**Université de Montréal**

# Environnement de test d'un serveur de négociations électroniques

Par
**El mostafa Ben Najim**

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la faculté des études supérieures
en vue de l'obtention du grade de
Maître ès sciences (M. Sc.)
En informatique

Mai 2001



Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:
Environnement de test d'un serveur de négociations électroniques

Présenté par
El mostafa Ben Najim

A été évalué par un jury composé des personnes suivantes:

M. Michel Gendreau, président
Mme Rachida Dssouli, membre et directrice de recherche
M. Ferhat Khendek, membre et codirecteur de recherche
M. Jean Vaucher, membre

Mémoire accepté le: 12-07-01

# SOMMAIRE

Les nouvelles technologies de l'information sont de plus en plus utilisées pour atteindre de nouveaux objectifs commerciaux. Grâce à elles, les limites imposées dans le passé par le temps et l'espace sont réduites. L'inforoute a profondément bouleversé les pratiques commerciales traditionnelles comme en témoignent l'adoption des normes EDI (Electronic Document Interchange), l'expansion des réseaux et la création de modes sécuritaires d'encryptage et de paiement électronique. Les entreprises innovatrices ne tardent pas à se servir de ces technologies, nous seulement pour automatiser les processus d'affaires existants, mais les transformer en profondeur en vue de créer de nouveaux types de services. Les marchés électroniques en sont un exemple. Plusieurs types de ces marchés ont vu le jour. Ils varient d'un simple catalogue en ligne jusqu'aux marchés synchronisés tout en passant par les divers systèmes d'enchères. De nos jours, le commerce électronique n'est plus l'apanage des grandes entreprises et des réseaux privés. Le réseau ouvert internet, et surtout le « world wide web» ouvrent de nouveaux horizons commerciaux aux grandes entreprises en plus d'offrir aux moyennes et aux petites entreprises un point d'entrée viable. Cependant, la qualité de ces systèmes demeure la priorité majeure de ces entreprises. Le test constitue un des moyens pour l'assurance de cette qualité. De nos jours, plusieurs définitions du test sont utilisées. Chacune essaie de voir le test d'un angle différent. Cependant, elles convergent toutes sur son rôle de détection des erreurs de programmation et de conformité. Les pratiques traditionnelles de test s'avèrent peu efficaces dans ce nouveau contexte d'où la nécessité de prévoir de

nouvelles solutions de test de ces systèmes de plus en plus intégrés et de plus en plus compliqués.

Dans ce mémoire, nous proposons un environnement de test pour un serveur de marché électronique. Le but est de tester une plate forme expérimentale de négociations implantant divers mécanismes d'enchères électroniques. L'environnement tel que développé se compose d'un formalisme de description des scénarios de test conforme au paradigme MSC, d'une démarche facilitant la construction des scénarios suivant ce formalisme, et, finalement, d'un traducteur dont la fonction est de traduire les scénarios écrits en MSC en des scripts *Jpython* exécutables.

**Mots-clé** : tests – scénario – MSC – enchère électronique – traduction .

# TABLE DES MATIERES

Sommaire ..... i

Table des matières ..... iii

Liste des figures ..... vi

Liste des tables ..... vii

Remerciements ..... viii

1 Introduction ..... 1

2 Enchères électroniques ..... 6

2.1 Marché virtuel ..... 6
2.2 Le commerce électronique ..... 7
2.3 Quelques modèles d'enchères ..... 10
2.3.1 Enchère hollandaise ..... 10
2.3.2 Enchère fermée ..... 11
2.3.3 Enchère ouverte ..... 12
2.3.4 Enchère synchronisée ..... 13
2.3.5 Enchère ouverte synchronisée « multi-items ». ..... 13
2.3.6 Marché optimisé ..... 14
2.4 Efficacité des enchères ouvertes ..... 15
2.5 Enchère ouverte vs enchère fermée ..... 16
2.6 Mise en oeuvre des enchères ouvertes ..... 17

3 GNP : Description technique ..... 18

3.1 Architecture de GNP ..... 18
3.2 Modèles d'utilisation de GNP ..... 21
3.3 Définition XML des documents GNP ..... 23
3.3.1 Langage XML ..... 23
3.3.2 DTD : Création d'un vocabulaire commun ..... 24
3.3.3 DOM : Document Object Model ..... 25

3.3.4 Définition des documents XML ..... 26
3.4 Déroulement d'une négociation ..... 29
3.4.1 Envoi de l'annonce par modèle ..... 30
3.4.2 Envoi de l'annonce directement ..... 30
3.4.3 Séquence d'actions après l'envoi d'une annonce ..... 30

4 Spécification et test de logiciel ..... 32

4.1 Spécification ..... 33
4.1.1 Machines à états finis ..... 33
4.1.2 Machines à états finis communicantes ..... 34
4.1.3 Machines à états finis étendus ..... 35
4.1.4 Réseaux de Petri ..... 36
4.1.5 Langage Message Sequence Chart - MSC ..... 36
4.2 Techniques de test ..... 40
4.2.1 Test structurel ..... 42
4.2.2 Test fonctionnel ..... 43
4.3 Principaux types de test logiciel ..... 47
4.3.1 Test de conformité ..... 47
4.3.2 Test de performance ..... 48
4.3.3 Test de sécurité ..... 50
4.3.4 Test de configuration ..... 51
4.3.5 Test de robustesse ..... 51
4.4 Automatisation du processus de test ..... 51
4.5 Test des enchères électroniques ..... 52

5 Description de l'environnement de test ..... 53

5.1 Dérivation des scénarios de test ..... 53
5.2 Exemple de scénarios ..... 55
5.3 Verdict de test ..... 59
5.4 Architecture de l'environnement de test ..... 60
5.4.1 *Jpython* : Survol technique ..... 60
5.4.2 Architecture de l'environnement de test ..... 64
5.4.3 Module de traduction ..... 65
5.4.4 Module des paramètres ..... 66
5.4.5 Module des utilitaires ..... 66
5.4.6 Module des prédicats ..... 67

6 Expérimentations et résultats ..... 69

6.1 Présentation de l'enchère PSP ..... 69
6.1.1 Règles de l'enchère ..... 69

6.1.2 Constantes du marché ..... 70
6.1.3 Mises de la ronde ..... 70
6.1.4 Allocation intérimaire ..... 71
6.1.5 Calcul de l'allocation intérimaire ..... 72
6.2 Sélection des scénarios de test. ..... 73
6.3 Cas concret d'utilisation ..... 75
6.3.1 Présentation du scénario ..... 75
6.3.2 Traduction et exécution du scénario ..... 77
6.4 Architecture de test ..... 81

7 CONCLUSION ..... 83

Bibliographie ..... 86

ANNEXE A: Exemple de documents GNP en XML ..... 89

ANNEXE B: Types supportés par *Jpython* et opérations associées ..... 98

ANNEXE C: Scénarios de test ..... 103

ANNEXE D: Code source *Jpython* des modules ..... 109

## LISTE DES FIGURES


| | | |
|---|---|---|
| **Figure 2.1** | Niveaux de complexité des marchés électroniques | 07 |
| **Figure 3.1** | Architecture 5-tiers de GNP | 18 |
| **Figure 3.2** | Architecture en deux couches de GNP | 20 |
| **Figure 3.3** | Rôle des différents acteurs de GNP | 22 |
| **Figure 4.1** | Exemple d'une FSM | 34 |
| **Figure 4.2** | Système communicant à base de deux MEFCs. | 35 |
| **Figure 4.3** | Éléments de base de MSC | 37 |
| **Figure 4.4** | Processus de test dans le processus de développement | 42 |
| **Figure 4.5** | Architecture du test structurel | 43 |
| **Figure 4.6** | Architecture du test fonctionnel | 44 |
| **Figure 4.7** | Architecture de test | 45 |
| **Figure 4.8** | Architecture de test externe | 46 |
| **Figure 4.9** | Capture et rejoue du trafic brut HTTP | 49 |
| **Figure 4.10** | Capture et rejoue des événements de la souris et du clavier | 49 |
| **Figure 5.1(a)** | Scénario MSC/GR d'une annonce | 56 |
| **Figure 5.1(b)** | Scénario MSC/GR d'une mise | 56 |
| **Figure 5.2(a)** | Scénario MSC/PR d'une annonce | 56 |
| **Figure 5.2(b)** | Scénario MSC/PR d'une mise | 56 |
| **Figure 5.3** | Architecture de l'environnement de test | 65 |
| **Figure 6.1** | Valeur des mises possibles | 73 |
| **Figure 6.2** | Architecture de test de GNP. | 81 |
| **Figure 6.3** | Architecture de test externe de GNP. | 82 |

