, l'onglet *Relations communautes* est subdivisé en autant de sous-onglets que de communautés définies dans l'univers de travail. Cette subdivision est illustrée par la capture suivante (Figure 56)

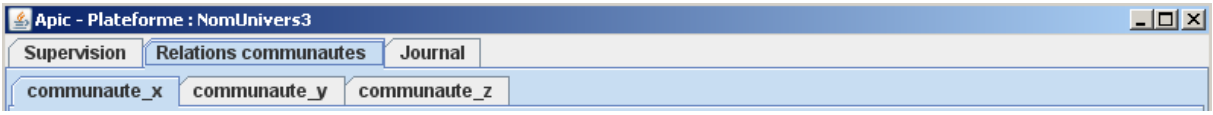


Figure 56 : Onglets communautaires

Chaque sous-onglet est divisé en trois zones, les deux premières consacrées au paramétrage des valeurs d'associations et la dernière à la sauvegarde des paramètres saisis.

La première des zones de paramétrage est destinée à la saisie des valeurs d'associations intercommunautaires de la communauté avec les communautés qu'elle influe, la seconde est pour sa part destinée à la saisie des valeurs d'associations intercommunautaires de la communauté.

Afin de permettre la saisie des matrices de paramétrage, un tableau de saisie à double en-tête (Figure 57) a été implémenté admettant comme en-tête de ligne le nom des agents constituants la communauté paramétrée et comme en-tête de colonne le nom des agents des communautés influées.

	z1	z2	z3	z4	z5	z6	z7	z8	z9	z10	z11	z12	z13	z14	z15
y1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
y2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
y3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
y4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
y5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
y6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
y7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
y8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
y9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
y10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 57 : Définition des valeurs

De manière à offrir une saisie plus aisée des matrices de grandes tailles dans ce tableau, des barres de défilement horizontales et verticales sont activées dès que la matrice dépasse respectivement 15 colonnes et 10 lignes. Un déplacement au clavier est aussi possible à l'intérieur du tableau.

De plus la description détaillée du nom de l'agent est disponible en aide contextuelle (Figure 58) lorsque le curseur de la souris est statique sur l'en-tête d'une ligne ou d'une colonne.

Figure 58 : Aide à la saisie

Enfin, afin d'assurer une cohérence des données saisies, une vérification de la valeur saisie est effectuée à chaque modification. Dans le cas où la valeur saisie n'appartient pas à l'intervalle [0 ;1], un message d'erreur est communiqué à l'utilisateur puis l'ancienne valeur est immédiatement réappliquée . De plus, lorsque une association symétrique est présente dans le tableau (associations intracommunautaires de type x1-x2 et x2-x1), toute modification de l'une des valeurs provoque la modification identique de sa valeur miroir. Enfin dans le cas d'une relation réfléchie (associations intracommunautaires de type x1-x1) seule une valeur égale à 1 est acceptée.

L'interface globale obtenue pour cet onglet Supervision est représentée par la capture suivante (Figure 59) réalisée lors de l'utilisation du jeu de test de la chaise présenté précédemment :

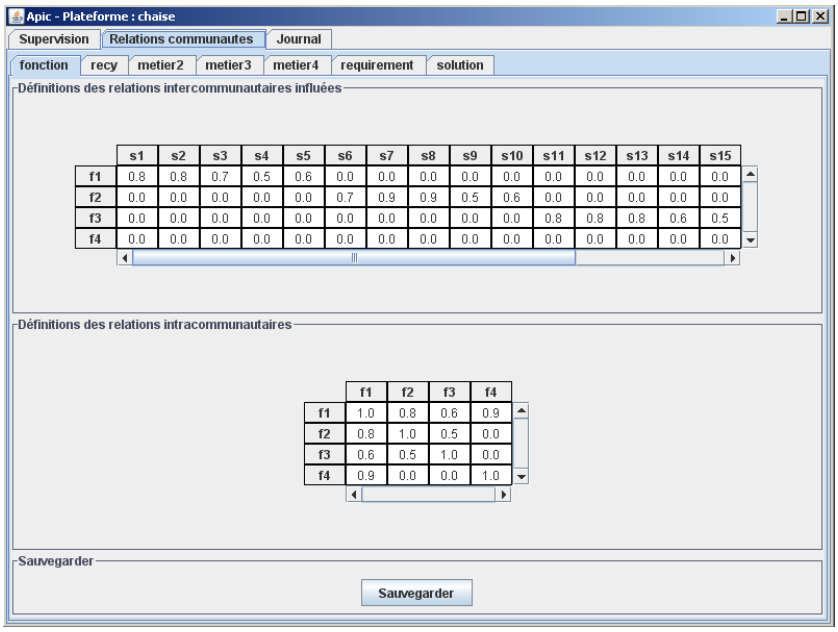


Figure 59 : Vision globale de l'onglet de paramétrage

3.2.6 Onglet journal

De manière à suivre les différents échanges (spécifications, contraintes, solutions consensuelles et optimales) entre les différents acteurs de la configuration de produits, un micro-outil journal, au sein duquel ces échanges sont écrits, a été implémenté et est illustré par la capture ci-dessous (Figure 60) issue de l'utilisation du jeu de test de la chaise :

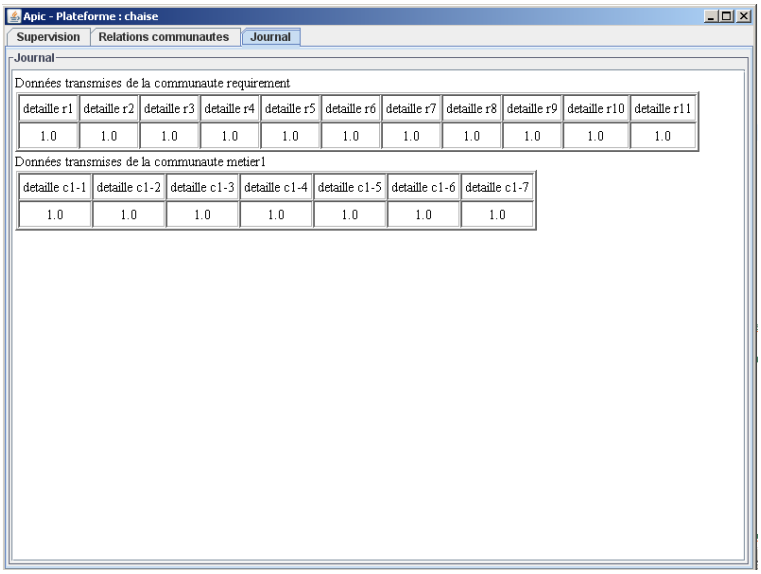


Figure 60 : Vision globale de l'onglet journal

Comme le montre cette capture (Figure 60), les différents échanges sont mis en forme sous forme de tableaux horizontaux en utilisant la description détaillée de chaque agent présent dans l'échange ainsi que sa valeur propre. Cette mise en forme est obtenue en formatant au format html les valeurs transmises par les différentes communautés concernées.

3.3 Les micro-outils client et métiers

Ces micro-outils sont destinés aux différents acteurs intervenant dans la réalisation de la configuration de produit. Etant très similaires, ils sont présentés ici d'une manière commune.

