Université de Sherbrooke

Faculté de genie

Département de génie électrique et informatique

Intégration du cahier des charges à la CAO en utilisant les « Agents logiciels »

Par

Dounia Habhouba

Mémoire présenté au département de génie électrique et informatique en vue de l'obtention du grade de maître en informatique

Sherbrooke, Québec, Canada, Janvier 2004

Your file Votre référence
ISBN: 0-494-05920-6
Our file Notre référence
ISBN: 0-494-05920-6

Table des matières

4. Architecture multi-agents proposée

5. Applications type de l'architecture proposée

RÉSUMÉ

Les agents logiciels ont été jusque là utilisés dans plusieurs domaines, ils ont prouvé une utilité et un intérêt inconstatable. Cette technologie a été utilisée notamment dans l'ingénierie mécanique et a donné des résultats très encourageant. Dans notre travail nous nous proposons d'intégrer ces agents logiciels au cahier des charges dans des projets de conception mécaniques. Autrement dit nous voulons concevoir un système qui s'occupe de la vérification automatique des contraintes de tous types (mécanique, hydraulique...) qui figurent dans le cahier des charges d'un produit mécanique.

Tout d'abord nous allons présenter dans ce mémoire le contexte du projet, c'est-à-dire la conception assistée par ordinateur, le concept d'ingénierie simultanée qui est le plus utilisé de nos jours pour le cycle de développement d'un produit et enfin les systèmes à base d'agents pour mieux comprendre le fonctionnement global des agents logiciels.

Nous allons ensuite présenter tous les travaux qui ont été déjà faits en utilisant les agents logiciels dans le domaine de l'ingénierie mécanique. Nous jugeons que cette partie est très intéressante du moment qu'elle nous donne une idée sur où est ce que nous sommes rendu maintenant avec les agents logiciels et où se situ notre travail dans tout cela.

Dans les trois dernières parties nous présentons l'architecture du système proposé, les applications types de notre architecture et les résultats obtenus du prototype réalisé. Notre système se compose de plusieurs parties. Nous avons conçu en premier lieu un éditeur de contraintes pour les saisir, les corriger et les interpréter. Nous avons réalisé ensuite un sous-système pour organiser toute l'information pertinente pour les contraintes comme les types des modèles et les attributs des contraintes. Et enfin nous avons mis au point le sous-système qui s'occupe de la vérification des contraintes.

REMERCIEMENTS

J'aimerai tout d'abord remercier mes deux directeurs de recherche Madame **Soumaya Cherkaoui** et Monsieur **Alain Desrochers** pour leurs patience, leurs soutien et leurs encouragement. Leur encadrement et leurs conseils m'ont beaucoup aider à réussir ce projet de maîtrise.

Je tiens aussi a remercier mon très cher mari **Mehdi Haitami** pour son grand soutien et pour ses encouragements continus tout au long de ces deux années d'études.

Je remercie aussi toute ma famille pour leurs grand soutien moral même s'ils sont si loin de moi.

Je voudrais finalement remercier tous les membres du groupe de recherche **INTERLAB** pour avoir formé tous ensemble un groupe très dynamique et enrichissant.

Liste des tableaux

Liste des figures

CHAPITRE 1

INTRODUCTION

L'ingénierie des produits est une tâche qui est de plus en plus réalisée de manière distribuée. Les travaux à différentes étapes du processus de réalisation de produits se font souvent dans différentes filiales d'une organisation, situées dans des villes ou même des pays différents. De plus, les compagnies sous-traitent aujourd'hui une partie importante des tâches dans des entreprises spécialisées afin de diminuer le coût final de production. Ces tâches peuvent appartenir à des domaines d'expertises différents : électriques, mécaniques, hydrauliques etc... qui dépendent en fait les uns des autres. Les différentes tâches doivent donc respecter les contraintes et les exigences du cahier des charges pour que le système électrique par exemple, corresponde bien au système mécanique conçu ainsi qu'au système hydraulique. De cette manière, tous les sous-systèmes qui constituent un produit vont converger finalement vers le produit désiré. Ceci nécessite justement une collaboration entre les différentes filières d'une organisation, en d'autres termes, entre les différents concepteurs des sous-systèmes d'un produit donné. Des outils de collaboration efficaces et intelligents sont donc de plus en plus nécessaires pour réaliser cette coopération globale.

Le travail présenté s'inscrit dans le cadre de cette problématique. Il s'agit de la conception et de la réalisation d'un outil de collaboration intelligent pour **l'intégration du cahier des charges à la conception assistée par ordinateur (CAO) en utilisant les agents logiciels dans un contexte d'ingénierie simultanée.**

Plus spécifiquement, dans un contexte de conception assistée par ordinateur, il arrive souvent que certaines contraintes spécifiées lors de la définition initiale du cahier de charge soient oubliées et non respectées à la fin du processus de conception. Le projet a donc pour but de concevoir une approche et définir des mécanismes qui permettront la gestion automatique de ces contraintes de manière intégrée à la totalité du processus de conception.

Pour implémenter le concept proposé, nous avons choisi la technologie « agent », car elle constitue un outil efficace qui pourrait aider à intégrer les différentes données hétérogènes appartenant à un projet donné, ce qui augmente par conséquent l'efficacité du processus de conception. Dans cette perspective, un prototype du système a été déjà développé et validé dans le contexte de l'aéronautique [13].

Les objectifs principaux du projet dans sa globalité sont :

- Traduire les besoins du client sous forme de contraintes mathématiques.
- Proposer une architecture qui permette la vérification automatique des spécifications techniques dans un système de CAO.
- Établir un lien entre la phase de spécification des besoins et la phase de conception détaillée.
- Maximiser la communication entre les différents utilisateurs d'un système de CAO dans un cadre d'ingénierie simultanée.

En fait, nous désirons intégrer les agents à l'étape de conception préliminaire au niveau des outils de CAO, notre objectif étant l'intégration du cahier des charges dans un projet d'ingénierie, par exemple en aéronautique ou en automobile, dans un contexte d'ingénierie simultanée. Dans tout projet de conception, il y a présence de contraintes. À mesure que le projet avance, les contraintes se précisent et tous les documents reliés au projet se modifient. Par contre, la vérification des reste cependant assez limitée, car il est difficile de vérifier constamment l'avancement d'un projet et les modifications qui y sont apportées. Cela est vrai parce qu'il y a toujours de multiples personnes qui travaillent sur un projet et que leur travail n'est répertorié que périodiquement. Notre projet a justement comme objectif de remédier à ce problème en utilisant des agents qui vont automatiser et contrôler le mécanisme de vérification de contraintes. Ces agents vont collaborer entre eux pour éliminer le risque de déviation par rapport aux spécifications lors du processus de conception. De cette manière, nous aurons intégré les agents aux outils de CAO en créant ainsi des outils de conception intelligents.

CHAPITRE 2

CONTEXTE DU PROJET

2.1. Conception assistée par ordinateur (CAO) et système de gestion de données techniques (SGDT)

2.1.1. Définition de la CAO

La CAO ou conception assistée par ordinateur vise à assister dans son travail tout concepteur dans plusieurs domaines (mécanique, électronique, informatique, etc.). En ce qui nous concerne, nous allons plutôt nous intéresser à la conception mécanique assistée par ordinateur.

La CAO permet la modélisation géométrique 2D avec les fonctions de dessin et de schématique par exemple, et la modélisation 3D pour traiter de la géométrie dans l'espace. Le taux d'utilisation des outils CAO dans les entreprises est important. Presque tous les domaines de l'industrie font appel aux logiciels CAO pour concevoir des produits à base de géométrie allant du simple au complexe et notamment dans le domaine de la mécanique.

La conception en ingénierie a bien évolué depuis l'ère de l'industrialisation jusqu'à l'ère de l'intégration. Au 18^{ième} siècle, le développement des machines assistant ou remplaçant l'homme a vu le jour, surtout pour les tâches répétitives dans les chaînes de fabrication. À cette époque, le monde a connu les premières chaînes de montage automatisées (Ford). Au 20^{ième} siècle, l'ordinateur assistant ou remplaçant l'homme dans les calculs les plus complexes est apparu. Selon la capacité des ordinateurs, les logiciels commençaient à apparaître sur le marché, les ingénieurs de l'époque étaient donc poussés à redéfinir des tâches suivant les fonctions réalisées par ces logiciels. En 1963 Ivan Sutherland [1] a entamé les premiers travaux sur les systèmes de dessin assisté par ordinateur (DAO) qui ont évolué petit à petit vers la CAO et ce, en incluant des fonctions de calculs qui exploitent les maquettes virtuelles. Vers la fin du 20^{ième} siècle et le début du 21^{ième} siècle, l'Internet est apparu ce qui a permis le partage de l'information. Par conséquent, des standards universels d'échanges d'information

ont commencé à se développer (norme STEP[2]). Cette époque a été caractérisée aussi par l'explosion des fonctions dans les systèmes CAO.

Les systèmes de CAO permettent d'assister les concepteurs dans la gestion de la complexité des produits dans un contexte où la pression du marché incite à produire plus rapidement et à moindre coût. Ces logiciels de CAO s'occupent des tâches routinières et permettent ainsi aux concepteurs de se concentrer sur la conception moins routinière et la créativité.

Dans l'industrie, le travail de conception se fait d'une manière distribuée. Or, il n'est pas du tout évident de gérer les problèmes générés par cette distribution, notamment l'incohérence des données. Nous avons pensé que l'intégration des agents avec les logiciels CAO pourrait présenter une solution intéressante à ce problème vu que ces entités logicielles sont capables de coopérer et de se coordonner entre elles. Les agents pourraient ainsi constituer un mécanisme de coopération sous-jacent dans ces systèmes.

2.1.2. Logiciels de CAO

De nos jours, la CAO est largement utilisée dans le domaine de l'aéronautique ainsi que dans l'industrie automobile. La conception des différents modèles d'avions ou d'automobiles se fait à l'aide de systèmes CAO [3] très puissants, entre autres CATIA[1], Solidworks[2], Autocad[3], etc. Ces différents logiciels offrent un environnement de conception, d'analyse et de simulation mécanique.

Les différents éléments d'un logiciel CAO sont (figure 2.1) :

1- Modeleur géométrique : permet la représentation de la géométrie et de la topologie;
2- Fonctions de manipulation : fonctions d'édition, de placement, de copie et fonctions de transformation géométrique;
3- Génération d'images et infographie : représentation 2D et 3D, lignes cachées, rendu réaliste, éclairage…
4- Interface utilisateur : contrôle du programme par commandes au clavier, menus et méthodes de sélection d'éléments, gestion des écrans, fenêtres….
5- Gestion de la base de données des modèles.

1 : Est un produit de Dassault Systems
2 : Est un produit de SolidWorks Corporation, propriété de SolidWorks
3 : Est un produit de Autodesk

6- Applications : modules utilisant le modèle géométrique pour la réalisation des analyses et des simulations pour évaluer le produit ou pour le préparer à la fabrication. Ces modulent intègrent aussi des connaissances propres à une expertise.
7- Utilitaires : ensemble de fonctions qui affectent l'opération du système.

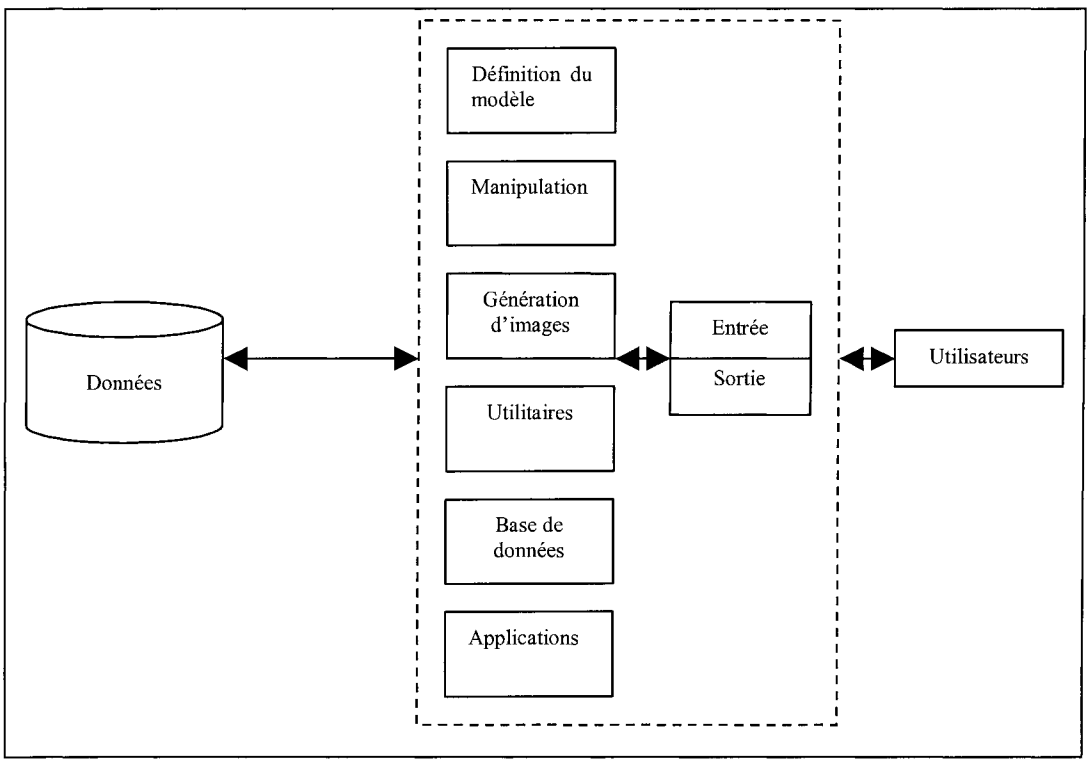


Figure 2.1 : Architecture d'un système CAO

L'utilisation d'un SGDT (voir 2.1.3) avec les logiciels de CAO permet de bien gérer l'accès aux différents modèles du système à développer dans un cadre d'ingénierie simultanée. Le logiciel de CAO que nous avons décidé d'utiliser pour le projet est CATIA puisque nous remarquons que c'est le logiciel de CAO le plus utilisé sur le marché actuellement. Toutefois notre système est conçu de manière à ce qu'il soit indépendant de tout système CAO que ce soit CATIA ou un autre.

2.1.3. Logiciels de Systèmes de gestion de données techniques (SGDT)

Les systèmes de gestion de données fournissent un moyen sécurisé pour modéliser, stocker, fédérer et gérer des données relatives aux produits et services offerts par une entreprise donnée. Ils permettent à chaque acteur au sein de ces organisations distribuées de disposer d'une vue sur les informations relatives aux produits, constituants et assemblages, afin d'assurer un premier niveau de cohérence pour des produits complexes. Les organismes qui s'intéressent à l'ingénierie mécanique étaient les premiers à découvrir le besoin pour un système de gestion de données techniques (en anglais : Product Data Management ou PDM), mais, une fois développé ce type de logiciel a été utilisé en dehors de ces domaines d'application originaux.

Dans le domaine de la mécanique, les logiciels de PDM fournissent des structures pour modéliser les relations, parfois très complexes, qui peuvent exister entre les différents constituants d'un assemblage, les personnes, les groupes de personnes, les schémas, ainsi que tous les documents liés par des liens technologiques. Les logiciels de PDM permettent aussi de stocker les différents fichiers reliés au système CAO dans une machine sécuritaire, l'accès à ces données étant bien entendu contrôlé.

En outre, avec les systèmes PDM, certaines tâches concourantes peuvent être facilement accomplies. L'ingénieur de fabrication est par exemple avisé du changement du concepteur et a accès au document de conception avant qu'il ne soit validé. Les travaux de fabrication sont étroitement liés aux travaux de conception; à chaque fois qu'il y a un changement dans la conception, le personnel de la fabrication doit être avisé pour faire les changements nécessaires.

Les systèmes de PDM peuvent être considérés comme des outils qui contribuent de très près à l'amélioration de la qualité et de la fiabilité d'un produit, accélèrent le temps de mise en marché, et participent à la réduction des coûts de développement du produit. Le problème avec ces outils c'est qu'ils ne prévoient pas la validation des changements de conception; autrement dit, la vérification des contraintes sur les modèles n'est pas prévue. Le but de notre projet est justement de réaliser un outil de vérification de contraintes qui va valider les changements effectués sur un modèle avant que le service de fabrication ne soit avisé et ce,

pour assurer que le produit soit bien conforme aux spécifications du cahier des charges. Sur la figure 2.2 nous illustrons une vue fonctionnelle d'un SGDT ou PDM[4][29].

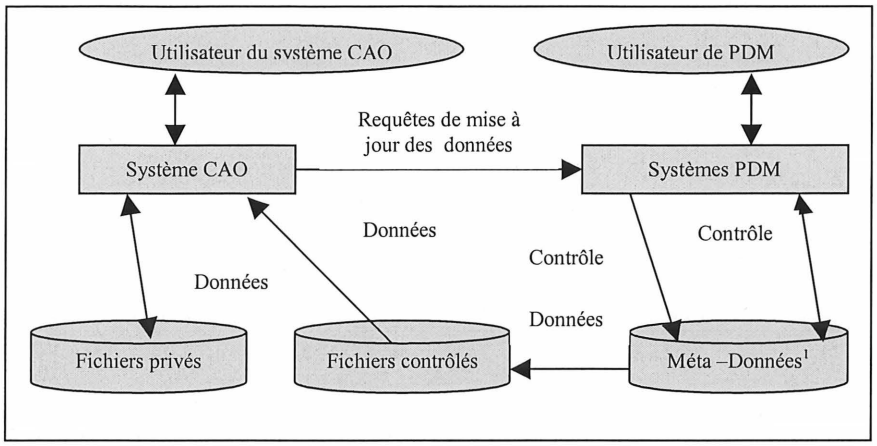


Figure 2.2 : Vue fonctionnelle d'un SGDT[4][29]

Comme illustré sur la figure 2.2, quand les utilisateurs du système de CAO agissent sur des fichiers de données, la mise à jour des méta-données[1] est faites automatiquement via le système PDM. De même si ce sont les utilisateurs du système PDM qui modifient des données sur un modèle CAO, la mise à jour est faite automatiquement sur le fichier physique du modèle. Ceci illustre la liaison étroite et l'intégration réussie des systèmes PDM et des systèmes CAO.

2.2. Ingénierie simultanée

2.2.1. Ingénierie de conception

"Engineering design" est la conception organisée et réfléchie d'un nouveau produit qui a une configuration particulière ou effectue quelques fonctions désirées, tout en respectant les besoins du client [5].

Dans l'industrie, la conception et la fabrication sont complètement séparées. Le processus de fabrication ne peut pas démarrer avant la fin du processus de conception. Dans certains domaines, notamment le domaine de l'électronique, la conception peut durer des années alors que la fabrication peut juste prendre quelques jours voire même quelques heures[5].

1 : Méta-Données : Les méta-données fournissent une description du contenu de la base de données, une description de sa structure et une définition technique des champs.

Le document fourni après la phase de conception doit être très clair pour ne pas générer d'erreurs dans la phase de fabrication et c'est pour cette raison que les formes les plus utilisées pour assurer une bonne communication entre les concepteurs et les fabricants sont les plans et les schémas qui donnent une description exacte du produit [6]. Les formes utilisées doivent elles-mêmes respecter des lois, des codes et des conventions. Avant de passer à la fabrication, le concepteur doit vérifier si le document produit respecte bien les spécifications et les contraintes.

Le concepteur peut être confronté à un certain nombre de problèmes lors de la conception. Cela peut arriver quand la problématique ne peut pas avoir une seule formulation, les objectifs paraissent vagues et les contraintes inconnues. Ou bien quand plusieurs solutions toutes valides sont disponibles, il devient alors difficile de trancher et de choisir la bonne solution. Certains problèmes peuvent même dépendre de la solution choisie; c'est-à-dire qu'on ne peut pas formuler le problème sans faire référence à une certaine solution. Pour éviter ce genre de problèmes, on doit donner une plus grande importance à l'étape de spécification et rassembler le maximum d'informations avant de passer à la phase de conception.

2.2.2. Cycle de développement d'un produit (CDP)

Le modèle le plus simple de processus de conception contient trois étapes, la première étape est la *génération* : le concepteur propose des solutions. La deuxième étape est l'*évaluation* : on vérifie si le document de conception est bien conforme aux objectifs et aux contraintes données au départ par le client. Et enfin à l'étape de *communication* le document est fourni aux manufacturiers pour la fabrication. Toutefois il existe d'autres modèles qui sont aussi utilisés que le premier. Ainsi sur la figure 2.3 nous présentons un autre modèle qui contient cinq étapes [7] : La première étape est la *définition du problème* ; le concepteur défini alors bien son problème et se prépare à la deuxième étape. La deuxième étape est l'*analyse conceptuelle;* elle permet de trouver différentes solutions et alternatives possibles. La troisième étape est la *conception préliminaire*; le concepteur choisit la solution la plus adéquate, celle qui satisfait aux besoins et objectifs du client. La quatrième étape est la *conception détaillée*; elle permet de raffiner la solution choisie par le concepteur et de la présenter en détail. Enfin la dernière étape est la *communication.*

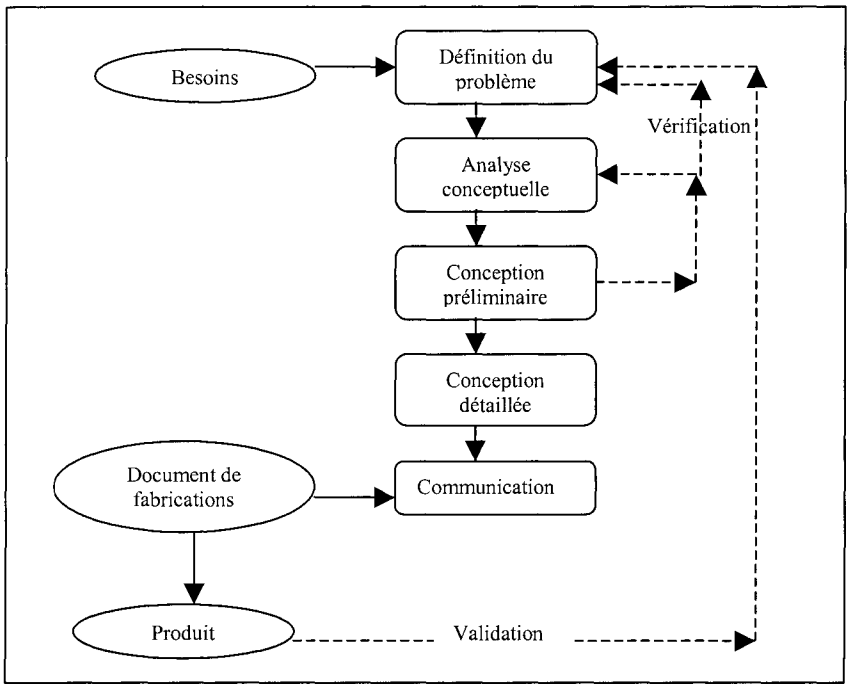


Figure 2.3 : Processus de conception [6]

Nous remarquons sur la figure 2.3 que le processus de conception n'est pas tout à fait linéaire ou séquentiel, la rétroaction est nécessaire pour que le processus fournisse de meilleurs résultats. En fait, il y a deux types de rétroactions : la *rétroaction interne* et la *rétroaction externe*. La rétroaction interne se fait pour vérifier si ce qui a été fait dans l'étape de *conception préliminaire* est bien valide et répond aux objectifs du projet. La rétroaction externe est la validation des utilisateurs après le lancement du produit dans le marché auquel il est destiné. Il y a un autre élément aussi important que la rétroaction et très utilisé dans les processus de conception : l'*itération*. Ce concept aide les concepteurs à garder en tête les objectifs établis au départ et permet de ne pas les perdre de vue au cours du processus. Il s'agit en fait de répéter certaines méthodes à plusieurs niveaux d'abstraction et ce, pour s'assurer que le produit répond bien aux spécifications du projet. Notre système épargnera aux concepteurs une tâche de taille puisque ce sont les agents qui vont s'occuper d'un certain nombre de ces itérations, ils effectueront les vérifications eux-mêmes pour voir si les spécifications sont respectées et ce, d'une manière automatique et systématique.

En ce qui concerne la CAO, le développement d'un produit exige la mise en oeuvre de plusieurs modèles [2] :

- Modèles des exigences fonctionnelles et des exigences des clients ;
- Modèles des contraintes associées;
- Modèles des chargements imposés aux mécanismes ;
- Modèles utilisés dans l'évaluation des performances tels les analyses de contraintes.

Certes, les systèmes de CAO actuels se distinguent par la multitude des modèles qu'ils permettent d'intégrer et par la capacité d'analyse qu'ils offrent. Dans le contexte de la CAO et de l'ingénierie simultanée, toutes les modifications effectuées ou générées par des analyses devraient être appliquées à un modèle central unique qui conserve la forme (géométrie, topologie, structure et dimension) du produit.

En fait, la CAO peut être appliquée à toutes les étapes du processus de conception, mais en principe elle a peu d'impact dans les premières phases puisqu'elle n'apporte rien au niveau de la génération d'idées et de solutions et pour les aspects impliquant des raisonnements complexes ou des jugements dans l'évaluation d'un concept, par exemple les méthodes et les coûts de fabrication. En revanche, il y une multitude de recherches qui s'orientent actuellement vers l'intégration maximale de la CAO dans le processus de conception [2]. Notre projet vise justement à renforcer cette intégration tout en facilitant la tâche des concepteurs. Ainsi les contraintes qui sont appliquées sur les différents modèles de CAO peuvent être vérifiées automatiquement et le processus d'itération est assuré par les agents.

2.2.3. L'ingénierie simultanée

Confrontées quotidiennement à l'émergence de nouveaux marchés et à la croissante de la compétition, les entreprises soucieuses d'accroître leur productivité et leur compétitivité ne peuvent plus suivre le cycle classique séquentiel de développement d'un produit. Ces entreprises doivent produire à moindre coût et de plus en plus rapidement. C'est dans ce but que le concept d'ingénierie simultanée a été développé [8].

L'ingénierie simultanée est une approche qui intègre simultanément les différentes phases de développement d'un produit[30], notamment l'identification des besoins du client, les spécifications, la conception et la fabrication du produit, incluant le service après-vente, l'entretien, et même le recyclage. En travaillant dans un environnement d'équipes multidisciplinaires, il est possible de développer rapidement des produits de qualité, à des coûts compétitifs.

L'accélération du cycle de conception et l'augmentation de la qualité des produits deviennent essentielles pour affronter la concurrence sur le marché. Pour parvenir à améliorer le processus de production, il faut être en mesure de concevoir des produits qui satisfont parfaitement aux besoins et aux attentes des clients, et raccourcir le temps de mise en marché.

L'ingénierie concurrente ou simultanée permet de franchir une étape nouvelle dans la gestion de projets de développement de systèmes complexes. L'ingénierie simultanée est fortement utilisée dans le domaine aérospatial qui, par exemple, sur des projets de fabrication d'avions a permis à plusieurs équipes de développement impliquant des milliers d'ingénieurs représentant plusieurs disciplines et domaines différents (aérodynamique, propulsion, matériaux, structure, stabilité, contrôle, etc.) de mener à bien leurs tâches. L'ensemble des équipes travaille parfois sur les mêmes bases de données à partir de plusieurs terminaux répartis sur les différents continents. L'institut d'ingénierie simultanée (IIS) [28] a affirmé aussi que les résultats obtenus par l'utilisation de l'ingénierie simultanée sont très satisfaisants. Toutefois, l'ingénierie concurrente présente des limites très vite ressenties dans les gros projets. Il existe toujours quelques bases de données relatives à certaines disciplines qui restent isolées et non connectées au reste des autres bases de données durant le processus de développement. Dans notre projet, nous gérons cet aspect de l'ingénierie simultanée. Quand les ingénieurs conçoivent ou modifient un modèle, le système vérifie systématiquement et automatiquement si les contraintes sur le modèle sont bien respectées, car sinon le système doit aviser les personnes concernées situées probablement à distance et qui utiliseront probablement le modèle. De cette manière le système contrôle la distribution et évite aux industriels des pertes considérables causées par une erreur d'inattention.

Notre projet se situe aussi dans un contexte d'ingénierie simultanée, puisque les modèles CAO reliés à un projet de conception peuvent être situés sur des machines ou des sites différents et

les contraintes qui doivent être vérifiées appartiennent à des domaines d'expertises différents[33].

2.3. Système à base d'agents

2.3.1. Les agents logiciels

2.3.1.1. Définition

Il existe de nombreuses définitions du concept d'«agent». Il n'existe pas de consensus sur une définition unique. Certes, elles sont à notre sens très comparables, mais différentes selon le type d'application pour laquelle serait conçu l'agent. Le point commun entre toutes ces définitions est qu'un agent est un système informatique (matériel ou logiciel) ayant une ou plusieurs propriétés le différenciant des autres systèmes [9].

Parmi ces propriétés, nous pouvons retrouver les suivantes :
Autonomie : l'agent est capable d'agir sans l'intervention d'un tiers et contrôle ses propres actions ainsi que son état interne.
Action : un agent agit à la demande de quelqu'un pour accomplir une tâche donnée.
Flexibilité : l'agent doit être capable de percevoir son environnement et élaborer une réponse dans les temps requis. Il doit donc être proactif, c'est à dire être capable de prendre l'initiative au bon moment. Il doit aussi être sociable; capable d'interagir avec les autres agents quand la situation l'exige afin de compléter ses tâches ou aider ces agents à accomplir les leurs.
Mobilité : l'agent est capable de se déplacer d'une machine à une autre.
Compétition : L'agent est capable d'agir dans un environnement où d'autres agents interviennent. Leur but est le même mais un seul l'atteindra.

2.3.1.2. Les agents intelligents

Les agents que nous utilisons dans notre prototype n'implémentent pas vraiment un concept d'intelligence artificielle parmi ceux que nous présentons dans cette partie, mais c'est envisageable pour le développement futur du système proposé surtout dans le cadre de l'optimisation des tâches effectuées par les agents.

Les agents intelligents sont des programmes qui utilisent de l'intelligence artificielle pour réaliser leurs objectifs. Ce sont des entités logicielles capables de penser, d'apprendre. Chacun dispose d'une base de connaissance qui leur permet d'agir d'une manière autonome face à une situation donnée.

Un agent intelligent peut aussi avoir des croyances, des désirs et des intentions. Ceci dit, ses réactions peuvent être expliquées par ce qu'il croit vrai dans une situation donnée, ce qu'il veut accomplir et comment il peut atteindre ses buts de la façon la plus optimale et la plus correcte possible.

Les agents logiques : sont un exemple d'agents intelligents; ils sont basés sur les connaissances disponibles concernant le monde et un raisonnement (logique) portant sur les actions possibles sur ce monde. Ils doivent connaître :

- L'état actuel du monde ou domaine
- Comment le monde change-t-il dans le temps ?
- Qu'est ce qu'il faut accomplir ?
- Quelles sont les conséquences des actions dans différentes circonstances ?

L'utilisation des agents logiques pour la résolution de problèmes par recherche d'alternative est très utile puisqu'ils ajoutent un raisonnement logique qui permet de proposer une résolution plus efficace. Ce type d'agents intelligents est très utilisé dans les applications de bureautique et de linguistique.

Parmi les autres types d'agents intelligents sont **les agents flous**. Ils utilisent, comme leur nom l'indique, la logique floue. La logique floue permet la formalisation des imprécisions associées à la connaissance globale d'un système très complexe et l'expression du comportement d'un système par des mots. Elle permet donc la standardisation de la description d'un système et du traitement de données aussi bien numériques qu'exprimées symboliquement par des qualifications linguistiques. Les ensembles flous prennent une

approche différente des autres méthodes logiques, la valeur de la vérité étant un nombre entre 0 et 1, non pas simplement vrai ou faux (0 ou 1).

Les agents flous sont généralement utilisés dans les applications suivantes :

- Représentation simple et rapide de processus numériques continus.
- Modélisation d'une fonction de transfert non linéaire par des règles formulées en termes linguistiques.
- Contrôle de systèmes numériques à entrées bruitées ou imprécises.

Les Agents neuronaux sont aussi des agents intelligents. Leurs programmes implémentent des réseaux de neurones ce qui leurs permet d'apprendre et d'agir d'une manière intelligente dans des situations données. Les réseaux de neurones sont le résultat d'une tentative de modélisation mathématique du cerveau humain. L'idée est d'interconnecter des unités simples (neurones) capable de réaliser des calculs simples dans le but de résoudre un problème plus complexe. Les neurones possèdent une fonction d'activation qui permet d'agir sur les autres neurones du réseau. Les connexions entre les neurones sont pondérées et servent de canal de communication entre les neurones, propageant ainsi leurs activités.

2.3.1.3. Les agents mobiles

Les agents mobiles sont des programmes qui sont capable de voyager et de migrer de machine en machine pour exécuter un traitement et accomplir une tâche donnée. Nous distinguons deux types de mobilité :

1- La mobilité forte (Figure 2.4) : c'est quand l'agent migre tout en transportant son état d'exécution. L'état d'exécution de l'agent est l'ensemble d'informations caractérisant le point d'arrêt de son exécution. Après une migration forte, l'agent continue son exécution exactement à partir du point où il s'est arrêté avant sa migration [10].

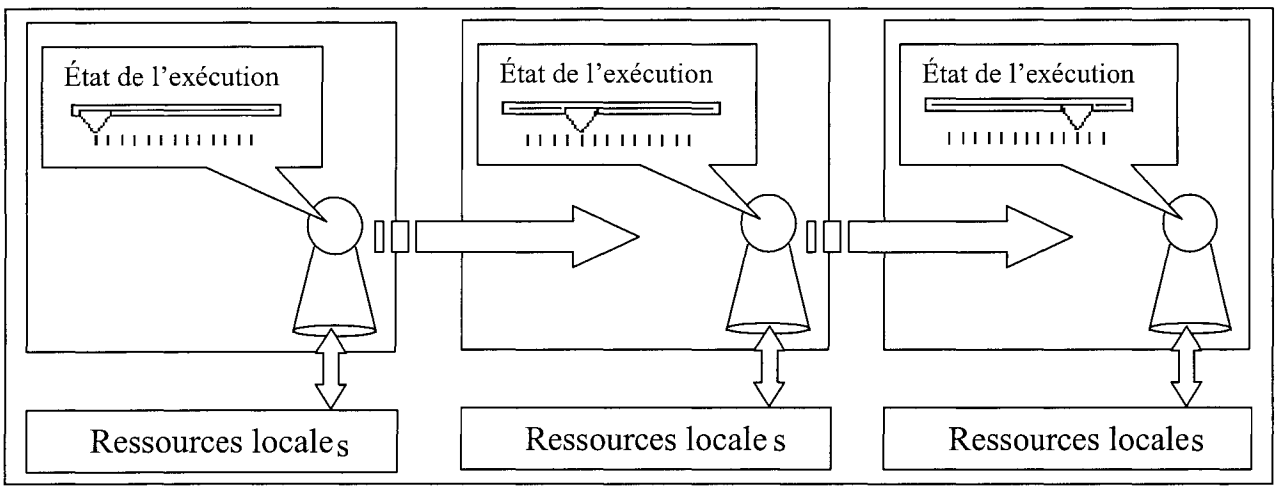


Figure 2.4 : Migration forte d'un agent mobile

2- La mobilité faible : c'est lorsque l'agent maintient son état de données en se déplaçant d'une machine à l'autre mais à chaque fois qu'il est sur une machine il s'exécute de nouveau depuis le début. Son état de données est en fait constitué de valeurs stockées dans des variables définies lors de la création de l'agent. Ces données sont envoyées via le réseau et récupérées par l'agent dans la machine destination. C'est le programmeur qui décide quelles variables feront partie de l'état de données [10].