# LISTE DES TABLES

**Table 5.1**: Liste partielle des méthodes de la «session beans» AnnouncerSession . . . **54**
**Table 5.2**: Liste partielle des méthodes de la «session beans» NegotiatorSession . . . **54**
**Table 5.3**: Associations entre GNP et MSC . . . . . **55**
**Table 5.4**: Équivalences entre les éléments de GNP et le langage MSC . . . . . **57**
**Table 5.5**: Liste et taille des modules de test . . . . . **67**
**Table 6.1**: Exemple de tableau des allocations intérimaires . . . . . **71**
**Table 6.2**: Mises et résultats observés pour l'objectif (i) . . . . . **74**
**Table 6.3**: Mises et résultats observés pour l'objectif (ii) . . . . . **75**

# REMERCIEMENTS

Je remercie en premier lieu les professeurs Rachida Dssouli et Ferhat Khendek qui ont bien voulu encadrer mon travaille de recherche. Leur support et leurs remarques toujours pertinentes m'ont été d'une grande utilité.

Je remercie également Bell Canada et le CIRANO (Centre Inter-universitaire de recherche en Analyse des Organisations) Pour leur contribution financière et pour leur accueil chaleureux dans leurs locaux pendant une partie importante de ma recherche et particulièrement Robert Gérin-Lajoie pour ses brillantes idées et son aide inestimable.

Je saisis cette occasion pour remercier tous les membres de ma famille et en particulier mes parents pour leur encouragement et leur support moral.

Finalement j'aimerais remercier toute personne ayant contribué de près ou de loin à la réalisation de ce travail, notamment Ennouaary Abdesslam.

Merci à toutes et à tous.

CHAPITRE 1

# INTRODUCTION

En l'absence d'une place de marché définie et efficace, la recherche d'occasions d'affaires pour combler un besoin pressant ou pour écouler un stock est potentiellement coûteuse en termes de temps et d'argent. D'où l'intérêt pour les entreprises d'avoir accès à une place de marché virtuelle où elles peuvent en tout temps vendre et/ou acheter. Le développement de l'autoroute de l'information aura un impact majeur sur l'organisation des échanges et des relations entre firmes et sur l'accélération de la globalisation des marchés. Les technologies de l'internet ont et auront un impact important sur la structure de nos économies modernes. Ces technologies permettent non seulement d'informatiser les processus d'affaires existants, mais aussi de transformer les organisations et les échanges pour les rendre plus efficaces. L'inforoute a d'ailleurs déjà commencé à modifier en profondeur la valeur des échanges commerciaux, comme en témoignent l'adoption des normes EDI, l'expansion des réseaux de l'inforoute et la création de modes sécuritaires d'encryptage et de paiement électronique. Même les gouvernements entendent se convertir au commerce électronique, notamment en ce qui concerne les achats de biens et services vu qu'il est généralement très coûteux de réunir dans une même salle les concurrents à un appel d'offres. Les technologies de communication permettent aujourd'hui d'organiser des millions d'enchères virtuelles entre participants répartis à travers le monde, favorisant ainsi l'apparition et l'utilisation de nouvelles règles pour les appels d'offres.

Une des formes les plus simples d'un serveur de négociations électroniques peut être assimilée à un catalogue en ligne. Il s'agit d'une liste d'informations de produits à vendre ou à acheter, accessible à un nombre donné d'usagers à l'aide d'un ordinateur branché à

l'internet. Judicieusement nommé, le e-commerce « commerce électronique », est la combinaison des pratiques commerciales traditionnelles avec l'ordinateur et les technologies de l'information et des communications. Le e-commerce a toujours tiré profit des systèmes et des outils innovateurs. Quand de nouvelles technologies font leur apparition, les entreprises astucieuses ne tardent pas à repérer les occasions et à étendre leurs activités commerciales. De nos jours, le e-commerce n'est plus l'apanage des grandes entreprises et des réseaux privés. Le réseau ouvert internet, et surtout le World Wide Web, ouvrent de nouveaux horizons commerciaux aux grandes organisations, en plus d'offrir aux petites et moyennes entreprises (PME) un point d'entrée viable. Pour plusieurs organisations, la qualité de ces systèmes est de plus en plus importante. Progressivement, de plus en plus de mesures sont prises en vue d'améliorer cette qualité. Malgré les résultats encourageants enregistrés dans ce domaine, l'objectif d'un système avec zéro défaut est encore à trouver. Les causes des défauts sont variées et imprévisibles, aussi faudra-t-il, par conséquent, consacrer plus de temps et d'efforts pour les traquer. Le test est, et va encore rester pour longtemps, une activité importante dans le processus de développement de logiciel, coûtant plus de 30-40% du budget global de l'activité du développement logiciel [Cou 94]. L'étape de test est celle qui garantit une certaine qualité d'une implantation. Elle consiste à valider le comportement de l'implantation par rapport à des données de test (cas de test) sélectionnées à l'avance. L'objectif de cette étape est la stimulation des fautes, l'observation de leurs manifestations (erreurs), leur localisation et éventuellement leur correction. Plusieurs définitions du test existent. En voici quelques unes [Bou 99]:

- *Le processus de test consiste en une collection d'activités qui visent à démontrer la conformité d'un programme par rapport à sa spécification. Ces activités se basent sur une sélection systématique des cas de test et l'exécution des segments et des chemins du programme* [Was 77].

- *Les tests représentent l'essai d'un programme dans son milieu naturel. Ils nécessitent une sélection de données de test à soumettre au programme. Les résultats des traitements de ces données sont alors analysés et confirmés. Si on découvre un résultat erroné, l'étape de débogage commence* [Cla 76].

- *L'objectif du processus de test est limité à la détection d'éventuelles erreurs d'un programme. Tous les efforts de localisation et de détection et de correction d'erreurs sont classés comme des tâches de débogage. Ces tâches dépassent l'objectif des tests. Le processus du test est donc une activité de détection d'erreurs, tandis que le diagnstic est une activité plus difficile consistant en la localisation et la correction des erreurs détectées* [Whi 78].

Durant les dernières années, l'activité de test a assez mûri. Des outils et des techniques de test d'un niveau satisfaisant sont disponibles. Cependant, le futur a constamment besoin de nouvelles solutions. L'important défi au test est constitué principalement de l'émergence des nouvelles technologies de l'information et de l'apparition d'une multitude de nouveaux standards.

Pour notre mémoire, nous avons developpé un environnement de test automatique pour un Serveur de Négociations électroniques ("Generic Negotiation Platform" ou GNP). GNP est développé dans le cadre du projet TEM (Toward Electronic Marketplaces) mené par l'équipe de recherche sur les marchés électroniques au CIRANO (Centre Interuniversitaire de Recherche et Analyse des Organisations : www.cirano.qc.ca). L'objectif de ce projet est le développement des compétences scientifiques et techniques nécessaires à la mise en place de mécanismes de marchés, d'échanges et d'enchères sur internet susceptibles d'être greffés aux Serveurs de Négociations Électroniques. Le GNP sert alors de plate-forme d'expérimentation. Grâce à sa flexibilité, plusieurs algorithmes économiques se basant presque tous sur le mécanisme des enchères ouvertes sont implantés dans GNP. Notre objectif est de tester la conformité de l'implantation de ces

algorithmes par rapport à leurs spécifications de référence. L'environnement de test proposé comporte trois éléments:

- un formalisme de description des scénarios de test,
- une démarche pour la dérivation des scénarios,
- un traducteur.

Le formalisme de description des scénarios de test est conforme au langage MSC (Message Sequence Chart) [Z.120]. La démarche à suivre pour la dérivation des scénarios apporte une assistance pour exprimer ces scénarios de façon conforme au formalisme. La fonction du traducteur est de transformer ces scénarios en des robots de test. Ces scripts exécutables jouent le rôle de testeurs. Le verdict de test est obtenu pendant l'exécution des tests.