3.3.1 Connexion au serveur distant

Le micro-outil de connexion au serveur distant est initialisé au démarrage des applications clientes (client et métiers). Ce micro-outil permet dans un premier temps le choix du serveur de connexion par le biais d'une fenêtre (Figure 61) où l'adresse du serveur désiré doit être saisie.

Figure 61 : Connexion à la plateforme

Une fois l'adresse serveur validée et la connexion réalisée, une seconde fenêtre (Figure 62) propose le choix de la communauté de connexion si plusieurs communautés sont disponibles pour le micro-outil client ou métiers initialisé.

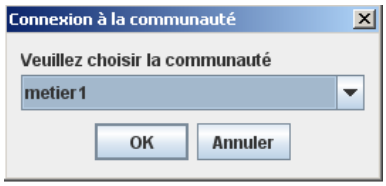


Figure 62 : Choix de la communauté de connexion

3.3.2 Présentation générale

Les autres micro-outils associés aux interfaces client et métiers ont été regroupés au sein de deux onglets (Communauté et Journal) illustrés par la capture ci-dessous

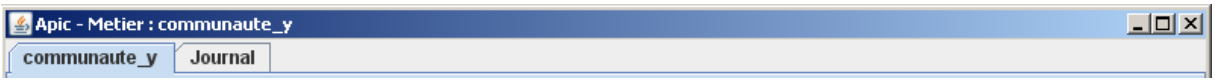


Figure 63 : Onglets des micro-outils client et métiers

3.3.3 Onglet de saisie

L'onglet de saisie est subdivisé de manière verticale en deux principales parties, la première consacrée à la saisie et l'affichage d'informations, la seconde orientée sur les échanges entre les différents acteurs.

3.3.3.1 Saisie et affichage des informations

La saisie des informations est réalisée à l'aide du micro-outil de saisie des valeurs des agents illustrée par la capture ci-après (Figure 64). Ce dernier est composé de deux éléments dont le premier permet le choix de l'élément à définir (*Définition*) et le second permet la définition propre de la valeur de l'élément sélectionné (*Valeur*).

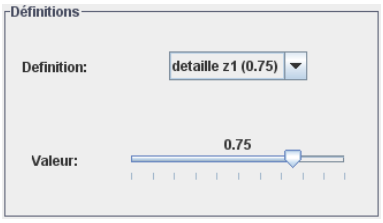


Figure 64 : Paramétrage des contraintes et spécifications

Comme l'illustre cette capture (Figure 64), la zone de choix (*Définition*) a été implémentée sous la forme de liste déroulante permettant un choix plus aisé de l'élément à définir lorsque les communautés comportent un grand nombre d'agents. De plus la description détaillée ainsi que le rappel de la valeur actuellement définie de l'agent permettent aussi une identification plus aisée de l'agent à sélectionner. La zone de définition de la valeur (*Valeur*) a été implémentée à l'aide d'une barre coulissante surmontée d'un rappel numérique de la valeur définie avec cette dernière. Cette barre permet de représenter l'aspect « flou » des agents de la plateforme.

L'affichage des différentes informations transmises par la plateforme telles que la requête client, les résultats consensuels obtenus ou bien les solutions optimales est réalisé à l'aide du micro-outil d'affichage des informations transmises présenté par la capture suivante (Figure 65) :

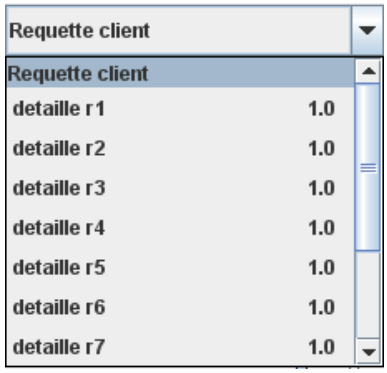


Figure 65 : Affichage des spécifications client

Comme l'illustre cette capture (Figure 65), les informations sont représentées à l'aide d'une liste déroulante qui comporte en partie gauche le nom ou la description détaillée de l'information puis en partie gauche la valeur de l'information. Le choix d'affichage du nom ou des informations détaillées de l'information listée est réalisé en fonction de la quantité d'information à afficher.

3.3.3.2 Echanges entre les différents acteurs

De manière à répondre aux spécifications du besoin de communication entre les différents acteurs de la configuration de produit, un espace d'envoi de messages instantanés de type Tchat a été implémenté. Cet aspect de communication étant complètement indépendant du processus de configuration de produits, le micro-outil de tchat dispose d'un accès spécifique à la plateforme illustré à l'aide du schéma (Figure 66) suivant :

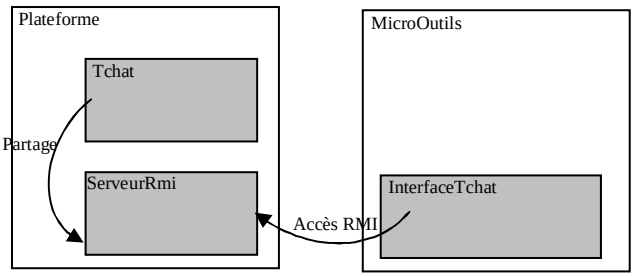


Figure 66 : Interactions avec la plateforme

Ainsi un nouvel objet rmi *Tchat* est partagé sur le *serveurRmi* de la plateforme puis y joue le rôle de serveur de tchat. Le micro-outil de tchat des différentes applications client et métiers y accède à l'aide de l'interface *InterfaceTchat* et y joue le rôle de client de tchat.

Comme l'illustre le schéma ci-après (Figure 67), pour palier le même problème de transfert de référence d'objet Swing à travers RMI, le micro-outil de tchat instancie un objet *IhmPlateformeListener* (1) qu'il enregistre auprès de son serveur de tchat (2) de manière à lui permettre de lui transmettre des événements (3).

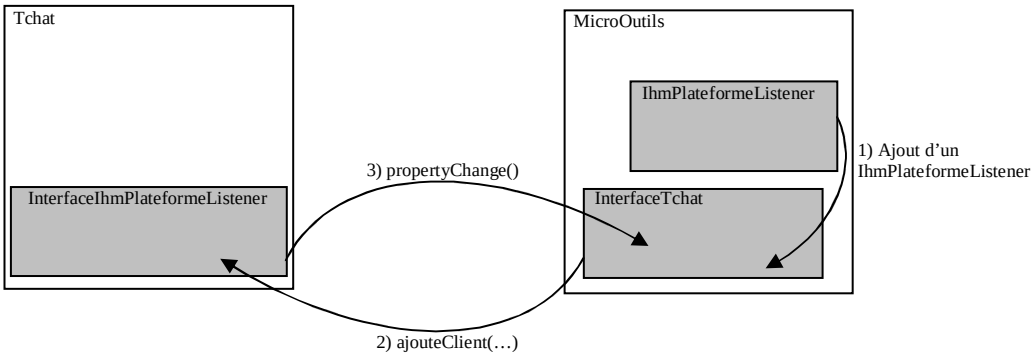


Figure 67 : Interactions avec le Tchat

Du point de vue graphique le micro-outil de tchat se décompose en trois éléments distincts correspondants aux besoins élémentaires de choix des destinataires, de saisie du message et de consultation des messages échangés.