Les agents mobiles sont des programmes capables de se mouvoir au sein d'un réseau d'une manière autonome. La protection de ces agents contre des hôtes malveillants représente certainement un problème difficile. Certes, il existe certains mécanismes de sécurité implémentés dans quelques plates-formes d'agents mobiles. Les trois principaux mécanismes sont l'authentification et le contrôle d'accès aux ressources d'un hôte exécutant un agent mobile, l'intégrité des agents et la confidentialité des informations véhiculées par les agents mobiles [10].

Pour diminuer l'importance des calculs effectués par les agents dans notre système et le trafic sur le réseau, nous avons décidé d'utiliser des agents mobiles qui vont se déplacer d'une machine à une autre au besoin, et utiliser les ressources des machines puissantes sur lesquelles ils vont s'exécuter et faire leurs calculs. D'autre part, ces agents mobiles vont éliminer le

besoin de faire des requêtes puisqu'ils se déplacent pour chercher l'information dont ils ont besoin, ce qui peut éviter éventuellement un encombrement du réseau informatique.

2.3.2. Systèmes multi-agents

2.3.2.1. Interactions et coopération dans un système multi-agents

Dans un système multi-agents, les agents interagissent entre eux pour effectuer une tâche ou atteindre un but particulier. Deux agents peuvent interagir directement ou par l'intermédiaire d'un autre agent. Chaque agent est caractérisé par sa capacité à effectuer des tâches avec les ressources dont il dispose. Un agent X peut avoir besoin des ressources d'un agent Y pour l'accomplissement de son plan d'action.

La topologie des interactions possibles entre les agents est présentée comme suit :

Indépendance : Les agents ont des buts compatibles, et des ressources et des capacités suffisantes; il n'y a donc pas vraiment d'interactions entre eux.

Collaboration simple : Les agents ont des buts compatibles et des ressources suffisantes, mais les capacités sont insuffisantes. Les interactions consistent essentiellement en des allocations de tâches et en du partage d'informations. Se référer par exemple à la collaboration de spécialistes pour la résolution d'un problème qu'aucun n'a la capacité de résoudre seul.

Encombrement : Les agents ont des buts compatibles et des capacités suffisantes, mais des ressources insuffisantes.

Collaboration coordonnée : Les agents ont des buts compatibles mais leurs ressources et leurs capacités sont insuffisantes. C'est la coopération la plus complexe car elle nécessite non seulement la coordination des tâches des agents, mais aussi la gestion des ressources limitées. C'est le cas de la fabrication des produits industriels, le contrôle de réseaux de communication et le partage du temps de spécialistes travaillant simultanément sur plusieurs projets.

Compétition individuelle pure : Les agents ont des buts incompatibles, mais des ressources et des capacités suffisantes. Les agents doivent négocier et lutter pour atteindre leurs buts. Il n'y a pas d'interaction entre les agents.

Compétition collective pure : Les agents ont des buts incompatibles, des ressources suffisantes et des capacités insuffisantes. On voit donc se former des groupes d'agents unis par des liens de collaboration et on voit ces groupes s'affronter entre eux.

Conflits individuels pour des ressources : Les agents ont des buts incompatibles, des ressources insuffisantes et des capacités suffisantes. C'est une situation de conflit dans laquelle l'enjeu est l'acquisition de ressources suffisantes.

Conflit collectif pour des ressources : Les agents ont des buts incompatibles, des ressources et des capacités insuffisantes. Dans ce cas les agents doivent s'associer en groupes de manière à s'affronter pour acquérir une ressource.

On peut caractériser un système par le type de coopération mis en œuvre qui peut aller de la coopération totale (buts communs, ressources et capacités suffisantes) à l'antagonisme. Des agents totalement coopératifs peuvent changer leurs buts pour résoudre les besoins des autres agents afin d'assurer une meilleure coordination entre eux. Le coût de communication est dans ce cas élevé. Les agents antagonistes par contre ne vont pas coopérer, le coût de communication dans de tels systèmes est minimal. La plupart des systèmes réels se situent entre les deux extrêmes : coopération totale et antagonisme total [9].

2.3.2.2. Coordination dans un système multi-agents

Dans ce contexte les agents doivent être capables d'allouer des ressources rares et d'échanger des résultats intermédiaires (deux personnes qui se parlent à tour de rôle en se passant un micro). Les agents doivent par conséquent être capables de communiquer entre eux pour échanger les résultats intermédiaires. Pour l'allocation des ressources partagées, les agents doivent être capables de faire des transferts de ressources.

Il y a trois types de coordination [9] :

Ajustement mutuel : la forme de coordination la plus simple qui se produit quand deux ou plusieurs agents se coordonnent pour partager des ressources pour atteindre un but commun. Les agents doivent donc échanger de nombreuses informations et faire plusieurs ajustements à leurs propres comportements en tenant compte des comportements des autres agents. Dans cette forme de coordination, aucun agent n'a un contrôle sur les autres agents et le processus de décision est conjoint.

Supervision directe : La supervision directe apparaît quand un ou plusieurs agents ont déjà établi une relation dans laquelle un des agents a un contrôle sur les autres. Cette relation est établie par ajustement mutuel comme par exemple dans le cas d'un employé ou d'un sous-contractant qui accepte de suivre les instructions du superviseur. Dans cette forme de coordination, l'agent superviseur contrôle l'utilisation des ressources partagées (comme par exemple les ressources humaines, le temps de calcul, les ressources financières) par les agents subordonnés. Il peut aussi imposer certains comportements.

Coordination par standardisation : l'agent superviseur coordonne les activités en établissant des procédures que doivent suivre les agents subordonnés dans des situations identifiées. On trouve par exemple de telles procédures dans les entreprises, mais aussi dans les systèmes informatiques.

Il existe deux grandes catégories de relations entre les actions accomplies simultanément par plusieurs agents : les relations négatives et les relations positives. Les relations négatives sont celles qui empêchent plusieurs actions de s'accomplir simultanément et sont dues en général à des incompatibilités de buts ou des conflits de ressources. Par exemple dans une vente aux enchères, agent X et agent Y veulent acquérir un même meuble M. Les relations positives sont celles qui permettent aux actions de bénéficier les unes des autres. Par exemple les agents X, Y et Z sont dans une pièce dont les fenêtres sont fermées et les stores baissés. X a chaud et Z aimerait avoir de la lumière. Y monte les stores et ouvre une fenêtre.

2.3.2.3. Négociation au sein d'un système agents

La négociation joue un rôle fondamental dans les activités de coopération en permettant aux agents de résoudre des conflits qui pourraient mettre en péril leur coopération. En général les chercheurs en intelligence artificielle distribuée utilisent la négociation comme un mécanisme pour coordonner un groupe d'agents.

Un des protocoles les plus étudié pour la négociation est le protocole du réseau contractuel **Contract-Net** [11][31]. C'est une des approches les plus utilisées pour les systèmes multi-agents (SMA). Les agents coordonnent leurs activités grâce à l'établissement de contrats pour atteindre des buts spécifiques. Un agent, agissant comme un gestionnaire, décompose son contrat (une tâche ou un problème) en sous-contrats qui pourront être traités par des agents contractants potentiels. Le gestionnaire annonce chaque sous-contrat sur un réseau d'agents. Les agents reçoivent et évaluent l'annonce. Les agents qui ont les ressources appropriées, l'expertise ou l'information requise envoient au gestionnaire des soumissions qui indiquent leurs capacités à réaliser la tâche annoncée. Le gestionnaire évalue les soumissions et accorde les tâches aux agents les plus appropriés. Ces agents sont appelés des contractants. Enfin, gestionnaires et contractants échangent les informations nécessaires durant l'accomplissement des tâches.

Un autre protocole de négociation assez important a été proposé pour le domaine du contrôle de trafic aérien [9] avec le but de permettre à chaque agent (représentant un avion) de construire un plan de vol qui permette de garder une distance sécuritaire par rapport aux autres avions et de satisfaire des contraintes telles qu'atteindre la destination désirée avec une consommation de carburant minimale. La stratégie choisie permet aux agents impliqués dans une situation conflictuelle potentielle (des avions se rapprochant trop, compte tenu de leurs caps respectifs), de choisir l'un d'eux pour résoudre le conflit. L'agent choisi agit comme un planificateur centralisé et développe un plan multi-agents qui spécifie les actions concurrentes de tous les avions impliqués. Les agents utilisent la négociation pour déterminer qui est le plus apte à réaliser le plan. Cette aptitude est évaluée à partir de divers critères permettant d'identifier par exemple l'agent le mieux informé ou celui qui est le plus contraint. Les

protocoles de négociation proposés supposent que les agents sont coopératifs, et donc qu'ils poursuivent un but commun.

La principale caractéristique de la négociation est la présence d'un confit qui doit être résolu de façon décentralisée par des agents ayant des informations incomplètes. Ces agents communiquent et échangent des propositions et des contre-propositions.

Les recherches en négociation peuvent être divisées en trois catégories. Les recherches sur les langages de négociation qui s'intéressent aux primitives de communication pour la négociation, à leur sémantique et à leur usage dans les protocoles, les recherches sur les décisions en négociation qui s'intéressent aux algorithmes pour comparer les sujets de négociation, les fonctions d'utilités et la caractérisation des préférences des agents. Il y a enfin les recherches sur les processus de négociation qui étudient des modèles généraux de comportement de négociation des agents.

2.3.2.4 Planification au sein d'un système multi-agents

Pour avoir une plus grande coordination entre les agents, leurs comportements doivent être orientés vers des buts communs en établissant explicitement une division du travail entre eux. Des techniques, comme la planification centralisée pour des groupes d'agents, la conciliation de plans, la planification distribuée, l'analyse organisationnelle, sont toutes des façons d'aider les agents à aligner leurs activités en assignant les tâches après avoir raisonné sur les conséquences de réaliser ces tâches dans des ordres particuliers. On peut distinguer plusieurs approches pour la planification :

La planification pour un seul agent : elle ne fait que construire une séquence d'actions en ne considérant que les buts de l'agent, ses capacités et les contraintes imposées par son environnement. Par contre, dans un environnement multi-agents, on se doit de tenir compte des contraintes que les actions des autres agents placent sur le choix des actions de l'agent. Les engagements de l'agent envers les autres agents lui imposent un certain choix d'action. L'intelligence artificielle distribuée est principalement concentrée sur des groupes d'agents qui

poursuivent des buts communs. On choisit alors de privilégier une approche selon laquelle on planifie avant d'agir [9].

Planification multi-agents centralisée[39] : un agent est responsable de la création du plan qui spécifie les actions planifiées pour tous les agents concernés. C'est cette approche qui a été proposée pour le problème de contrôle de trafic aérien par exemple. Une manière d'implémenter cette approche, c'est de créer tout d'abord les plans de façon individuelle, ensuite un agent centralisateur rassemble ces plans et les analyse pour identifier les conflits. L'agent centralisateur en question essaye de résoudre les conflits en modifiant les plans locaux des autres agents et en introduisant des commandes de communication afin que les agents se synchronisent de façon appropriée [9].

La Planification Partielle Globale [12][34] : (en anglais GPP pour Global and Partial Planning) est une approche flexible qui permet aux divers agents d'un système de se coordonner dynamiquement. Les agents interagissent en se communiquant leurs plans et leurs buts selon un niveau d'abstraction approprié. Ces communications permettent à chacun d'anticiper quelles seront les actions futures d'un ou de plusieurs autres agents, augmentant ainsi la cohérence de l'ensemble des agents. Comme les agents coopèrent, le receveur d'un message peut utiliser les informations reçues afin d'ajuster sa propre planification.

En général, la planification multi-agents nécessite une forme ou une autre de synchronisation de plans qui peut être réalisée à divers moments: pendant la décomposition de plan, pendant la construction de plan ou après celle-ci. Les plans des agents peuvent être en conflit en raison d'incompatibilités d'états des systèmes, de l'ordre des activités ou de l'usage des ressources. De tels conflits peuvent être résolus par un agent en particulier (coordonnateur ou médiateur) ou une solution peut être obtenue par négociation.

Pour assurer, dans un développement futur, une optimisation dans l'exécution des différents plans de nos agents, nous aurons recours à ces différentes stratégies de coordination, de planification et de négociation.

2.4. Conclusion

Nous avons essayé, dans cette partie, de faire le tour des technologies qui encadrent notre projet. En effet notre système s'inscrit dans le cadre de l'ingénierie simultanée pour le domaine de la CAO. Il vise à intégrer le cahier des charges au processus de conception et ceci, en utilisant des agents logiciels capables de communiquer et de coopérer entre eux. La question qu'il convient de poser à ce niveau c'est de savoir s'il existe déjà des systèmes qui intègrent aussi bien les agents logiciels à la CAO. Nous tenterons de répondre à cette question dans le prochain chapitre.

CHAPITRE 3

REVUE DE LA LITÉRATURE

3.1. Présentation de la problématique

Les objectifs principaux du projet dans sa globalité sont :

- Proposer une architecture qui permette la vérification automatique des spécifications techniques dans un système de CAO.
- Traduire les besoins du client sous forme de contraintes mathématiques.
- Établir un lien entre la phase de spécification des besoins et la phase de conception détaillée.
- Maximiser la communication entre les différents utilisateurs d'un système CAO dans un cadre d'ingénierie simultanée.

Nous désirons intégrer les agents à l'étape de conception au niveau des outils CAO, notre objectif étant l'intégration du cahier des charges dans un projet d'ingénierie mécanique dans un contexte de conception assistée par ordinateur.

Notre projet se divise en trois parties : une première partie pour configurer notre système, une deuxième pour gérer les contraintes et enfin une troisième et dernière partie pour gérer la vérification des contraintes et la notification des utilisateurs. L'architecture globale du système est illustrée dans le chapitre qui suit.

3.2. Les Agents logiciels et la CAO

Pour augmenter la qualité de la conception et réduire les efforts de re-conception il est nécessaire de construire un environnement CAO coopératif. Ceci est pour coordonner les informations entre les différents concepteurs. Dans les grands projets de mécanique, comme on a déjà vu, le développement doit être organisé selon la philosophie de l'ingénierie simultanée. Le processus de conception nécessite une certaine coordination entre différents

groupes de concepteurs. Avec un système de CAO traditionnel il y a presque toujours des imprévus qui arrivent et qui retardent le processus de conception, tels que les conflits et l'inconsistance des données, autrement dit, lorsque le travail des différents concepteurs engendre des oppositions ou des actions répétitives. Le partage d'information, des idées et des données ne sont malheureusement pas suffisant pour éviter ces conflits. La conception coopérative peut être utilisée pour éviter les problèmes communs aux systèmes de CAO classique permettant ainsi de faciliter la coordination entre les différents concepteurs. Les problèmes de coopération dans les systèmes CAO peuvent être résolus en utilisant des systèmes à base d'agents.

3.3. Mise en situation par rapport aux travaux déjà fait

Il y a une multitude de projets qui ont été déjà réalisés en utilisant la technologie agent au service de l'ingénierie simultanée; PACT, SHARE, SiFA, DIDE, Co-designer et d'autres. Nous allons présenter quelques-uns uns de ces projets afin de montrer leur apport au domaine de l'ingénierie simultanée et à la conception mécanique coopérative.

3.3.1. PACT ou PACE (Palo Alto Collaborative Testbed/Environment)

PACT est un projet supervisé par un groupe de recherche de l'université Stanford, Lockheed, Hewlett-Packard et Enterprise Integration Technologie. L'objectif de ce projet est d'explorer une nouvelle méthodologie pour résoudre des problèmes d'ingénierie en se basant sur le partage de la connaissance et sur la collaboration. Nous présentons dans la figure 3.1 le schéma de l'architecture PACT.

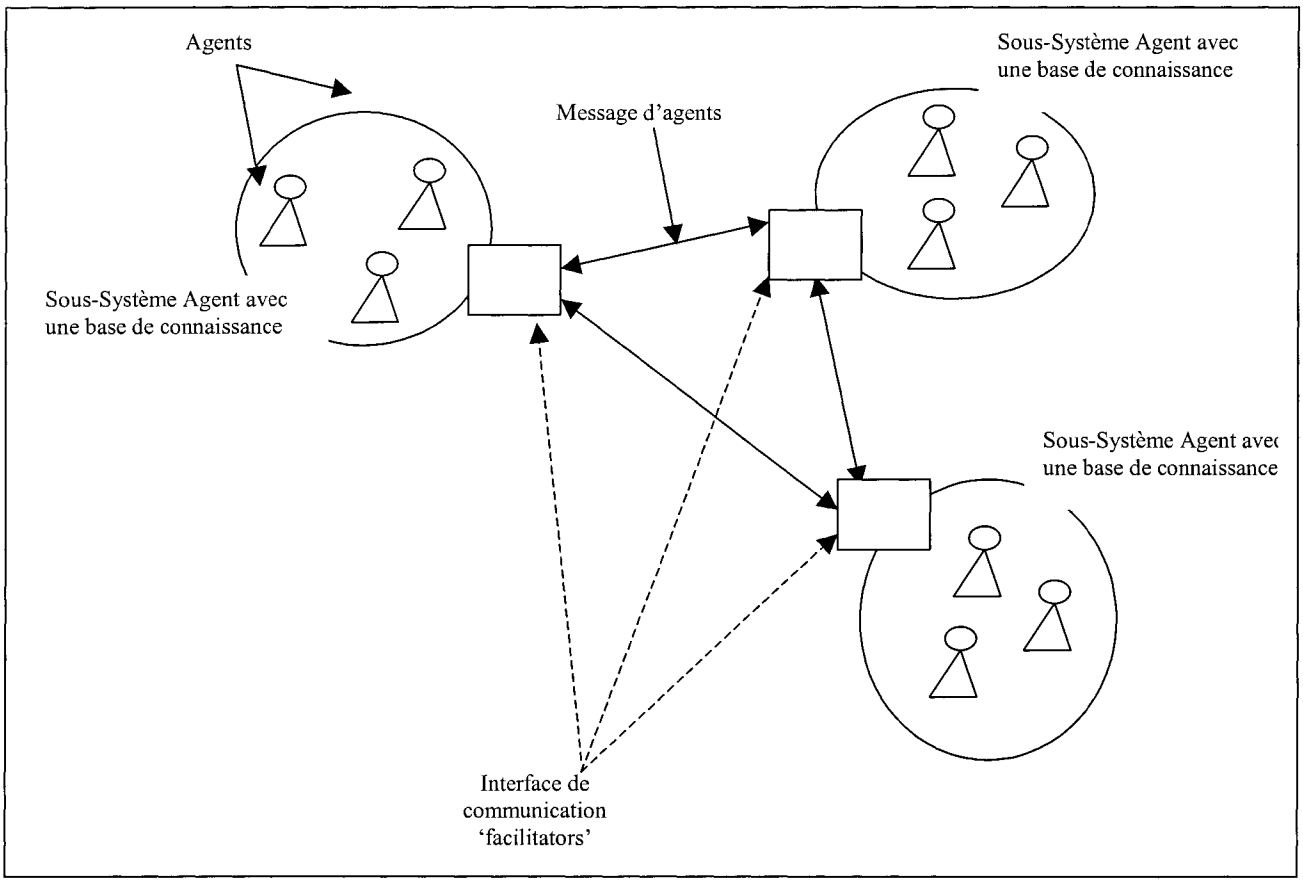


Figure 3.1 : Architecture PACT [35][14]

Les différents groupes d'agents communiquent entre eux via les interfaces de communications ou « facilitators » qui traduisent les différentes requêtes en un langage standard (ACL). L'architecture PACT a été conçue pour être intégrée dans un système ouvert qui supporte différents systèmes d'exploitation et formats de connaissances.

Le groupe de recherche a réalisé des expérimentations sur un robot manipulateur. Chaque sous-système dans l'architecture PACT modélise un aspect différent du robot des points de vues mécanique, électronique, software). Les différents sous-systèmes coopèrent entre eux via des langages et des services pour synchroniser les modifications effectuées sur le design du robot [16].

3.3.2. SHARE

SHARE est une architecture distribuée qui permet aux agents de communiquer entre eux pour résoudre les problèmes d'ingénierie, les objectifs de ce projet sont :

1- Supporter la communication et la collaboration entre les ingénieurs concepteurs.
2- Permettre aux différents outils utilisés par les différents concepteurs de partager les données.

SHARE permet la capture et l'organisation des messages multimédias concernant un produit; il incorpore plusieurs outils comme NOTEMAIL qui permet de créer, visualiser et partager des documents multimédias. Tous les documents sont stockés dans une machine. Les utilisateurs ont des pointeurs sur les différents documents [16].
Les agents dans ce système communiquent via Internet (figure 3.2), chaque agent représentant l'une des entités suivantes:

- Un concepteur et ses outils CAO personnels.
- Une base de donnée ou autres informations de services.
- Un service qui supporte le processus d'ingénierie.

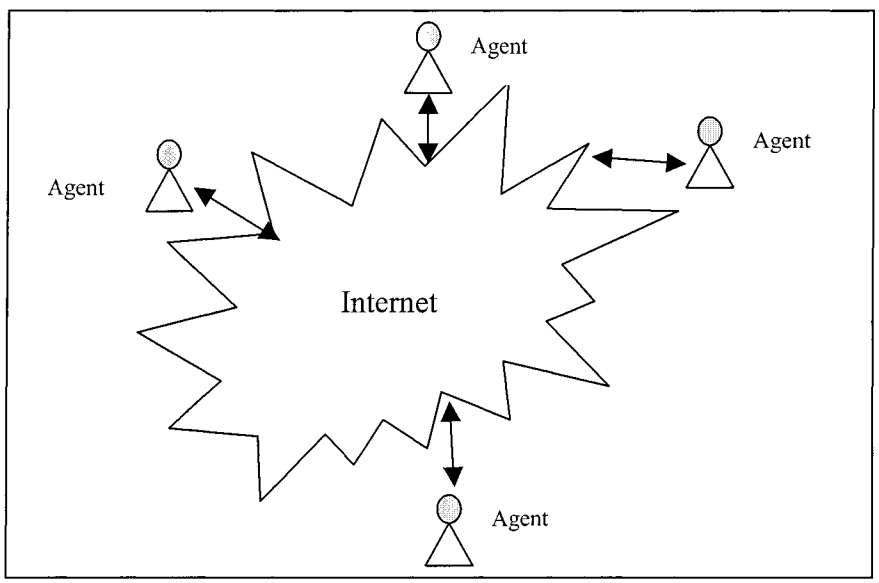


Figure 3.2 : Architecture SHARE

SHARE a été en fait remplacé par DSC (Design space colonization) 1996 [36]

3.3.3. DIDE (Distributed Intelligent Design Environment)

DIDE est une architecture basée sur la technologie agent; elle est utilisée pour développer des systèmes de CAO distribués. L'objectif essentiel de ce projet est d'intégrer les outils d'ingénierie tels que les systèmes de CAO et de FAO (Fabrication assistée par ordinateur), les bases de données les systèmes à base de connaissances dans un système complètement ouvert et dédié à l'ingénierie de conception tout en permettant une certaine intelligence lors de la fabrication [18].

Les agents dans ce système échangent les informations entre eux via Internet et n'utilisent pas de médiateurs ou de « facilitators » comme dans l'architecture PACT (figure 3.3). Tous les agents sont autonomes, indépendants et communiquent entre eux directement via le protocole OSACA (Scalabrin 1996) avec cinq types de messages : *Request, Inform, Notice, Announce* et *Bid* [19].

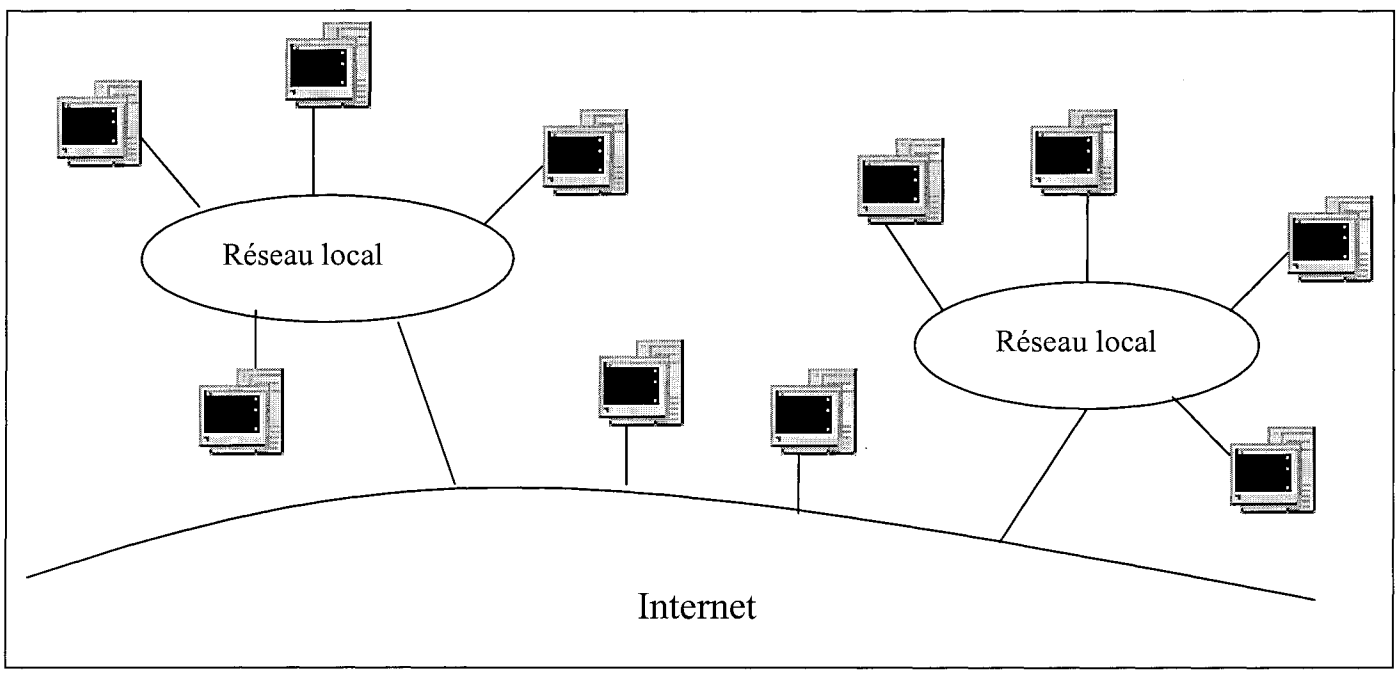


Figure 3.3 : Architecture générale de DIDE [37][16]

Dans l'architecture DIDE, chaque groupe est contrôlé par un « gestionnaire d'agents ». Chaque concepteur au sein du groupe contrôle un certain nombre d'agents et utilise une interface qui est vue comme un agent dans le système. Chaque groupe de concepteurs peut modifier un modèle localement et les résultats sont gardés sous plusieurs versions et c'est le « gestionnaire de produits » qui s'occupe de la synchronisation de toutes les versions existantes pour ne garder à la fin qu'une seule version finale.

Tous les agents dans ce système sont connectés au réseau. Quand un agent veut échanger de l'information avec un autre agent, il envoie son message en « émission » et c'est l'agent qui arrive à interpréter le message qui est le destinataire de celui-ci. Une fois qu'il reçoit le message, il commence automatiquement le travail puisque les tâches qui lui sont affectées par les autres agents sont prioritaires par rapport à son travail courant.

3.3.4. SiFA (Single Function agent)

Cette architecture vise à explorer les différentes formes d'interactions, de communication et de résolution de conflits. Dans SiFA, le processus de conception est divisé en plusieurs tâches et chaque fonctionnalité est assignée à un agent. Chaque agent dans SiFA a une seule fonction (le genre de travail que l'agent fait) qui s'applique sur une cible dans un certain point de vue.

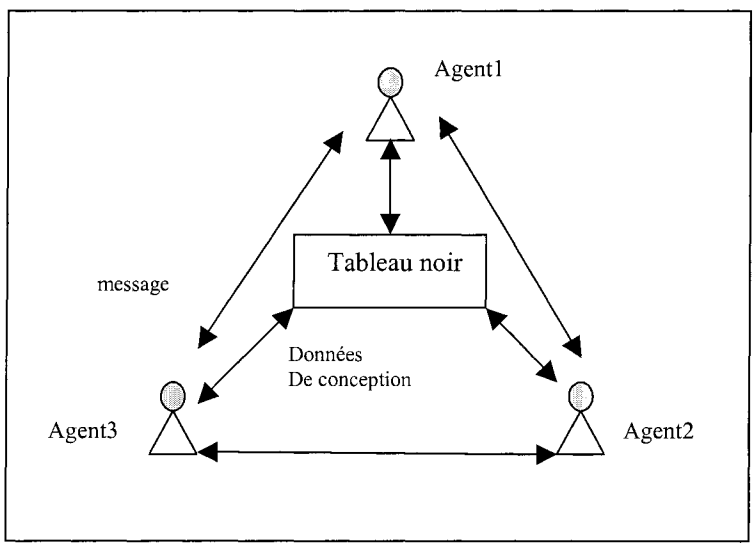


Figure 3.4 : Architecture de SiFA

Dans ce système on distingue plusieurs types d'agents. On présente dans ce qui suit, quelques-uns de ces types :

- Le « sélecteur » : qui se contente de choisir une valeur parmi d'autres.
- L'estimateur : qui donne une valeur approximative à partir des valeurs des attributs d'un modèle.
- L'évaluateur : qui utilise les données du modèle ou du produit pour voir pour quelles limites le « design » rencontre encore les objectifs fixés initialement.
- Le « suggesteur » : qui suggère des solutions alternatives pour arriver aux buts fixés au départ.

Dans le système SiFA, l'exécution des tâches est contrôlée par un agenda. Il y a une certaine planification qui donne l'ordre d'exécution des agents. Dans l'architecture agent proposée, les informations concernant l'état courant du « design » sont stockées dans un « tableau noir », ce qui les rend accessible à tous les agents et permet à ces derniers de mettre à jour l'état du « design » en cas de modifications par les concepteurs. Les agents communiquent aussi entre eux directement et c'est ce qui distingue le système SiFA des systèmes « tableau noir » classiques (voir la figure 3.4).

3.3.5. Architecture ABCDE (Agent-Based Concurrent Design Environment)

ABCDE est un système qui a été implémenté en utilisant la programmation orientée objet. C'est un système modulaire, et les différents agents du système se coordonnent entre eux via l'environnement de gestion qui comporte la conception du système. Les autres agents qui sont coordonnés par le « Sélectionneur d'agents » correspondent aux ressources de production du système de fabrication (voir la figure 3.5).

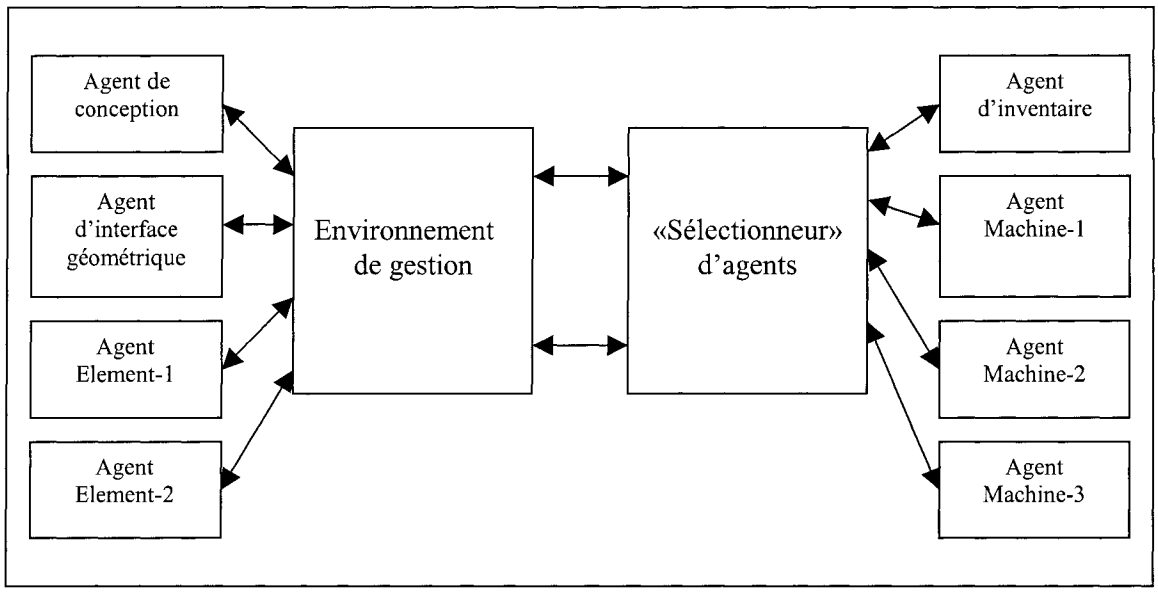


Figure 3.5 : Architecture Agent de ABCDE

Le concepteur dans le système ABCDE peut démarrer le processus en créant un modèle vide. Quand le concepteur demande à « l'environnement de gestion » d'instancier un élément dans le modèle vide, cette requête est redirigée vers l'élément correspondant. L'« Agent-Élement » demande des informations telles que la dimension, la localisation, la tolérance, et d'autres informations de fabrication en terme d'accessibilité, de stabilité et de compatibilité pour évaluer la demande. Si l'évaluation est réussie, l'« Agent-Élement » demande à « l'environnement de gestion » d'évaluer sa requête. Ce dernier la redirige vers le « Sélectionneur d'agents » qui à son tour la redirige vers l'« Agent-Machine » correspondant. Chaque machine vérifie sa capacité à traiter la demande et renvoie sa réponse au « Sélectionneur d'agents ». Ce dernier choisit un agent parmi tous ceux qui se portent volontaires pour effectuer la tâche demandée et ce, en se basant sur des critères de performance et d'optimalité [15].