Le présent mémoire est organisé de la manière suivante :

Dans le chapitre 2, nous présentons une description des marchés électroniques. Nous nous attardons d'avantage sur les enchères électroniques qui présentent un niveau de complexité plus élevé qu'un simple catalogue en ligne. Plusieurs modèles d'enchères sont alors abordés. Ensuite nous faisons une comparaison entre les mécanismes de l'enchère ouverte et ceux de l'enchère fermée.

Dans le troisième chapitre, nous faisons un survol technique de la plate-forme GNP. Cette plate-forme de négociations électroniques utilise le standard XML pour véhiculer les données entre ses différents composants. La manipulation des documents XML est faite par une implantation Java de l'interface DOM (Document Object Model). Nous décrivons brièvement dans ce chapitre ces deux technologies avant d'aborder le rôle de chaque document XML manipulé par GNP.

Dans le quatrième chapitre, nous décrivons dans la première partie l'activité du génie logiciel et les cycles de développement des logiciels. Dans une deuxième partie, nous

présentons l'intérêt des modèles formels de spécification ainsi que quelques modèles et langages très utilisés surtout dans le domaine des télécommunications. Nous abordons successivement les machines à états finis, les machines à états finis communicantes, les machines à états finis étendus et les réseaux de Pétri. Finalement, nous introduisons le langage MSC que nous avons choisi pour la description des scénarios de test.

Nous consacrons le cinquième chapitre à la description de notre environnement de test. Nous commençons ce chapitre par présenter le formalisme utilisé pour dériver les scénarios de test à partir des «session beans » de GNP. Le traducteur et le rôle de chaque module le composant sont ensuite présentés.

Le chapitre 6 est réservé à l'expérimentation de l'environnement de test. Nous présentons la version GNP de l'enchère de Vickrey, connue également sous le nom PSP (Progressive Second Price). Une étude de couverture des scénarios de test dérivés par rapport à des objectifs déterminés, ainsi que les résultats obtenus sont décris en fin du chapitre. Nous concluons dans le dernier chapitre.

CHAPITRE 2

# ENCHERES ELECTRONIQUES

## 2.1 Marché virtuel

En l'absence d'une place de marché définie et efficace, la recherche d'occasions d'affaires pour combler un besoin pressant ou pour écouler un stock est potentiellement coûteuse, en termes de temps et d'argent. D'où l'intérêt pour des entreprises d'avoir accès à une place de marché où elles peuvent en tout temps vendre ou acheter. Dans certaines industries, l'alimentation par exemple, la place du marché traditionnelle où les négociations et les échanges se font en personne peut être toujours une pratique. Par contre, dans un marché où les distances entre acheteurs et vendeurs sont importantes, une place de marché virtuel peut constituer toute une amélioration au « magasinage » traditionnel. Typiquement, une place de marché virtuel est le serveur informatique auquel peuvent accéder, via l'internet, acheteurs et vendeurs.

La collecte des informations nécessaires pour conclure une transaction avantageuse (prix, quantité disponible, qualité, caractéristiques du produit, etc.) demeure très coûteuse. Les nouvelles technologies de l'information et des communication sont en voie d'abolir cet obstacle.

La variété des technologies de l'internet a et aura un impact important sur la structure de nos économies modernes. Ces technologies permettent non seulement d'informatiser les processus d'affaires existants, mais aussi de transformer les organisations et les échanges pour les rendre plus efficaces.

Dans sa forme la plus simple, un serveur de négociations électroniques (SNE) peut être assimilé à un catalogue en ligne. Il s'agit d'une liste d'informations de produits à vendre ou à acheter, accessible à un nombre donné d'usagers à l'aide d'un ordinateur branché à l'internet.

Le niveau de complexité d'un SNE est d'autant plus élevé que les procédures électroniques sont fortement impliquées dans le processus de négociation (voir Figure 2.1).

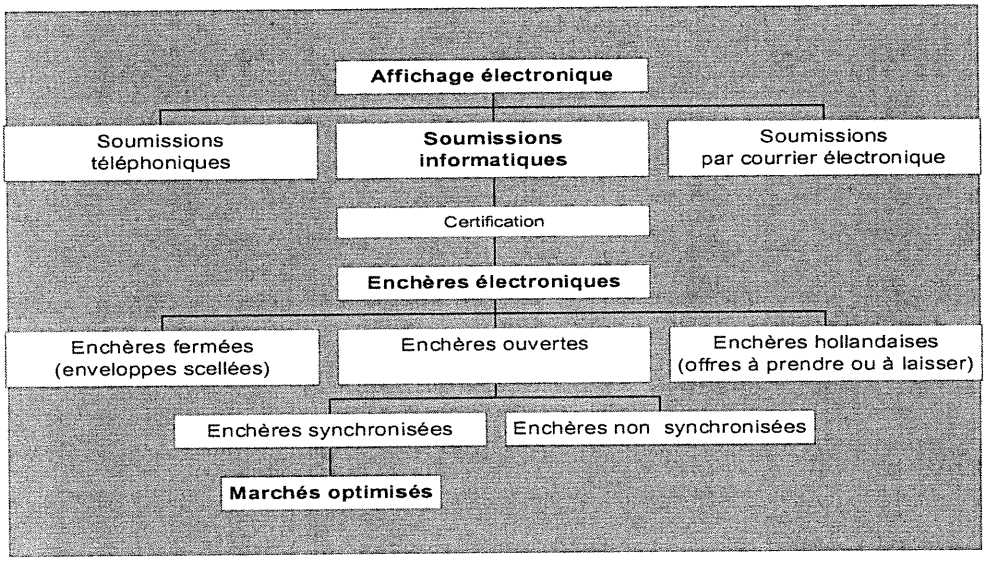


Figure 2.1 Niveaux de complexité des marchés électroniques

Dans sa forme la plus simplifiée, un SNE se résume donc à une série de pages web où des acheteurs peuvent exprimer leurs besoins, et des vendeurs peuvent offrir leurs stocks disponibles.

## 2.2 Le commerce électronique

Les systèmes de commerce électronique (e-commerce) sont des instances spécialisées des systèmes distribués. Un système distribué est un système dans lequel le traitement de l'information est distribué entre plusieurs ordinateurs au lieu d'être confié à une seule

machine. Évidemment, l'ingénierie de ces systèmes a beaucoup en commun avec l'ingénierie des autres systèmes logiciels. Cependant, certains aspects qui leur sont spécifiques doivent être pris en considération. Coulouris et al. [Cou 94] identifient six caractéristiques des systèmes distribués :

1. Partage des ressources : Un système distribué permet le partage des ressources logiciels et matériels comme les disques, par exemple, les imprimantes, les fichiers, les processeurs, etc. qui sont associés à d'autres ordinateurs sur le réseau.

2. Ouverture : L'ouverture d'un système dépend de sa capacité à être étendu par l'ajout de nouvelles ressources non propriétaires. Les systèmes distribués sont des systèmes ouverts qui, normalement, incluent des logiciels et matériels provenant de plusieurs constructeurs différents.

3. Concurrence : Dans un système distribué, plusieurs processus peuvent tourner en même temps sur différents ordinateurs ou processeurs sur le réseau. Ces processus peuvent communiquer entre eux durant leur fonctionnement normal.

4. Capacité de monter en charge : Au moins en principe, les systèmes distribués sont capables de monter en charge dans la mesure où leur capacité peut être augmentée en ajoutant de nouvelles ressources capables de répondre aux nouveaux besoins du système. Cette montée en charge peut cependant être limitée par les liaisons réseau reliant les ordinateurs individuels. Si plusieurs autres ordinateurs sont ajoutés au système, alors la capacité du réseau peut devenir inadéquate.

5. Tolérance aux pannes : La disponibilité de plusieurs ordinateurs et la puissance de réplication des informations signifient que les systèmes distribués sont tolérants à certaines pannes logicielles et matérielles. Dans la plupart des systèmes distribués, un service, bien que dégradé, peut toujours être fourni même si une panne se produit.

L'arrêt définitif du service peut néanmoins survenir si, par exemple, une panne réseau se produit.

6. Transparence : La transparence du système est relative à l'ignorance totale de l'usager de sa nature distribuée. L'objectif de la conception du système peut être basé sur une transparence complète de la part des usagers les déchargeant de savoir ce qui se passe dans les coulisses du système tout en leur permettant un accès complet à ses fonctionnalités. Cependant et dans plusieurs cas, une connaissance de l'organisation du système favorise un meilleur usage de la part des utilisateurs.