Ainsi le premier élément, illustré par la capture ci-dessous (Figure 68), répertorie les clients connectés au serveur. Son actualisation se réalise à chaque connexion ou déconnection d'un client qui provoque l'émission côté serveur d'un événement transmis à ses clients à l'aide de la méthode *informeIhmPlateformeListeners()*. Suite à la réception de cet événement, le micro-outil client provoque l'appel de la fonction *getListeClients()* sur son *interfaceTchat* et réalise la mise à jour de la liste des clients connectés. Enfin la sélection d'un ou plusieurs noms de la liste provoque leur ajout dans la liste des destinataires du message qui sera envoyé.

Figure 68 : Choix des destinataires

Le second élément, comme le montre la capture ci-dessous (Figure 69), permet la saisie d'un message et son envoi. L'envoi du message saisi est réalisé à l'ensemble des destinataires sélectionnés dans le précédent élément.

Figure 69 : Saisie d'un message

L'envoi d'un message est réalisé à l'aide de l'appel de la méthode *envoyerMessage()* de l'*interfaceTchat* du micro-outil tchat. Cette méthode admet en paramètres l'expéditeur (String *expediteur*), le message (String *message*) ainsi que la liste des destinataires (Vector *destinataires*)

Le dernier élément, représenté par la capture ci-dessous (Figure 70), permet l'affichage chronologique des messages reçus des autres clients ainsi que ceux envoyés par le client.

Figure 70 : Historique des messages

La réception d'un message par le micro-outil de tchat est réalisée par la réception d'un événement *message* émis par l'objet *Tchat* hébergé sur le *serverRmi* de la plateforme. Ce dernier produit cet événement lors de la réception d'un message envoyé par un client grâce l'appel de sa méthode *envoyerMessage()* et le transmet à l'ensemble des destinataires du messages (Vector *destinataires*).

L'interface globale obtenue pour cet onglet Supervision est représentée par la capture suivante (Figure 71):

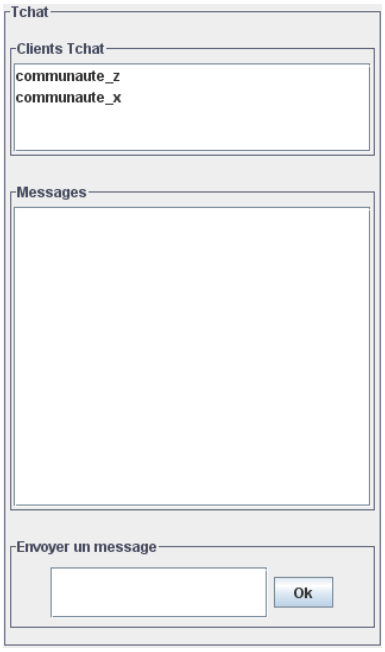


Figure 71 : Vue globale du micro-outil de Tchat

3.3.3.3 Présentation des interfaces client et métiers

Les interfaces globales des micro-outils client et métiers obtenues sont représentées par les captures suivantes (Figure 72) réalisée lors de l'utilisation du jeu de test de la chaise présenté précédemment :

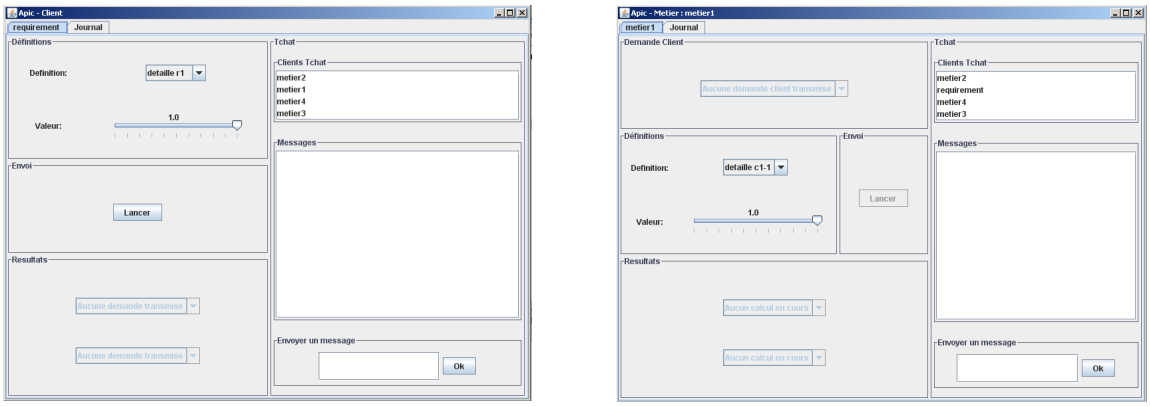


Figure 72 : Captures des interfaces client et métiers

3.3.4 Onglet journal

Tout comme le micro-outil de gestion de la plateforme, le micro-outil journal, présenté en 3.2.6 Onglet journal (p. 57) est proposé au sein des micros-outils client et métiers de manière à suivre les différents échanges (spécifications, contraintes, solutions consensuelles et optimales) entre les différents acteurs de la configuration de produits.

OPTIMISATION : RECHERCHE DE SOLUTION(S) OPTIMALE(S)

Ce dernier chapitre porte sur la dernière étape de réalisation du projet qui consiste à proposer et mettre œuvre une solution d'optimisation afin de permettre la détermination de solutions optimales au problème de configuration d'un produit.

1. Emergence d'une solution globale des solutions locales

La nouvelle plateforme étant désormais agentifiée, la recherche de solutions de configurations optimales doit elle aussi être orientée agent. Ainsi, chaque agent solution, de part ses connaissances, sa valeur et ses échanges avec les autres agents de sa communauté, peut déterminer la meilleure solution le concernant, cette solution étant locale à l'agent. Cependant la plateforme doit fournir la ou les solutions optimales répondant aux critères saisis lors de l'initialisation de la configuration de produit, cette ou ces solutions étant globales. Dès lors, il convient de mettre en place, grâce aux échanges entre les agents portant sur leurs solutions locales, un mécanisme permettant de faire émerger une solution globale à l'ensemble des agents de la communauté solution.