3.3.6. Modèle de systèmes multi-agents pour la coopération dans les systèmes CAO : Théorie de la communication

En utilisant les systèmes multi-agents dans le cadre de la conception mécanique assistée par ordinateur nous pouvons avoir plus de coordination et de flexibilité au sein du système. Le modèle multi-agents présenté ici est proposé par Pan Xufeng [24]. Il nous propose les principes de bases pour la construction d'un système de design coopératif. La théorie de la communication au sein d'un système multi-agents peut supporter les communications inter-agents d'une manière efficace.

Pan Xufen [24] a proposé un modèle de conception coopérative d'un châssis de voiture. Au début, le modèle proposait une base de données centralisée pour le projet. Les différents concepteurs communiquent entre eux via un contrôleur qui assure la consistance et l'unicité des données. Mais cette manière de faire conduit à une lourdeur dans le réseau et une non-flexibilité dans la conception. En prenant en considération tous ces problèmes, les chercheurs ont proposé alors une architecture multi-agents pour la conception du châssis de voiture mais qui peut être utilisée dans n'importe quel autre projet mécanique nécessitant une conception coopérative.

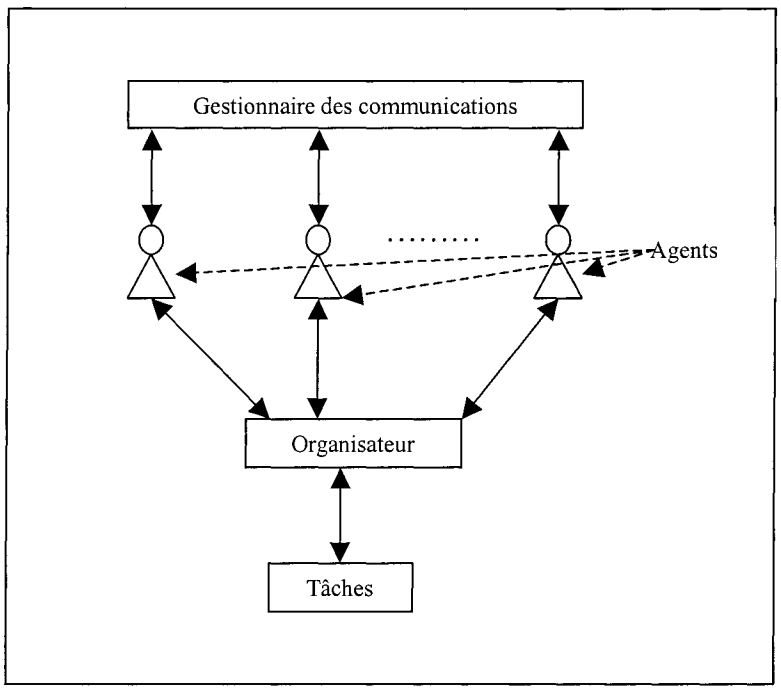


Figure 3.6 : L'architecture multi-agents proposée pour la conception coopérative [22]

Chaque module (associé à la base de données) est représenté par un agent (figure 3.6). Ces agents coopèrent entre eux et mettent à jour leurs comportements selon les changements de l'environnement de conception.

Dans ce modèle, l'organisation lors de la conception est prise en charge par l'organisateur. Chaque agent du système est reconnu par ce dernier. Au début, tout le travail de conception et d'analyse est décomposé par l'organisateur en plusieurs tâches (l'organisateur peut être lui-même un agent). Par la suite, il informe les agents concernés des tâches qu'il doit répartir et dès la réception d'une réponse de leur part, il se charge d'assigner les différentes tâches aux agents concernés. Les agents peuvent interagir entre eux via le gestionnaire de communication.

a) La fonction et la structure de l'organisateur

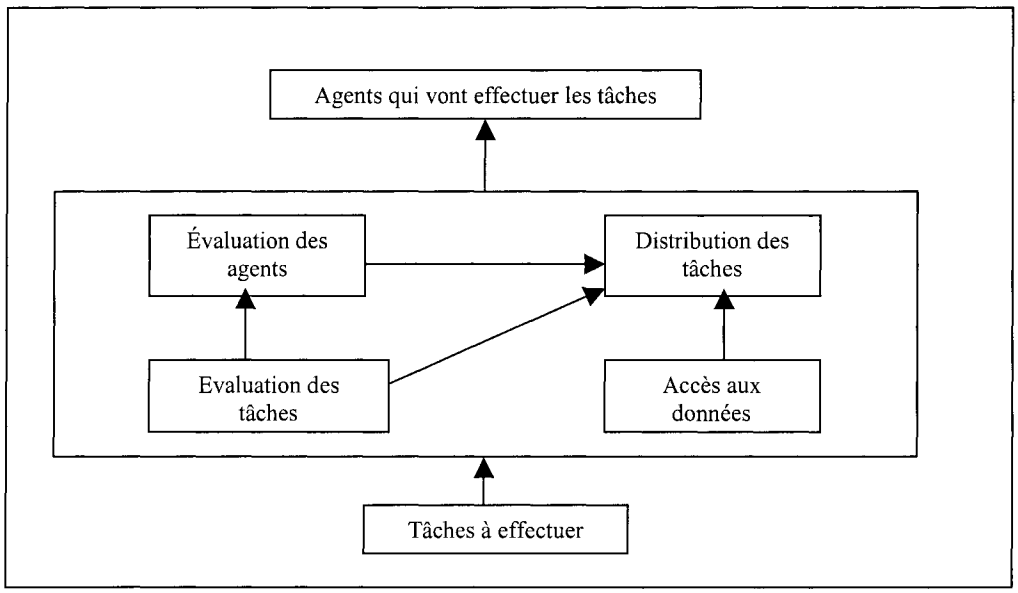


Figure 3.7 : Structure de l'organisateur [22]

Le module d'**Évaluation des tâches** évalue toutes les tâches existantes. Le module d'**Évaluation des agents** évalue le potentiel des agents à effectuer les tâches, et invite quelques agents qualifiés à faire des propositions. Le module de **Distribution de tâches** fait une synthèse et décide de la distribution effective des tâches. Le module Accès aux données sert à partager les données essentielles et pertinentes pour la coopération (figure 3.7).

b) La fonction et la structure d'un agent

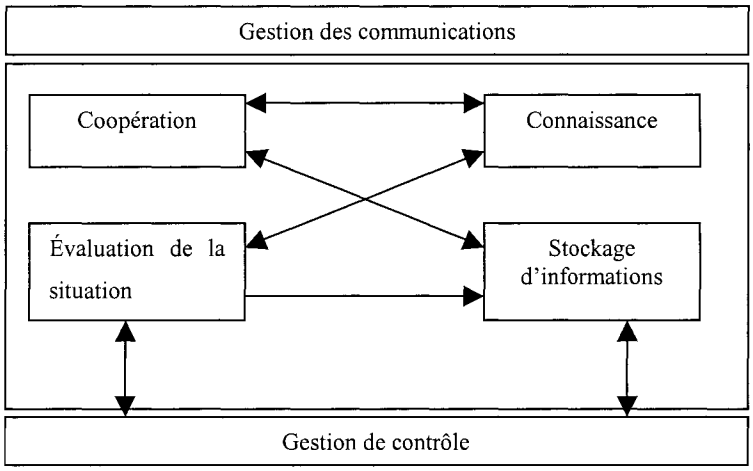


Figure 3.8 : Structure d'un agent [9]

Le module de **Gestion de contrôle** est l'interface par laquelle on peut accéder à l'agent. Le module d'**Évaluation de la situation** décide des activités qui vont être effectuées localement et de ce qui doit être transmis à un autre agent. Le module de **Coopération** est responsable de la gestion des activités d'interactions entre les agents. Le module de **Stockage d'informations** contient toutes les données qui ont été générées ou reçues suite aux activités d'interactions. Le module de **Connaissance** contient la représentation des autres agents et des systèmes locaux. Le module de **Gestion des communications** envoie et reçoit les messages des différents agents du système (figure 3.8).

c) La structure du Gestionnaire de communication

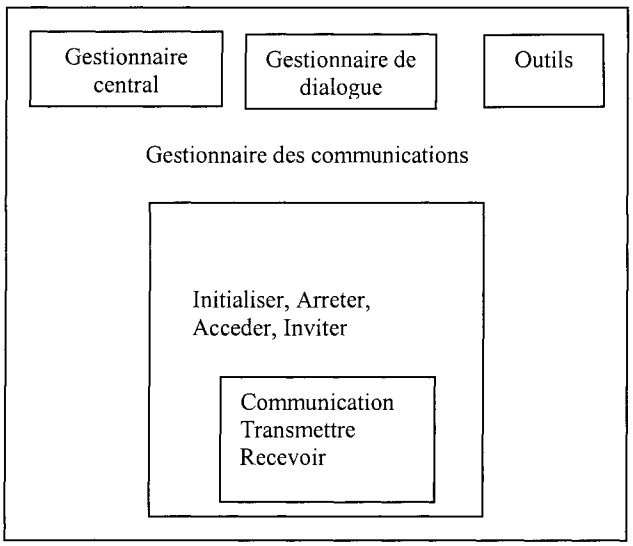


Figure 3.9 : Structure du Gestionnaire des communications [22]

Le module d'**Outils** contient quelques applications incluant des applications multimédias qui peuvent réaliser une communication complète dans le futur. Le **Gestionnaire central** contrôle l'utilisation des **Outils**. Le **Gestionnaire de dialogue** contrôle les communications entre les agents ou entre des agents et l'organisateur (figure 3.9).

d) Communications entre les agents

La conversation dans un système multi-agents peut être formulée comme un dialogue. Si la communication inter-agents est standardisée, les requêtes de communication vont être réduites et l'efficacité de la coopération entre les agents va augmenter. Donc, la gestion du dialogue dans le système multi-agents doit être fondée sur une vraie théorie de dialogue pour que la communication inter-agents soit mieux organisée [22].

Habermas [26][32], un grand sociologue allemand, a développé les bases de la théorie de la communication entre les individus. Selon Dietz's [27], la théorie de Habermas peut être considérée comme la base pour le développement de la communication inter-agents. Cependant cette théorie doit être développée davantage et adaptée aux systèmes multi-agents. Selon la théorie de la communication de Habermas, dans n'importe quelle conversation, nous

avons un orateur et un auditeur. Cette discussion est basée en fait sur trois principes : la vérité, la justice, et la sincérité. Les deux interlocuteurs négocient le degré de vérité dans une déclaration (figure 3.10) puis le degré de justice et le degré de sincérité. À la fin ils sont, ou bien tout à fait d'accord, ou bien tout à fait en désaccord.

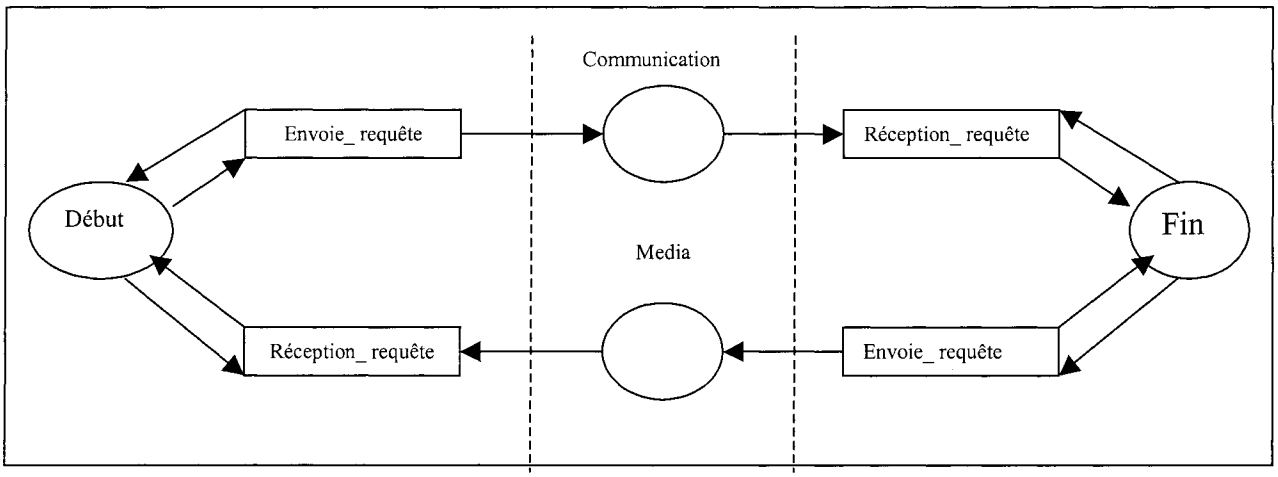


Figure 3.10 : La théorie de communication [22]

Cette théorie peut s'appliquer à la communication inter-agents. Cependant, dans les systèmes multi-agents ainsi que dans la vie de tous les jours, on peut avoir plus de deux personnes dans une conversation. En fait, le même principe est appliqué à la seule différence que nous aurons plusieurs auditeurs ou plusieurs orateurs à la fois.

3.3.7. Co-Designer

Dans cette partie, nous allons présenter le prototype co-designer qui a été développé par Hu bin, Lin Zong-kai, Lin Shou-xun et Ma Xian-lin [23]. En fait, ce prototype est basé sur la technologie agents et suit le même principe que les autres architectures présentées. Les agents au sein de ce système coopèrent entre eux pour atteindre un objectif donné.

a) Présentation du prototype

Les systèmes multi-agents peuvent être classifiés en trois catégories [23] : le modèle « Tableau noir », le modèle « Acteur » et le modèle « Contrat-net ». Dans le modèle « Tableau noir » il y a différents modèles de connaissances qui se nomment « Sources de connaissance ».

Ils sont utilisés pour accéder à une base de données globale qui constitue le « Tableau noir ». Chaque source de connaissance décrit une partie de la connaissance qui va contribuer à la résolution du problème. Dans le modèle « Acteur », les différents acteurs forment des blocs principaux du système, ces blocs peuvent communiquer entre eux directement par des messages. Les comportements d'un acteur sont définis par un script qui lui est associé et qui détermine la prochaine action la plus appropriée. Pour le modèle « Contrat-Net », l'agent gestionnaire décompose une tâche donnée en plusieurs sous-tâches; cet agent affecte alors chaque tâche à l'agent le plus approprié.

Le prototype Co-Designer présenté dans cette partie fait partie du modèle « Acteur ». Dans co-designer, chaque agent a les mêmes droits que les autres agents et il opère en mode autonome. Autrement dit, l'action que va entreprendre chaque agent est effectuée selon son état et ses conditions. Pour la communication inter-agents, c'est le mode point à point qui est utilisé. Les agents peuvent donc échanger leurs informations par des messages. Ces messages influencent aussi l'action entreprise par l'agent après la réception de ces derniers.

b) L'architecture de Co-Designer

L'architecture du Co-Designer est décomposée en plusieurs niveaux et les fonctionnalités de chaque niveau, appelées « Composantes », peuvent être programmées par des langages différents et maintenus indépendamment l'une de l'autre.

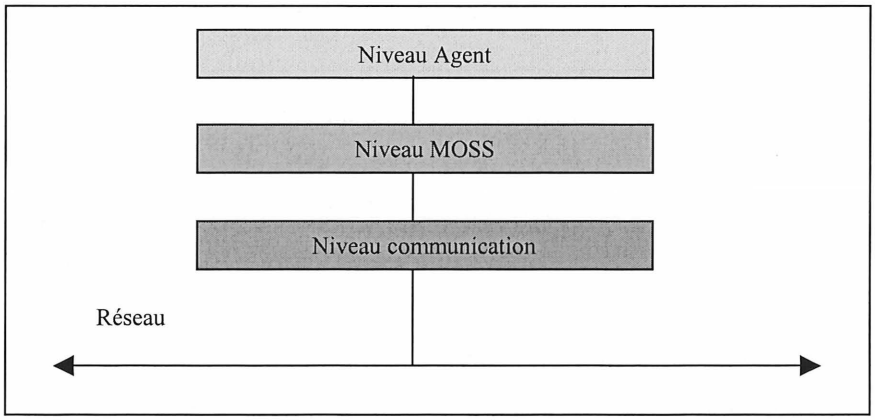


Figure 3.11 : Les trois niveaux de Co-Designer [23]

Le niveau communication (figure 3.11)

La fonction « Socket » dans UNIX, qui est basée sur le protocole TCP/IP, est utilisée à ce niveau. Ce niveau inclue aussi une librairie de fonctions programmées en C. Puisque ces fonctions cachent les détails de la communication, elles peuvent être utilisées dans n'importe quel réseau TCP/IP.

Le niveau MOSS [38] (figure 3.11)

Le but de ce niveau est de développer un environnement qui maintienne une grande flexibilité pour supporter les applications coopératives et multi–utilisateurs. MOSS est le prototype d'un environnement objet développé par l'Université de Technologie de Compiègne en France.

Le niveau Agent (figure 3.11)

Ce niveau construit un environnement de conception coopératif basé sur le niveau MOSS. L'agent est l'unité de base de ce niveau. Un agent peut contacter les autres selon un protocole de communication qui est défini par les deux fonctions « envoie_message » et « réception_message ». Dans le prototype Co-designer, il y a deux types d'agents ; des agents systèmes, et des agents outils. Plus précisément, le niveau agent offre quatre agents systèmes pour gérer les groupes d'agents :

L'agent communication-serveur : doit être exécuté en premier car il gère les groupes d'agents et permet une communication asynchrone entre les agents. Il construit aussi des tables de mémoire pour permettre aux différents agents de coopérer entre eux.

L'agent moniteur : contrôle l'état de chaque agent. Quand quelques erreurs arrivent pour certains agents, il avise les autres agents de ces erreurs et arrête les agents erronés pour qu'ils ne causent pas de tort au système.

L'agent mode de travail : gère le mode du travail du système pour augmenter son efficacité et améliorer la qualité de la conception [20].

L'agent interface humain-humain (HHI) : est une interface multimédia d'humain à humain. Les différents utilisateurs peuvent ainsi discuter d'un problème par la voix ou par le texte en utilisant une communication de type synchrone ou asynchrone. Il y a deux types d'interfaces;

l'interface « **Maître** » pour ajouter et supprimer des participants et l'interface « **Participant** » pour dresser des plans qui vont par la suite être évalué par l'interface « **Maître** » [21].

c) Application : un assemblage mécanique dans le système Co-Design

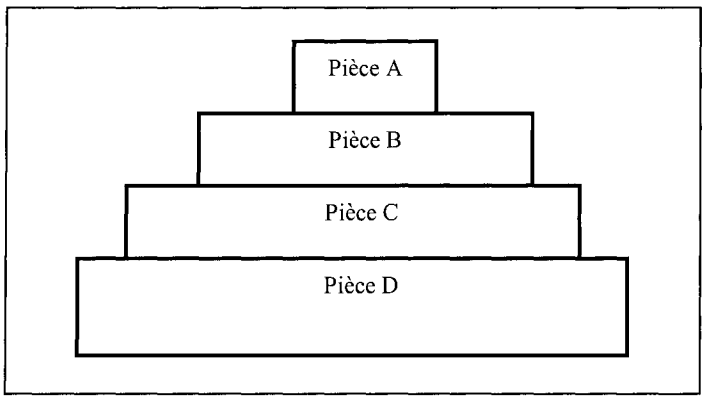


Figure 3.12 : Un assemblage mécanique [23]

L'assemblage de la figure 3.12 est affecté par trois paramètres qui vont complètement changer l'assemblage s'ils sont modifiés. Dans ce qui suit, on va construire un système multi-agents associé à cet assemblage (figure 3.13) :

1- **L'agent communication-serveur :** pour gérer le groupe d'agents et les communications entre eux.
2- **L'agent moniteur** : pour gérer l'état des différents agents.
3- **L'agent mode de travail** : pour contrôler le mode de travail dans le système.
4- **L'agent interface humain-humain (HHI)** : qui offre une interface multimédia pour discuter des paramètres reliés à l'assemblage.
5- **L'agent calculateur** : qui fait les calculs selon les paramètres en entrée.
6- **L'agent pièce A** : qui sélectionne la pièce A de la base de données et la dessine.
7- **L'agent pièce B** : qui sélectionne la pièce B de la base de données et la dessine.
8- **L'agent pièce C** : qui sélectionne la pièce C de la base de données et la dessine.
9- **L'agent pièce D** : qui sélectionne la pièce D de la base de données et la dessine.
10- **L'agent assembleur** : qui assemble et dessine le mécanisme.

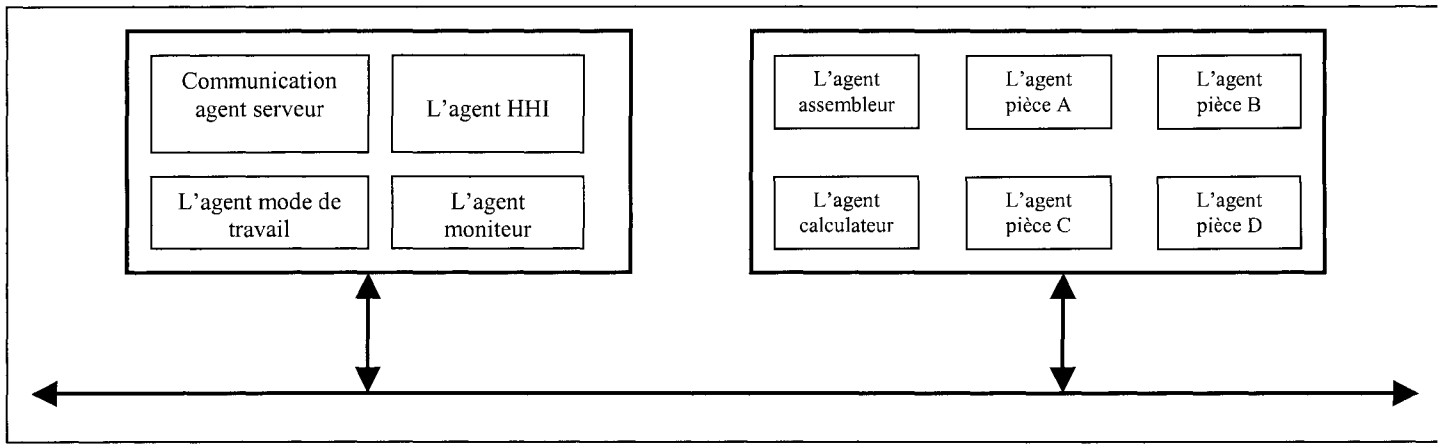


Figure 3.13 : Le système multi-agents pour l'assemblage mécanique présenté [23]

- Le processus de conception

Le processus de conception mécanique pour l'assemblage présenté plus haut implique ce qui suit :

1- Créer chaque agent en ordre ;
2- Dans **l'agent mode de travail,** définir le mode de travail du système ;
3- Dans **l'agent HHI,** discuter et définir les paramètres reliés à l'assemblage et les envoyer à **l'agent mode de travail ;**
4- **L'agent mode de travail** informe l'agent calculateur; l'agent calculateur calcule le diamètre de la pièce D selon les paramètres définis à l'étape précédente puis renvoie le résultat à **l'agent mode de travail ;**
5- **L'agent mode de travail** informe l'agent pièce A, l'agent pièce B, l'agent pièce C et l'agent pièce D. Ces agents sélectionnent par la suite les pièces adéquates de la base de données en respectant le diamètre de la pièce D et envoient ces données à **l'agent mode de travail ;**
6- **L'agent mode de travail** vérifie les données reçues. Si tout est conforme, **l'agent mode de travail** demande aux différents agents de dessiner chaque partie de l'assemblage et d'envoyer le résultat à l'agent assembleur pour assembler les pièces. Si par contre l'information reçue par **l'agent mode de travail** est erronée, il demande aux différents agents de refaire leur travail ;

7- **L'agent assembleur** assemble les différentes parties de l'assemblage puis le dessine.

3.4. Conclusion

Dans la partie précédente, nous avons présenté des systèmes qui visent à intégrer les agents dans le processus de conception et de fabrication. Ce sont tous de gros projets qui permettent de réaliser un environnement d'ingénierie simultanée ouvert, flexible et robuste. Comme nous pouvons le remarquer, les différents systèmes s'intéressent peu à l'intégration du cahier des charges au processus de conception de même qu'au respect des contraintes spécifiées initialement. Comme on l'a vu plus haut, l'architecture PACT ou PACE s'intéresse surtout au partage de l'information et à la communication entre les différents sous-systèmes qui sont basés sur des disciplines différentes. L'architecture SHARE va un peu dans le même sens puisqu'elle permet surtout le partage de l'information concernant la conception, et ce, via des courriels multimédias. L'architecture DIDE se concentre plutôt sur le problème de mise à jour des bases de connaissances des agents quand une modification est effectuée par les concepteurs.

L'architecture SiFA par contre, donne un peu d'importance aux contraintes du cahier des charges qui sont spécifiées pour le produit car elle permet à son agent "Évaluateur" de donner une certaine évaluation du modèle mais sans affirmer si le modèle est conforme ou non aux contraintes. L'architecture SiFA se concentre surtout sur la gestion des conflits.

Dans ABCDE, les chercheurs se sont concentrés surtout sur la synchronisation du besoin et des ressources disponibles et ont mis l'accent sur la communication inter-agents dans les systèmes de CAO coopératifs. Enfin, dans le projet Co-Designer les quatre chercheurs ont surtout insisté sur le design collaboratif à l'aide d'agents logiciels.

Dans notre projet, nous visons justement à bonifier ces gros projets déjà existant et à combler l'absence du concept d'intégration du cahier des charges au processus de conception. Cette approche permettra de valider le modèle mécanique au fur et à mesure qu'il est conçu. Ceci permettra évidemment de gagner du temps et de l'argent, car on ne pourra avancer dans la conception tant qu'elle ne sera pas conforme aux spécifications du cahier des charges.

CHAPITRE 4

ARCHITECTURE MULTI-AGENTS PROPOSÉE

4.1. Concepts

L'architecture multi-agents proposée permet à des experts de spécifier un ensemble de contraintes dans les premières phases de conception d'un produit. Chaque contrainte spécifiée doit être satisfaite tout au long du processus de conception. Le système multi-agents doit alors assurer la satisfaction de ces contraintes selon deux perspectives différentes [25] :

1- La première considère la diversité des domaines et des conditions appliquées au produit.
2- La deuxième concerne l'évolution du produit à travers le processus de conception.

Pour assurer une bonne vérification des contraintes selon ces deux perspectives, nous avons besoin d'utiliser différentes catégories d'agents. Chaque agent a une expertise dans un domaine précis, ce qui lui permet de contrôler le produit tout au long du processus de conception et d'évaluer si les contraintes reliées à son domaine d'expertise sont respectées. Nous allons présenter dans ce qui suit les attributs des contraintes, des stratégies, des modèles et des utilisateurs que nous proposons de définir dans le système avant de commencer la vérification.

4.1.1. Contraintes

Les contraintes définies dans notre système peuvent être générales, elles s'appliquent automatiquement à chaque nouveau produit, ce qui exprime en fait l'expertise de l'organisation dans la fabrication de ses produits. Les contraintes peuvent aussi être spécifiques à un produit en fabrication; elles concernent en général les caractéristiques qui différencient un produit d'un autre. Les contraintes sont classifiées par les agents lors de la

vérification selon différents aspects. Ces aspects sont exprimés sous forme d'attributs associés aux contraintes (tableau 4.1) :

Attribut	Valeurs de l'attribut	Nombre de valeur de l'attribut
Type	Fabrication, Électrique, Hydraulique…	1
Formule	Une formule mathématique	1
Priorité	Obligatoire, Importante, Facultative	1
Collaboration	Liste des départements	0,n
Activation	Tout le temps, Jamais, à la conception détaillée….	1
Planification	Peut être « 12h : 00 de chaque jour »	0,1
Stratégie	Nom de la stratégie	1

Tableau 4.1 : Les attributs des contraintes

Type : Le type d'une contrainte indique son domaine d'appartenance aux différents domaines de conception (fabrication, électricité, hydraulique etc.…).

Formule : La formulation d'une contrainte utilise les opérateurs conditionnels (if…then) et | ou, les opérateurs inconditionnels tel que les opérateurs booléens (and, or..), algébriques (=, +, -, /, <, >) et géométriques (fonctions généralement disponibles à travers le système CAO comme le calcul d'interférences). Cet attribut contribue au choix des agents de vérification et ne contient pas la formule de la contrainte ; il mentionne simplement les types d'éléments utilisés dans la formule de la contrainte.

Priorité : La priorité d'une contrainte peut prendre une valeur « obligatoire », « importante » ou « facultative ». Cet attribut permet de déterminer les contraintes prioritaires au cas ou nous aurions recours à la négociation lors du processus d'optimisation pour la vérification des contraintes.

Collaboration : Cet attribut indique quels sont les gens (ou les départements) qui doivent être avisés dans le cas ou une contrainte ne serait pas respectée.

Activation : Permet de spécifier à quelle étape de la conception une contrainte doit être déclenchée. Les différentes valeurs de cet attribut peuvent être « tout le temps », « jamais », « à la conception détaillée », etc.

Planification : Cet attribut précise à quel moment et à quelle fréquence une contrainte doit être déclenchée une fois que l'attribut **A** (Activation) est vérifié. Sa valeur peut être par exemple « chaque jour à 12h00 ».

Stratégie : Cet attribut indique le nom de la stratégie de vérification adoptée.

4.1.2. Stratégie

Chaque contrainte est associée à une stratégie de vérification qui précise le déploiement des agents pour la vérification de celle-ci. Les attributs d'une stratégie sont (tableau 4.2) :

Attribut	Valeurs de l'attribut	Nombre de valeur de l'attribut
Nom de la stratégie	Un nom	1
Type de calcul	Léger ou lourd	1
Fonctionnement	Par modèle ou par contrainte	1
Déclenchement	Hors ligne ou en ligne	1
Nombre de pièces	Élevé ou faible	1
Parallélisme	Parallélisable ou non-Parallélisable	1
Agent	Agent « nom de la stratégie » i	1

Tableau 4.2 : Les attributs des stratégies

Nom de la Stratégie : Indique le nom de la stratégie utilisée.

Type de calcul : Peut être « Léger » ou « Lourd ». Léger veut dire que la vérification demande un seul agent. Lourd indique que la vérification nécessite autant d'agents qu'il y a de machines dans le réseau.

Fonctionnement : La valeur de cet attribut peut être « par modèle » ou « par contrainte ». Par modèle indique qu'il faut vérifier si nous avons plusieurs contraintes qui impliquent le même modèle; dans ce cas il faudrait vérifier toutes les contraintes sur le modèle avant de passer à un autre. C'est en quelque sorte de l'optimisation, car l'opération d'examen d'un modèle est lourde en traitement. Quand la valeur de l'attribut est « par contrainte », les agents doivent vérifier les contraintes l'une à la suite de l'autre, sans se soucier d'une optimisation possible.

Déclenchement : Les valeurs de cet attribut peuvent être « Hors ligne » ou « En ligne». Le fonctionnement hors ligne veut dire que la vérification est faite quand les machines ne sont pas utilisées; ceci évite de déranger les concepteurs quand ils sont en train de travailler, surtout dans le cas ou la vérification utilise un maximum de ressources. Le fonctionnement en ligne permet de vérifier les contraintes même si les machines sont déjà utilisées par les concepteurs. Ce type de fonctionnement est peu recommandable et devrait plutôt être utilisé dans le cas d'une urgence.

Nombre de pièce : Peut être « Élevé » ou « Faible ». Élevé veut dire que le nombre de pièces impliquées dans les modèles vérifiés par cette stratégie est important. Faible veut dire que ce nombre est faible.

Parallélisme : La valeur de cet attribut peut être « parallèlisable » quand les calculs faits pour la vérification au sein de la stratégie sont parallélisable ou « non-parallèlisable » si les calculs ne peuvent pas être faits en même temps.

Agent : Indique le nom de l'agent ou des agents sous la forme suivante : Agent « nom de la stratégie ».

4.1.3. Modèles

chaque modèle au sein d'un produit a des attributs qui illustrent les différents changements effectués. Ces attributs sont (tableau 4.3) :

Attribut	Valeurs de l'attribut	Nombre de valeur de l'attribut
Géométrie	Changements sur la géométrie	1
Position	Changements sur la position	1
Statut	Étape dans le processus de conception	1
Fonction	Type du modèle	N

Tableau 4.3 : Les attributs des modèles

Géométrie : Montre les changements effectués sur la géométrie en incluant les changements topologiques et dimensionnels.

Position : Illustre les changements dans la position du modèle par rapport à une référence donnée.

Statut : Désigne l'étape dans le processus de conception où le modèle CAO est rendu. Par exemple, dans le domaine de l'industrie aéronautique on utilise les termes suivants : WIP (Work In Progress) si le modèle n'est pas encore terminé, APP (Approval Pending) lorsqu'en attente d'approbation ou REL (Released) s'il est déjà approuvé.