Ceci va sans dire que ces systèmes présentent également quelques désagréments :

1. Complexité : Les systèmes distribués sont plus complexes que leurs homologues centralisés. Ceci rend la compréhension de leurs fonctionnalités plus difficile et leur test plus ardu. Par exemple, au lieu que leur performance dépende de la puissance d'un seul processeur, celle-ci est fonction de plusieurs autres facteurs dont la puissance des processeurs des autres ordinateurs, la bande passante du réseau, la charge du réseau etc. en outre, Migrer des ressources d'une partie du système vers une autre peut changer radicalement la performance globale du système.

2. Sécurité : Un système distribué peut être accédé de plusieurs points (ordinateurs) différents et le trafic réseau peut être mis en écoute. Ceci rend l'assurance de la sécurité des systèmes distribués parmi les tâches les plus difficiles.

3. Gestion : Les différents ordinateurs du système peuvent être de plusieurs types et peuvent fonctionner à base de différents systèmes d'exploitation. Une erreur survenant sur une machine peut se répercuter fatalement sur les autres machines avec des conséquences imprévisibles. Ceci implique des efforts supplémentaires pour la gestion et le maintien en marche de ces systèmes.

4. Imprévu: Comme tous les usagers du WWW le savent, les systèmes distribués sont imprévisibles dans leurs réponses. La réponse dépend de la charge totale du système, de son organisation et de la charge du réseau. Comme tous ces éléments peuvent changer dans un temps très bref, le temps nécessaire pour répondre à une requête d'usager peut considérablement varier d'un moment à l'autre.

## 2.3 Quelques modèles d'enchères

Les mécanismes d'enchères électroniques sont couramment utilisés pour l'émission des bons de Trésor ou des valeurs boursières, pour l'attribution de marchés, pour des concessions d'exploitation, pour la vente d'œuvres d'art, etc. De tels mécanismes créent un cadre compétitif censé faire émerger le « prix du marché ». Leur caractère équitable en fait un cadre particulièrement attrayant, cadre qui est d'ailleurs la doctrine officielle de l'administration en matière d'achats publics. Dans la section suivante, nous allons présenter quelques célèbres modèles d'enchères.

### 2.3.1 Enchère hollandaise

À supposer que le prix soit le seul critère d'évaluation d'une soumission (ce qui est le cas lorsque l'offre d'achat ou de vente est définie avec précision), la manière la plus élémentaire de procéder consiste à afficher sur le babillard électronique une offre de vente à prendre ou à laisser. Typiquement, un participant affiche une offre à un prix donné et se déclare prêt à recevoir des soumissions pour une période de temps limitée. Le premier acheteur intéressé se voit octroyer l'item mis en vente et la transaction est officialisée de quelconque façon. Par contre, si aucun acheteur ne se déclare intéressé par l'item en question avant que la période de soumission ne vienne à échéance, le participant désireux de conclure une transaction doit publier une nouvelle offre. Dans le cas d'un vendeur, il lui faut annoncer un nouveau prix, plus bas que celui de l'offre de vente initiale, et une nouvelle limite de temps. Le vendeur réduit ainsi son prix jusqu'à ce qu'un acheteur se déclare intéressé par le lot au prix demandé.

Un processus semblable peut aussi bien se dérouler de l'autre « côté » du marché, c'est-à-dire par la publication d'une offre d'achat à prendre ou à laisser. Un participant affiche au babillard une offre d'achat pour une quantité donnée d'un produit répondant à des critères précis à un prix donné, initialement le plus bas possible. Il augmente le prix qu'il offre tant et aussi longtemps qu'aucun vendeur ne se déclare intéressé à vendre cette quantité du produit demandé (répondant aux critères établis dans l'offre d'achat) au prix offert.

Ce processus simple peut s'interpréter comme un encan ; à strictement parler, il s'agit de l'enchère hollandaise. Dans ce type d'encan, le rôle de l'opérateur se limite à «entretenir» un babillard électronique et à en assurer, selon certaines règles, l'accessibilité aux entreprises. Il n'est pas nécessaire d'avoir recours à des soumissions informatiques. Le processus n'est alors pas uniformisé et les annonceurs disposent de toute la discrétion voulue dans le traitement des soumissions. La discrimination est possible, ce qui peut être considéré ou non comme désirable par les entreprises d'une industrie. De plus, ce type de processus de soumission n'est pas anonyme, en ce sens qu'un annonceur connaît en tout temps l'identité de tous les soumissionnaires. Toutefois, un mécanisme de marché objectif et non discriminatoire est souvent considéré par une entreprise comme une condition de participation.

### 2.3.2 Enchère fermée

Les soumissions par enveloppes scellées sont un bon exemple d'un mécanisme de négociation conçu dans une optique d'absence de discrimination. De nombreux appels d'offre gouvernementaux se déroulent selon ce processus, où le montant des soumissions n'est connu qu'à la fin de la période de mise, lors de l'ouverture des enveloppes. On appelle enchère fermée ce type de processus. L'application électronique de ce mécanisme implique un rôle plus important pour l'opérateur de marché, qui doit ici être une tierce partie objective et neutre. Le montant des soumissions demeure secret jusqu'à la fin de la

période déterminée pour la réception des mises, c'est-à-dire que ni l'annonceur ni les autres participants n'ont accès à cette information. Elle est recueillie et archivée par l'opérateur du marché. Les soumissions par courrier électronique ou les soumissions informatiques peuvent être utilisées.

À la fin de l'enchère, l'ensemble des soumissions respectant les critères établis sont transmises à l'annonceur. Il est également possible que le meilleur soumissionnaire soit automatiquement identifié par le système informatique et que seule cette soumission soit transmise à l'annonceur. Le système peut aussi gérer l'officialisation de la transaction, par exemple en expédiant un courrier électronique aux deux parties.

### 2.3.3 Enchère ouverte

Des enchères sont ouvertes quand les mises envoyées par certains participants sont connues de tous les autres participants qui peuvent réagir en conséquence. Cela entraîne un processus dynamique d'enchères dans la mesure où chaque participant connaît à chaque instant la valeur de la mise «leader» de l'enchère et peut la dépasser pour en devenir «leader» lui-même. Les enchères à la criée, qui rassemblent commissaire-priseur et participants dans une même salle, en sont l'exemple le plus courant. Le serveur électronique de négociation GNP est habilité à gérer l'ensemble des variantes associées aux enchères ouvertes. Une enchère ouverte, telle que gérée par GNP, comprend en principe deux phases : la phase continue et la phase chronométrée. La phase continue est de durée fixe, les mises sont reçues par le serveur qui détermine en tout temps la mise «leader». À la fin de cette phase, l'enchère ne se ferme pas mais entre en phase chronométrée, ou encore en prolongation. À partir de ce moment, le processus de mise reste inchangé (ou peut admettre quelques changements de règles), mais la fin effective de la négociation va dépendre de l'activité des participants : l'enchère se ferme si aucune mise n'est envoyée pendant un délai fixé par l'opérateur appelé délai *d'adjudication*. Tant que des mises sont reçues à intervalle inférieur au délai, l'enchère reste en prolongation.

Il est à noter que l'enchère fermée vue précédemment, peut être appréhendée comme une variante de l'enchère ouverte. Ainsi, une enchère fermée peut être tenue à l'aide d'un mécanisme d'encan électronique ouvert en limitant l'information disponible au cours de la période de mise et en réduisant à une le nombre de soumissions qu'un participant peut envoyer. La notion de mise «leader» et de phase chronométrée n'existe plus, une *mise gagnante* de l'enchère est déterminée à l'heure de fermeture prévue.

### 2.3.4 Enchère synchronisée

Dans un mécanisme d'enchères ouvertes en présence de plusieurs négociations simultanées, on peut envisager de les synchroniser entre elles. Des négociations simultanées ont des heures d'ouverture et de fermeture identiques. Elles sont synchronisées à partir du moment où leurs phases chronométrées sont aussi de durées identiques. Ainsi, toutes les enchères se fermeront si aucune mise n'est reçue sur aucune des enchères pendant un délai d'adjudication entier. Grâce à la technologie des enchères synchronisées, l'opérateur de marché peut décider de synchroniser toutes ou certaines enchères selon une périodicité donnée (une fois par jour, par semaine, par mois, etc.). On parle alors d'un marché périodique synchronisé. Tous les items y sont négociés et adjugés en même temps. En terme d'efficacité, le principal avantage d'un marché périodique est la concentration des offres d'achat et de vente dans le temps, ce qui facilite la création d'une masse critique de transactions sur le marché.