2. Evaluation de la solution

De manière à déterminer la ou les solutions optimales, il s'avère nécessaire d'évaluer la manière dont la solution concernée répond aux critères de la fonction objectif définie. Cette valeur, dénommée ultérieurement *valeur de solution* ou *valeur de fonction objectif*, pour être cohérente doit tenir compte des valeurs propres des agents constituant cette solution mais aussi de l'ensemble des interactions entre ces différents composants, qu'elles soient directes ou indirectes. Cet ensemble de valeurs à prendre en compte pour la détermination de la valeur de la fonction objectif d'une solution sont représentées par le schéma ci-dessous (Figure 73) :

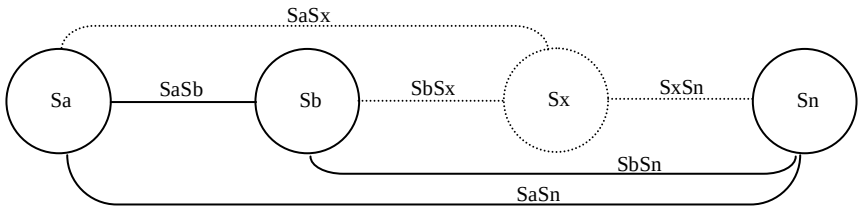


Figure 73 : Représentation d'une solution optimale

Ainsi comme le montre ce schéma (Figure 73), une solution formant un réseau composé de n composants devra être évaluée sur les valeurs propres de ses composants (*Sa,Sb,…,Sx,…,Sn*) ainsi que sur les relations directes, de proche en proche, de ses composants (*SaSb,SbSx,…,SxSn*) mais aussi sur les relations indirectes entre composants non contigus de la solution (*SaSx,…,SaSn,…,SbSn,…*).

Les valeurs propres des composants constituant la solution évaluée correspondent aux valeurs consensuelles issues de la première phase de la configuration de produit, les relations directes et indirectes des composants extraites de la matrice SxS de relations intracommunautaires de la communauté solution.

La détermination de la valeur de la fonction objectif de la solution évaluée peut-être abordée selon différentes approches :

- Par le produit, cette approche discriminante possède pour principal défaut qu'appliquée au monde floue, l'évaluation a très vite tendance à tendre vers zéro et ceci même pour des valeurs initialement haute. En effet même pour un petit réseau de quatre composants ne représentant que dix valeurs à prendre compte pour l'évaluation, toutes par exemple à 0.8, cette dernière retournerait une valeur égale à $0.8^{10} \approx 0.11$, valeur très faible.
- Par la somme, cette approche très peu discriminante ne permet pas de différencier une solution moyenne dans son ensemble d'une solution moyenne dans les extrêmes. En effet, dans le cas du réseau précédemment présenté de quatre composants, il est impossible de discerner une solution qui posséderait l'ensemble des valeurs à 0.5 d'une solution qui comprendrait cinq valeurs à 0.8 et cinq autres à 0.3, toutes deux ayant une évaluation égale à 5.

- Par la moyenne, cette approche pondérante a pour principal attrait de ne pas affaiblir fortement une solution en cas de mauvaises valeurs éparses ou au contraire un renforcement de cette dernière en cas de très bonnes valeurs occasionnelles. En effet, dans le cas du réseau précédemment présenté, une seule valeur à 0,4 par exemple, bien qu'influant sur l'évaluation, retournera une valeur de 0.76. Si à l'inverse, l'ensemble des valeurs était égal à 0.4 à l'exception d'une valeur à 0.8, l'évaluation retournerait 0.44.
- Par le maximum, cette approche très tolérante permet de faire ressortir une solution dès l'instant où une valeur entant dans l'évaluation est bonne. En effet une évaluation d'un réseau ne contenant par exemple qu'une seule valeur à 0.8 et l'ensemble des autres valeurs à 0.4 retournerait 0.8.
- Par le minimum, cette approche discriminante possède pour principal intérêt de tenir extrêmement compte des mauvaises valeurs présentes dans un réseau de solution optimale tout en favorisant les meilleures valeurs. En effet, dans le cas où un réseau ne serait composé que de valeurs à 0,8 par exemple, obtiendrait une évaluation très favorable à 0,8. En revanche si le même réseau venait à posséder une valeur plus faible à 0,4 par exemple, l'évaluation serait alors nettement moins favorable avec une valeur à 0,4.

Ces différentes approches peuvent naturellement être défendues en fonction du type et de la qualité de solution souhaitée. C'est l'évaluation par le minimum qui a été retenue pour l'approche algorithmique proposée, présentée ultérieurement, et implémentée par défaut sur la plateforme. De manière à pouvoir comparer les différentes approches algorithmiques, ce dernier type d'évaluation sera considéré à chaque étude algorithmique.

2.1 Approche des meilleurs voisins

Initialement implémentée dans la version c disponible au démarrage de ce projet, l'approche de recherche de solution optimale par les meilleurs voisins est illustrée par le schéma ci-dessous (Figure 74) :

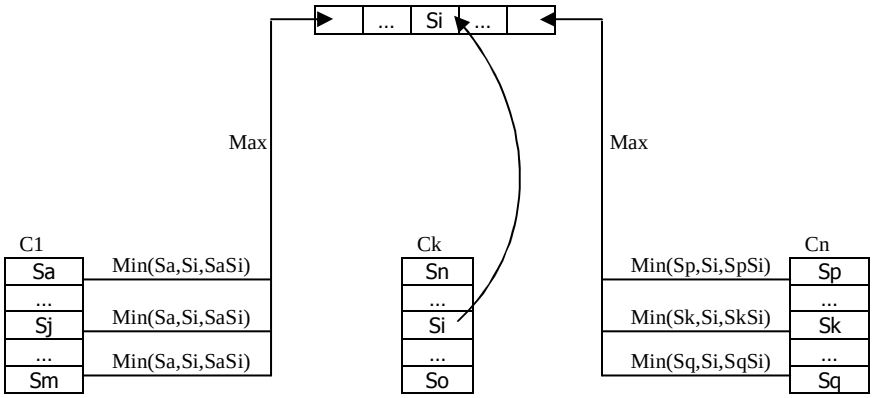


Figure 74 : Approche du meilleur voisin

Comme l'illustre ce schéma (Figure 74), chaque agent (*Si*) recherche le meilleur agent de chaque classe de composant (*C1,…,Cn*) en effectuant le minimum entre sa valeur (*Si*), celle de l'agent évalué (*Sj*) et leur valeur d'association issue de la matrice SxS (*SaSi*). La référence à l'agent de la classe étudiée qui obtient la meilleure évaluation est alors ajoutée au vecteur solution et le résultat de cette dernière sommée à la valeur de fonction objectif de l'agent.

La formalisation de l'algorithme est donnée par le schéma ci-dessous (Figure 75) :

```
Pour chaque Si

Si ∈ Ck                                   Si composant de la classe Ck
Sl ∈ Cm                                   Sl composant de la classe Cm

Vsi                                       Valeur de l'agent Si
VAsi                                      Valeur d'association entre Si et Sl
VOvi                                      Valeur de la fonction objectif pour Si
RESEAUri                                  Réseau optimal de Si

FOR EACH Cm                               Parcours des classes de composants
        RES=0
        If  Ck=Cm                         La classe courante est celle de Si
                VOvi=VOvi+Vsi
        ELSE                              La classe courante n'est pas celle de Si
                MAX=0
                FOR EACH Sl               Recherche  de la meilleure entente
                        EVAL=Min (Vsi,Vsl, VAsisl)
                        IF EVAL>MAX
                                RES=Sl
                                MAX=EVAL
                        ENDIF
                NEXT
                VOvi=VOvi+MAX
                RESEAUri=RESEAUri ∪ RES
        ENDIF
NEXT
```

Figure 75 : Algorithme formalisé du meilleur voisin

Cette approche initiale ne répond pas aux exigences définies précédemment de construction de solution optimales (2. Evaluation de la solution, p.64) et ne propose qu'un ensemble de solutions locales qui ne font pas émerger une solution globale.