Fonction : Il représente la fonction du modèle et correspond à un type qui lui est associé. Cet attribut peut avoir plusieurs valeurs simultanément. La valeur de N peut être par exemple « longeron aile droite » pour un longeron qui se trouve dans la partie droite de l'aile d'un avion. Il est préférable, pour exprimer une contrainte, d'utiliser les types de modèles au lieu des identificateurs de modèles dans le système de CAO. Ceci permet premièrement aux experts d'exprimer les contraintes avant même que les modèles géométriques ne soient créés, deuxièmement ils permettent de formuler une expression générale de la contrainte sans se lier

à une instance précise. Ces attributs sont organisés dans une structure d'arbre XML dans le système. L'arbre des attributs est structuré de la manière suivante (Figure 4.1):

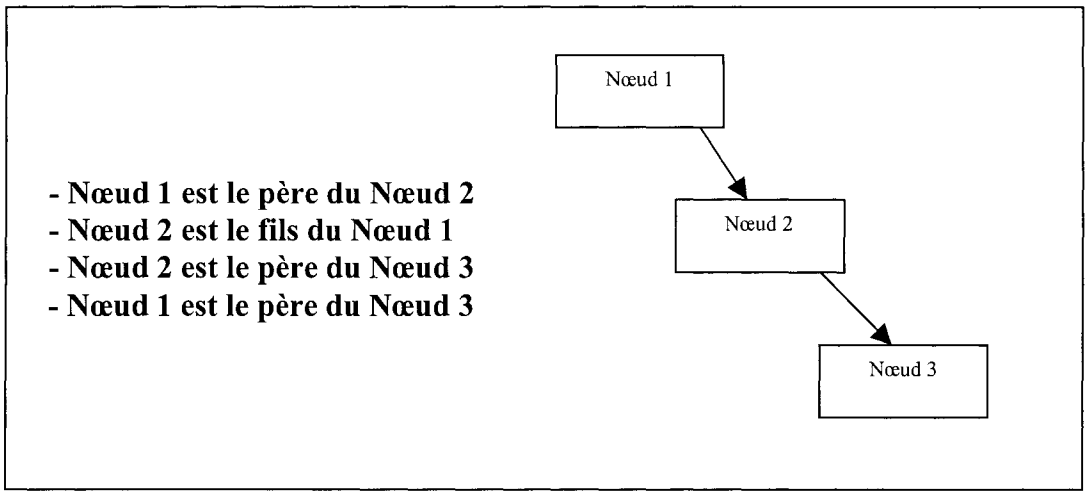


Figure 4.1 : Structure d'un arbre de types

Si un modèle a comme type Noeud2, alors il est implicitement aussi du type Nœud1. Par itération, chaque modèle qui a comme type Noeud2 a aussi comme type Nœud1 et tous ses pères. Lorsqu'on exprime une contrainte avec un attribut N1 alors cette contrainte doit être vérifiée pour tous les types N1 et Nx ou Nx sont les fils de N1. Dans la figure suivante (Figure 4.2) nous illustrons un exemple d'une arborescence de types :

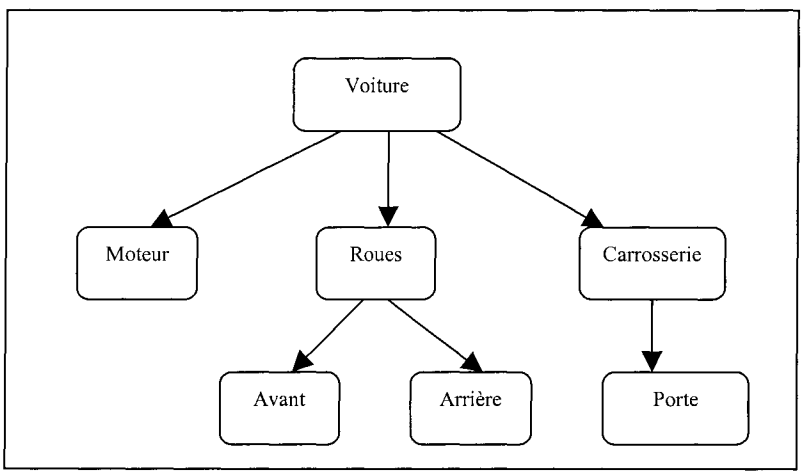


Figure 4.2 : Arborescence des types

4.1.4. Utilisateurs

Les attributs des utilisateurs sont spécifiés dans l'environnement SGDT (Système de Gestion de Données Techniques) ou PDM (Product Data Management) :

Attribut	Valeurs de l'attribut	Nombre de valeur de l'attribut
Fonction utilisateur	La fonction de l'utilisateur	1
Département	Le département de l'utilisateur	1

Tableau 4.4 : Les attributs des utilisateurs

Fonction utilisateur : cet attribut indique la fonction de l'utilisateur dans l'organisation pour un projet donné.

Département : Détermine à quel domaine l'utilisateur est associé.

Il est utile d'utiliser ces attributs lors de la définition de la politique de notification au lieu des noms des personnes car cela permet, comme avec les contraintes, de lier la notification à une fonction ou un domaine au lieu d'une personne. Deuxièmement nous pouvons réutiliser les contraintes dans un autre projet sans changer le mode de notification.

4.2. Architecture globale du système

L'architecture détaillée du système proposé (figure 4.3) contient trois sections. Tout d'abord nous avons la « section environnement » qui regroupe les logiciels externes pour la gestion des modèles et notamment le système de CAO, la gestion des usagers par le SGDT et la gestion des contraintes par les KBE (Knowledge-Based Environment). Ensuite nous avons la « section expert »; elle permet d'interfacer l'application à base d'agents avec les différents logiciels externes vus précédemment (logiciels de CAO et PDM) et d'adapter le système à un métier donné. Enfin, nous avons la « section produit »; elle représente le contexte dans lequel les agents sont actifs. Plusieurs « sections experts » représentant plusieurs métiers peuvent être

reliés cette section. La « section produit » est également liée à la « section environnement » pour une éventuelle utilisation de quelques logiciels externes.

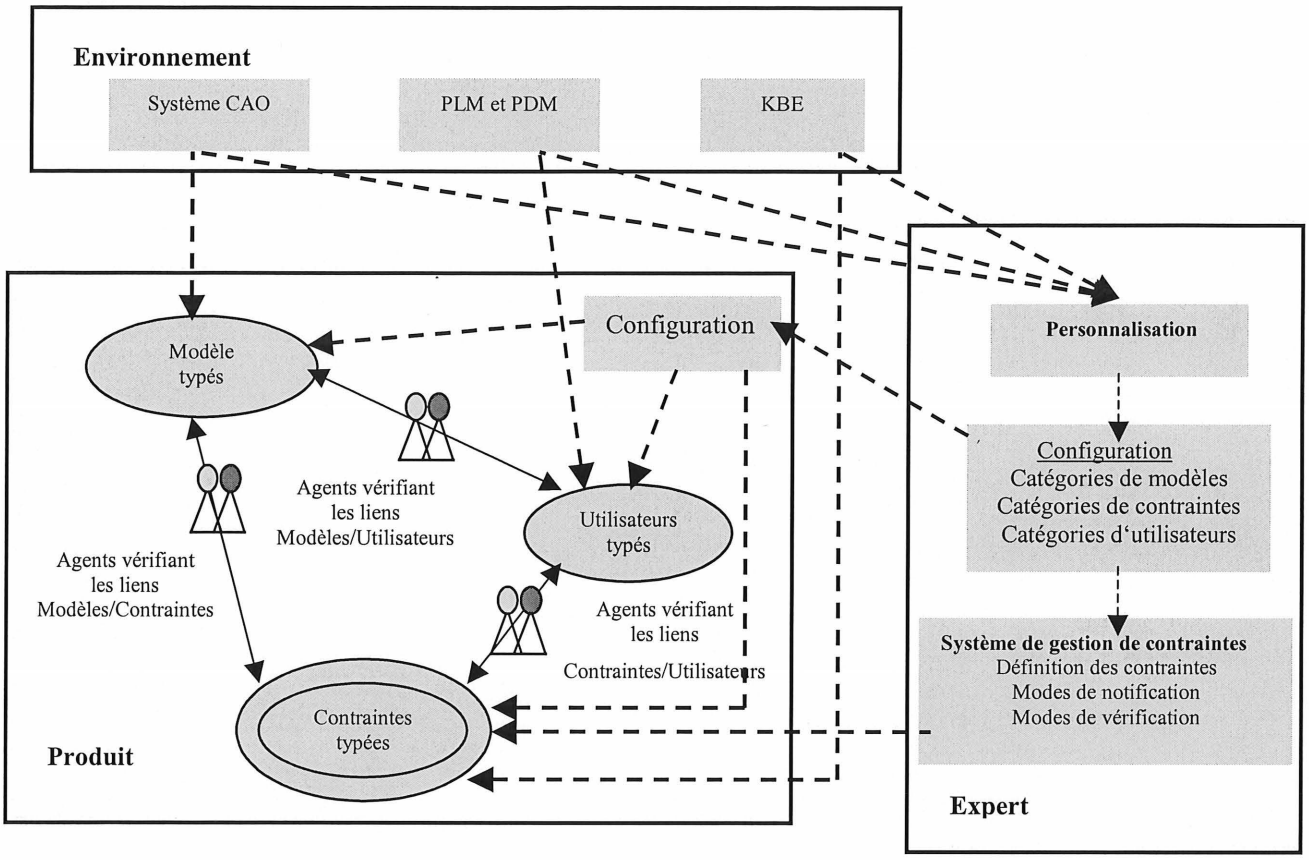


Figure 4.3 : Architecture globale du système multi-agents proposé

4.2.1. Section environnement

La « section environnement » contient des logiciels déjà existants à partir desquels nos agents vont extraire les données dont ils ont besoin. Ces différents systèmes sont présentés comme suit :

4.2.1.1. Système de Conception Assistée par Ordinateur (CAO)

Ce système fournit aux agents toutes les informations géométriques, topologiques et même physiques (matériel, masse, moment d'inertie...) des modèles CAO. Le système à base d'agents que nous avons développé utilise aussi des fonctions du système CAO que nous utilisons (CATIA[1]) et ce, à travers son API (« Application Programming Interface ») CAA[2].

4.2.1.2. Système PLM (Product Lifecycle Management), PDM (Product Data Management)

Ces systèmes fournissent des informations concernant le produit et ses configurations multiples à travers le processus de conception. Toutefois, nos agents logiciels vont s'intéresser uniquement aux données relatives à la gestion des utilisateurs. Les vérifications des contraintes doivent être envoyées à la personne ou au département adéquat et le choix de ce dernier peut dépendre de l'état du produit; par exemple, s'il est en WIP (Work In Progress) le résultat de la vérification peut être envoyé aux concepteurs, mais si par contre il est REL (Released for production) le résultat doit être envoyé au chef de projet.

4.2.1.3. Système KBE (Knowledge-Based Enterprises)

Ce système s'occupe de la gestion des connaissances ou des contraintes pour une organisation donnée. Nos agents peuvent alors s'occuper de la vérification de ces contraintes. Toutefois ce système n'est pas vraiment nécessaire pour nous, puisque nous avons développé notre propre système de gestion de contraintes. Nous tenons à le représenter quand même dans notre architecture pour montrer l'interaction que pourrait avoir notre système avec d'autres systèmes KBE.

4.2.2. Section Expert

La « section expert » agit comme « une interface de communication » entre les différents modules du système. Elle permet de configurer notre système selon les différents systèmes extérieurs que nous pourrions utiliser et de définir tout ce dont a besoin notre système pour qu'il soit parfaitement fonctionnel dans une organisation.

1 : Est un produit de Dassault Systemes
2 : Est un produit de Dassault Systemes

4.2.2.1. Personnalisation

Il s'agit d'un module générique de notre système qui fait la correspondance entre les attributs générés par les différents systèmes de gestion (usagers, contraintes, modèles) et les attributs manipulés par l'application, et ce, via le sous-module configuration. Il constitue l'interface entre les différents systèmes de CAO, PLM/PDM, KBE (section environement) et notre application (section produit). Ceci rend notre application plus flexible et plus indépendante des systèmes propriétaires. Si nous décidons de changer de plate-formes ou de logiciels nous n'aurions alors qu'à changer notre fichier de personnalisation créé au sein de ce module. On fait appel à ce module au moment de l'installation.

4.2.2.2. Configuration

C'est un module de notre système qui permet de définir les types des modèles, des contraintes et des utilisateurs. Les types des modèles, des contraintes et des utilisateurs sont spécifiés respectivement dans les attributs **Fonction, Type** et **Fonction utilisateurs.** Le module de configuration permet de définir toutes les valeurs possibles pour ces attributs en les organisant sous forme de structure d'arbre. Ces arbres vont être utilisés par le système multi-agents pour la vérification des contraintes. Le module de configuration est générique, il est instancié pour chaque projet.

4.2.2.3. Système de gestion de contraintes

Le système de gestion des contraintes est un module générique qui permet de gérer les contraintes et de définir les modalités de notification et de vérification. Il fait appel au module de configuration, et ce, pour récupérer les différents types de contraintes, d'usagers et de modèles. Les modes de vérification spécifient les conditions qui doivent êtres satisfaites avant que les agents ne commencent la vérification. Les modes de notification, quant à eux, indiquent les personnes à notifier dans le cas d'une violation de contraintes lorsqu'une notification automatique (qui dépend du statut du produit) ne serait pas utilisée.

4.2.3. Section produit

La « section produit » est considérée comme étant l'environnement de travail. C'est à ce niveau là que les différents projets, les produits ainsi que le processus de conception sont définis. C'est en fait le domaine d'action des agents logiciels.

4.2.3.1. Configuration

Le module de configuration est une instance du module « configuration » de la « section expert ». Ceci, pour mettre à la disposition de chaque projet, des informations de configuration. Lors du lancement d'un projet, le chef de projet ajoute une nouvelle liste contenant les différentes catégories de contraintes, d'utilisateurs et de modèles qui appartiennent spécifiquement à ce projet.

4.2.3.2. Modèles typés

La liste des modèles typés est fournie par le système de CAO et ne fait pas partie de l'étendue de notre projet dans la mesure ou ces types doivent être fournis par les concepteurs du projet ou bien par le chef du projet. Une fois que les modèles sont typés, ils peuvent être vérifiés par le système multi-agents.

4.2.3.3. Contraintes typées

Ce module est une instance du module « gestion des contraintes » dans la « section expert ». Seules les contraintes qui concernent le projet en cours sont prises en considération dans ce module et sont vérifiées par le système multi-agents.

4.2.3.4. Utilisateurs typés

La liste des usagers typés est fournie par un système de PLM, PDM ou SGDT, et ne fait pas non plus partie de l'étendue de notre projet. En général, les types fournis reflètent la hiérarchie de l'organisation. Une fois que tous les utilisateurs sont typés, ils peuvent, à ce moment là, être notifiés en cas de violation de contraintes.

4.2.4. Le flux d'information

Dans l'architecture du système présenté, nous montrons aussi le flux d'information circulant entre les différents modules et dans chaque module. Certains flux sont représentés avec des lignes continues (**Catégorie1**) et d'autres sont représentés par des lignes pointillées (**Catégorie2**) et ce, pour souligner la différence entre ces deux flux.

Catégorie1 : Ce flux représente les liens inter-modules au sein du processus de conception. Dans la « section produit », ces liens relient les modules « contraintes typées », « modèles typés » et « utilisateurs typés ». Ils sont générés et maintenus par le système multi-agents. En effet, les agents examinent les contraintes et déterminent les catégories de modèles qui interviennent dans ces contraintes, les modèles réels correspondant à ces catégories de modèles, ainsi que les types d'utilisateurs qui doivent être notifiés. Les différents liens sont ainsi générés. Toutefois ces liens doivent aussi être maintenus dans la mesure où, si un modèle change de statut ou si d'autres utilisateurs sont ajoutés, les agents de vérification et de notification doivent prendre ces changements en considération. En externe, les lignes du flux relient la « section produit » et la « section environnement » indiquant que tous les modèles, les contraintes et les utilisateurs peuvent être récupérés respectivement à partir des systèmes de CAO, KBE et PDM.

Catégorie2 : Le flux de la deuxième catégorie représente les différents liens qui s'établissent lors de l'installation et de la configuration du système. Dans la « section expert », le module de personnalisation indique au module de configuration où il peut trouver les informations sur les modèles et les utilisateurs. Le module de configuration informe à son tour le module de gestion des contraintes, des catégories d'utilisateurs et de modèles dont il dispose. Ces informations s'avèrent utiles lors de la définition des contraintes. Dans la « section produit », le flux d'information indique le vocabulaire de configuration qui est rendu disponible dans un projet par le module de configuration. En externe, le flux d'information relie le module de configuration de la « section expert » à la « section environnement » ce qui rend notre système complètement indépendant des systèmes de CAO, PDM et KBE utilisés. Enfin ce type de lien relie aussi le module de gestion des contraintes de la « section expert » au module des

contraintes typées de la « section produit »; ce lien indique le transfert des contraintes entre les deux modules.

4.3. Déploiement des agents

Nous allons présenter dans cette section le déploiement de nos agents logiciels afin de vérifier les contraintes du cahier des charges.

4.3.1 Attributs des agents

Les attributs des agents proposés dans cette section découlent des attributs issus des stratégies présentées dans la section 4.1.2. Ces attributs sont classés dans le tableau 4.5 :

Attribut	Valeurs de l'attribut	Nombre de valeurs de l'attribut
Nom	Agent « nom de la stratégie associée »	1
Rang	Le rang de l'agent	0 ou 1
Structure	Agent seul ou Agent en groupe	1
Collaboration	Agent coordonnateur ou Agent non coordonnateur	1
Déclenchement	Agent hors ligne ou Agent en ligne	1
Fonctionnement	- Agent de vérification : Agent modèle ou Agent contrainte - Agent de classification	1

Tableau 4.5 : Les attributs des agents

Nom : Le nom de l'agent est composé du nom de la stratégie associée plus un nombre qui représente le numéro de l'agent dans le cas ou il y a plusieurs agents.

Rang : Est un entier représentant le numéro de l'agent crée dans le cas ou il y a plusieurs agents dans le système.

Structure : L'attribut défini dans cette section donne une idée sur le nombre d'agents utilisés dans une application; les deux valeurs possibles sont mutuellement exclusives :

«Agent Seul » : Ce type indique qu'un seul agent est crée pour s'occuper de la vérification d'une contrainte donnée. Cependant cela ne veut pas dire pour autant qu'il n'y ait qu'un seul agent dans le système. Nous pouvons effectivement avoir plusieurs agents qui ont comme type « Agent seul » au sein du système, mais chacun s'occupe de vérifier ses contraintes sans communiquer avec les autres agents.

« Agent en groupe » : Ce type nous informe que l'agent qui vérifie une contrainte donnée travaille en groupe et non tout seul. Il s'occupe d'une partie des calculs prévue pour la contrainte et partage ses résultats avec les autres agents du groupe.

Collaboration : Cette section indique le rôle de l'agent au sein du système, ces agents ne sont pas mutuellement exclusifs; on peut avoir dans le système un agent coordonnateur et plusieurs agents non coordonnateurs :

« Agent non coordonnateur » : Désigne un agent qui s'occupe de calculer une partie de la contrainte et remet le résultat de ses calculs à l'agent coordonnateur.

« Agent coordonnateur » : Désigne un agent qui s'occupe de la récupération des résultats intermédiaires effectués par les agents non coordonnateurs et se charge de fournir le résultat final.

Déclenchement : Les agents de cette section reflètent le type de déclenchement de l'agent; ils sont mutuellement exclusifs :

«Agent hors ligne» : Ceci veut dire que la vérification ne sera déclenchée que si la machine n'est pas utilisée par un concepteur. Cela permet de ne pas déranger le concepteur lors de son travail.

« Agent en ligne » : Cet agent peut faire la vérification pendant que la machine est utilisée. Ce type d'agent n'est pas vraiment recommandable en tout temps; il peut être plutôt utilisé en cas d'urgence.

Fonctionnement : Dans cette section, les agents sont caractérisés par le type de vérification qu'ils font, soit par contrainte ou par modèle; ils sont mutuellement exclusifs :

« Agent modèle » : Cet agent cherche à optimiser son fonctionnement. Il vérifie toutes les contraintes qui impliquent un modèle donné.

« Agent contrainte» : Cet agent vérifie une contrainte à la suite de l'autre sur tous les modèles concernés sans se soucier de l'optimisation.

« Agent de classification » : C'est un agent qui s'occupe de la classification des contraintes selon leurs stratégies et leurs priorité. C'est lui qui s'occupe de la création des agents de vérification (modèle et contrainte)

Les attributs des agents constituent une sorte de carte d'identité pour l'agent.

4.3.2. Stratégie globale de vérification

Une fois que les contraintes sont sélectionnées et que la vérification est déclenchée par l'utilisateur, un agent de classification est crée. Il forme à partir de l'ensemble des contraintes sélectionnées, des classes de contraintes de même priorité. Pour chaque classe de même priorité, notre agent sectionne d'abord les contraintes légères. Pour cet ensemble, il forme des classes de contraintes de même stratégie. Pour chaque classe de même stratégie, il crée des classes de même nom de stratégie. L'agent de classification crée par la suite un agent ou des agents de vérification. Il associe à chaque agent une carte d'identité selon les attributs de la stratégie associée aux contraintes (tableau 5.1). Les agents vérifient alors l'ensemble des contraintes qui leurs sont fournies. Une fois que toutes les contraintes légères sont vérifiées, l'agent de classification s'occupe des contraintes lourdes, il forme des classes de contraintes

lourdes de même stratégie et puis de même nom de stratégie et crée un groupe d'agents pour les vérifier de la même manière que les contraintes légères (figure 4.4).

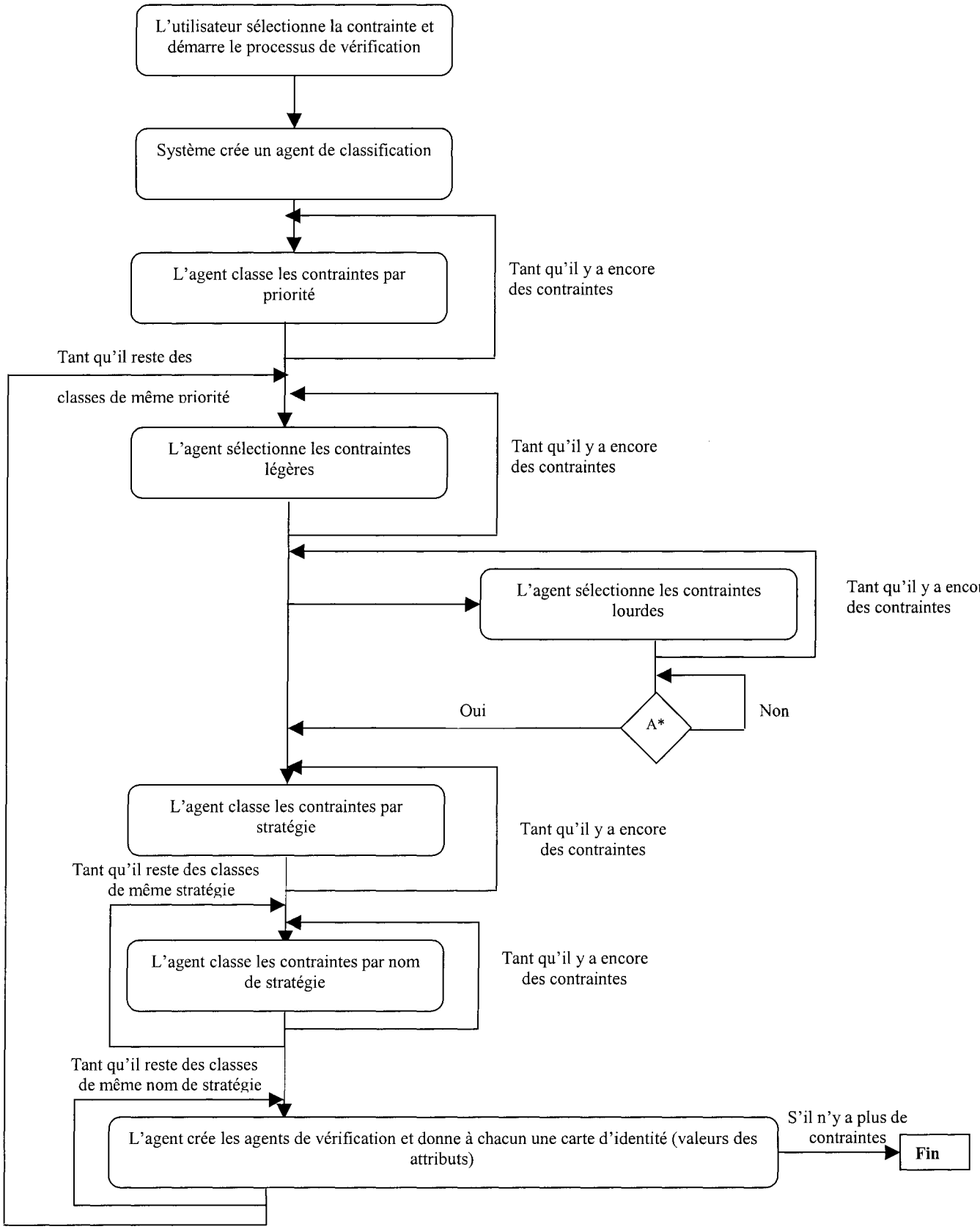


Figure 4.4 : Algorithme global de vérification des contraintes

A* : s'il ne reste plus de contraintes légères non vérifiées.

CHAPITRE 5

APPLICATIONS TYPES DE L'ARCHITECTURE PROPOSÉE

5.1. Introduction

Comme nous le savons, de nos jours, le domaine d'activité des entreprises qui travaillent dans le domaine de la CAO en mécanique est assez varié. Le domaine d'activité des agents serait alors assez grand. Pour bien maîtriser les calculs à faire par les agents, nous définissons des stratégies de vérification pour chaque groupe de contraintes associé à un métier donné. Cela donne naissance à des agents spécialisés pour chaque groupe.

Dans cette partie, nous allons présenter les applications types pour notre architecture définie dans le chapitre qui précède. Tout d'abord nous présentons un tableau des spécifications des attributs des agents qui permet de montrer comment nous avons choisi nos agents pour chaque application présentée. Le choix des agents dépend fortement des valeurs des attributs de la stratégie associée aux contraintes à vérifier (tableau 5.1). Il est clair que notre architecture peut être appliquée à plusieurs contextes, mais nous n'en retenons qu'un seul pour la phase d'implémentation qui suit.

5.2 Le choix des attributs des agents

Le tableau 5.1 nous montre comment choisir les valeurs des attributs des agents de vérification à créer selon les attributs de la stratégie associée aux contraintes à vérifier : fonctionnement, déclenchement, structure et collaboration. Pour chaque application présenter un agent de classification est toujours créé au départ pour classer les contraintes et ainsi permettre aux agents de vérification de faire leurs travails correctement

		Par modèle				Par contrainte			
		Parallélisable		Non-parallélisable		Parallélisable		Non-parallélisable	
	Hors ligne	Nombre de Pièces élevé	Nombre de Pièces faible	Nombre de Pièces élevé	Nombre de Pièces faible	Nombre de Pièces élevé	Nombre de Pièces faible	Nombre de Pièces élevé	Nombre de Pièces faible
Léger		Agent modèle Agent hors ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent modèle Agent hors ligne Agent seul Null	Agent modèle Agent hors ligne Agent seul Null	Agent modèle Agent hors ligne Agent seul Null	Agent contrainte Agent hors ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent contrainte Agent hors ligne Agent seul Null	Agent contrainte Agent hors ligne Agent seul Null	Agent contrainte Agent hors ligne Agent seul Null
	En ligne	Agent modèle Agent en ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent modèle Agent en ligne Agent seul Null	Agent modèle Agent en ligne Agent seul Null	Agent modèle Agent en ligne Agent seul Null	Agent contrainte Agent en ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent contrainte Agent en ligne Agent seul Null	Agent contrainte Agent en ligne Agent seul Null	Agent contrainte Agent en ligne Agent seul Null
Lourd	**Hors ligne**	Agent modèle Agent hors ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent modèle Agent hors ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent modèle Agent hors ligne Agent seul Null	Agent modèle Agent hors ligne Agent seul Null	Agent contrainte Agent hors ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent contrainte Agent hors ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent contrainte Agent hors ligne Agent seul Null	Agent contrainte Agent hors ligne Agent seul Null
	En ligne	Agent modèle Agent en ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent modèle Agent en ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent modèle Agent en ligne Agent seul Null	Agent modèle Agent en ligne Agent seul Null	Agent contrainte Agent en ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent contrainte Agent en ligne Agent en groupe Agent coordonnateur **ou** Agent non coordonnateur	Agent contrainte Agent en ligne Agent seul Null	Agent contrainte Agent en ligne Agent seul Null

Tableau 5.1 : Spécifications des attributs des agents

5.3. Poids et centrage

Dans cette section nous expliquons comment les agents logiciels créés au sein de notre système pourrait vérifier des contraintes poids et centrage.

5.3.1 Calcul de la masse et du centre de gravité par les agents

Dans l'exemple de poids et centrage, les agents logiciels vont calculer le poids d'un sous-assemblage d'un avion et son centre de gravité.

Pour calculer la masse d'un sous-assemblage, les agents doivent récupérer la masse de chaque pièce qui a comme type le sous-assemblage à vérifier et aussi toutes les pièces qui ont comme type les fils du type père (figure 5.1). Les agents calculent par la suite la masse totale du sous-assemblage en faisant la somme de toutes les masses des pièces récupérées pour ensuite la comparer à la masse spécifiée dans le cahier des charges afin de vérifier si la contrainte est conforme aux spécifications ou pas.

Pour vérifier si le centre de gravité du sous-assemblage est bien conforme aux spécifications nous procédons pratiquement de la même manière; les agents récupèrent toutes les pièces qui ont comme type le sous assemblage à vérifier, ainsi que toutes les pièces qui ont comme type tous les fils de ce type. Puis ils calculent le centre de gravité total de la pièce en utilisant bien entendu les centres de gravité déjà calculés pour chaque pièce. Enfin nous comparons la position calculée du centre de gravité avec celle qui est spécifiée dans le cahier des charges.

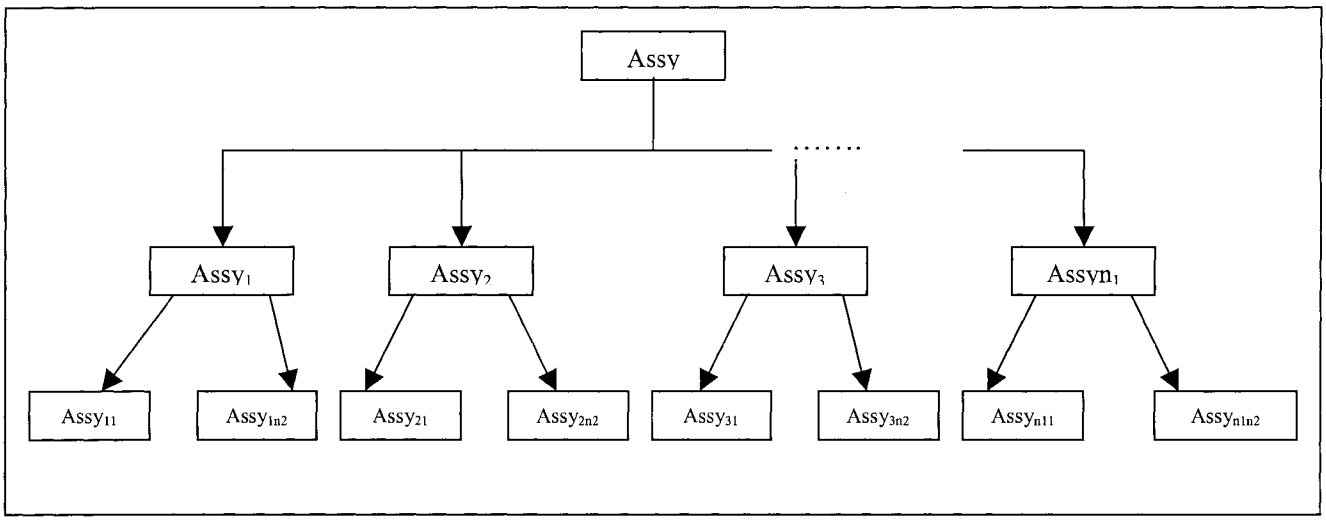


Figure 5.1 : structure d'un assemblage

$$Masse(Assy) = \sum_{i=1}^{n1} Masse(Assy_i) \qquad Masse(Assy_i) = \sum_{j=1}^{n2} Masse(Assy_{ij})$$

$$Centre(Assy) = 1/n_1 \sum_{i=1}^{n1} Centre(Assy_i) \qquad Centre(Assy_i) = 1/n_2 \sum_{j=1}^{ne} Centre(Assy_{ij})$$

5.3.2 Déploiement des agents : Déploiement poids et centrage

Les valeurs des attributs de la stratégie poids et centrage sont comme suit :

- Nom de la stratégie : Poids et centrage
- Type de calcul : Léger
- Fonctionnement : Par contrainte
- Déclenchement : Hors ligne
- Nombre de pièces : Élevé
- Parallélisme : Parallélisable
- Agent : Agent poids et centrage

Selon le tableau des spécifications des attributs des agents qui relie les attributs des stratégies et les types d'agents créés, la valeur des attributs des agents créés est :

- Nom : Agent poids et centrage
- Rang : Le numéro de l'agent dans le système multi-agents
- Structure : Agent en groupe
- Collaboration : Agent Coordonnateur ou Agent non coordonnateur
- Déclenchement : Agent hors ligne
- Fonctionnement : Agent par contrainte

Lors du déclenchement des vérifications, il y aura plusieurs agents de vérification dans le système, un agent coordonnateur et des agents non coordonnateurs. Puisque le calcul est parallélisable, les agents non coordonnateurs vont calculer les masses des sous-assemblages composant l'avion. Chacun d'eux effectue les calculs indépendamment des autres agents. L'agent coordonnateur coordonne les travaux et donne le résultat final.

Tous les attributs des contraintes sont définis par l'utilisateur lors de la définition de la contrainte. L'utilisateur peut aussi définir la stratégie la mieux adaptée au groupe de contraintes.

Quand la vérification est lancée, plusieurs agents stationnaires sont crées; un agent coordonnateur sur la machine de départ puis des agents non coordonnateurs sur chaque machine du réseau. Chaque agent non coordonnateur calculera la masse de toutes les pièces incluses dans l'avion et existante sur sa machine et donne le résultat à l'agent coordonnateur qui calcule le résultat final. Si l'agent coordonnateur se rend compte qu'il y a violation de contraintes, l'agent envoie immédiatement une notification aux personnes concernées qui se trouvent être dans l'attribut collaboration (figure 5.2).