### 2.3.5 Enchère ouverte synchronisée « multi-items ».

Le contraste entre les enchères ouvertes et fermées devient plus marqué dans le cas d'items multiples, c'est-à-dire lorsque plusieurs items complémentaires sont mis en appel d'offres. Il ne fait aucun doute qu'une enchère fermée simultanée pour plusieurs items soit un très mauvais choix. Considérons par exemple l'allocation de routes de déneigement ou de prélèvement des ordures. Un entrepreneur pourrait être capable d'offrir ses services pour un nombre limité de routes et aimerait si possible obtenir des routes voisines. Cependant, dans le cas d'enchères scellées, comment peut-il miser ? Doit-il miser sur

toutes les routes pour au moins obtenir le nombre désiré au risque d'en obtenir plus qu'il ne peut fournir ? Doit-il miser agressivement sur un nombre limité de routes et si oui lesquelles ? On voit bien les difficultés imposées par les enchères scellées aux fournisseurs. Au moment de miser sur une route particulière, ces derniers ne savent pas quelles autres routes ils vont gagner aux enchères. Un appel d'offres par enveloppe scellée ne fournit pas l'information nécessaire pour éviter le regret des participants et assurer une allocation efficace des contrats. Il devient en effet impossible pour les participants de miser intelligemment sur une série d'items sans savoir lesquels ils obtiendront.

Un mécanisme ouvert où les items sont adjugés simultanément et où la compétition reste ouverte sur tous les items tant que de nouvelles mises sont soumises est une conception beaucoup plus appropriée. Ce mécanisme permet aux participants de modifier leurs stratégies au fur et à mesure que les prix évoluent, leur permettant ainsi de se constituer des combinaisons efficaces d'items. Le processus de découverte dynamique des prix permet d'augmenter substantiellement l'efficacité.

### 2.3.6 Marché optimisé

A un niveau de complexité plus élevé (voir Figure 2.1), on retrouve les marchés optimisés. Un marché optimisé est un marché centralisé qui optimise les échanges après avoir recueilli une certaine quantité d'informations auprès des participants. Il gère les offres et les demandes des agents du marché de manière à déterminer les quantités de marchandises échangées et le montant des transactions. Contrairement à ce qui se passe sur un marché continu ou décentralisé, toute l'information disponible (disposition à payer de tous les participants, coûts du transport, exigences techniques, qualité, etc.) est utilisée par le marché optimisé afin de déterminer l'allocation des ressources qui maximise les profits d'une industrie dans son ensemble.

## 2.4 Efficacité des enchères ouvertes

L'objectif des enchères (pour les ventes ou pour les appels d'offres) est essentiellement de fixer le prix et d'allouer une ressource, un droit ou un contrat dans un environnement incertain. Chaque enchère vise donc à répondre à deux questions : qui devrait obtenir l'item et à quel prix ?

L'intérêt des mécanismes d'enchères ouverts est de répondre à ces deux questions de la manière la plus transparente possible. Dans le cas d'un appel d'offres, celui qui obtient le contrat est celui qui est prêt à demander le moins pour l'obtenir et il reçoit le montant tout juste nécessaire pour battre ses concurrents.

La caractéristique des enchères ouvertes est d'offrir un système dynamique et transparent de découverte des prix. Dans une enchère ouverte, la meilleure offre est publique et les agents sont invités à battre ce prix. Ainsi, les prix montent progressivement dans le cas d'une vente ou descendent dans le cas d'un appel d'offres. L'enchère s'arrête lorsque, après un certain temps, aucune nouvelle offre n'est soumise et que personne n'accepte de battre le meilleur prix.

Le résultat d'une enchère ouverte est le moins contestable qui soit : les perdants ont tous eu l'opportunité de faire mieux, mais ont décliné cette possibilité, ils ne peuvent donc pas se plaindre du résultat de l'enchère. Le gagnant doit offrir un prix légèrement supérieur (vente) ou légèrement inférieur (appel d'offres) aux autres pour gagner, il ne peut se plaindre de payer trop ou de recevoir trop peu par rapport à ce que la concurrence était prête à offrir.

Finalement, il est toujours difficile d'évaluer le coût d'un contrat. Or dans une enchère ouverte, le processus dynamique de négociation permet aux participants d'obtenir de l'information des autres participants et de réévaluer en conséquence les coûts attendus du contrat avant qu'il ne soit trop tard. Ceci permet de limiter le problème associé à ce que l'on appelle la malédiction du gagnant. Cet avantage des enchères ouvertes est peut-être moins évident à priori, mais est l'un des plus cruciaux.

## 2.5 Enchère ouverte vs enchère fermée

La principale vertu du processus de découverte de prix dans une enchère ouverte est l'efficacité. On dit qu'une enchère est efficace si elle alloue l'item (le contrat) à celui qui est le plus disposé à l'obtenir. Dans le cas d'une enchère pour un item, les enchères ouvertes sont efficaces sauf dans des cas très exceptionnels. Il est, par contre, très exceptionnel que les enchères fermées soient efficaces [Rob 99].

Pour comprendre pourquoi, il faut se rendre compte que dans une enchère ouverte, les stratégies de participation sont très simples : il suffit d'enchérir tant que le prix à battre est acceptable. L'enchère va s'arrêter lorsque le deuxième meilleur participant préfère se retirer.

Dans une enchère ouverte, le prix payé par le gagnant correspond donc, à un incrément près, à la mise ultime du second meilleur participant. Le gagnant n'a pas à révéler jusqu'où il était prêt à aller. La simplicité des stratégies permet de limiter les erreurs stratégiques et les tentatives de manipulation du marché.

Dans une enchère fermée où le gagnant paie (ou reçoit) sa mise, les stratégies des participants sont complexes, même dans le cas d'une enchère pour un seul item. Comme aucune information sur les intentions des autres participants n'est révélée durant le processus même de l'enchère, un participant doit prévoir le comportement des autres participants pour fixer sa mise et sa marge bénéficiaire. Ceux qui se débrouillent le mieux sont ceux qui connaissent bien l'état de la concurrence et du marché. Dans ce contexte, il est plus utile de se demander ce que les autres vont miser que de faire une évaluation précise de son propre seuil de rentabilité. Comme la marge bénéficiaire dépend de considérations stratégiques, celui qui mise le plus bas dans un appel d'offres n'est pas nécessairement celui qui aurait été capable de produire l'item au plus bas coût. Ainsi l'allocation n'est pas toujours efficace et certaines firmes peuvent regretter à posteriori de ne pas avoir misé plus bas ou plus haut.

## 2.6 Mise en oeuvre des enchères ouvertes

Pour mettre en place un mécanisme d'enchère ouverte sur l'internet, il faut créer une succession de courtes rondes de mises. À la fin de chaque ronde, la meilleure offre est affichée et les participants sont invités à soumettre de nouvelles mises. Le temps entre les rondes est contraint par une série de considérations techniques : le temps requis pour traiter les mises, la variation dans les délais réseau, le nombre de participants, etc. De plus, il faut éviter d'avoir un nombre fixe de rondes. Le danger dans ce cas est que les participants attendent la toute dernière ronde, ne laissant pas la possibilité aux autres de réagir tel que souhaité. Les règles d'arrêt doivent se baser sur un critère d'activité : par exemple, le mécanisme s'arrête après 15 minutes d'inactivité. Des règles d'activités individuelles peuvent aussi être mises en oeuvre : par exemple, un participant est exclu de l'enchère s'il n'a pas soumis une mise pertinente lors des 15 dernières minutes. La durée des rondes, les règles d'activités et d'arrêt sont des éléments importants de la conception et de la mise en oeuvre des mécanismes ouverts d'enchères sur l'internet.