2.2 Approche du meilleur réseau voisin

Elaborée par Alain-Jérôme Fougères en parallèle de l'approche proposée et présentée dans le point suivant, l'approche du meilleur réseau voisin s'inspire de l'algorithme de routage de « vecteur distance », utilisant l'algorithme de Bellman-Ford [Cormen et al, 2001], pour le calcul du meilleur chemin de routage au sein d'un réseau informatique.

2.2.1 Initialisation

A l'initialisation de l'algorithme, chaque agent initialise son réseau optimal (*RESEAUri*) en le complétant par la référence aux agents ayant la meilleure évaluation partielle pour chacune de ces classes voisines, puis évalue le réseau obtenu et stocke le résultat de cette évaluation (*VOvi*). Enfin chaque agent communique son réseau optimal ainsi que le résultat de son évaluation aux agents composant ses classes voisines. Il réitèrera ensuite cette opération à chaque fois qu'il recevra un réseau optimal meilleur que le sien.

2.2.2 Déroulement

Une fois l'initialisation achevée et l'émission des premiers messages effectuée, le déroulement de l'algorithme est représenté par le schéma ci-dessous (Figure 76) :

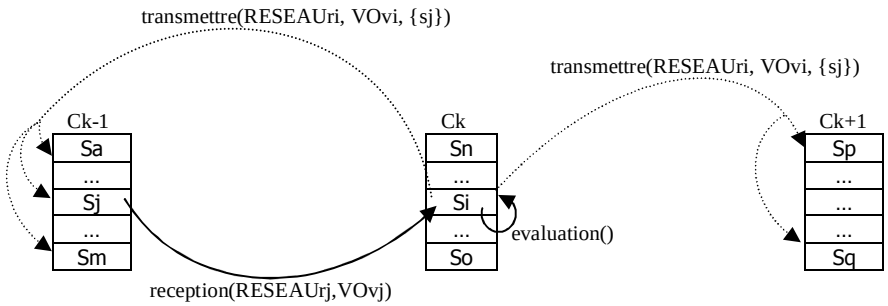


Figure 76 : Approche du meilleur réseau voisin

Comme l'illustre ce schéma (Figure 76), à la réception d'une solution provenant d'une classe voisine, l'agent *Si* évalue en premier lieu s'il appartient à la solution reçue. En second il évalue (*evaluation*) si la solution reçue de l'agent *Sj* (*RESAUrj*) est meilleure que la solution locale optimale actuelle (*RESEAUri*) en comparant les valeurs respectives de fonctions objectif (*VOvj* et *VOvi*). Dans le cas où la solution reçue (*RESEAUrj*) est meilleure que celle locale(*RESEAUri*), il écrase sa solution locale ainsi que sa valeur avec la solution reçue et sa valeur de fonction objectif associée (*RESEAUri*=*RESEAUrj* et *VOvi*=*VOvj*) ; puis il communique ses nouvelles valeurs à l'ensemble des agents *Sj* des classes voisines.

2.2.3 Convergence

La convergence de cet algorithme est obtenue à l'arrêt de l'émission et de la réception de messages entre les agents de la communauté solution. Chaque agent est alors en possession de son réseau optimal (solution locale). En l'état actuel des choses, il reviendra à la plateforme de comparer ces solutions afin de déterminer une solution optimale globale.

2.2.4 Algorithme formalisé

```
Pour chaque Si

Si ∈ Ck                                          Classe de composants de Si
Si ∈ RESEAUri                                    Réseau optimal de Si
Sj ∈ Ck ∪ Ck-1 ∪ Ck+1                            Sj est un voisin de Si
VOvi = val(RESEAUri)                             Valeur de la fonction objectif pour Si

transmettre(RESEAUri, VOvi, {Sj})
WHILE (agent actif)
        IF reception(RESEAUrj,VOvj)
                IF Si ∈ RESEAUrj ∩ VOvj > VOvi
                        RESEAUri ← RESEAUrj      Nouveau réseau optimal
                        VOvi ← VOvj              Nouvelle valeur de fonction objectif
                        transmettre(RESEAUri, VOvi, {sj})
                ELSE                             Le réseau n'est pas optimal
                ENDIF
        ENDIF
END
```

Figure 77 : Algorithme formalisé du meilleur réseau voisin

2.3 Approche proposée

L'approche proposée dans ce point s'inspire de l'algorithme « p-median » largement utilisé dans le domaine de l'optimisation.

2.3.1 Objectifs de l'algorithme

L'algorithme présenté a pour principal objectif de fournir l'ensemble des solutions optimales répondant aux critères fournis lors du processus de configuration de produit. Ces solutions devront naturellement répondre aux exigences définies précédemment de construction de solutions optimales (2. Evaluation de la solution, p.64)

Le second objectif de cet algorithme est de ne pas alourdir les connaissances des agents en utilisant ces dernières telles qu'elles ont été précédemment définies, à savoir pour chaque agent avec lequel un agent est lié, ce dernier ne connait de celui-ci que son nom, sa valeur propre ainsi que sa valeur d'association avec lui.

Le troisième objectif est de limiter au maximum les échanges de messages liés à la phase de recherche de solutions optimales.

2.3.2 Initialisation

L'initialisation de la phase de recherche de solutions optimales est réalisée par la communauté solution sur le premier agent solution qui la compose. A cet agent, la communauté fournit une valeur de la solution optimale (*VSC*) égale à 1 ainsi que les vecteurs solution (*SC*) et exclusion (*EXC*) vides. Ce dernier transmet cette demande aux agents de sa classe.

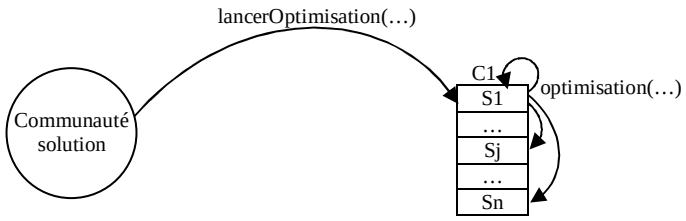


Figure 78 : Approche proposée

La détermination de la classe des agents, illustrée par le schéma ci-dessous (Figure 79), est réalisée en utilisant les relations intracommunautaires définies à -1.