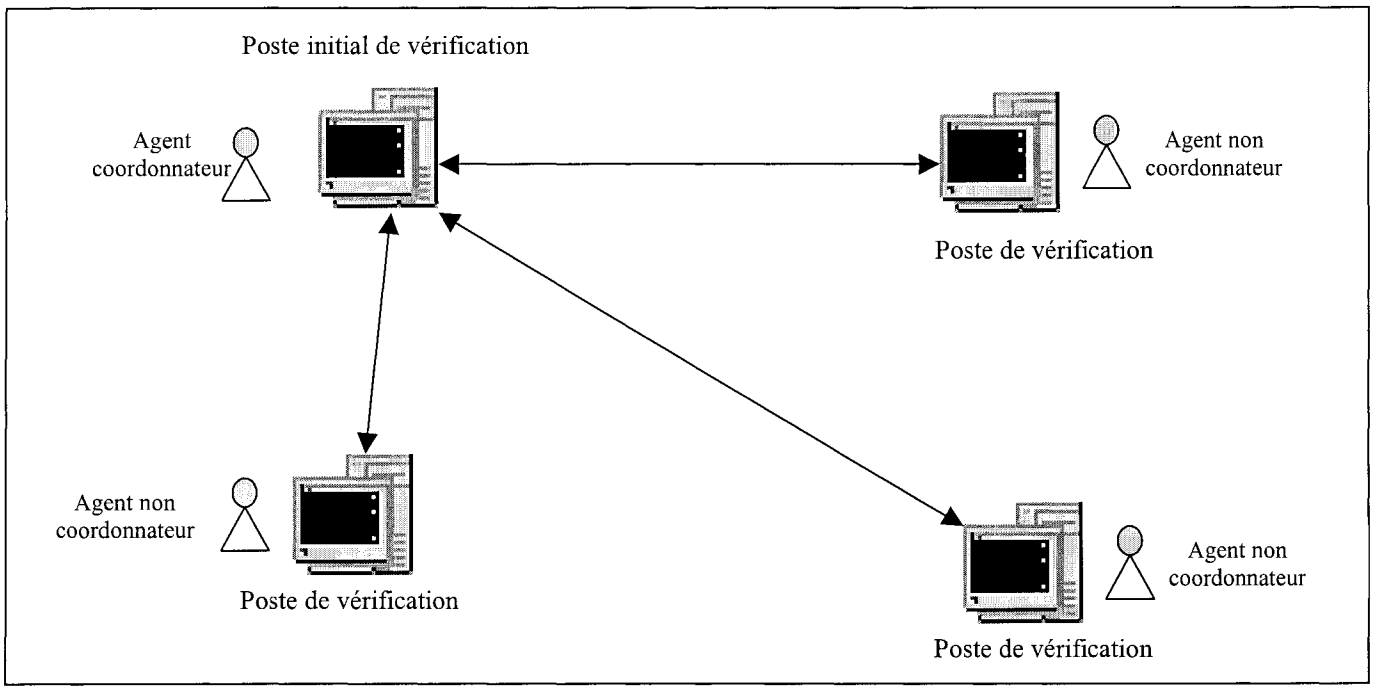


Figure 5.2 : Déploiement poids et centrage

5.4. Volume d'un sous assemblage

5.4.1. Calcul du volume d'un sous assemblage

Pour calculer le volume d'un sous-assemblage, l'agent de vérification récupère le volume de chaque pièce du sous-assemblage et fait la somme.

$$Volume(Assy) = \sum_{i=1}^{n1} Volume(Assy_i) \qquad Volume(Assy_i) = \sum_{j=1}^{n2} Volume(Assy_{ij})$$

5.4.2. Déploiement des agents : Déploiement volume

Les valeurs des attributs que nous proposons pour la stratégie volume sont:

- Nom de la stratégie : Volume
- Type de calcul : Léger
- Fonctionnement : Par contrainte
- Déclenchement : Hors ligne

- Nombre de pièces : faible
- Parallélisme : Parallélisable
- Agent : Agent volume

Les valeurs des attributs de l'agent créé pour la vérification sont donc comme suit :

- Nom : Agent volume
- Rang : 1
- Structure : Agent seul
- Collaboration : Nulle
- Déclenchement : Agent hors ligne
- Fonctionnement : Agent par contrainte

La stratégie utilisée pour la vérification des contraintes de volume est pratiquement la même que celle de poids et centrage à la différence que le nombre de pièces impliquées est plus faible. Donc nous avons besoin d'un seul agent pour la vérification.

5.5. Manufacturabilité

5.5.1. Calcul de la « manufacturabilité »

Pour calculer la « manufacturabilité » d'un sous assemblage, il faut calculer le rayon des congés de raccordement pour chaque pièce dans le sous assemblage et vérifier s'il existe des outils pour le fabriquer :

Manufacturabilité (sous-assemblage) = Manufacturabilité (pièces)

5.5.2. Déploiement des agents : Déploiement « Manufacturabilité »

Les attributs de la stratégie manufacturière que nous proposons sont :

- Nom de la stratégie : « Manufacturabilité »
- Type de calcul : Lourd
- Fonctionnement : Par modèle

- Déclenchement : Hors ligne
- Nombre de pièces : faible
- Parallélisme : Parallélisable
- Agent : Agent manufacturabilité

La valeur des attributs des agents créés pour la vérification selon le tableau des spécifications des attributs des agents (tableau 5.1) est :

- Nom : Agent manufacturabilité
- Rang : Le numéro de l'agent dans le système multi-agents
- Structure : Agent en groupe
- Collaboration : Agent coordonnateur ou non coordonnateur
- Déclenchement : Agent hors ligne
- Fonctionnement : Agent modèle

L'agent classificateur du système va créer autant d'agents qu'il n'y a de machines. Chaque agent va récupérer les pièces qui se trouvent dans sa machine, va faire ses calculs puis renvoie les résultats à l'agent coordonnateur qui se trouve dans le poste initial de vérification (Figure 5.2). Les autres agents sont considérés comme des agents non coordonnateurs.

5.6. Capacité d'un avion

5.6.1. Calcul de la charge utile de l'avion

Pour calculer charge utile de l'avion, c'est-à-dire sa capacité à supporter les passagers et leurs bagages, il faudrait que nos agents logiciels récupèrent d'abord la masse de l'avion chargé, qui est déjà prévue dès le départ et qui est stockée dans la base de donnée, pour y retrancher la masse de l'avion à vide. Nous avons déjà vu que le calcul de la masse totale d'un avion se fait en calculant la masse de chaque pièce le constituant (voir section 5.1).

$$\text{Capacité (avion)} = \text{Masse (charge)} - \text{Masse (vide)}$$

$$Masse(Assy) = \sum_{i=1}^{n1} Masse(Assy_i) \qquad \text{voir figure 5.1}$$

5.6.2. Déploiement des agents : Déploiement Capacité

Les attributs de la stratégie de charge utile sont :

- Nom de la stratégie : Capacité
- Type de calcul : Léger
- Fonctionnement : Par contrainte
- Déclenchement : Hors ligne
- Nombre de pièces : faible
- Parallélisme : Parallélisable
- Agent : Agent capacité

La valeur des attributs des agents créés pour cette application selon le tableau 5.1 est :

- Nom : Agent capacité
- Rang : Le numéro de l'agent dans le système multi-agents
- Structure : Agent en groupe
- Collaboration : Agent coordonnateur ou non coordonnateur
- Déclenchement : Agent hors ligne
- Fonctionnement : Agent par contrainte

Pour calculer la capacité ou charge utile de l'avion, le système considère toutes les pièces de l'avion pour faire les calculs; ce qui prend du temps. Ainsi, pour le calcul des contraintes de capacité, le système créé un nombre d'agents égal aux machines disponibles pour faire la vérification. Le déploiement capacité est exactement le même que le déploiement poids et centrage puisqu'il s'agit toujours de calcul de masse (figure 5.2).

CHAPITRE 6

RÉALISATION D'UN PROTOTYPE INFORMATIQUE

6.1. Introduction

Nous présentons dans cette partie, l'implémentation d'un prototype qui a pour mission de vérifier des contraintes de volume sur des sous-assemblages. Plus précisément, il vérifie si le volume de certains modèles CAO est bien conforme aux spécifications du cahier des charges. Le prototype est l'implémentation de l'application type «Volume d'un sous assemblage»

Les contraintes de volume sont vérifiées par des agents logiciels qui sont implémentés grâce à l'algorithme global de vérification présenté au chapitre 4. Nous présentons dans ce qui suit l'environnement de développement, l'architecture du prototype et son fonctionnement.

6.2 Environnement de développement

1) Langage Java

Pour la gestion des contraintes, des arbres, des attributs et des stratégies, nous avons utilisé le langage Java qui est parfaitement adapté à n'importe quel type de machine. Cette notion d'adaptabilité doit être reliée à la notion de portabilité assurée par la machine virtuelle Java (dans la mesure où celle-ci existe pour chaque type de machine).

2) Langage C++ et API CAA[1]

- **L'API CAA**

CAA[1] est l'API (Application Programming Interface) de CATIA. Elle est disponible en C++ et en Java et permet d'accéder à toutes les fonctions internes de CATIA à des fins de personnalisation. CAA permet donc de créer des librairies de partage que d'autres programmes en Java ou en Viusal Basic pourraient utiliser pour accéder aux fonctions CATIA.

1 : CAA est un produit développé par Dassault Systèmes

- **Développement sous CAA**

Pour développer la fonction calculant le volume d'un modèle CAO, nous avons utilisé le langage C++ sous l'environnement CAA. Bien que l'API CAA fut disponible en C++ et en Java, nous n'avons néanmoins pas vraiment eu le choix que d'utiliser celle en C++, puisqu'elle était la mieux documentée. La fonction que nous avons utilisée pour le calcul du volume d'une pièce est CATDynCreateMassProperties3D.

CATDynMassPropreties3D CATDynCreateMassProperties3D(const CATBody body)*

Cette fonction renvoie un pointeur sur la classe CATDynMassPropreties3D qui a une méthode getvolume() qui à son tour permet d'extraire le volume du corps analysé

3) Java Native Interface (JNI)

Pour assurer l'interface entre les modules Java que nous avons développés et le module C++, Nous avons utilisé les JNI (figure 6.1). Les JNI permettent au langage Java de communiquer avec les librairies conçues par d'autres langages de programmation tel C ou encore C++ en utilisant des méthodes natives. Le schéma suivant montre comment les JNI établissent le lien entre la partie C++ et la partie Java d'une application.

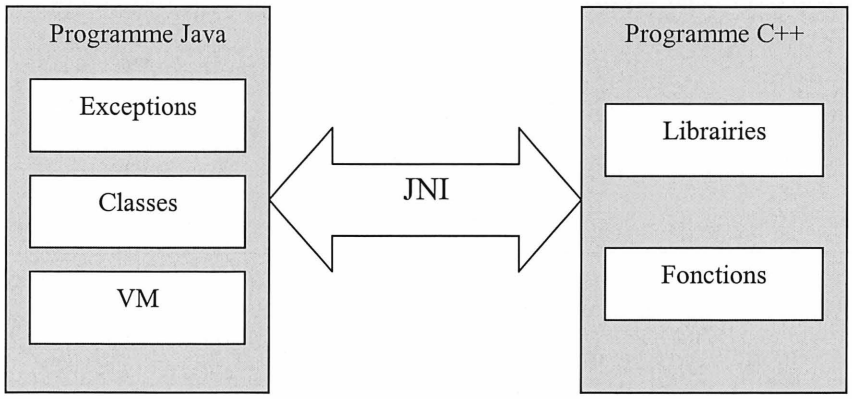


Figure 6.1 : Lien entre java et C++ avec les JNI

Dans notre travail, nous avons généré une DLL où nous avons intégré la fonction de calcul de volume, et ce à partir de notre programme C++ implémenté sous l'environnement CAA. Nous avons pu accéder à cette fonction dans le programme Java. Le diagramme à la figure 6.2 montre clairement la procédure suivie. Nous commençons premièrement par écrire et

compiler le code Java à partir duquel nous voulons accéder à la fonction C++ de calcul de volume. Par la suite nous générons, via une commande JNI, le fichier d'en-tête ou le « header file » qui sera utilisé par le programme C++. Nous implémentons ensuite la méthode native que nous voulons exposer et nous générons, à partir de ce programme, une DLL. En exécutant le programme Java, ce dernier charge la DLL puis accède à la méthode de calcul de volume.

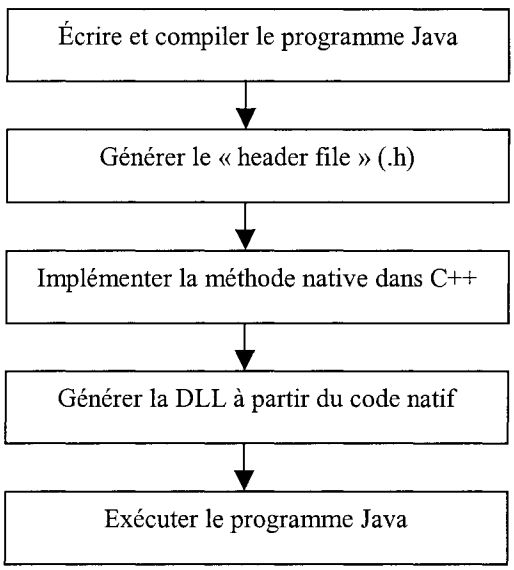


Figure 6.2 : Procédure suivie pour la réalisation de l'interface entre CAA et Java

4) XML(Extensible Markup Language)

Nous avons décidé d'utiliser le langage XML pour présenter les modèles CAO, les contraintes et même les utilisateurs de l'application. Les trois entités se présentent en fait sous forme d'arborescences, nous avons donc estimé que XML est le meilleur choix pour les modéliser. Nous avons développé un module dans l'application qui permet de représenter l'arborescence de chaque entité et de générer le fichier XML associé. Nous présentons dans la partie 6.4 l'exemple du modèle CAO que nous utilisons pour la validation du prototype représenté sous forme d'arborescence et le fichier XML généré.

5) Base de données Access

Nous avons utilisé une base de données Access pour stocker les attributs, les contraintes, les stratégies, les différentes pièces du projet avec leurs volumes et le résultat de l'interprétation des contraintes (figure 6.3).

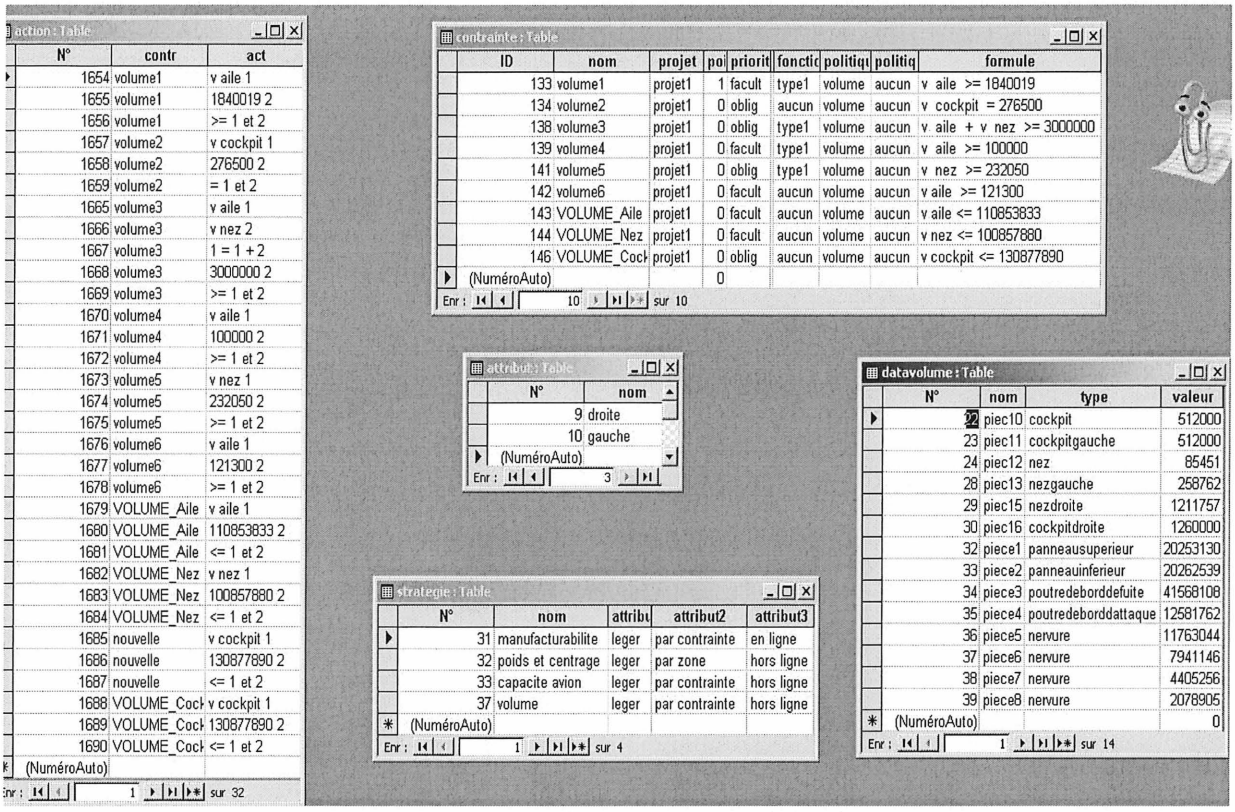


Figure 6.3 : Base de données Access

6.3. Architecture du prototype développé

a) Parties implémentées pour le prototype

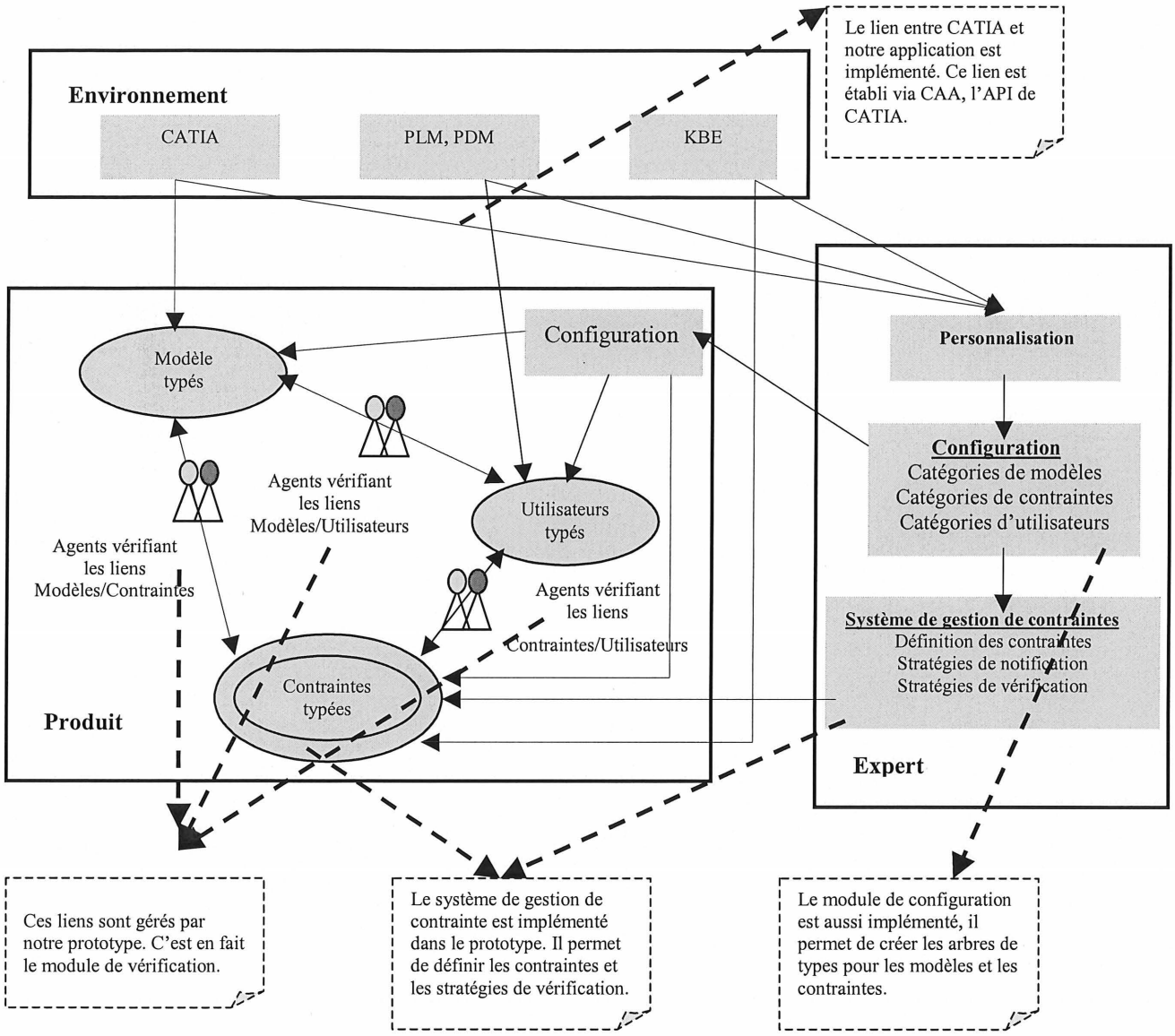


Figure 6.4 : Parties implémentées pour le prototype

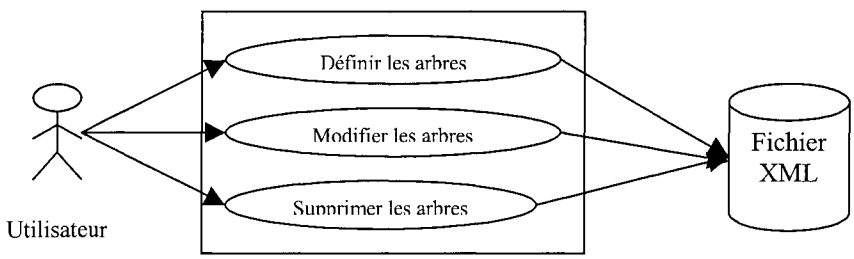
Le prototype de calcul du volume implémente les parties clés de l'architecture globale (figure 6.4). Dans la « section produit », l'algorithme de vérification détermine si telle contrainte liée à tels modèles est correcte ou pas. Dans le cas où la contrainte comporterait des erreurs, le système notifie les utilisateurs concernés. Dans la « section expert », la partie configuration défini les types de modèles et les types de contraintes. Le module de gestion des contraintes

défini et interprète nos contraintes et les stratégies de vérification. Nous n'avons pas implémenté la partie « personnalisation » car nous supposons que tous les types de modèles qui sont utilisés dans le calcul du volume et qui sont défini dans CATIA correspondent bien aux types définis dans notre application. Nous avons enfin implémenté l'interface entre la « section environnement » et la « section produit ». Ainsi, pour vérifier le volume d'un sous-assemblage, l'algorithme de vérification récupère le nom des modèles concernés par chaque contrainte à partir d'une base de données, puis il appelle la fonction CATIA de calcul de volume pour chaque modèle.

b) Diagrammes de cas d'utilisation et de séquence

Nous présenterons, dans ce qui suit, les diagrammes de séquences pour les différents modules de notre application.

1) Module de gestion des arbres de typage



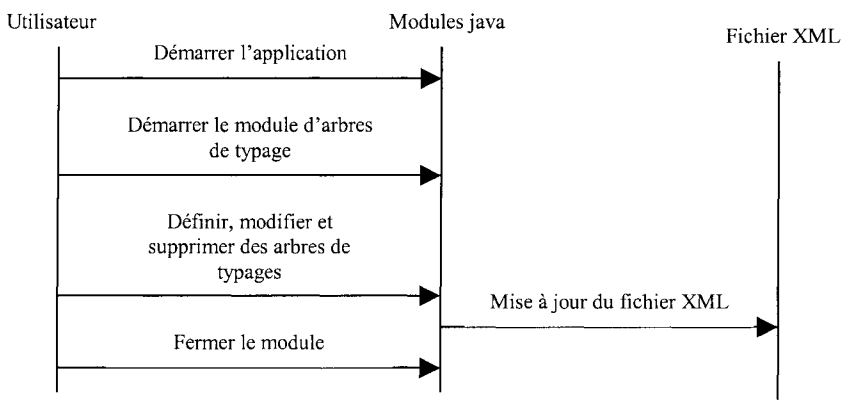


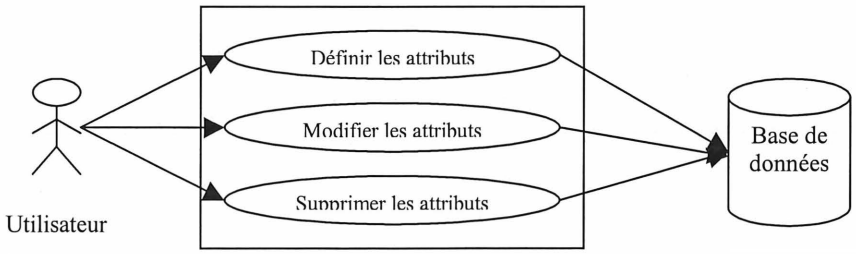
Figure 6.5 : Diagrammes d'utilisation et de séquence du module de gestion des arbres de typages

Ce module implémente la gestion des arbres de typage pour les modèles, les contraintes et les usagers (figure 6.5). Quand l'utilisateur démarre le module de gestion des arbres de typage, il a la possibilité de créer, modifier et supprimer des structures d'arbres. Tous les arbres sont stockés sous forme de fichiers XML (Extensible Markup Language) (figure 6.6). Ce module peut être utilisé par un utilisateur concepteur.


```
- <contrainte xmlns="personal.xsd">
    <typecontrainte xmlns="">mecanique</typecontrainte>
    <typecontrainte xmlns="">electrique</typecontrainte>
    <typecontrainte xmlns="">hydrolique</typecontrainte>
  </contrainte>
```

Figure 6.6 : Fichier XML pour les contraintes

2) Module de gestion des attributs



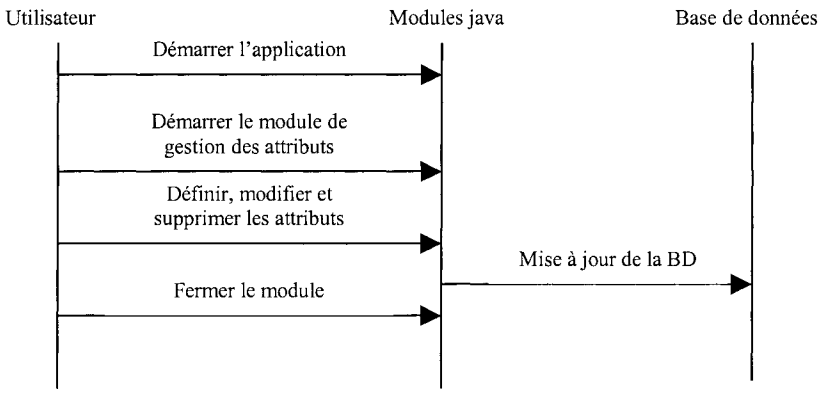


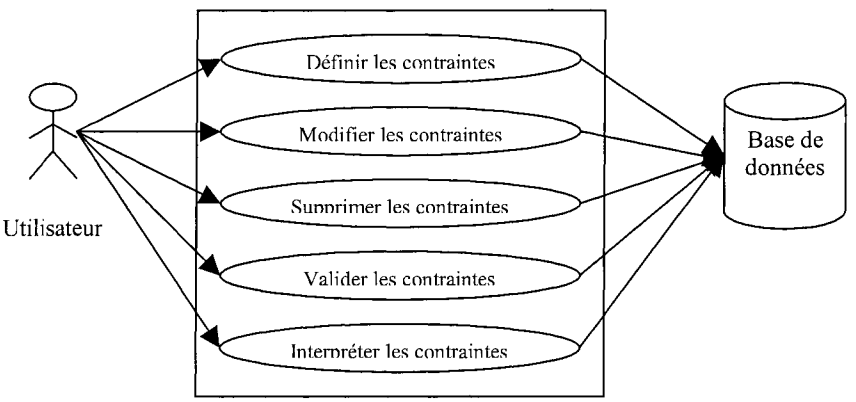
Figure 6.7 : Diagrammes d'utilisation et de séquence du module de gestion des attributs

Le module de gestion des attributs créé les attributs des types de modèles (figure 6.7). Ces attributs peuvent être utilisés pour la définition des contraintes. Nous pouvons donner comme exemple la contrainte suivante:

v aile droit >= 12

L'attribut dans ce cas est **droit**.

3) Module de gestion des contraintes



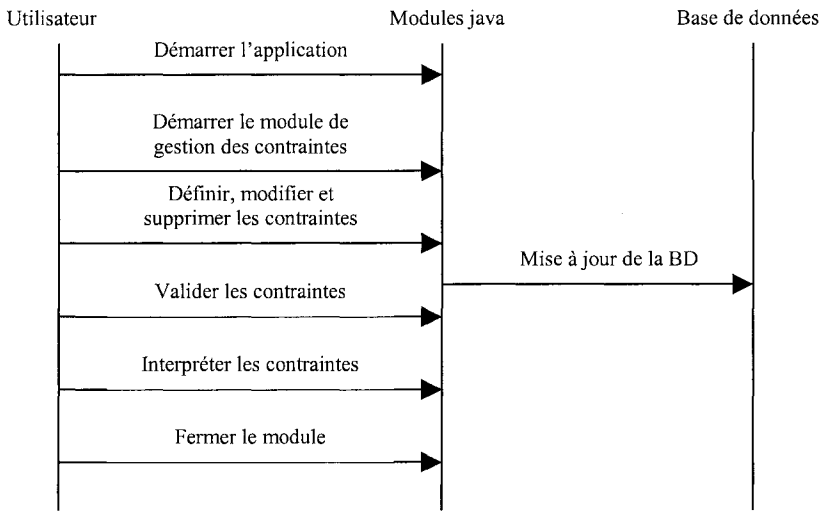


Figure 6.8 : Diagrammes d'utilisation et de séquence du module de gestion des contraintes

Ce module gère les contraintes : création, modification, enregistrement etc. Nous avons défini un langage pour exprimer nos contraintes (figure 6.8). Nous présentons dans ce qui suit la représentation BNF du langage :

Alphabet terminal

0..9, v, m, px, py, pz

+, -, *, /, <, <=, =>, >,#,=

f1, f2, f3, f4, f5, f6, f7

Alphabet non terminal

< contrainte>, < Expr>, < Opcomp>, < Chiffre>, < Exprsimp>, < f>, < Opmath>, < Constante>, < Motclé>

Règles

< contrainte> ::= < Expr> < Opcomp> < Exprcomp>

< Exprcomp> ::= < Expr> | < Chiffre>

< Expr> ::= < Exprsimp> | < Exprsimp > < f> < Exprsimp >

< Exprsimp> ::= < Chiffre> | < Opmath > < Constante> < Motclé >

< f> ::= <f1>| <f2>|<f3>|<f4>|<f5>|<f6>|<f7>

<Chiffre>::=[0..9]

<Opcomp>:: =< |<= | => | > | # | =

<Opmath>::= +| - | * | /

<Constante>::= v | m | px | py | pz

<Mot clé> ::= « Aile » | « Cockpit » | « nez » | « Aucuntype » | « Toutlestypes »

Une fois créée, la contrainte est stockée avec tous ses attributs dans une base de données. Dans ce module, il existe deux autres sous-modules; celui pour la validation et celui pour l'interprétation. Le sous-module de validation sert à valider la syntaxe des contraintes; il s'agit tout simplement d'un compilateur de contraintes. Le sous-module d'interprétation sert, quant à lui, à générer toutes les actions nécessaires pour faire la vérification des contraintes. Ces actions sont utilisées par la suite par les agents logiciels pour les intégrer à leurs plans.

4) Module de gestion des stratégies

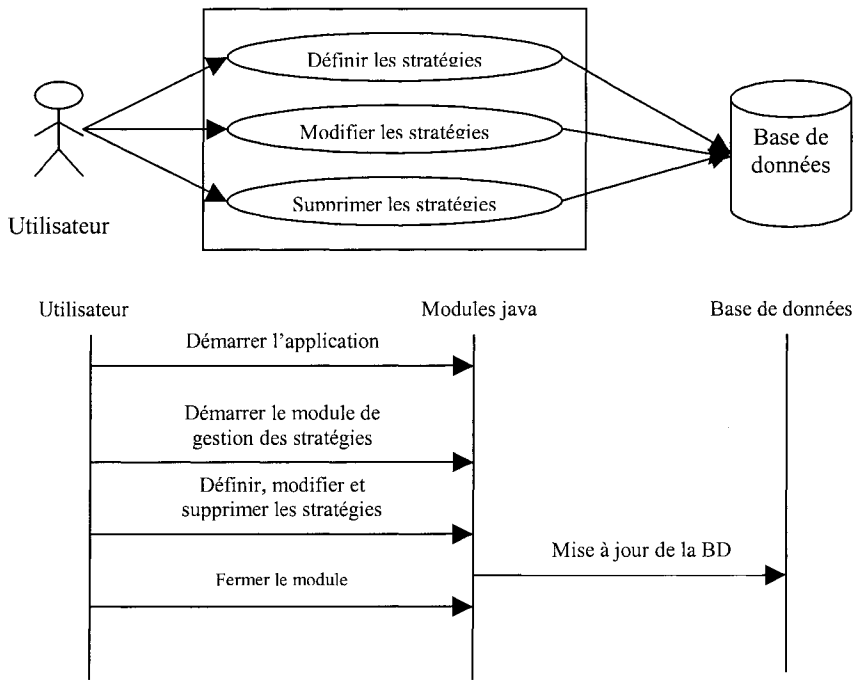


Figure 6.9 : Diagrammes d'utilisation et de séquence du module de gestion des utilisateurs

Ce module permet de créer des stratégies de vérification, si nécessaire. En effet, l'utilisateur pourrait avoir besoin d'une stratégie qui ne figure pas parmi les stratégies que nous avons définies; il aura donc la possibilité de la définir lui-même (figure 6.9).