Un autre des éléments à considérer est la possibilité d'élargir le concept de mise en permettant d'inclure un prix ferme et un prix limite. Une mise avec prix limite peut être interprétée comme une instruction donnée par le participant au marché de surenchérir en son nom, et ce, seulement si nécessaire jusqu'à ce que ledit prix limite soit atteint. Les mécanismes d'enchères sur l'internet permettent généralement de telles mises. C'est le cas notamment de e-Bay [http:// www. ebay. com] qui est classé numéro 1 mondial dans le domaine. Dans les enchères de e-Bay, si certains se contentent de soumettre uniquement des prix fermes, les plus expérimentés offrent des mises avec prix limite. L'avantage de telles mises est que le participant n'a pas à suivre à la minute le déroulement de l'enchère tout en restant assuré que son prix s'ajustera à ce qui se passe sur le marché.

CHAPITRE 3

# GNP : DESCRIPTION TECHNIQUE

## 3.1 Architecture de GNP

La plate-forme de négociation générique (GNP) dans sa version Alpha (septembre 2000) est conçue pour s'insérer dans une place de marché telle que décrite précédemment. Le GNP est un système multi-composants avec 5 composants [Gér 01] (voir le schéma de la Figure 3.1):

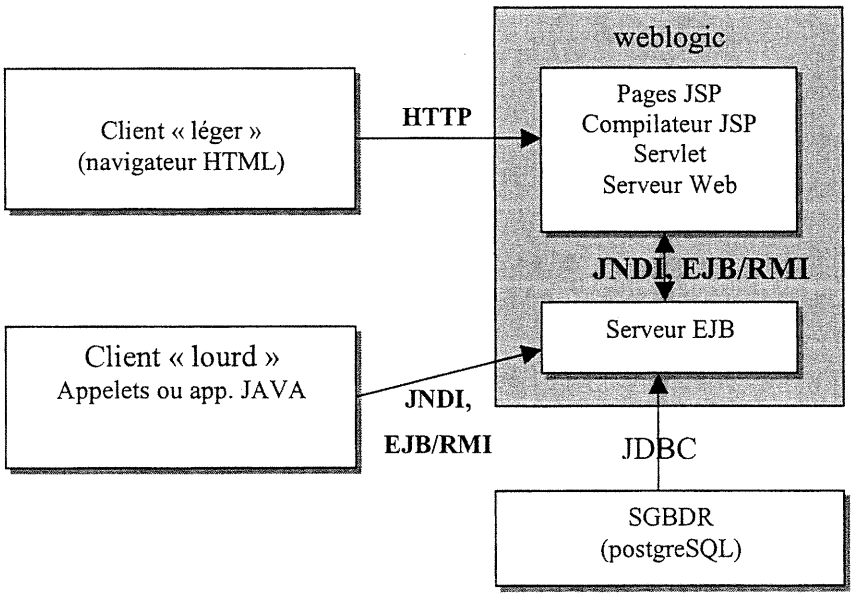


Figure 3.1 Architecture 5-tiers du GNP

- Les participants (client lourd/légérs) : acheteurs, vendeurs et régisseurs de marché rejoignent les serveurs GNP avec leurs fureteurs, soit par des pages HTML, soit par des composants Java (des Applets) activés à l'intérieur du fureteur ou directement dans des agents Java ou *Jpython* ;
- Le portail HTTP de contenu (serveur Web), utilisant des servlets Java et des pages « Java Server Pages », JSP [06]. Ce serveur est bâti sur l'infrastructure J 2EE fournit par le serveur BEA-Weblogic;
- Le serveur EJB de requêtes, utilisant les « Enterprises Java Beans » pour les sessions et les entités [01, 02];
- Le serveur des encanteurs, des « Enterprises Java Beans » activés par des messages asynchrones « Java Message Service » (JMS);
- Le serveur de données SQL. GNP utilise la base de données PostgreSQL et Oracle8i.

Ce système est conçu pour supporter de nouvelles règles économiques nécessaires aux système de négociations inter-entreprises (Business to Business ou B2B). Il est bâti autour de plusieurs concepts parmi lesquels on retient essentiellement des négociations conduites avec des phases et des rondes (voir le paragraphe 2.3.3 du deuxième chapitre). Les négociations se déroulent sur plusieurs phases lorsque les règles économiques changent au cours du déroulement de la négociation. Si par contre ces règles demeurent inchangées, alors la négociation se déroule dans une seule phase. Chaque phase est composée d'une ou plusieurs rondes. Une ronde est une suite de trois événements: (i) Une cote émise par l' encanteur, suivie par (ii) une ou plusieurs mises des participants et finalement (iii) l'évaluation par l'encanteur des mises reçues qui donne lieu à une allocation temporaire. Ce cycle se poursuit avec une nouvelle ronde et ainsi de suite.
Le GNP supporte également plusieurs modèles économiques grâce à une architecture flexible isolant tout ce qui est économique (documents d'affaires et scripts économiques)

de tout ce qui est purement informatique (les sessions et les entités « Entreprise Java Beans») (voir pour cela la Figure 3.2).

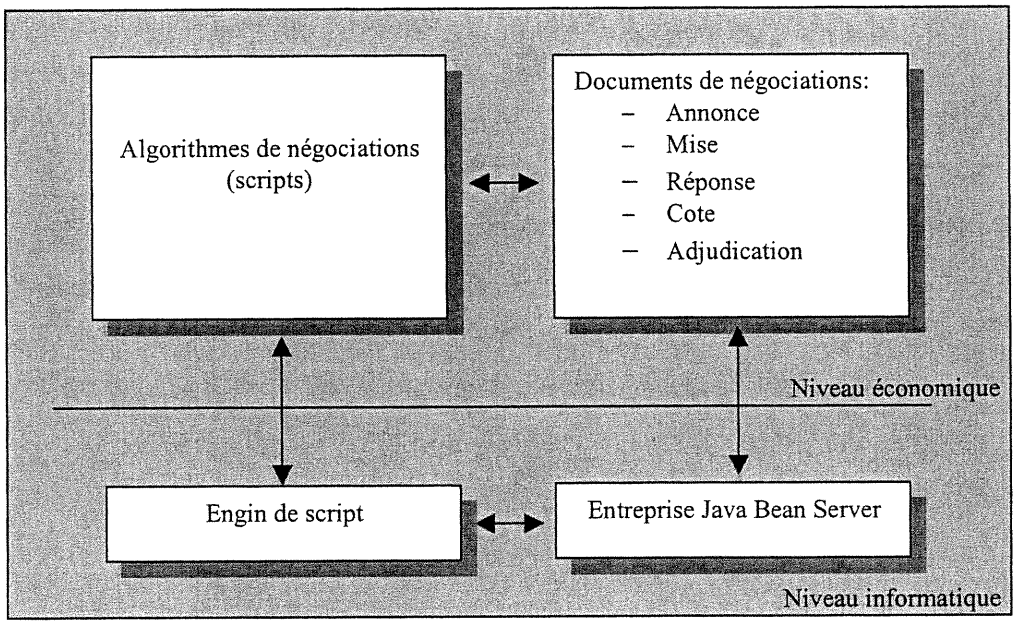


Figure 3.2 Architecture en deux couches de GNP

Les modèles d'enchères supportés par GNP sont :

- Enchère anglaise :

Reçoit une ou plusieurs mises pour chaque ronde et ferme après une période d'inactivité. Une enchère anglaise demande un nouveau prix à battre pour chaque nouvelle ronde.

- Enchère hollandaise :

Demande une mise sur un prix pour chaque nouvelle ronde. Si aucune mise n'est reçue pendant la période de la ronde alors le prix demandé est revu à la baisse (vente) ou la hausse (achat). L'enchère ferme dès la première mise reçue.

- Enchère cadencée :

Demande à tous les participants s'ils acceptent le nouveau prix à chaque ronde et ferme lorsqu'un seul participant accepte le prix proposé.

- Enchère synchronisée :

Utilise la même chorégraphie pour piloter des négociations sur des items reliés.

## 3.2 Modèles d'utilisation de GNP

En fonction du type de l'interaction d'un participant avec Le GNP, celui-ci lui accorde un rôle bien déterminé (Figure 3.3) [Gér 01].

**Lecteur - « Reader »**

Tous les participants peuvent à priori être des lecteurs. Ils peuvent consulter les produits, les annonces d'achat et de vente, voir les cotes publiques et finalement sélectionner celles sur lesquelles ils vont réagir et soumissionner.