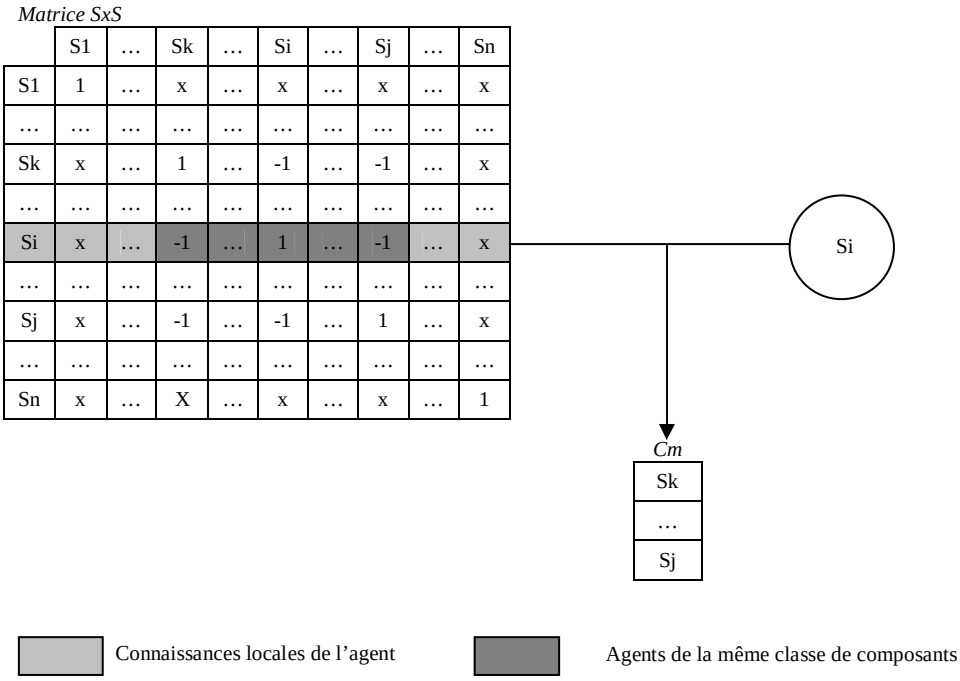


Figure 79 : Les connaissances d'un agent

A la première réception d'un message demandant la recherche de solutions optimales, un agent solution instancie alors son vecteur de solutions optimales (*SO*) et sa valeur de solution optimale, puis initialise cette dernière à -1.

2.3.3 1ère phase : Construction de solutions complètes

La première phase de l'algorithme consiste à construire les solutions optimales répondants aux critères présentés dans les objectifs de l'algorithme (2.3.1 Objectifs de l'algorithme, p.70). Cette construction, représentée par le schéma ci-dessous (Figure 80), est réalisée par une vague se transmettant en premier lieu classe par classe (*lanceOptimisation(…)*) et ensuite composants de classe par composants de classe (*optimisation(…)*):

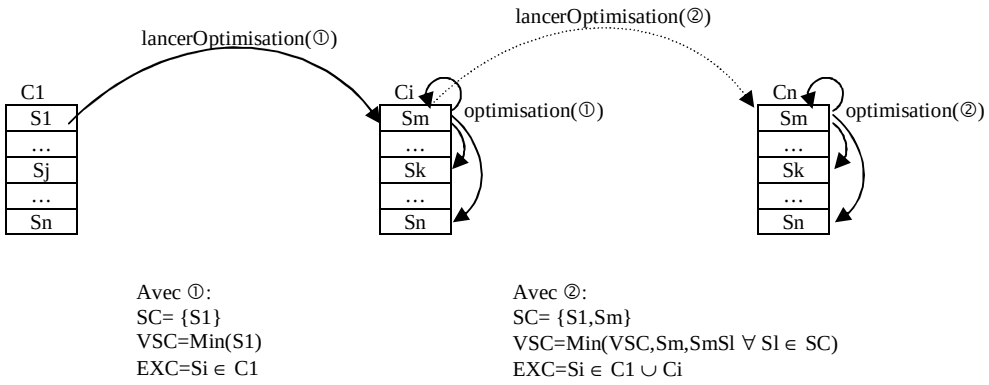


Figure 80 : Construction des solutions optimales

Comme l'illustre ce schéma (Figure 80), à chaque saut, un agent d'une classe de composants reçoit une demande de construction de solution pour sa classe. Ce dernier transmet cette demande aux agents de sa classe qui complètent alors la solution selon le schéma ci-dessous (Figure 81) :

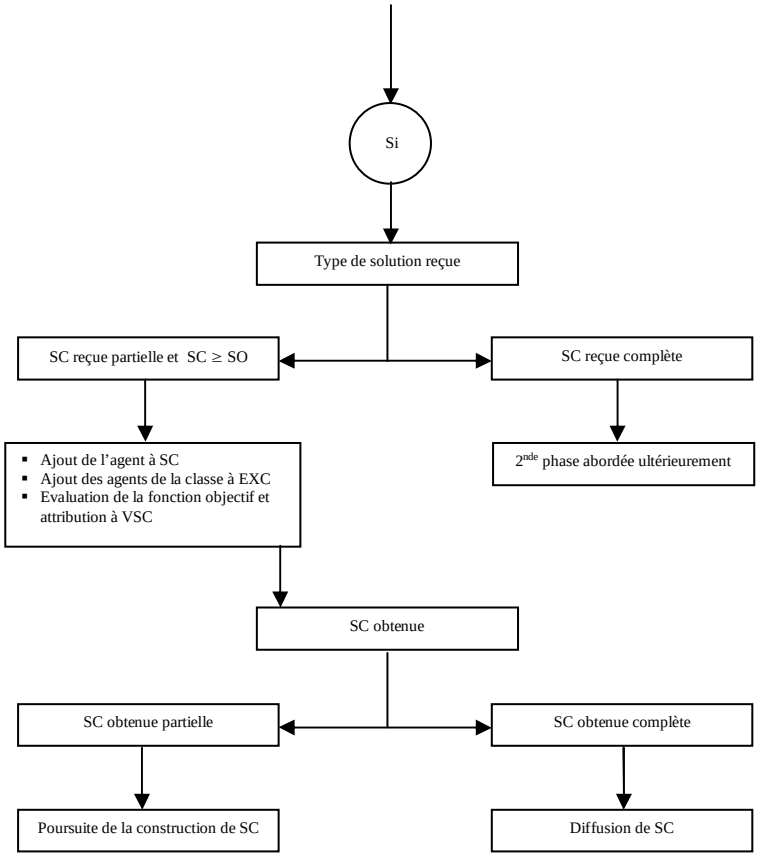


Figure 81 : Traitement des messages

Ainsi comme le montre ce schéma (Figure 81), lors de la réception d'une solution partielle, l'agent complète la solution avec son nom. De manière à limiter les propagations de constructions de solutions partielles alors qu'une solution complète connue est déjà meilleure, et ainsi la quantité de messages échangés comme annoncé dans les objectifs de l'algorithme (2.3.1 Objectifs de l'algorithme, p.70), l'agent n'effectue cette opération que si la valeur de fonction objectif reçue (*VSC*) est supérieure ou égale à la valeur de fonction objectif locale (*VSO*).

Puis pour éviter un ajout d'un autre composant de sa classe à la solution ou bien un bouclage de l'algorithme, il ajoute au vecteur d'exclusion (*EXC*) les agents de sa classe ainsi que lui-même.

Ensuite il évalue la fonction objectif en réalisant le minimum entre la valeur de fonction objectif reçue (*VSC*), sa valeur propre (*Sm*), ainsi que chaque valeur d'association qu'il possède avec les composants constituant la solution actuelle (*SmSi*). Cette approche permet de tenir compte de l'ensemble des relations de l'agent avec les autres composants comme décrit dans les objectifs de l'algorithme (2.3.1 Objectifs de l'algorithme, p.70).