5) Module de vérification

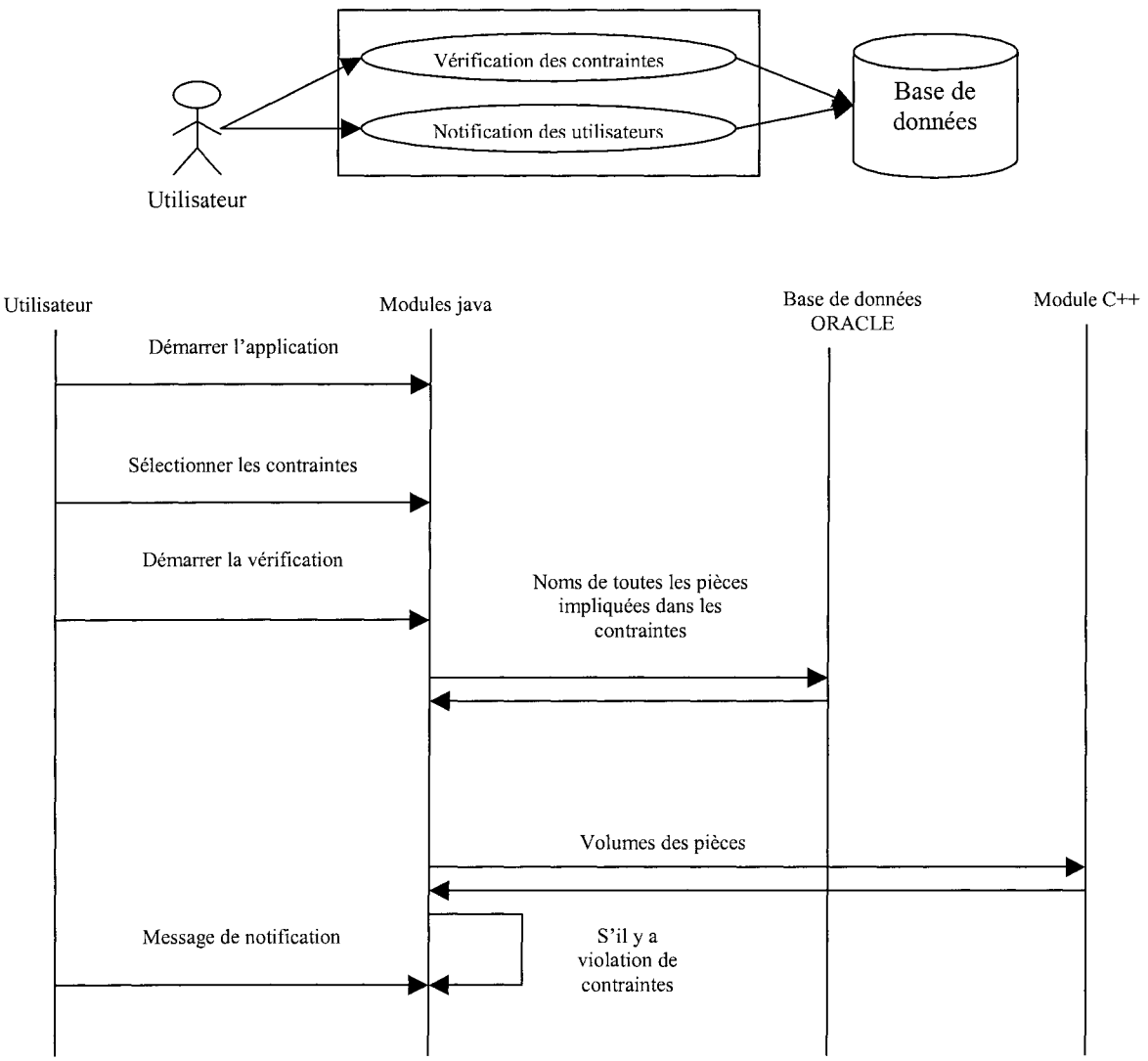


Figure 6.10 : Diagrammes d'utilisation et de séquence du module de vérification

Le module de vérification permet à son tour de vérifier les contraintes selon l'algorithme global de vérification (chapitre 4). Une fois que nous choisissons les contraintes à vérifier et que nous démarrons la vérification, les agents récupèrent les types mentionnés dans la formule de chaque contrainte et cherchent dans l'arbre XML des types de modèles, tous les fils des types mentionnés. Ensuite, ils vont chercher dans la base de donnés des modèles, tous les modèles qui ont comme type, le type mentionné dans la contrainte et ses fils. Les agents

communiquent par la suite avec le logiciel CATIA à travers CAA, ils invoquent la fonction de calcul de volume pour avoir les volumes des modèles récupérés de la base et effectuent enfin les calculs et déterminent si les contraintes sont bien respectées (figure 6.10).

6.4. Définition des données pour le prototype

a) Définition des arbres de typage

Les arbres de typage définissent une arborescence de types de modèles CAO. Ils sont représentés sous la forme d'un fichier XML. Le menu principal permet de créer un nouvel arbre (modèle, contrainte et usagers) ou de modifier un arbre déjà existant. Dans l'arbre XML, on peut insérer des nœuds ou en supprimer. Lors de l'insertion d'un nœud, nous pouvons spécifier ses attributs s'il en a. Nous avons aussi prévu la création d'arbre de types de contraintes et de types d'usagers, mais nous ne les utilisons pas dans l'étude de cas.

b) Définition des attributs

Notre application permet aussi de définir les attributs que nous pouvons associer aux types de modèles créés comme par exemple droit, gauche, avant, arrière.

c) Définition des stratégies

Le système permet de définir les stratégies de vérification qui sont le mieux adaptées aux contraintes. Lors de la définition d'une stratégie, nous spécifions les valeurs de ses différents attributs. Dans notre application nous définissons juste la stratégie « volume » vu que notre programme calcule essentiellement le volume des pièces « CATPART » mais dans une perspective de développements futurs, nous pouvons bien entendu définir d'autres stratégies comme par exemple la stratégie de masse ou encore la stratégie de poids et centrage etc.

d) Définition, validation et interprétation des contraintes

Le module de gestion des contraintes permet de définir des contraintes, de valider leur syntaxe, de les modifier ou de les supprimer. Ce module nous permet aussi d'interpréter la contrainte et générer toutes les actions qui vont être entreprises par les agents logiciels lors de la vérification. Lors de la définition d'une contrainte, nous la relions à un projet, nous lui

donnons un poids, une priorité, nous spécifions le type de la formule et la stratégie de vérification appropriée. Après avoir compilé et enregistré la contrainte, nous pourrons donc l'interpréter.

e) Scénario de vérification des contraintes

Le scénario de vérification des contraintes que nous avons défini respecte exactement l'algorithme global de vérification voir figure 4.4.

6.5. Validation du prototype et étude de cas

6.5.1. Introduction

Nous présentons, dans ce qui suit, une étude de cas qui illustre clairement l'utilisation de chaque partie de notre prototype en commençant par la création des contraintes à leur vérification en passant par toutes les fonctions de personnalisation. Ce scénario est illustré par des saisies d'écrans.

6.5.2. Scénario d'exécution

Nous présentons dans cette partie le plan d'exécution de notre prototype avec l'outil utilisé pour le développement de chaque étape. (figure 6.11)

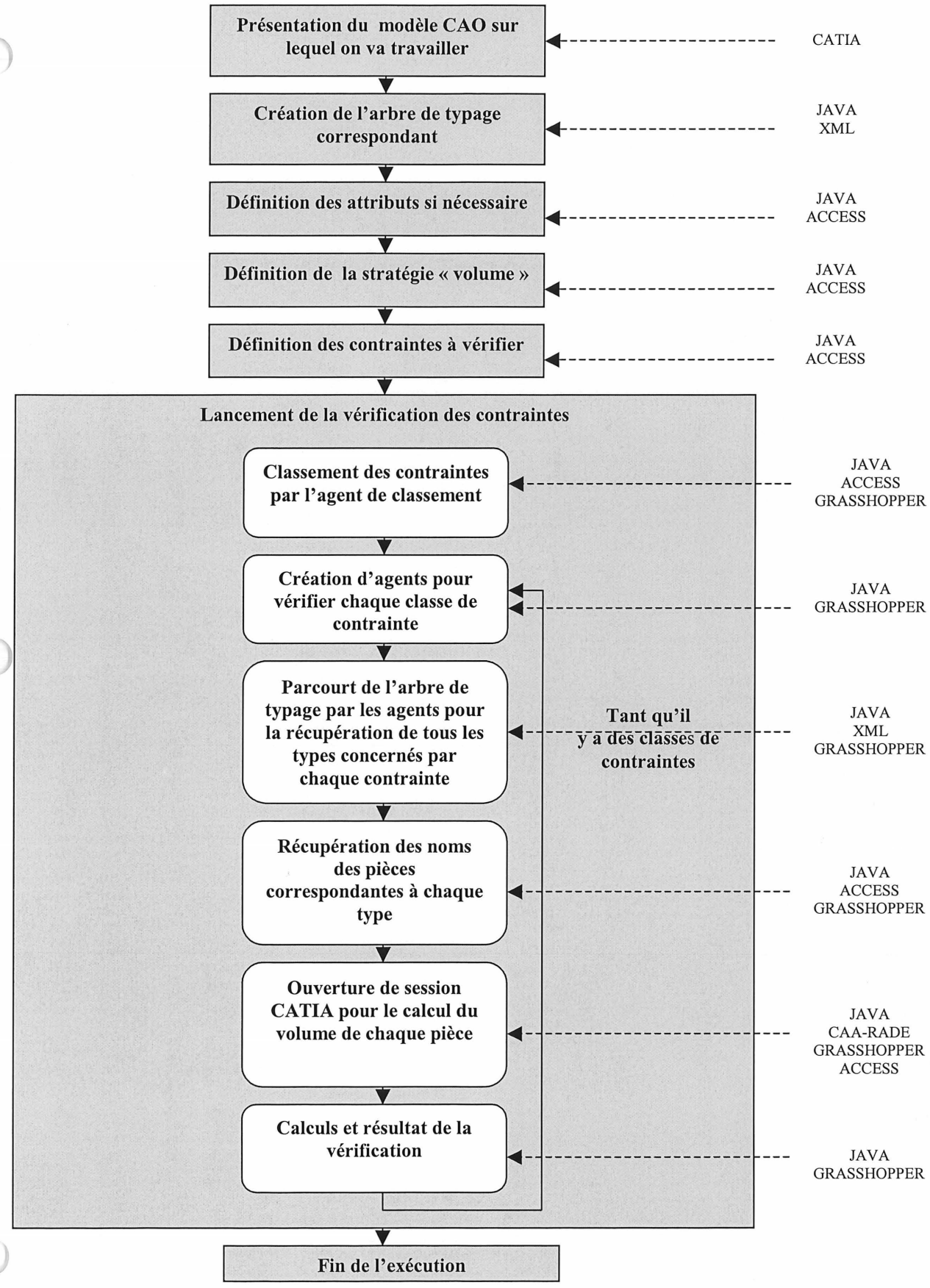


Figure 6.11 : Plan d'exécution et plates-formes de développement

a) Présentation du modèle CAO

Le modèle que nous proposons d'utiliser pour le prototype est l'aile d'un avion que nous avons conçue sur CATIA V5. (figure 6.12).

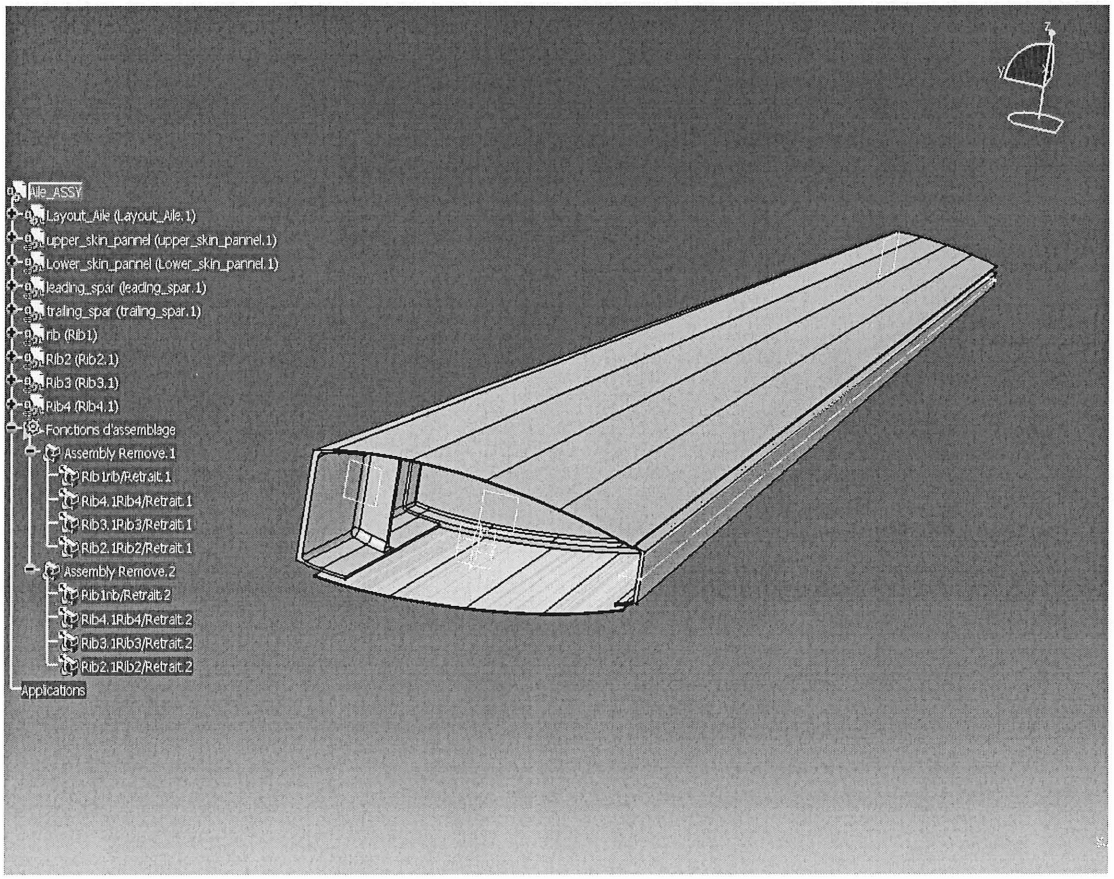


Figure 6.12 : L'aile d'un avion

L'aile de l'avion est composée en fait de huit pièces : Le panneau supérieur, le panneau inférieur, la poutre de bord de fuite, la poutre de bord d'attaque et quatre nervures.

b) Création d'un arbre de typage

Nous allons créer dans l'arbre de typage le nœud représentant le type du modèle représenté ci-dessus qui est en fait « Aile », ainsi que deux autres types que nous rajoutons « nez » et cockpit » pour pouvoir vérifier plus d'une contrainte (pour les types « nez » et « cockpit »

nous allons utiliser des modèles CAO que nous ne définirons pas dans cette section pour alléger le chapitre). (figure 6.13)

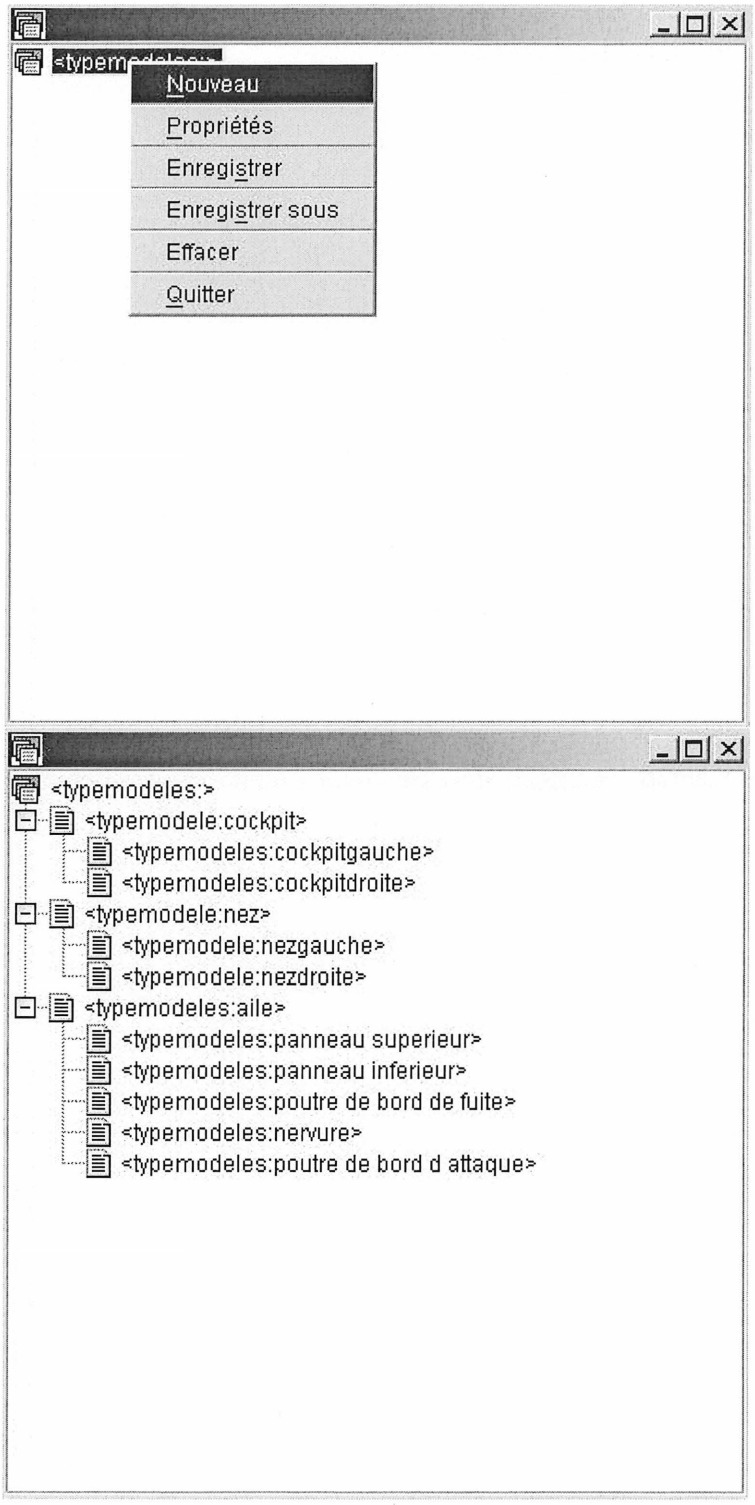


Figure 6.13 : L'arbre de typage

c) Création des attributs associés au modèle CAO proposé

Dans notre cas il n'y a aucun attribut qui se trouve être associé au type du modèle que nous avons choisi, nous présentons quand même l'interface qui permet la création des attributs.

(figure 6.14)

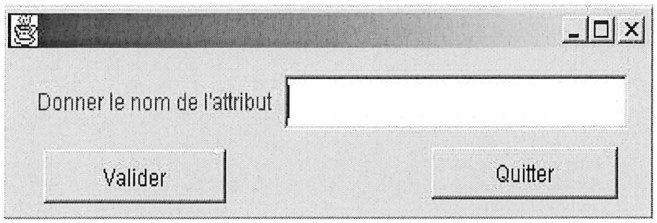


Figure 6.14 : Définition d'attributs

d) Définition de la stratégie « volume »

La stratégie « volume » que nous proposons de créer dans cette partie (figure 6.15), va être reliée à toutes les contraintes que nous allons définir dans la section d'après vu que l'application que nous avons développée sous CAA-Rade est une application de calcul de volume. Cette stratégie détermine en fait le déploiement approprié des agents pour la vérification de ces contraintes.

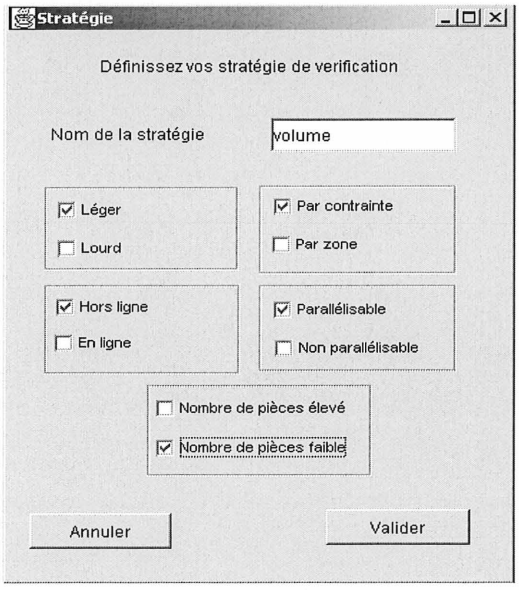


Figure 6.15 : Définition de la stratégie « volume »

Les attributs de la stratégie volume sont : Léger, Par contrainte, Hors ligne, Paralléllisable, Nombre de pièces faibles.

e) Définition des contraintes à vérifier

Nous proposons de définir trois contraintes; une contrainte sur le modèle de l'aile défini plus haut, une contrainte sur le nez de l'avion et une contrainte sur son cockpit. Les trois contraintes sont exprimées comme suit :

VOLUME_Aile : V aile <= 110853833 mm^3, Priorité : Facultative, Stratégie : Volume
VOLUME_Nez : V nez <= 100857880 mm^3 , Priorité : Facultative, Stratégie : Volume
VOLUME_Cockpit : V cockpit <= 130877890 mm^3 , Priorité : Obligatoire, Stratégie : Volume

Nous allons présenter dans la figure 6.16 l'interface de création des contraintes et le résultat de la compilation de la contrainte VOLUME_Aile en cliquant sur le bouton « Valider ». Une fois que la contrainte est enregistrée nous pourrons l'interpréter pour faciliter le processus de vérification par les agents. Pour les autres contraintes c'est exactement la même chose. En créant une contrainte il faut obligatoirement lui donner une priorité et la lier à une stratégie qui est dans notre cas la stratégie « volume ».

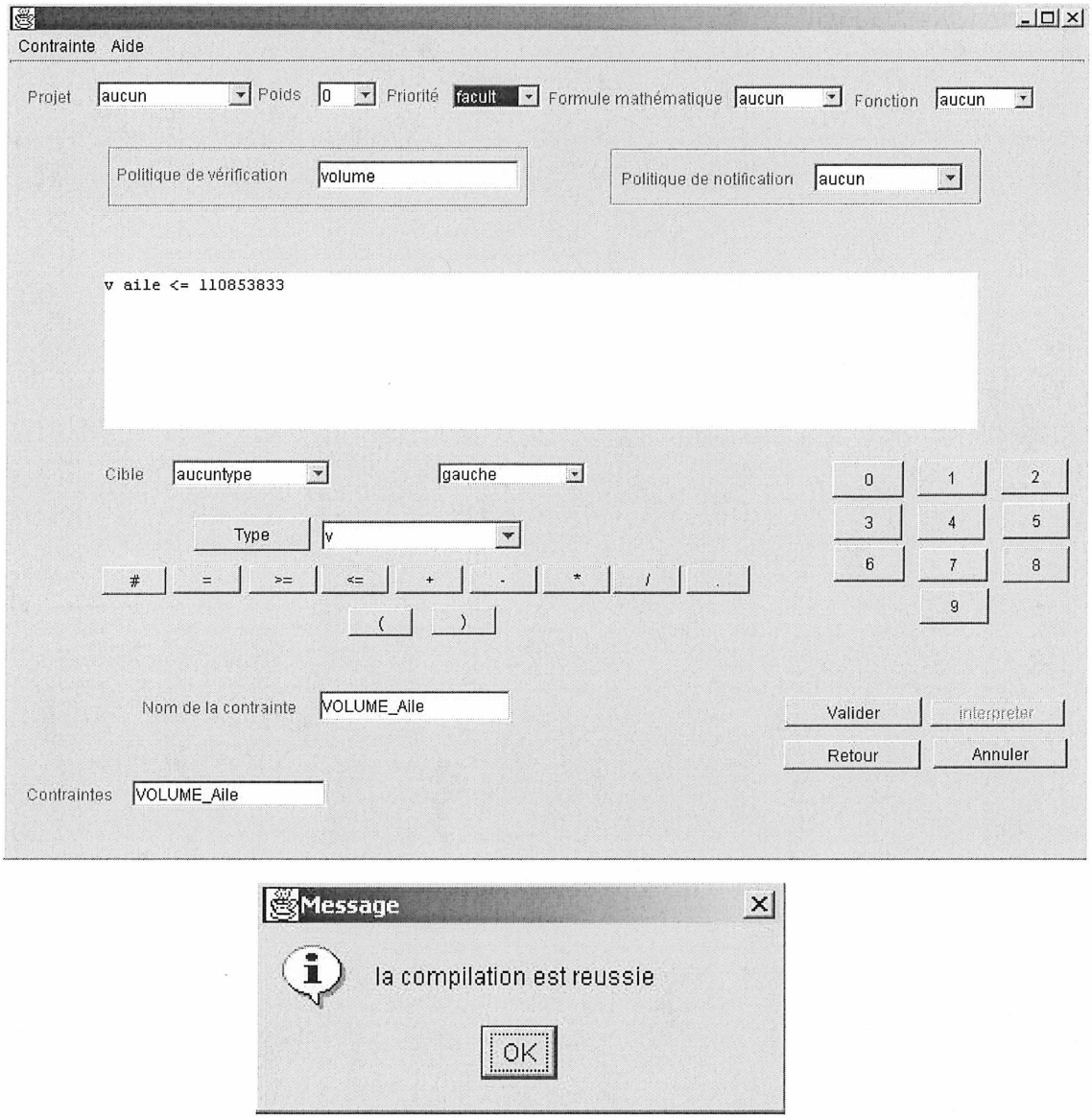


Figure 6.16 : Interface de la définition des contraintes et résultat de la compilation

f) Lancement de la vérification des contraintes

Nous présentant dans l'interface à la figure 6.17 le lancement de la vérification des trois contraintes définies plus haut.

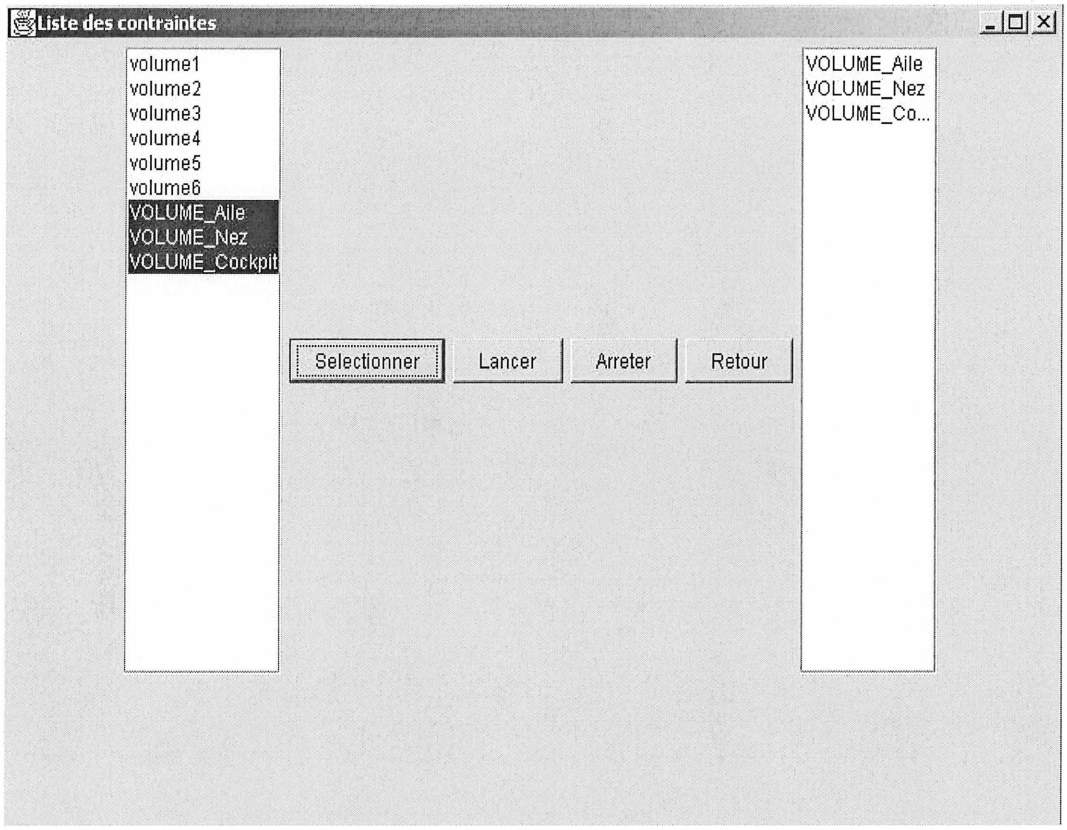
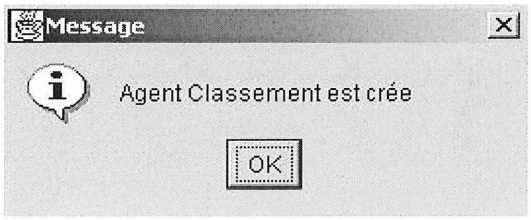


Figure 6.17 : Sélection et lancement de la vérification des trois contraintes définies plus haut

Une fois on clique sur le bouton « Lancer » la vérification est déclenchée.

1) Classement des contraintes par l'agent de classement

Un agent de classement est créé, il se nomme « CADAgent ». CADAgent classe les contraintes selon leurs priorités et leurs stratégies (figure 6.18). Dans notre cas, nous avons une seule stratégie, donc le classement va être fait selon la priorité de chaque contrainte.



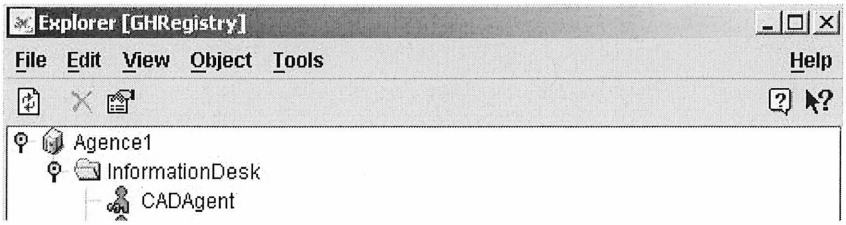


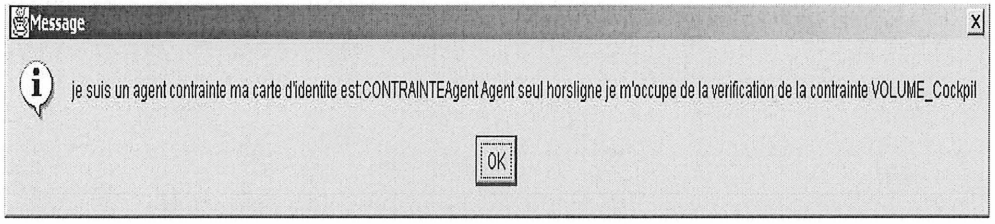
Figure 6.18 : Création de l'agent de classement

2) Procédure de vérification

- **Création d'un agent « Contrainte » pour la vérification de la contrainte VOLUME_Cockpit**

La contrainte VOLUME_Cockpit est obligatoire donc elle va être vérifiée en premier par un agent même si on l'a sélectionné en dernier, cette contrainte constitue à elle seule une classe.

Les deux contraintes VOLUME_Aile et VOLUME_Nez constituent une autre classe de contraintes puisqu'elles sont facultatives, donc elles vont être vérifiées par un autre agent par la suite. Nous présentons dans la figure 6.19 les saisies d'écrans qui montrent l'agent créé pour la première contrainte.



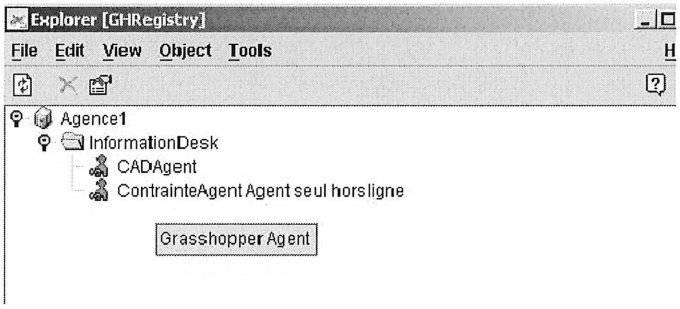


Figure 6.19 : Création d'un agent pour la vérification de la contrainte VOLUME_Cockpit

- **Parcours de l'arbre de typage et récupération des types concernés par la contrainte VOLUME_Cockpit**

L'agent parcourt l'arbre de typage défini plus haut pour récupérer les sous types du type « cockpit ». L'agent trouve : « cockpitgauche » et « cockpitdroite ». (figure 6.20)

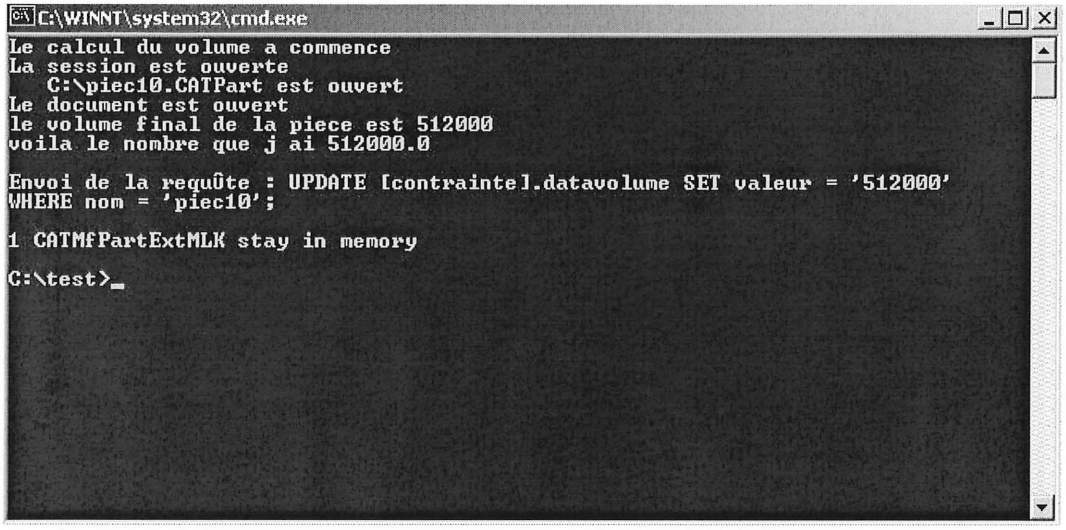
Figure 6.20 : Les sous types de Cockpit

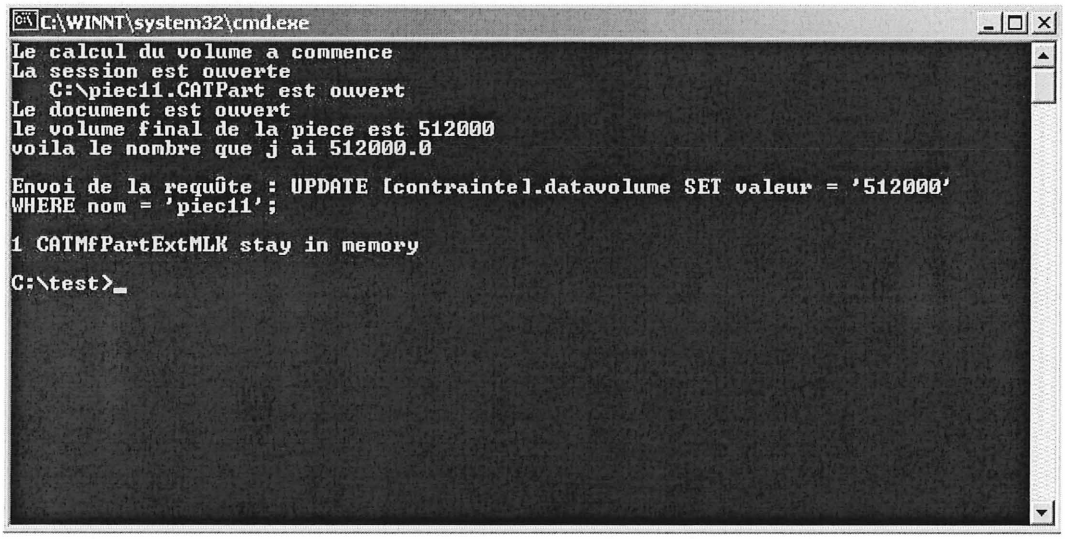
- **Récupération des noms des pièces de types : « cockpit », « cockpitgauche » et « cockpitdroite »**

L'agent se connecte à une base de donnée Access pour récupérer toutes les pièces de types « cockpit », « cockpitgauche » et « cockpitdroite ». L'agent trouve respectivement : piec10.CATPart, piec11.CATPart et piec16.CATPart.