**Annonceur - « Announcer »**

Un annonceur crée une négociation. Il annonce un intérêt d'achat ou de vente d'un ou plusieurs produits ou service. Les 3 étapes de la création d'une négociation sont :

- Créer les règles de la négociation à partir de modèles existants.
- Créer les références aux objets de la négociation, produits ou services.
- Créer un ordre initial d'achat ou de vente, définissant le prix et la quantité de départ.

**Négociateur et soumissionnaire - « Negotiator and Submitter »**

Un négociateur émet les ordres d'achats ou de vente. Un acheteur et un vendeur peuvent donc être négociateurs. Le négociateur émet l'ordre initial d'achat ou de vente. Le soumissionnaire est un négociateur répondant à une cote avec un ordre d'achat ou de vente.

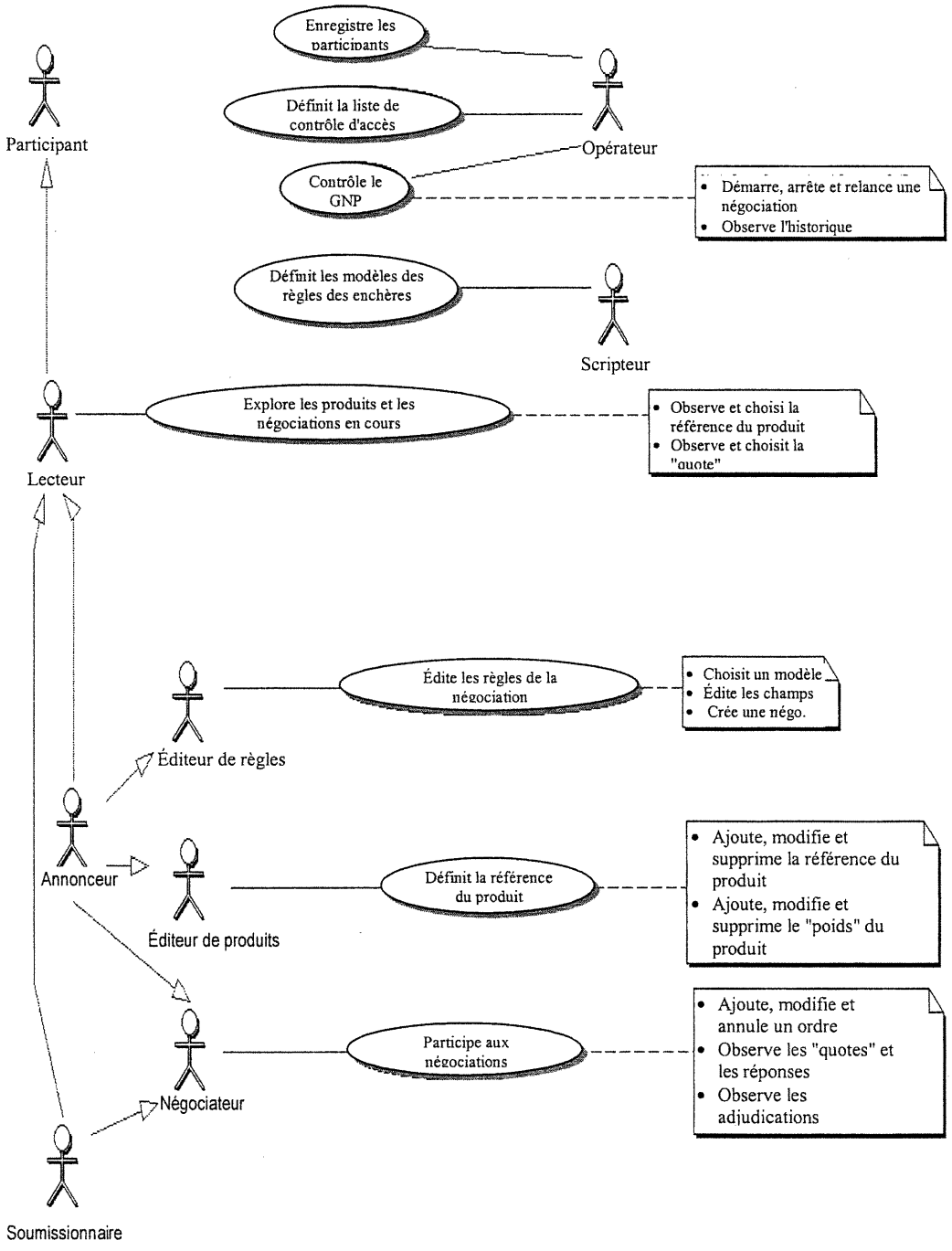


Figure 3.3 Rôle des différents acteurs de GNP

**Administrateur et opérateur - « Administrator and Operator »**

Les administrateurs et les opérateurs gèrent les participants et leurs propriétés, les marchés et les négociations. Sur les marchés et les négociations, ils peuvent non seulement les ajouter, modifier et enlever, mais aussi les démarrer, suspendre et repartir.

## 3.3 Définition XML des documents GNP

Toutes les données manipulées par le GNP, que ce soit pour la description des règles économiques, ou pour la sauvegarde des calculs intermédiaires, utilisent le format standard XML. Ces documents sont par la suite sauvegardés dans une base de données dans leur forme brute. Cependant, pour accélérer certaines opérations de recherche, quelques attributs importants sont extraits de certains documents et sauvegardés comme champs de tables accessibles plus rapidement. Avant d'introduire les différents documents manipulés par le GNP, nous allons présenté dans la section suivante un bref aperçu sur XML et quelques normes associées.

### 3.3.1 Langage XML

*EXtensible Markup Language* (XML) [Wil 99, 03] est une norme universelle de codage de l'information développé par le W3C-*World Wide Web Consortium* (http://www.w3c.org). Son atout majeur est qu'elle permet d'identifier clairement chaque information, non pas en fonction de son rendu visuel, mais en fonction de sa nature. Par exemple, un numéro de fax est marqué comme tel, et non pas comme du texte en Arial noir de 13 points souligné. Ainsi, toute application compatible XML ayant besoin de cette information est capable de la retrouver et de l'utiliser selon ses besoins, que ce soit pour envoyer une page par Fax, permettre une recherche via le Web, imprimer une plaquette ou envoyer une carte de visite électronique. On constate ainsi que le XML représente l'interface idéale entre les pages web et les bases de données.

La définition de XML permet de l'appliquer à un énorme éventail d'applications en conservant un format unique et des structures adaptées à chaque besoin. Cette norme est

dérivée à la fois de HTML (*HyperText Markup Language*) et de SGML (*Standard Generalized Markup Language*), XML étant un sous-ensemble de la norme SGML (donc compatible avec SGML). La norme utilise un format de texte brut garantissant une excellente portabilité quelle que soit la plate-forme et une grande simplicité d'édition. L'usage de balises (tags) semblables à celle du langage HTML permet à tout le monde de comprendre très simplement la structure de tout fichier XML. De nombreuses autres normes autour de XML garantissent une grande variété et facilité d'utilisation :

- XSL ou « eXtensible Style Language » qui devient de plus en plus un standard complémentaire à XML. Le XSL est utilisé pour préparer des feuilles de style ou des modèles de style HTML.

- DTD ou « *Document Type Definition* » permet de définir un format particulier de documents XML (par exemple XHTML est une DTD qui définit un document XML comme étant du HTML).

- SAX [04] et DOM qui sont deux APIs de programmation standardisées destinées à la manipulation des documents XML.

### 3.3.2 DTD : Création d'un vocabulaire commun

Lorsque des informations doivent être partagées, il est indispensable de communiquer avec la même langue. En XML, c'est la même chose : il faut communiquer avec les mêmes balises pour décrire les mêmes informations. Ainsi, tout le monde se comprendra. Dans cette optique, on utilise une DTD (*Document Type Definition*) ou un schéma XML (*XML Schema*). Il s'agit à peu près des mêmes mécanismes. La différence cruciale entre les deux est que la DTD est un outil de SGML, et que XML Schema a été créé pour XML. Le principe est que l'on veut décrire le contenu d'un document XML. On va y préciser le nom des balises que l'on *peut* ou l'on *doit* utiliser dans le document, et les sous-éléments qu'elles peuvent contenir. On va préciser également quel type de données ces éléments pourront contenir : numériques, textuelles, alphanumériques, etc. Le fait de

définir le document à réaliser permet d'être sûr qu'il sera compris à l'arrivée.