L'évaluation de la fonction objectif réalisée, l'agent évalue ensuite si la solution obtenue est une solution partielle ou complète en comparant la taille du vecteur exclusion au nombre d'agents constituant la communauté solution. Si la solution obtenue est partielle, il transmet alors une demande de poursuite de construction de la solution au premier agent qui ne fait pas encore partie du vecteur exclusion (*EXC*). Dans le cas où la solution obtenue est complète et meilleure que la solution locale déjà connue, l'agent diffuse alors cette solution aux agents composant la solution, comme l'illustre le schéma suivant (Figure 82) :

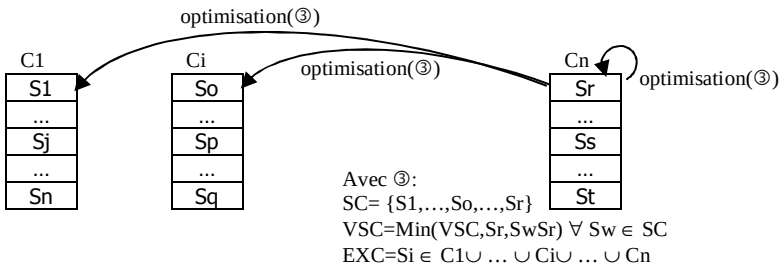


Figure 82 : Transmission des solutions complètes

Comme le montre ce schéma (Figure 82), la diffusion de la solution obtenue par le dernier agent constituant la solution aux autres agents la constituant permet une diffusion horizontale de cette dernière. Cette diffusion est volontairement réduite de manière à limiter la diffusion des messages entre agents comme annoncé dans les objectifs de l'algorithme (2.3.1 Objectifs de l'algorithme, p.70). En effet la diffusion de la solution obtenue à l'ensemble des agents de la communauté solution, ne sera nécessaire que si cette dernière est meilleure que les solutions optimales trouvées précédemment ou au moins égale. Il conviendra alors à chaque agent composant la solution de déterminer si cette diffusion est nécessaire ou non. Cette diffusion verticale est illustrée dans le point suivant.

2.3.4 2nde phase : Mémorisation des meilleures solutions

La seconde phase de l'algorithme porte sur la mémorisation et la diffusion des meilleures solutions. Comme annoncé dans le point précédent et illustré par le schéma ci-dessous (Figure 83), chaque agent constituant de la solution trouvée a en charge la diffusion de celle-ci au niveau de sa classe.

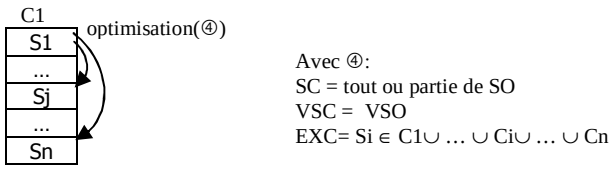


Figure 83 : Mémorisation des solutions complètes

Cette diffusion n'est pas systématique et dépend de la qualité de la solution trouvée, comme illustré par le schéma suivant (Figure 84) :

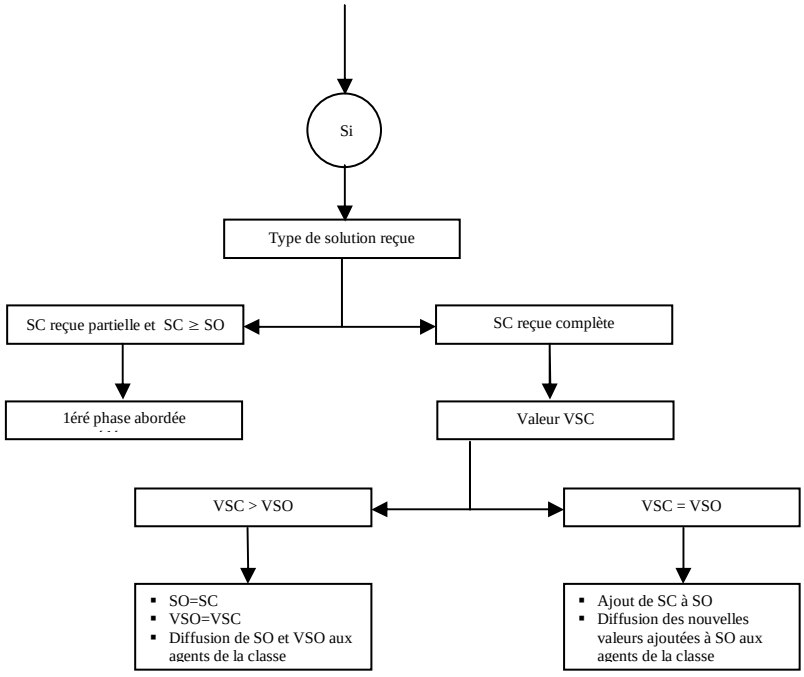


Figure 84 : Traitement des messages

Comme l'illustre ce schéma (Figure 84), à la réception d'une solution complète, l'agent détermine en fonction de la valeur de fonction objectif reçue (*VSC*) et de sa valeur locale de fonction objectif (*VSO*) s'il y a lieu ou non de diffuser la solution reçue.

Ainsi si la solution reçue est meilleure que la solution locale (*VSC* > *VSO*), la solution locale tout comme la valeur locale de fonction objectif sont alors remplacées par la solution reçue et sa valeur (*VSO*= *VSC* et *SO*=*SC*).

Sinon, si la solution reçue est de qualité égale à la solution locale (*VSC* = *VSO*), la solution reçue est alors ajoutée aux solutions locales ($SO = SO \cup SC$) puis la solution ajoutée est diffusée aux agents de la classe.

Enfin, dans le cas où la solution reçue est moins bonne que la solution locale (*VSC* < *VSO*), aucune diffusion de solution n'est effectuée.

2.3.5 3ème phase : Emergence des solutions optimales

La troisième phase de l'algorithme fait émerger de l'ensemble des solutions locales une solution globale. Cette émergence est réalisée de part l'échange des solutions optimales, selon l'algorithme présenté dans le paragraphe précédent. Cette émergence est progressive, les agents s'échangeant par classe leurs meilleures solutions, tendant progressivement à une solution locale identique pour chaque agent. La plateforme détecte l'émergence de la solution optimale par l'arrêt des échanges de messages par les agents, du à la convergence de l'algorithme, comme présenté dans la détection des différentes phases de la configuration de produit (2.2.3 Détection des phases de configuration, p.29)