- **Ouverture d'une session CATIA (en background) pour chaque pièce trouvée et récupération de son volume**

L'agent ouvre par la suite une session CATIA en « background » pour les trois pièces pour calculer leurs volumes. (figure 6.21)





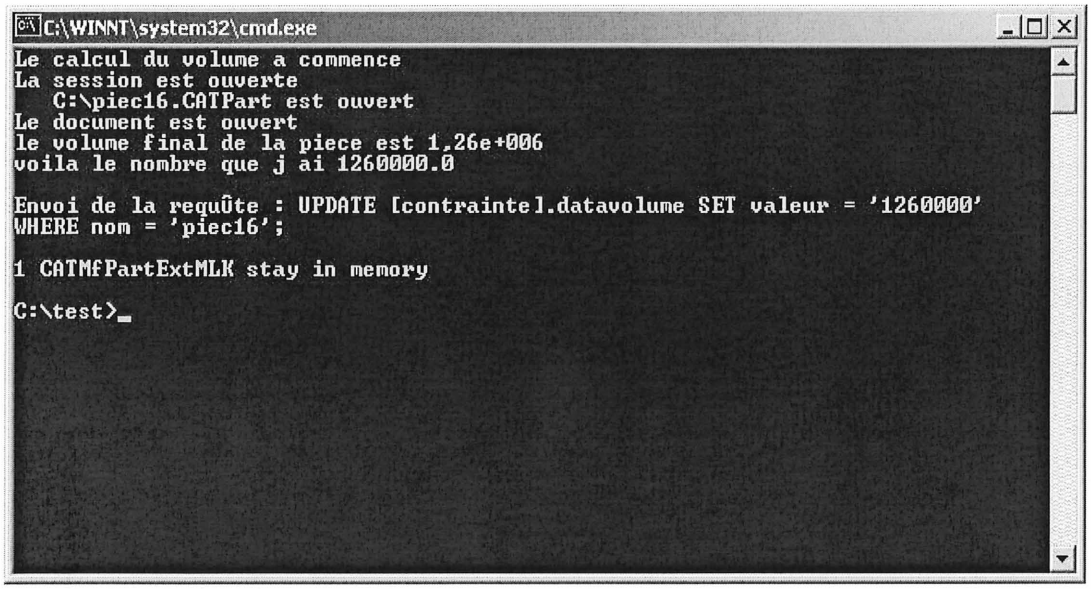


Figure 6.21 : Ouverture de sessions CATIA pour le calcul du volume des pièces : piec10.CATPart, piec11.CATPart, piec16.CATPart

- **Calculs et résultat de la vérification**

L'agent créé fait la somme de tous les volumes des pièces qu'il a trouvé et vérifie si la contrainte est respectée. Dans le cas de la contrainte VOLUME_Cockpit la contrainte est vérifiée. (figure 6.22)

Figure 6.22 : Résultat de la vérification de la contrainte VOLUME_Cockpit

- **Création d'un agent « Contrainte » pour la vérification des contraintes VOLUME_Aile et VOLUME_Nez**

Les deux contraintes VOLUME_Aile et VOLUME_Nez sont facultatives, elles vont donc être vérifiées par un seul agent « Contrainte ». (figure 6.23)

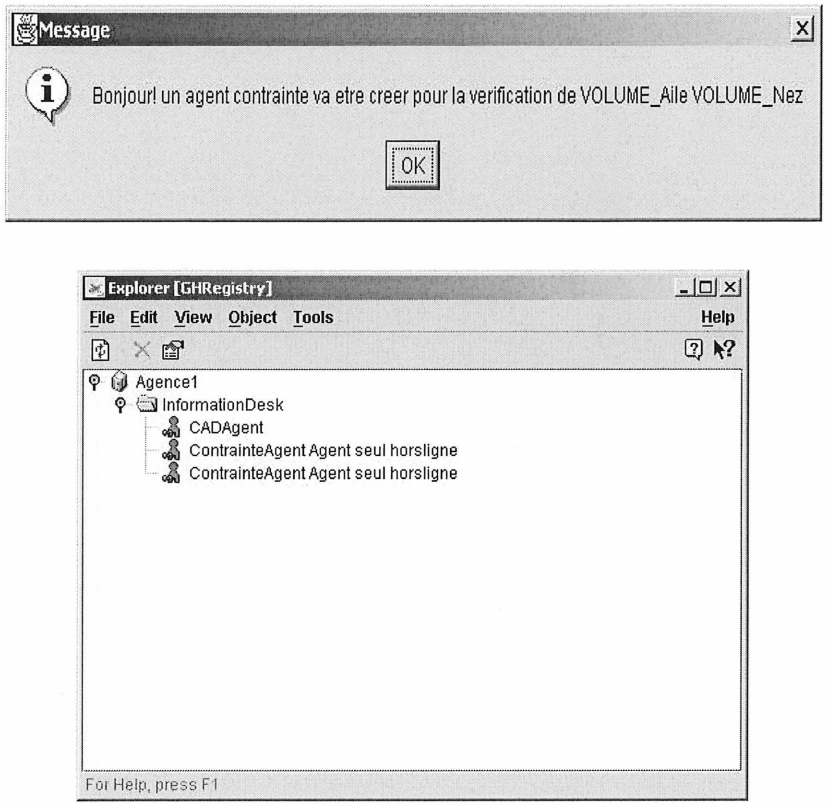


Figure 6.23 : Création d'un agent pour la vérification des contraintes VOLUME_Aile et VOLUME_Nez

a) Contrainte VOLUME_Aile

La figure 6.24 montrent l'identité de l'agent qui va vérifier la contrainte VOLUME_Aile.

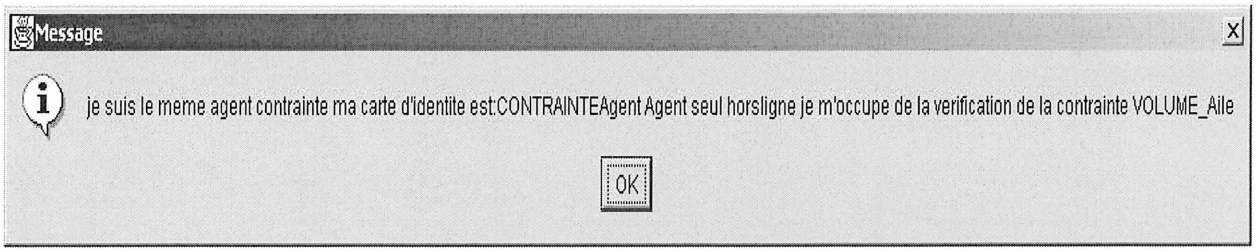


Figure 6.24 : Identité de l'agent « Contrainte » créé pour vérifier VOLUME_Aile

- **Parcours de l'arbre de typage et récupération des types concernés par la contrainte VOLUME_Aile**

L'agent parcourt l'arbre de typage et récupère les sous types du type « aile ». L'agent trouve : « panneau supérieur », « panneau inférieur », « poutre de bord de fuite », « poutre de bord d'attaque » et « nervure ». (figure 6.25)

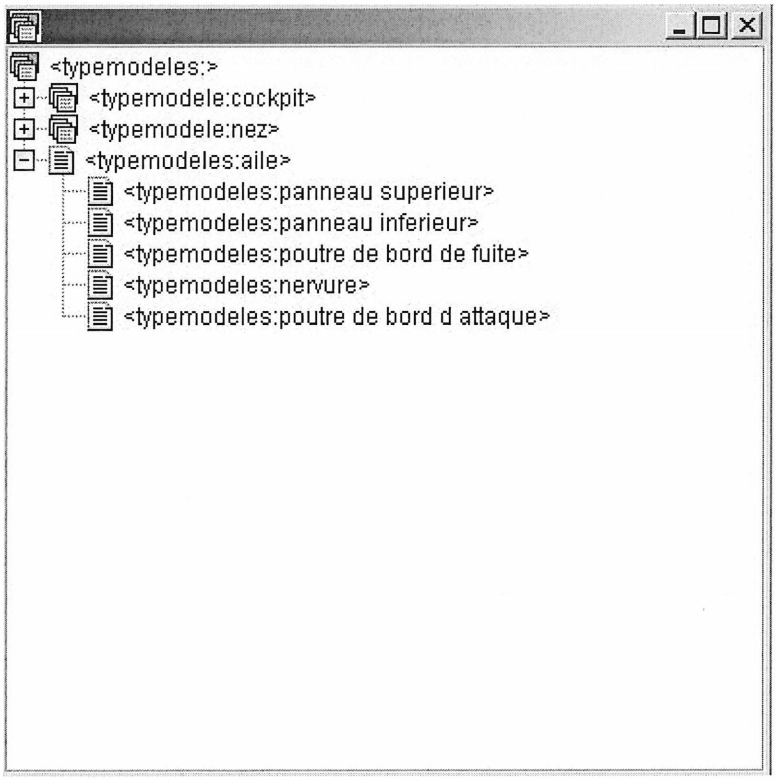


Figure 6.25 : Les sous types de Aile

- **Récupération des noms de pièces de types : « panneau supérieur », « panneau inférieur », « poutre de bord de fuite », « poutre de bord d'attaque » et « nervure »**

L'agent se connecte à une base de donnée Access pour récupérer toutes les pièces de types « panneau supérieur », « panneau inférieur », « poutre de bord de fuite », « poutre de bord d'attaque » et « nervure ».

piece1.CATPart : panneau supérieur

piece2.CATPart : panneau inférieur

piece3.CATPart : poutre de bord de fuite

piece4.CATPart : poutre de bord d' attaque

piece5.CATPart : nervure

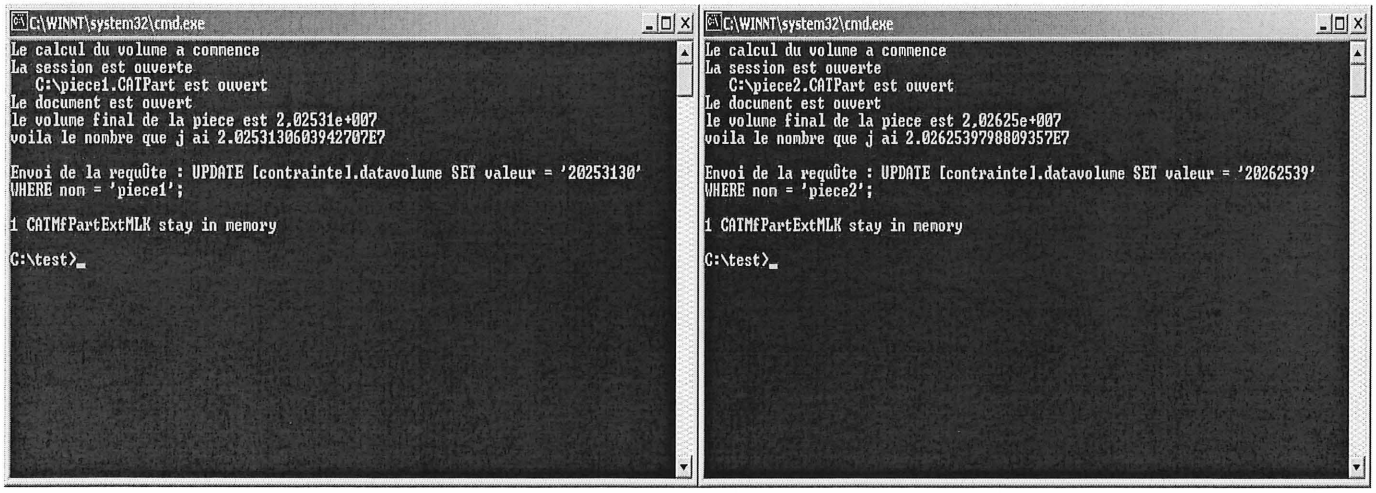
piece6.CATPart : nervure

piece7.CATPart : nervure

piece8.CATPart : nervure

- **Ouverture de session CATIA pour chaque pièce trouvée et récupération de son volume**

L'agent ouvre par la suite une session CATIA pour les huit pièces pour calculer leurs volumes. (figure 6.26)



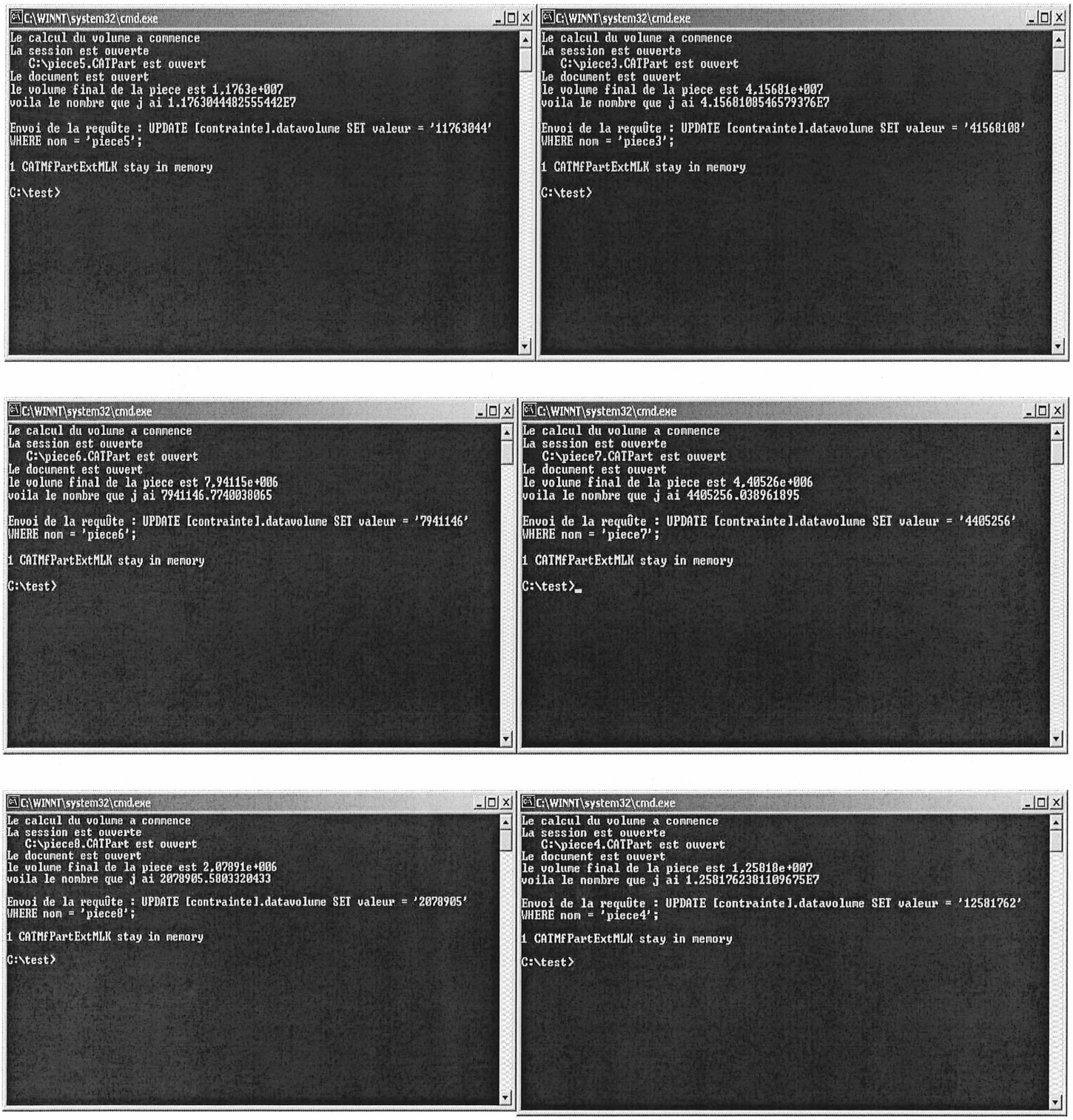


Figure 6.26 : Ouverture de sessions CATIA pour le calcul du volume des pièces : piece1.CATPart, piece2.CATPart, piece3.CATPart, piece5.CATPart, piece6.CATPart, piece7.CATPart, piece8.CATPart, piece4.CATPart

- **Calculs et résultat de la vérification**

L'agent de vérification trouve que la contrainte est non respectée. (figure 6.27)

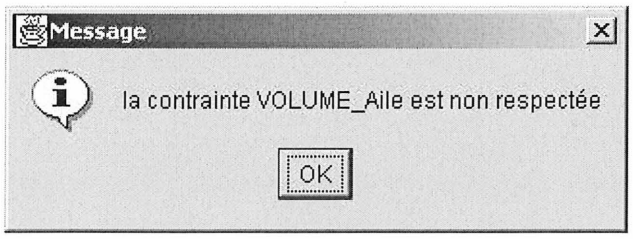


Figure 6.27 : Résultat de la vérification de la contrainte VOLUME_Aile

b) Contrainte VOLUME_Nez

L'agent qui vérifie la contrainte VOLUME_Nez est le même agent qui a déjà vérifié la contrainte VOLUME_Aile puisque les deux contraintes appartiennent à la même classe de contraintes « Contrainte facultative ». (figure 6.28)

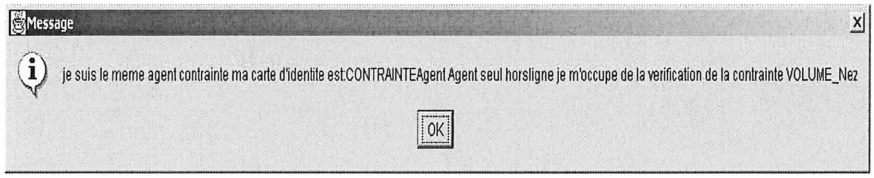


Figure 6.28 : Identité de l'agent « Contrainte » créé pour vérifier VOLUME_Nez

- **Parcours de l'arbre de typage et récupération des types concernés par la contrainte VOLUME_Nez**

L'agent parcourt l'arbre de typage et récupère les sous types du type « Nez ». L'agent trouve : « nez gauche », « nez droite ». (figure 6.29)

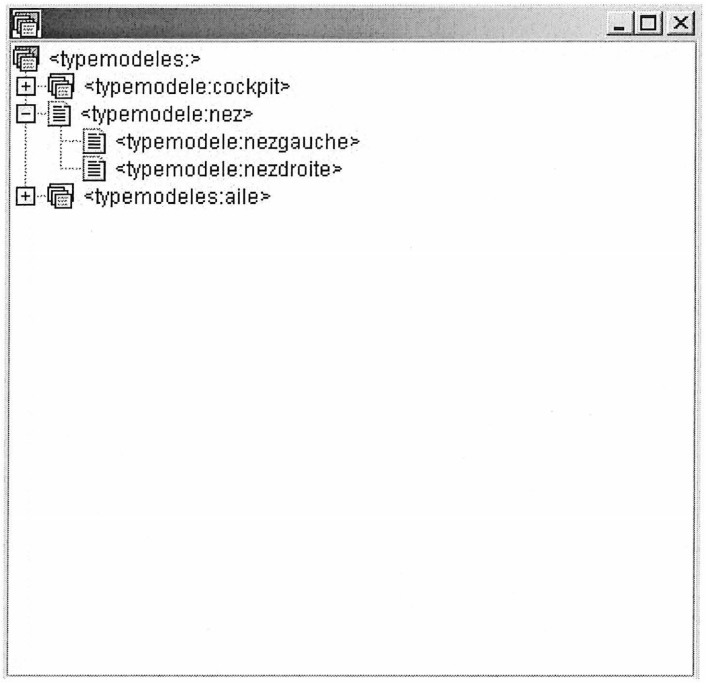


Figure 6.29 : Les sous types de Nez

- **Récupération des noms de pièces de types « nez », « nez gauche », « nez droite »**

L'agent trouve respectivement : piec12.CATPart, piec13.CATPart, piec15.CATPart.

- **Ouverture de session CATIA pour chaque pièce trouvée et récupération de son volume**

L'agent ouvre par la suite une session CATIA pour les trois pièces et calcul leurs volumes. (figure 6.30)

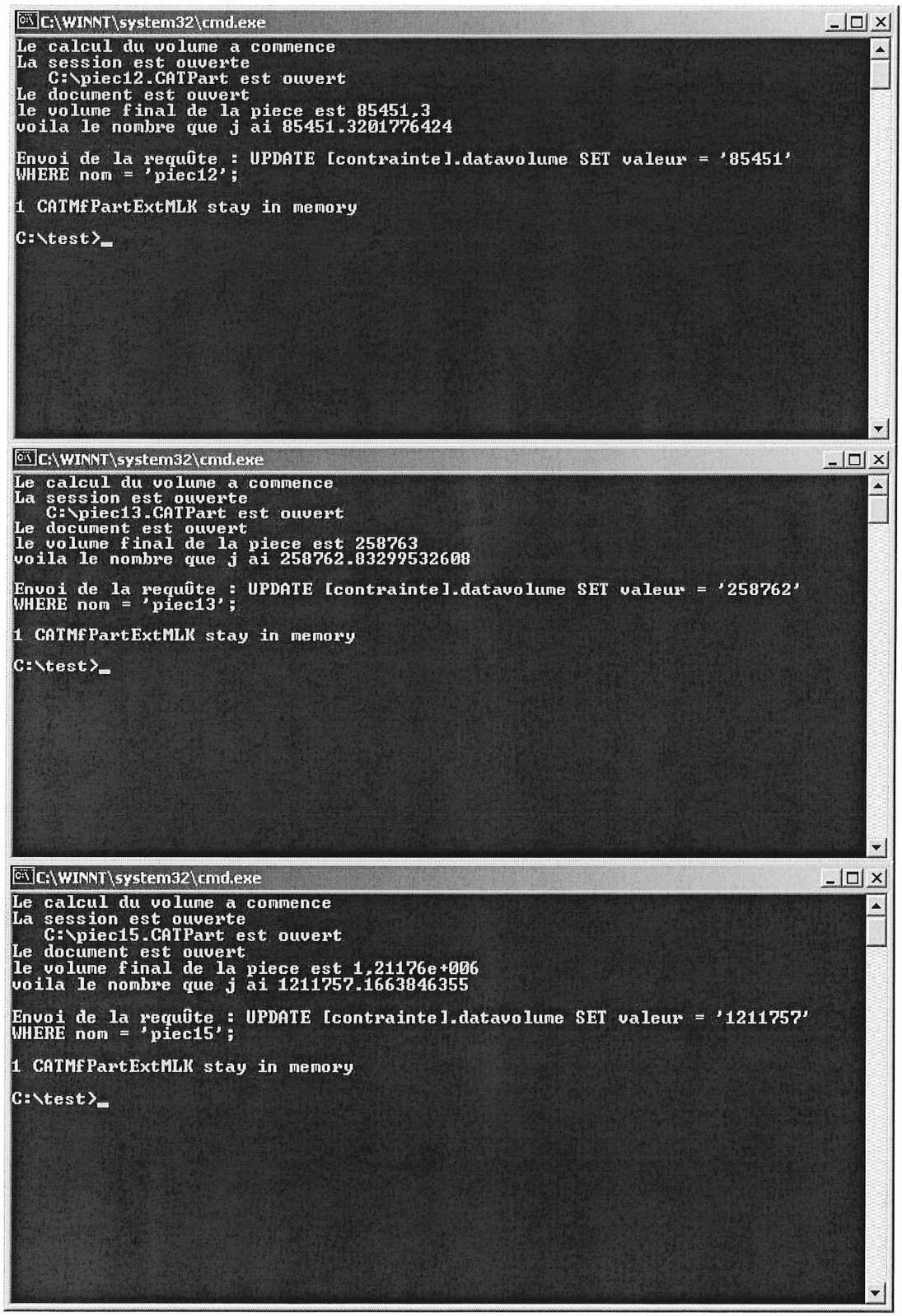


Figure 6.30 : Ouverture de sessions CATIA pour le calcul du volume des pièces : piec12.CATPart, piec13.CATPart, piec15.CATPart

L'agent trouve que la contrainte est bien respectée. (figure 6.31)

Figure 6.31 : Résultat de la vérification de la contrainte VOLUME_Nez

6.5.3 Validation des résultats

Les résultats du prototype valident parfaitement les grands axes de l'architecture proposée :

- Création d'une librairie partagée (dll) permettant d'accéder aux modèles (fichiers CatPart) en ouvrant des sessions CatiaV5 sans toutefois lancer CATIA (recherche du volume des pièces en "background");
- Création d'arbres de typage des modèles en utilisant le langage XML;
- Création, gestion et caractérisation de contraintes (vérification du volume d'une aile ici) basées sur une grammaire exploitant les types définis précédemment;
- Vérification des contraintes par des agents logiciels exploitant des stratégies de vérification prédéfinies en fonction du type de contraintes (plateforme Grasshopper);
- Interaction avec une base de donnée CAO créée sous ACCESS;

Le prototype est entièrement fonctionnel et ne requérant pas de licence CAA Rade ou autre (sauf une licence CatiaV5).

CONCLUSION

Nous avons réussi, dans le cadre de notre projet de maîtrise, à valider le concept d'agents intégrés à la conception assistée par ordinateur. Notre système assure la vérification des contraintes du cahier des charges sur des modèles CAO et notifie l'utilisateur en cas du non respect des contraintes. L'architecture du système que nous avons proposé est complète et nous avons pris le soin d'étudier et d'analyser chaque partie de notre système. Le prototype que nous avons développé valide chaque partie importante du système à savoir la gestion des contraintes, la gestion des arbres de typages, la gestion des stratégies et la vérification. Dans ce présent rapport, nous avons présenté également plusieurs applications types de notre système. Puisque ce dernier est assez générique, nous pouvons l'adapter à d'autres métiers en créant d'autres types de déploiement d'agents encore plus complexes. Nous pourrions même prévoir, dans un développement futur, de faire des tests d'échelle pour évaluer combien de contraintes notre application pourrait supporter.

A la fin de notre projet, nous avons constaté qu'il y a plusieurs perspectives qui s'ouvrent à nous pour des développements futurs du système, notamment dans le cadre d'un projet de doctorat. Premièrement, nous pourrions prévoir d'approfondir le travail actuel au niveau de l'optimisation et de la coordination des agents. Nous proposons dans ce cas d'utiliser des mécanismes de coordination, de planification et de négociation pour coordonner le travail au sein du système. Les contraintes peuvent être partagées entre les différents agents selon les types de contrainte et l'expertise de chaque agent. Chaque agent pourrait être spécialisé dans un domaine précis (hydraulique, mécanique..) et pourrait, dans ce cas, traiter les contraintes qui correspondent parfaitement à son domaine d'expertise, en planifiant ses tâches et en prenant en considération les tâches des autres agents dans le système. Autrement dit chaque agent pourrait planifier ses tâches d'une manière à ce qu'il n'effectue pas des tâches répétitives et inutiles. L'ensemble des agents pourrait coordonner dynamiquement ses actions au sein du SMA (Système Multi-Agents) afin de créer un plan d'action optimisé pour l'ensemble du système. On pourrait alors penser à une approche distribuée, compte tenu de ses

avantages par rapport à l'approche centralisée, qui implique de son côté une dépendance envers un seul superviseur qui pourrait en tout temps tomber en panne et bloquer le fonctionnement du groupe d'agents.

Par ailleurs, nous pourrions également prévoir faire la vérification de la cohérence des contraintes entre elles. Il pourrait arriver que des contraintes propres à différents métiers complètement distincts, ou encore propres à projet donné, soient complètement incohérentes. Il s'agirait donc de trouver une manière de détecter de telles incohérences et les corriger.

Toutefois, il faudrait d'abord faire une recherche bibliographique sur ces différents sujets qui s'offrent à nous, afin d'en vérifier l'originalité. C'est justement nous proposons de faire dans le cadre d'un projet de doctorat.

Bibliographie

[1] SUTHERLAND, I.E, and MIT Lincoln Laboratory, (1963) Sketchpad: *A Man-Machine Graphical Communication System*, Detroit, Michigan,USA, 18 p.

[2] DESROCHERS, A., (Automne 2002) *Conception assistée par ordinateur*, Université de Sherbrooke, Sherbrooke, Canada, Notes du cours.

[3] INGHAM, P., STOLL, H., (Mars 1991) *CAD systems in mechanical and production engineering*, New York, USA, 44 p.

[4] TANER, B., (1997) *Product data management systems: State-of-the-art and the future,* Toronto, Canada, 7 p.

[5] PUGH, S., (1991) *Total Design: Integrated methods for successful product engineering,* Massachusetts, Addison-Wesley, USA, 278 p.

[6] CROSS, N. (2000) *Engineering Design Methods*, New York, USA, 212 p.

[7] CLIVE, L.-D, LITTLE, P., (1999) *Engineering Design*, New York, USA, 278p.

[8] PRASAD, B., (1996) *Concurrent Engineering: Fundamentals Volume 1, Integrated Product & Process Organization*, New Jersey, USA, 502 p.

[9] CHAIB-DRAA, B., (1999) *Agents et Systèmes Multi-agents*, Université Laval, Québec, Canada, Notes de cours.

[10] Grasshopper Basics And Concepts: Release 2.2 IKV++, guide du programmeur.

[11] CHENG, G., TORU, I., (1996) *Analyzing the social behavior of contract net protocol,* Lecture Notes in Artificial Intelligence, Kyoto University, Japan, 116 p.

[12] DURFEE, E. H., LESSER, V. R., (1991) *Partial Global Planning: A Coordination Framework for Distributed Hypothesis Formation, IEEE Transactions on Systems, Man, and Cybernetics,* IEEE Transactions on Systems, Man and Cybernetics, University of Michigan, Michigan, 16 p.

[13] CHERKAOUI, S.,DESROCHERS, A., (2001) *Verification of Product Technical Specifications in CAD systems using software agents*, Sherbrooke, Canada, 9 p.

[14] SHEN, W., (2002) *Distributed Manufacturing Scheduling using Intelligent Agents*, IEEE Intelligent Systems and Their Applications, Integrated Mfg. Technol. Institute, National Research Council Canada, Ontario, 6 p.

[15] BALASUBRAMANIAN, S., NORRIE, D.H., (1996) *A multiagent Architecture for concurrent Design, Process Planning, Routing, and scheduling*, Concurrent Engineering, University of Calgary, Calgary, Alberta, Canada, 9 p.

[16] SHEN, W., MATURANA, F., NORRIES, D. H., (2000) *MetaMorph II: An agent-based architecture for distributed intelligent design and manufacturing,*, University of Calgary, Calgary, Alberta, Canada, 14 p.

[17] SHEN, W., BARTHÉS, J. P., (1996) *An Experimental Multiagents Environment For Engineering Design*, Université de Technologie de Compiègne, France, p 20.

[18] SHEN, W., BARTHÉS, J. P., (1995) DIDE: *A multiagent Environment for Engineering Design*, Université de Technologie de Compiègne, France, 7 p.

[19] SHEN, W., DOUGLAS, H. N., (1998) *An Agent-Based Approach for Dynamic Manufacturing Scheduling*, University of Calgary, Calgary, Alberta, Canada, 11 p.

[20] BIN, H., (1996) *Research on running mode in cooperative design*, Beijing, China, 4 p.

[21] MA, X. *A.*, (1996) *multimedia Interface for human-human interaction*, Bejing, China, 5 p.

[22] HONG, C., WANG, G. Y., LI, Z., (1999) *A multi-agent model and its speech act theory for cooperative mechanical CAD* , Journal of intelligent manufacturing, 6 p.

[23] BIN, H., LIN, S. X., (2000) *A multi-agent prototype in cooperative Design on multi-levels,* Shenzhen, China, 3p.

[24] XUFENG, P., (1997) *Computer Integrated Environment of Vehicle Body CAD system, automotive engineering,* China, 19 p.

[25] CHERKAOUI, S., DESROCHERS, A., HABHOUBA, D., (2003) *A Multi-agent Architecture for Automated Product Technical Specifications Verification in CAD Environments, Transactions of the SPDS*, Canada, 17 p.

[26] FERRY, HEBERMAS, J. M., (1987) *l'éthique de la communication*, Paris, p 587.

[27] DIETZ, J. L. G., (1999) *Understanding Business Processes on the basis of Habermas' Theory of Communicative Action*, Milwaukee, USA, 14 p.

[28] http://www.icare-espace.com/ing_sim1.html

[29] GAO, J. X., HAYDAR, A., MAROPOULOS, P. G., CHEUNG, W. M., (2003) *Application of product data management technologies for enterprise integration,* London, UK 10 p.

[30] LIU, D. Z., LIU, M., ZHONG, P. S., (2004) *Method of Product Development Process Analysis and Reengineering for Concurrent Engineering*, China, 5 p.

[31] NASA TECH BRIFS, (2002) *Solutions- Information sciences–Multiover coordinationbased on Contract Net protocol*, Californie, USA, 50 p.

[32] LEYDESDORFF, L., (2000) *Luhmann, Habermas, and the Theory of Communication* , Hollande, 16 p.

[33] DERELI, T., BAYKASOGLU, A., (2003) *Concurrent engineering utilities for controlling interactions in process planning*, Turkey, 9 p.

[34] DECKER, K., LI, J., (2000) *Coordinating Mutually Exclusive Resources using GPGP,* Hollande, 25 p.

[35] CUTKOSKY, M. R., ENGELMORE, R. S., FIKES, R.E., GENESERETH, M. R., GRUBER, T. R., MARK, W. S.,TENENBAUM, J. M., WEBER, J. C., (1993) *PACT: An Experiment in Integrating Concurrent Engineering Systems*, Sanford, USA, 9 p.

[36] http://www-cdr.stanford.edu/DSC/

[37] SHEN, W.,BARTHÈS, J. P., (1996) *An Experimental environment for exchanging design knowledge by cognitive agents, In M, Mantyla, S. Finger and T. Tomiyama, eds., knowledge Intensive CAD*, Compiègne, France, 9 p.

[38] BARTHÈS, J. P., (2003) *MOSS 4 : A primer*, Compiègne, France, 25 p.

[39] IADINE,A. A., (2000) *Les systèmes multi-agents*, France 24 p.

Annexes

Menu principal

```
public class MenuArbre extends JFrame {}

//Cette classe permet de definir le menu principale de l'application
```

Gestion des arbres de typages

1- Création d'arbres à partir de fichier XML

```
public class JDomTreeFrame extends CentralFrame {
//static JDomTree aa;
static JFrame j= new JFrame();
 public JDomTreeFrame(String l) {
  initComponents ();
  JDomTree aa = new JDomTree(l);
  aa.setCellRenderer(new JDomTreeCellRenderer(j));
  getContentPane().add (new JScrollPane(aa), BorderLayout.CENTER);
  pack ();
 }
.....}
 private void initComponents () {
  addWindowListener (new java.awt.event.WindowAdapter () {
 public void windowClosing (java.awt.event.WindowEvent evt)
       {
  if (!test.interfacontrainte.castype)
  test.Application1.frame.show();
  else
  {
  test.interfacontrainte.castype=false;
  test.Application6.frame.show();

  }
      }
  }
  );
       /*
    doButton.addActionListener (new java.awt.event.ActionListener () {
    public void actionPerformed (java.awt.event.ActionEvent evt) {
    doButtonActionPerformed (evt);
    }
   }
   );
       */

  // XMLTree aa = new XMLTree("personal-schema.xml");
```

```
        /*JDomTree aa = new JDomTree("ini.xml");
        aa.setCellRenderer(new JDomTreeCellRenderer(j));
      getContentPane().add (new JScrollPane(aa), BorderLayout.CENTER);*/

 }//(JFrame)this)

 private void doButtonActionPerformed (java.awt.event.ActionEvent evt) {

 }

 private void exitForm(java.awt.event.WindowEvent evt) {
 dispose();
 System.exit (0);
 }

 public static void main (String args[]) {
  //JDomTreeFrame f = new JDomTreeFrame();
  //f.setSize(400, 400);
 // f.setLocationToCenter();
  //f.setVisible(true);
 }

}

// Les trois classes suivantes sont également pertinente pour la gestion des arbres de typage

public class JDomTree extends JTree {}

class JDomTreeCellRenderer extends DefaultTreeCellRenderer {}

 class JDomTreeModel extends DefaultTreeModel implements Serializable {}
```

Gestion des attributs

```
public class attribut extends JFrame {} //créer des attributs

public class attributmodif extends JFrame {} //modifier les attributs

public class attributsuppr extends JFrame {} //supprimer les attributs
```

Gestion des contraintes

1- Création des contraintes

public class interfacontrainte extends JFrame {} //classe de création des contraintes

2- Compilation de la contrainte

```
public class Cadre1 extends JFrame {
   //Construire le cadre
  public Cadre1() {
//ouverture du fichier ou la formule est stockée
 try{
 ouvrir();}
 catch(Exception e){}

 petitexp();
   }

  //Initialiser le composant
 /* private void jbInit() throws Exception  {
   //setIconImage(Toolkit.getDefaultToolkit().createImage(Cadre1.class.getResource("[Votre
icône]")));
   contentPane = (JPanel) this.getContentPane();
   jTextArea1.setBounds(new Rectangle(36, 72, 309, 65));
   contentPane.setLayout(null);
   this.setSize(new Dimension(400, 300));
   this.setTitle("");
   jButton1.setBounds(new Rectangle(144, 161, 104, 24));
   jButton1.setText("Compiler");
   jButton1.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(ActionEvent e) {
       jButton1_actionPerformed(e);
     }
   });
   contentPane.add(jTextArea1, null);
   contentPane.add(jButton1, null);
 }
 //Remplacé, ainsi nous pouvons sortir quand la fenêtre est fermée
 protected void processWindowEvent(WindowEvent e) {
   super.processWindowEvent(e);
   if (e.getID() == WindowEvent.WINDOW_CLOSING) {
     System.exit(0);
   }
 }
```

```
/*void jButton1_actionPerformed(ActionEvent e) {
//String a = jTextArea1.getText();
petitexp();  }*/


// debut de programme de compilation
 void petitexp()
 {
 try {

//ouverture du fichier ou la formule est stockée

/*BufferedReader b = new BufferedReader( new InputStreamReader(
                        new FileInputStream("monfichier.txt")));
chaine= b.readLine();*/
//chaine= jTextArea1.();
 tokenizer = new StringTokenizer(chaine);
 int count = tokenizer.countTokens();

// debut de la compilation
 while (tokenizer.hasMoreTokens()&& nonstop)
 {
 System.out.println("je suis dans le while");
 token = tokenizer.nextToken();

/*if (token.compareTo("")==0)
{JOptionPane.showMessageDialog(null,"ecrivez votre formule");}*/

 if (testNumber.isNumber(token))
 {
 System.out.println("j ai trouve1"+ token);
 chiffre(token);}

// IF
 else if (token.compareTo(v)==0 | token.compareTo(m)==0 | token.compareTo(px)==0 |
token.compareTo(py)==0|token.compareTo(pz)==0)
   {
   System.out.println("j'ai trouve"+token);
   if (!tokenizer.hasMoreElements())
   finexpr(token);
   else
   {
   token= tokenizer.nextToken();
   constantemotcle(token);
   }
   } //END IF

// else if
```

```
 else if (token.compareTo(f1)==0 | token.compareTo(f2)==0|token.compareTo(f3)==0 |
token.compareTo(f4)==0
|token.compareTo(f5)==0|token.compareTo(f6)==0|token.compareTo(f7)==0)
   {
   System.out.println("j'ai trouve"+token);
   if (!tokenizer.hasMoreElements())
   finexpr(token);
  else
   {
   token= tokenizer.nextToken();
   fonction(token);
   }// end else
  }// END else if
//else
else {JOptionPane.showMessageDialog(null,"il faut commencer par une foction ou un chiffre ou v m
px py pz ");
     nonstop=false;}// END else
}// end while



} catch (Exception v){};
}// // fin de petitexpression

//*************************************************************************************
******************

// foction qui verifie si on a un v aile operateur ou f ( v aile ) operateur
void constantemotcle(String s)
{
      //IF 1
     if (token.compareTo("aile")==0 |
token.compareTo("nez")==0|token.compareTo("cockpit")==0|token.compareTo("aucuntype")==0|toke
n.compareTo("toutlestypes")==0)
        {
         System.out.println("j ai trouve2"+ token);
         //IF 1.2
         if (!tokenizer.hasMoreElements())
         finexpr(token);
         else // else de 1.2

          {
         token= tokenizer.nextToken();

      if (token.compareTo("gauche")==0 |
token.compareTo("droite")==0|token.compareTo("haut")==0|token.compareTo("bas")==0)
       { //IF 1.2
         if (!tokenizer.hasMoreElements())
         finexpr(token);
         else // else de 1.2
          {
```

```
          token= tokenizer.nextToken();
         //if 1.2.1
       if (token.compareTo("+")==0 | token.compareTo("-")==0|token.compareTo("*")==0|
token.compareTo("/")==0)
          {System.out.println("j ai trouve3"+ token);
          if (!tokenizer.hasMoreElements()) // if 1.2.1.1
          finexpr(token);
          else // else 1.2.1.1
           {
            token= tokenizer.nextToken();
            // if 1.2.1.1.1
            if (testNumber.isNumber(token))
            {System.out.println("j ai trouve4"+ token);
            chiffre (token);}

            else if (token.compareTo(v)==0  | token.compareTo(m)==0 | token.compareTo(px)==0 |
token.compareTo(py)==0|token.compareTo(pz)==0)
              ..........}}}}}}}}
```

3- Interprétation de la contrainte

```
public class interprete extends JFrame {}
```

Définition des stratégies

```
 public Stratégie() {} //Création des stratégies
```

Verification des contraintes

1- Verification

```
public class verification extends JFrame {


  public verification(Vector vect)
  {

    super( "Liste des contraintes" );

    Container c = getContentPane();
    c.setLayout( new FlowLayout() );
```

```
    listecontrainte = new JList( vect );
    listecontrainte.setVisibleRowCount( 25 );
    listecontrainte.setFixedCellHeight( 15 );
    listecontrainte.setSelectionMode(
      ListSelectionModel.MULTIPLE_INTERVAL_SELECTION );
    c.add( new JScrollPane( listecontrainte ) );

    // créer bouton ok.
   ok = new JButton( "Lancer" );
    ok.addActionListener(
      new ActionListener() {
        public void actionPerformed( ActionEvent e )
        {
        tab.clear();
        java.util.List list = new ArrayList();
          for(int x=0; x < listeCopie.getModel().getSize();x++)
           {
            list.add(listeCopie.getModel().getElementAt(x).toString());
           }
        System.out.println("voila la liste "+list.toString());
        tab.addAll(list);
        System.out.println("voila ce que j ai dans tab "+tab);
        int index = 0;
        counter=listeCopie.getModel().getSize();
        for(int i = 0; i < listeCopie.getModel().getSize(); i++) {
        s+= listeCopie.getModel().getElementAt(i).toString()+" ";
        }
        l= s.substring(4,s.length()-1);
        System.out.println(l);
        s="";

        hide();
//**********************création de la region et de l'agence et de l'agent*******************
                // Create a proxy on the currentRegionAddress
                  getLocalIp();
                  try{
                region = RegionRegistrationFactory.createRegionRegistration("GHRegistry");  // Use
default name -> bug in GH 2.1
                                currentRegionAddress = new GrasshopperAddress(localHost);
                                System.out.println("Region registry created to location:
"+currentRegionAddress.toString());
                                connect = true;

                        region = (IRegionRegistration)
ProxyGenerator.newInstance(IRegionRegistration.class,
                                 currentRegionAddress.generateRegionId(), currentRegionAddress);
                    }
                 catch(Exception e2){System.out.println("c'est ici 1");}
                // Try some method to see if the region is really living
                        try{
```

```
                                agentSystemInfo = region.listAgencies(new SearchFilter());
                        //System.out.println(connect+" je suis dans connect");
                                System.out.println("Region Registry found to
"+currentRegionAddress.toString());
                                //controlDetection = false;  // Region found
                                //connect = true;
                            // A region is living
                        }
                        catch(Exception e2){System.out.println("c'est ici 2");
                        //index++;    // If the method didn't worked, read another address and retry
                }
                //}//end of while

 //2-creation de l'agence

if(connect){

                //getAgencyName();// Get the agency name (the names are unique)
            currentAgencyName="Agence1";
                agency = AgentSystemFactory.createAgentSystem(currentAgencyName,
currentRegionAddress.toString()); // Create agency
                System.out.println(currentAgencyName+" has been created");
            AgentSystemFactory.start(agency);
            try{
            // a ce niveau la on peut tester sur les strategie et verifier les strategies qui ont leger comme
            //attribut c est a dire un seul agent!
            //apres on pourra prendre en consideration les contrainte lourde qui demande la creation de
plusieurs agents
            JOptionPane.showMessageDialog(null,"Agent Classement est crée");
            System.out.println("Agent has been created"); // Create an agent
            AgentInfo CADAgent = agency.createAgent("test.CADAgent",null,null,null);
            //while(test.CADAgent.agentcadtermine==false) {}
                        }
                catch(AgentCreationFailedException acfe){
                        //acfe.printStackTrace();
                System.out.println("ca n a pas marché!");
                }

    }
   //}


//**********************fin de la création du systeme agent*****************************
        }

     }
   );....}}}}}}}}}}
```

2- Classement des contraintes selon leurs priorités

```
public class filtragecontrainte {
public static String ligne=null;
boolean zone= false;
boolean contrainte= false;
Vector listnomstrategie = new Vector();
Vector liststrategie = new Vector();
Vector vect= new Vector();
Connection connexion;
Statement ordre;
ResultSet rs;
String resul1=null;
String resul2=null;
String resul3=null;
boolean existecontrainte1=false;
boolean existecontrainte2=false;
String[] tabcontrainte;
Vector taboblig= new Vector();
Vector tabfacult= new Vector();
Vector tablegere= new Vector();
Vector tablourde= new Vector();
Vector tabdiffstrategie= new Vector();
Vector tabdiffnomstrategie= new Vector();
Vector vectmemenomstrategie= new Vector();
Vector vectmemestrategie= new Vector();
public static Vector veczone= new Vector();
public static Vector veccontrainte= new Vector();
String interm="";
int k=0;
String[] strategie;
StringTokenizer tokenizer =null;
StringTokenizer tokenizer1 =null;
int s=0;
String token="";
int taillestrategie=0;
int f=0;
int compteur=0;
boolean trouve=false;
boolean trouve1=false;
Vector intem= new Vector();
boolean vrai=true;
boolean vrai1=true;
boolean memorise=false;
boolean diff=false;
int counter=0;
boolean memestrategie=false;
private GrasshopperAddress currentRegionAddress = null;
private IAgentSystem agency = null;
private AgentSystemInfo agentSystemInfo[] = null;
private IRegionRegistration region = null;
```

```
public static AgentInfo GeneralAgent = null;

   // Other variables
private boolean controlDetection = true;      // Used in region detection
private boolean connect = false;              // Used in agency creation
private String localHost = null;              // Used in region detection/creation
private String currentAgencyName = null;


public filtragecontrainte(Vector vect) {

String url= "jdbc:odbc:contrainte";
   try {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   connexion= DriverManager.getConnection(url);
     }
   catch (ClassNotFoundException e)  {}
   catch (SQLException e)  {System.out.println("la connexion n a pas marche");}


counter = vect.size();
System.out.println("je suis dans le prog filtrage le nombre de contrainte est "+counter);
tabcontrainte= new String[counter];
strategie=new String[counter];

for (int i=0; i<counter;i++)
{
tabcontrainte[i]=vect.elementAt(i).toString();
}

for(int i=0;i<counter;i++)
{
try {
        ordre = connexion.createStatement();
        String requete = "SELECT priorite, politique_verif FROM contrainte "+
        "WHERE nom = '" + vect.elementAt(i).toString()+ "'";
        //"WHERE nom = 'volume1'";
       //"WHERE nom = 'volume de carburant'";
        System.out.println( "\nLancement de la requête : " +
                connexion.nativeSQL( requete )
                + "\n" );
        rs = ordre.executeQuery( requete );
        insererpriostrat(rs);
        ordre.close();

     }
  catch ( SQLException sqlex ) {
      System.out.println("pb sql");
  }
```

```
}//fin de la boucle
recupererattribut();
filtrage0();


 }//fin constructeur

private void insererpriostrat (ResultSet resultat) {
 try
 {
 int i;

 /* Les informations telles que le nom des colonnes et leur nombre
 * sont dans l'objet ResultSetMetaData.
 * le nombre de colonnes nbcol s'obtient avec getColumnCount ()*/
  ResultSetMetaData rsmd = resultat.getMetaData ();
  int nbcol = rsmd.getColumnCount ();

 /* Pour afficher les en-têtes de colonnes*/
  for (i=1; i<=nbcol; i++) {
   if (i > 1) System.out.print("\t");
      System.out.print(rsmd.getColumnLabel(i));
  }
  /* Pour sauter une ligne */
  System.out.println("");
  /* Pour afficher chaque ligne de la table */
  boolean encore = resultat.next ();
  while (encore) {
  /* Pour lire et afficher toutes les colonnes en tabulant */
  for (i=1; i<=nbcol; i++) {
  if (i > 1) System.out.print("\t");
  if (i==1) resul1=resultat.getString(i);//la priorite
  if (i==2)//la strategie
  {
  resul2=resultat.getString(i);
  strategie[s]=resul2;
  System.out.println("voila ce que j ai dans le tableau strategie "+strategie[s]);
  s++;
  }
  //interm= interm+" "+resul;
  }
  System.out.println("");
  remplirtab1(resul1);

  /* Puis on passe à la ligne suivante */
  //calculfinal(resul);
  //System.out.println("voila ce que j ai dans mon tableau"+action[k]);
  encore = resultat.next ();
  System.out.println(encore);
  }
  }catch (Exception v) {}
```

```
 taillestrategie=s-1;
//System.out.println(positionmemoir);
 }


void remplirtab1(String data)
{
tabcontrainte[k]=tabcontrainte[k]+" "+data+" "+enleverspace(strategie[k]);
System.out.println("voila ce que j'ai dans mon tableau maintenanat a la case "+k+" "+tabcontrainte[k]);
k++;
}


public void recupererattribut()
{
for(int i=0;i<=taillestrategie;i++)
{
try {
        ordre = connexion.createStatement();
        String requete = "SELECT attribut1, attribut2, attribut3 FROM strategie "+
        "WHERE nom = '" + strategie[i]+ "'";
        //"WHERE nom = 'volume1'";
       //"WHERE nom = 'volume de carburant'";
        System.out.println( "\nLancement de la requête : " +
                connexion.nativeSQL( requete )
                + "\n" );
        rs = ordre.executeQuery( requete );
        insererstrategie(rs);

      }
  catch ( SQLException sqlex ) {
      System.out.println("pb sql");
 }//fin du catch


}// fin de la boucle

}//fin de la methode
private void insererstrategie (ResultSet resultat) {
 try
 {
 int i;

 /* Les informations telles que le nom des colonnes et leur nombre
 * sont dans l'objet ResultSetMetaData.
 * le nombre de colonnes nbcol s'obtient avec getColumnCount ()*/
  ResultSetMetaData rsmd = resultat.getMetaData ();
  int nbcol = rsmd.getColumnCount ();

 /* Pour afficher les en-têtes de colonnes*/
  for (i=1; i<=nbcol; i++) {
```

```
    if (i > 1) System.out.print("\t");
       System.out.print(rsmd.getColumnLabel(i));
  }
  /* Pour sauter une ligne */
  System.out.println("");
  /* Pour afficher chaque ligne de la table */
  boolean encore = resultat.next ();
  while (encore) {
  /* Pour lire et afficher toutes les colonnes en tabulant */
  for (i=1; i<=nbcol; i++) {
  if (i > 1) System.out.print("\t");
  if (i==1) resul1=resultat.getString(i);//attribut1
  if (i==2) resul2=resultat.getString(i);//attribut2.....}}}}}}}}}}}}}}}
```

3- Agent de classement

```
public class CADAgent extends StationaryAgent{



   public String getName(){
        return "CADAgent";
   }


   // The entry point of the agent (Grasshopper)
   // This agent does nothing except waiting for a incoming call from another agent
   public void live(){

   filtragecontrainte filtr = new filtragecontrainte(test.verification.tab);

   }

//trouvercontrainte

public void trouvercontrainte(String s)
{

String url= "jdbc:odbc:contrainte";
        try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        connexion= DriverManager.getConnection(url);
            }
        catch (ClassNotFoundException e)  {}
        catch (SQLException e)  {System.out.println("la connexion n a pas marche");}

        // log("hello je suis CADAgent et je suis entrain de verifier les contraintes souhaite moi bonne
chance!\n");
        //JOptionPane.showMessageDialog(null,"hello je suis CADAgent et je suis entrain de verifier
les contraintes souhaite moi bonne chance!\n");
//*recuperer la liste des contraintes selectionnee dans la liste**********************************
```

```
        s=test.verification.l;
        System.out.println(s);
        //int k =s.length();
        //System.out.println(k);


}
//****chercher les donnée concernant les contraintes selectionnée**************************

public void TrouverEnregistrement(String a,int i)
{
 try {
 Statement ordre = connexion.createStatement();
        String requete = "SELECT * FROM contrainte " +
                "WHERE nom = '" +
                a + "'";
        System.out.println( "\nLancement de la requête : " +
                connexion.nativeSQL( requete )
                + "\n" );
        ResultSet rs = ordre.executeQuery( requete );
        afficher( rs,i );
        System.out.println( "\nRequête réussie\n" );
        ordre.close();}
 catch ( SQLException sqlex ) {
      sqlex.printStackTrace();
      //sortie.append( sqlex.toString() );
    }

}
//*stocker ses données dans une table pour une utilisation
ulterieure*****************************
 public void afficher( ResultSet rs,int i )
  {
    try {
      rs.next();
      int numeroEnregistrement = rs.getInt( 1 );

      if ( numeroEnregistrement != 0 ) {

       // contrainte[0]=numeroEnregistrement;
        contrainte[count][1]=rs.getString( 2 ) ;//nom
        contrainte[count][2]=rs.getString( 3 ) ;//projet
        contrainte[count][3]=rs.getString( 4 ) ;//poids
        contrainte[count][4]=rs.getString( 5 ) ;//priorite
        contrainte[count][5]=rs.getString( 6 ) ;//type formule
        contrainte[count][6]=rs.getString( 7 ) ;//fonction
        contrainte[count][7]=rs.getString( 8 ) ;//politique verif
        contrainte[count][8]=rs.getString( 9 ) ;//politique notif
        contrainte[count][9]=rs.getString( 10 ) ;//formule
        /*while(count!=0)
        { System.out.println(contrainte[count][9]);
```

```
        count--;
        }*/
      }
      else
        System.out.println( "\nAucun enregistrement trouvé\n" );
    }
    catch ( SQLException sqlex ) {
      sqlex.printStackTrace();
      //sortie.append( sqlex.toString() );
    }
  } }
```

4- Agent de type « contrainte »

```
public class VERIFAgent extends StationaryAgent{

String nom= null;
static boolean agentcontermine=false;
 // Return the name of the agent (Grasshopper)
  public void init(Object[] creationArgs) {

   if (creationArgs != null) {
   nom = (String)creationArgs[0];
   System.out.println(nom);
       }
       }



   public String getName(){
         return "ContrainteAgent Agent seul "+test.filtragecontrainte.ligne;
   }


   // The entry point of the agent (Grasshopper)
   // This agent does nothing except waiting for a incoming call from another agent
public void live(){
//JOptionPane.showMessageDialog(null,"je suis un agent contrainte ma carte d'identite
est:ContrainteAgent Agent seul "+test.filtragecontrainte.ligne+ " je m'occupe de la verification de la
contrainte "+nom);
if (nom.compareTo("contrainte")==0)
{
//JOptionPane.showMessageDialog(null,"je suis un agent zone ma carte d'identite est:ZONEAgent
Agent seul "+test.filtragecontrainte.ligne+ " je m'occupe de la verification de la contrainte "+nom);
for (int i=0;i<test.filtragecontrainte.veccontrainte.size();i++)
   {
System.out.println("je suis dans verifagentgroupecontrainte voila le vect que j
ai"+test.filtragecontrainte.veccontrainte+test.filtragecontrainte.ligne);
JOptionPane.showMessageDialog(null,"je suis le meme agent contrainte ma carte d'identite
est:CONTRAINTEAgent Agent seul "+test.filtragecontrainte.ligne+ " je m'occupe de la verification de
la contrainte "+test.filtragecontrainte.veccontrainte.elementAt(i).toString());
```

```
System.out.println( "je suis le meme agent contrainte ma carte d'identite est:CONTRAINTEAgent
Agent seul "+test.filtragecontrainte.ligne+ " je m'occupe de la verification de la contrainte
"+test.filtragecontrainte.veccontrainte.elementAt(i).toString());
calculpourverification calculverif = new
calculpourverification(test.filtragecontrainte.veccontrainte.elementAt(i).toString());
   }

   }
else {
JOptionPane.showMessageDialog(null,"je suis un agent contrainte ma carte d'identite
est:CONTRAINTEAgent Agent seul "+test.filtragecontrainte.ligne+ " je m'occupe de la verification de
la contrainte "+nom);
System.out.println("je suis un agent contrainte ma carte d'identite est:CONTRAINTEAgent Agent seul
"+test.filtragecontrainte.ligne+ " je m'occupe de la verification de la contrainte "+nom);
calculpourverification calculverif = new calculpourverification(nom);}
agentcontermine=true;
 }}
```

5- Agent de type « modèle »

```
public class ZONEAgent extends StationaryAgent{

String nom= null;
static boolean agentzonetermine=false;
 // Return the name of the agent (Grasshopper)
  public void init(Object[] creationArgs) {

   if (creationArgs != null) {
   nom = (String)creationArgs[0];
   System.out.println(nom);
       }
       }



   public String getName(){
         return "ZONEAgent Agent seul "+test.filtragecontrainte.ligne;
   }


   // The entry point of the agent (Grasshopper)
   // This agent does nothing except waiting for a incoming call from another agent
public void live(){

if (nom.compareTo("zone")==0)
   {
//JOptionPane.showMessageDialog(null,"je suis un agent zone ma carte d'identite est:ZONEAgent
Agent seul "+test.filtragecontrainte.ligne+ " je m'occupe de la verification de la contrainte "+nom);
for (int i=0;i<test.filtragecontrainte.veczone.size();i++)
{
```

```
JOptionPane.showMessageDialog(null,"je suis le meme agent zone ma carte d'identite est:ZONEAgent
Agent seul "+test.filtragecontrainte.ligne+ " je m'occupe de la verification de la contrainte
"+test.filtragecontrainte.veczone.elementAt(i).toString());
System.out.println( "je suis le meme agent zone  tableau ma carte d'identite est:ZONEAgent Agent
seul "+test.filtragecontrainte.ligne+ " je m'occupe de la verification de la contrainte
"+test.filtragecontrainte.veczone.elementAt(i).toString());
calculpourverification calculverif = new
calculpourverification(test.filtragecontrainte.veczone.elementAt(i).toString());}

}
else
{
JOptionPane.showMessageDialog(null,"je suis un agent zone ma carte d'identite est:ZONEAgent
Agent seul "+test.filtragecontrainte.ligne+ " je m'occupe de la verification de la contrainte "+nom);
System.out.println( "je suis un agent zone ma carte d'identite est:ZONEAgent Agent seul
"+test.filtragecontrainte.ligne+ " je m'occupe de la verification de la contrainte "+nom);
calculpourverification calculverif = new calculpourverification(nom);

 }
agentzonetermine=true;
  }

}
```

Calcul du volume des pièces CATIA

1- Code source CAA-Rade : Récupération du volume à partir des modèles CAO en ouvrant une session CATIA en backgroud

```
#include <jni.h>
#include "CalculV.h"
#include <stdio.h>
#include <iostream.h>

// COPYRIGHT DASSAULT SYSTEMES 2000
//==================================================================================
//
// Mission        : Recolor fillets and planar faces in a Part document
//
//
// Illustrates    : o Document loading in session
//                  o Access to part within document
//                  o Access to hierarchy of features within part,
//                    filtering on fillets
//                  o Access to topology associated to features
//                  o Graphical modification of this topology
//                  o Direct access to all topological cells linked to part,
```

```
//                filtering on planar faces
//              o Graphical modification of these faces
//              o "save as" of the modified document
//
//=============================================================================
==
// How to execute :
//
//  mkrun -c "CAAMmrFeatureTopoBRep  FileNameIn FileNameOut"
//
//  where   FileNameIn  : The complete name of a Part document
//          FileNameOut : The complete name to saveas FileNameIn
//
//  ex:
//
//  CAAMmrFeatureTopoBRep
$WSROOT/CAAMechanicalModeler.edu/InputData/CAAMmrPart.CATPart
//                      PartModified.CATPart
//
//  In the current directory, PartModified.CATPart will be created
//
//=============================================================================
==================

// ObjectModelerBase Framework
#include "CATDocument.h"
#include "CATSessionServices.h"    // To Create,delete a session
#include "CATDocumentServices.h"
#include "CATInit.h"
#include "CATIContainer.h"

// ObjectSpecsModeler Framework
#include "CATIDescendants.h"
#include "CATISpecObject.h"
#include "CATLISTV_CATISpecObject.h"

// NewTopologicalObjects Framework
#include "CATBody.h"
#include "CATCell.h"

// GeometricObjects Framework
//#include "CATGeometry.h"

// MechanicalModeler Framework
#include "CATIPrtContainer.h"
#include "CATIMfGeometryAccess.h"
#include "CATMfBRepDecode.h"
#include "CATIPrtPart.h"
#include "CATIBodyRequest.h"
#include "CATIPartRequest.h"
```

```
// MecModInterfaces Framework
//#include "CATIBRepAccess.h"

// Visualization Framework
//#include "CATIVisProperties.h"          // To change faces color
//#include "CATVisPropertiesValues.h"

// TopologicalOperators
#include "CATDynMassProperties3D.h"

#include "CATUnicodeString.h"

#include "CATIAlias.h"



#define Slash    "\\"
#define path     "C:"



extern "C"
JNIEXPORT jdouble JNICALL
Java_CalculV_donnevolume(JNIEnv *env, jobject obj, jstring js)
{
 // return code

 int rcode = 0;

 // Checks number of arguments
 /*if( 2!=iArgc )
 {
        cout <<"modulevolume nomfichier" << endl;
        cout <<"le nom du fichier est le chemin de la piece" <<endl;
   return 1;
 }*/

 cout << "Le calcul du volume a commence" << endl;

 // Creates the session
const char* filename=env->GetStringUTFChars(js,0);
 //const char *filename = "C:\\cube.CATPart";
 char *pSessionName = "Sample session";
 CATSession *pSession = NULL;
 HRESULT rc = Create_Session(pSessionName, pSession) ;
 if( FAILED(rc) )
 {
        cout << "can not open session" << endl;
        return 0;
 }
```

```
cout << "La session est ouverte" << endl;

// Loads the input document
CATDocument *pDoc = NULL;
rc= CATDocumentServices::Open(filename, pDoc);

if( FAILED(rc) )
{
   cout <<"Error in opening the document: " << filename << endl ;
        return 0;
}
cout <<"   " << filename << " est ouvert" << endl;
cout <<"Le document est ouvert" << endl;

CATInit *pDocAsInit = NULL;
rc= pDoc->QueryInterface(IID_CATInit, (void**)&pDocAsInit) ;
if( FAILED(rc) )
{
        cout << "Error, the document does not implement CATInit"<< endl;
        return 0;
}

// Gets root container the document
CATIPrtContainer *pSpecContainer =
(CATIPrtContainer*)pDocAsInit->GetRootContainer("CATIPrtContainer");

pDocAsInit->Release();
pDocAsInit = NULL ;

if( NULL == pSpecContainer )
{
   cout <<"Error, the root container is NULL" << endl;
        return 0;
}

//Retrieves the MechanicalPart of the document
CATIPrtPart_var spPart ( pSpecContainer->GetPart() );
if ( NULL_var == spPart )
{
   cout <<"Error, the MechanicalPart is NULL" << endl;
        return 0;
}

pSpecContainer->Release();
pSpecContainer = NULL ;

//compute volume


CATDynMassProperties3D *pPropOp = CATDynCreateMassProperties3D (spPart->GetSolid());
```

```
        if (NULL != pPropOp)
        {
                cout << "le volume final de la piece est " << pPropOp->GetVolume() << endl;
          return pPropOp->GetVolume();
                delete pPropOp;
                pPropOp = NULL;

        }

// close the document and the session
rc = CATDocumentServices::Remove(*pDoc);
  if (FAILED(rc))
  {
   cout << "Error in closing document" << endl ;
   return 0;
  }
  pDoc = NULL ;

  // Deletes all documents in session
  rc = Delete_Session(pSessionName);
  if (FAILED(rc))
  {
   cout << "Error in deleting session" << endl ;
   return 0;
  }

  cout << "Le calcul du volume est termine" << endl;

  return 1;

}
```

2- Accés à CAA-Rade à partir de Java en utilisant une librairie partagée : CalculVolume.DLL

```
package test;


public class AppelCalculV {

private Thread lsThread=null;
static Process theProcess;

 public AppelCalculV(String s) {
  try {
   Runtime rt=Runtime.getRuntime();
   String[] cmd= {"cmd","/c","START Call java -classpath . CalculV "+s};
   Process proc = rt.exec (cmd);
   BufferedReader b =new BufferedReader(new
   InputStreamReader(proc.getInputStream()));
   String line = null;
   System.out.println("");
```

```
    while ( (line = b.readLine()) !=null)
    {System.out.println(line);
    System.out.println("");}
    int exitVal = proc.waitFor();
    System.out.println("Process exitValue: " + exitVal);
     } catch (IOException e1) {
      System.err.println(e1);
     }
      catch (java.lang.InterruptedException e1) {
       System.err.println(e1);
            }

}

}
```

```
import java.sql.*;

class CalculV {
   public native double donnevolume(String s);

   static {
      System.loadLibrary("CalculVolume");
   }

    public static void main(String[] args) {
    boolean rs ;
    Connection connexion= null;
    Statement ordre;
    String nom= args[0].substring(3,9);

   String url= "jdbc:odbc:contrainte";
   try {
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   connexion= DriverManager.getConnection(url);
    }
   catch (ClassNotFoundException e)   {}
   catch (SQLException e)   {System.out.println("la connexion n a pas marche");}


   //double nombre = new CalculV().donnevolume("c:\\cube.CATPart");
   double nombre = new CalculV().donnevolume(args[0]);
   Double nbr= new Double(nombre);
   System.out.println("voila le nombre que j ai "+nombre);


   try {

        ordre = connexion.createStatement();
```

```
        //String requete=  "INSERT INTO contrainte.attribut (" + "nom" + ") VALUES ( '" +
test+"')";;
        String requete = "UPDATE contrainte.datavolume SET " +
            "valeur='" + nbr.intValue() +
            "' WHERE nom = '" + nom + "'";
       //  String requete= "INSERT INTO valeurpiece (" +
        //   "nom, valeur" +
        //   ") VALUES ('" +
         //  nom + "', '" +
         //  nbr.intValue() + "')" ;

        //String requete = "UPDATE datavolume SET " + "type = " + "'"+test+"'"+
                  //"UPDATE datavolume SET " + "valeur = '"+ nombre +"'"+

                 //    " WHERE nom = '" + nom + "'";
          System.out.println( "\nEnvoi de la requête : " + connexion.nativeSQL( requete ) + "\n" );
          rs = ordre.execute( requete );
         //if ( rs == 1 )
        //{System.out.println( "\nMise à jour réussie\n" );}
        // else {System.out.println( "\nMise à jour non réussie\n" );}
    ordre.close();
    }
  catch ( SQLException sqlex ) {
    System.out.println(sqlex.getMessage());
    System.out.println("pb sql");
}

   }
}
```

Exemple d'un arbre de typage sous format XML

Un arbre de typage s'enregistre sous fornat XML voir la figure ci-dessous.