2.3.6 Algorithme formalisé

```
Si                                          Agent solution exécutant l'algorithme
SO                                          Solution(s) optimale(s) locale(s)
VSO                                         Valeur de la fonction objectif du réseau SO
SC                                          Solution(s) communiquée(s)
VSC                                         Valeur de la fonction objectif du réseau SC
EXC                                         Agents exclus de la poursuite de recherche optimale
TEXC                                        Taille de EXC
NAS                                         Nombre d'agents de la communauté solution

Sj ∈ Cm tq ∀ Sj, SiSj=0                     Classe de composants de Si
Sk  1er agent tq ∉ Cm ∪ EXC                 Elément d'une autre classe
Sl ∈ SO                                     Agent appartenant à la solution

IF TEXC < NAS                               Construction de solutions optimales complètes
        EXC = EXC ∪ Si ∪ Sj                 Ajout des agents de la classe et de l'agent à EXC
        IF  EXC ∩ SC = ∅                    Aucun élément de la classe dans la solution reçue
                SC = SC ∪ Si                Ajout de l'agent à la solution
                VSC = Min(VSC,Si,SiSl)      Calcul de la nouvelle valeur de la fonction objectif
                IF TEXC < NAS               Poursuite de la construction de solutions complètes
                        lanceOptimisation(SC,VSC,EXC,Sk)
                ENDIF                       Poursuite de l'algorithme à la classe suivante
                IF TEXC = NAS ∧ VSO < VSC
                        optimisation(SC,VSC,EXC,Sl)   Solution complète obtenue
                ENDIF                       Transmission de la solution aux agents la composant
        ENDIF
ELSE                                        Mémorisation des solutions optimales complètes
        IF VSC > VSO                        Meilleure valeur solution reçue
                SO=SC                       Ecrasement de la solution optimale locale
                VSO=VSC                     Ecrasement de la valeur de fonction objectif  locale
                optimisation(SC,VSC,EXC,Sl) Transmission aux éléments de la classe
        ENDIF
        IF VSC = VSO                        Valeur de solution reçue identique
                SO= SO ∪ SC                 Ajout des nouvelles solutions aux solutions locales
                DIFF= ∀ Sl ∉ SO ∩ SC
                optimisation(DIIF,VSC,EXC,Sl)   Transmission des nouvelles solutions aux éléments de
        ENDIF                               la classe
ENDIF
```

Figure 85 : Algorithme formalisé proposé

2.3.7 Pertinence

De manière à valider la pertinence de l'algorithme proposé, tant en nombre de solutions trouvées que messages échangés, une simulation a été réalisée comparant l'approche du meilleur voisin avec celle proposée.

Cette étude comparative a été réalisée sous Excel à l'aide de code VBA, évitant de manipuler et modifier le code source de la plateforme. En effet le test de l'algorithme du meilleur réseau voisin nécessitait la modification de la portée de la connaissance des agents solutions pour leur permettre de connaitre de manière globale les différentes classes de composants constituant la communauté solution.

Afin de pouvoir comparer les deux algorithmes sur une même base, la partie concernant l'émergence des solutions optimales globales a été supprimée de l'algorithme proposé.

En parallèle, une évaluation exhaustive de l'ensemble des solutions a été mise en place dans l'optique de vérifier la validité et le nombre des solutions optimales retournées par les deux algorithmes.

Il ressort de cette étude que l'algorithme du meilleur réseau voisin ne permet pas la construction de solutions si l'évaluation de la fonction objectif est basée sur la méthode du minimum, tandis que l'algorithme proposé s'adapte à plusieurs méthodes d'évaluation.

La comparaison des deux algorithmes a donc été réalisée sur l'utilisation de la somme comme méthode d'évaluation de la fonction objectif. Dans ce contexte, il ressort que les deux algorithmes recensent la totalité des solutions optimales ainsi que un gain moyen de 40%, variant de 25% à plus de 50%, de la quantité de messages échangés par l'approche proposée. Quelques exemples sont illustrés par le tableau suivant (Tableau 5) :

Jeu de test	Structures des classes de composants	Nombre de messages échangés			Nombre de solutions optimales trouvées		
		MRV (1)	AP (2)	Gain AP/MRV	Recherche exhaustive	MRV	AP
Chaise	4 classes de 5 composants	1049	678	35,36%	1	1	1
		931	676	27,38%	1	1	1
		966	676	30,02%	1	1	1
		842	664	45,43%	1	1	1
Eolienne	3 classes de composants de respectivement 6, 9 et 7 composants	876	478	54,57%	2	2	2
		894	478	47,38%	2	2	2
		854	466	45,43%	2	2	2
		894	478	46,53%	2	2	2

(1) MRV= Algorithme du meilleur réseau voisin ; (2) AP = Algorithme proposé

Tableau 5 : Exemples de comparaison des algorithmes

Le déport sur Excel de cette comparaison ne permet plus de bénéficier de l'aspect aléatoire du traitement des messages apporté par l'agentification de la plateforme. En effet il est impossible de déterminer à l'avance l'ordonnancement du traitement des différents processus de la plateforme alors que le traitement des messages sous Excel est réalisé de manière itérative agent par agent, correspondant à un ordonnancement particulier. Cependant le gain important en quantité de messages échangés n'est pas remis en cause par cet aspect.

CONCLUSION

La nouvelle plateforme répond aujourd'hui aux attentes des membres de l'équipe. En effet elle est dorénavant entièrement agentifiée et les calculs souhaités ont été adaptés à cette nouvelle structure. De plus, elle offre aujourd'hui le support réseau attendu ainsi que l'évolutivité demandée qui a déjà pu être vérifiée grâce à l'ajout de nouveaux calculs intercommunautaires ou bien encore le test de différentes méthodes d'évaluation de la fonction objectif. Enfin, elle permet aussi un paramétrage aisé de nouveaux univers de configuration et a déjà permis la création d'un nouveau jeu de test de configuration d'une éolienne. Un article scientifique comprenant une présentation de la nouvelle plateforme et de ses avancées est en cours de rédaction par l'équipe et devrait prochainement être publié.

La tenue de ce projet m'a permis de découvrir le domaine palpitant de la recherche, mais aussi de m'approprier de nouvelles notions qui m'étaient jusque là inconnues, comme la logique floue, ou bien méconnues, comme le monde agent. Enfin les problèmes liés à l'optimisation m'ont particulièrement captivé.

Les évolutions futures de la plateforme tant en termes de complexité des échanges entre agents que d'optimisation des algorithmes de recherches de solutions optimales ou la création de nouvelles heuristiques mèneront certainement à de nouvelles recherches et de nouvelles expérimentations.

REFERENCES

REFERENCES

A.-J. Fougères, De l'activité collaborative aux micro-outils: l'exemple de PLACID, une plate-forme agent , CITE'06, Nantes, 2006 48

Deciu E.-R., *Contribution à une démarche de conception pour la configuration des familles de produits*, p. 53-113, Thèse de Doctorat de l'UTBM, 2007 11

Fougères A.-J., *Agents to cooperate in distributed design*, IEEE Internationnal Conefrence on Systems, Man and Cybernetic, The Hague, Netherlands, October 10-13, 2004 14

Ostrosi E., Fougères A ;-J., Ferney M., *Fuzzy agntns to assist collaborative and distributed design for optimal product configuration*, 5th International Conference on Digital Enterprise Technology, Nantes, France, 22-24 October, 2008 16

Rasmussen J., *Skills, rules and knowledge, signals, signs, and symbols, and other distinctions in human performance models*, IEEE Transactions on Systems, Man and Cybernetics, SMC-13, p. 257-266, 1983 14

Shodam Y.,*Agent Oriented Programming*, Artificial Intelligence,60, p. 51-92, 1993 13

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms. MIT Press, second edition, 2001. 68

Zadeh L.A., *Fuzzy Sets*,Information and Control, 8, p.338-353, 1965 12