Si un document n'a pas de DTD (il n'est pas obligatoire d'en avoir une) et qu'il est conforme à la syntaxe XML, on dit qu'il est *bien formé*. Si, de plus, une DTD ou un schéma XML est déclaré et que le document XML le respecte, on dira que ce dernier est *valide*. Pour cela, il faut déclarer la DTD ou le schéma dans le code XML

### 3.3.3 DOM : Document Object Model

En tant que spécification du W3C, un objectif important du Modèle Objet de Document (DOM) [05] est de fournir une interface de programmation standard qui puisse être utilisée dans une grande variété d'environnements et d'applications. Il s'agit d'une interface créée pour un langage, ici XML, permettant son accès à d'autres langages de programmation (C, Java, Perl, *Jpython*, etc.) ou de script (comme JavaScript, jscript ou VBScript). Cette interface est nécessaire, car les langages de programmation ou de script ne sont pas prévus, à la base, pour comprendre et exploiter des fichiers écrits en XML. En revanche, ils prévoient l'utilisation d'une API servant de passerelle.

Dans la spécification DOM, le terme "document" est utilisé au sens large - XML est de plus en plus utilisé pour représenter tout type d'information, stockée sur tout type de système. La plupart d'entre elles auraient été traditionnellement vues comme des données plutôt que des documents. Cependant, XML représente ces données comme des documents, et DOM peut être utilisé pour gérer ces données.

Avec le Modèle Objet de Document, les programmeurs peuvent construire des documents, naviguer dans leur structure, et ajouter, modifier, ou supprimer soit des éléments soit du contenu. Tout ce qui peut être trouvé dans un document HTML ou XML peut être accédé, changé, détruit, ou ajouté en utilisant le Modèle Objet de Document, à quelques exceptions près.

La structure des documents SGML a traditionnellement été représentée par un modèle abstrait de données, et non par un modèle objet. Dans un modèle abstrait de données, le

modèle est centré autour des données. Dans les langages de programmation orientés objets, la donnée elle-même est encapsulée dans des objets qui masquent la donnée, la protégeant de toute manipulation externe directe. Les fonctions associées à ces objets déterminent comment les objets peuvent être manipulés, et elles font partie du modèle objet. Le nom "Modèle Objet de Documents" a été choisi parce qu'il s'agit d'un "modèle objet" au sens traditionnel de la conception orientée objet : les documents sont modélisés en utilisant des objets, et le modèle ne contient pas uniquement la structure du document mais également son comportement et celui des objets dont il est composé. En d'autres termes, les éléments d'un document ne représentent pas une structure de données, ils représentent des objets ayant des fonctions et une identité. En tant que modèle objet, DOM identifie :

- les interfaces et les objets utilisés pour représenter et manipuler un document,
- la sémantique de ces interfaces et objets - incluant le comportement et les attributs,
- les relations et collaborations entre ces interfaces et ces objets.

Une API DOM, DOMUtils, a été développée localement en Java. Le GNP s'en sert pour les différentes transformations des documents XML.

### 3.3.4 Définition des documents XML

Les documents XML dans le GNP ont été définis selon le format standard suivant :

```
<gnpDocument>
        <nom du document>
                contenu du document en format XML valide,  défini par le scripteur
        </nom du document>
</gnpDocument>
```

Les paramètres de construction du document, tel que sa clé primaire ( Global Primary Key ou GPK), sa date de création, son lien avec un autre document, etc., sont rajoutés par Le GNP au moment de la construction de l'objet. L'exemple suivant montre le format standard du document de négociation.

```
<gnpDocument GPK="589886" anounceGPK="589885" creationTime="957984417000">
state="OPENED">
 <negotiation/>
</gnpDocument>
```

L'annexe A contient un modèle de chaque document XML utilisé par le GNP.

#### 3.3.4.1 Documents de description et d'état d'un participant

*a- Description d'un participant*

Ce document sert pour rassembler les renseignements personnels sur un participant. Il contient des informations comme son nom, son adresse, et ainsi de suite.

*b- État d'un participant*

Ce document contient l'information dynamique d'un participant au cours d'une ou plusieurs sessions d'enchères. Par exemple, on y trouvera les crédits inutilisés et le formulaire initial.

#### 3.3.4.2 Document de description des règles - « Rules »

Ce document contient l'ensemble des informations relatives à la négociation et communes à chacun des produits. Dans le cas d'une négociation à plusieurs phases, chacune de ces phases y est décrite. Les noms des scripts encanteurs exécutant les règles économiques y sont fournis également ainsi que leurs paramètres comme la durée de chaque ronde, l'heure de clôture des phases, etc. Ces informations ne sont pas éditables pendant la durée de la négociation.

#### 4.3.4.3 Document de référence du produit - « Product Reference »

Ce document contient la référence du produit sur lequel vont porter tous les ordres liés à la négociation de ce produit. Dans une enchère synchronisée avec plusieurs objets, il y aurait plusieurs références, une par produit de l'enchère.

#### 3.3.4.4 Le document de description de l'ordre – « Order »

Un négociateur peut émettre un ordre de vente ou d'achat pour un produit dans une négociation. Le nom du participant, la référence au produit, le prix et la quantité sont les composantes majeures d'un ordre. Un nouvel ordre peut remplacer (« update ») un ordre précédent. Dans ce cas seulement, le plus récent est pertinent dans la négociation. Un ordre est toujours émis après une cote. Ainsi un lien vers la cote vue est inclus dans l'ordre.

#### 3.3.4.5 Document de description de la négociation – « Negotiation »

L'encanteur construit et maintient à jour ce document interne. Il contient l'ensemble des informations relatives à la négociation et susceptibles de changer de valeur lors de chaque ronde. Le contenu de ce document n'est pas destiné à être affiché, il tient à jour les données de calcul privé de l'encanteur pendant la négociation.

#### 3.3.4.6 Document de description de la cote – « Quote »

Ce document est un document de sortie. L'encanteur produit la cote après la fin de chaque ronde et avant le début de la ronde suivante. Ce document contient l'ensemble des informations affichées publiquement à tous les participants. Le prix à battre, ou prix « leader », le numéro de la ronde et de la phase ainsi que le temps d'expiration en sont les composantes usuelles. Le domaine de mise général pour tous les participants y figure aussi. Le domaine de mise contient l'intervalle de prix et de quantité dorénavant admissible.

#### 3.3.4.7 Document de description de la réponse – « Response »

L'encanteur construit ce document en évaluant un ordre et à chaque fin de ronde qui modifie une réponse précédemment faite. La réponse contient habituellement l'allocation temporaire que l'encanteur attribue à ce participant. Cette allocation (la quantité allouée et son prix) deviendra finale si l'enchère s'arrêtait à cette ronde. De plus, la réponse contient habituellement le domaine de mise du participant, c'est-à-dire quel intervalle de prix et de quantité le participant pourra émettre lors de la prochaine ronde.

#### 3.3.4.8 Le document de description de l'adjudication– « Adjudication »

L'adjudication est produite par l'encanteur à la fin de la négociation. Elle est basée sur la dernière allocation temporaire de l'enchère. Elle est la combinaison entre un ordre de vente et un ordre d'achat donnant lieu à une transaction.

#### 3.3.4.9 Document de description de l'annonce– « Announce »

L'annonce contient toujours une partie *règles* (rules) et, selon le mécanisme choisi, une partie *produit* (product Reference), et une partie *ordre* (order). La partie « rules » établit précisément les paramètres fixes de la négociation, tels que les heures d'ouvertures et de fermetures, l'incrément minimum, les conditions de fin de rondes et les conditions d'équilibrage du marché. Si ceux-ci doivent varier, la négociation est divisée en plusieurs phases. La partie « product reference » définit techniquement la nature du produit à négocier ou renvoie à sa description précise. La partie « order » représente les données de la mise initiale faite par le participant, telles que sont prix de départ, son prix de réserve, la quantité de produit mise en négociation et ses fractions permises.

## 3.4 Déroulement d'une négociation

L'initialisation d'une négociation sur GNP se fait par l'envoi d'un document XML initial. Il s'agit de l'annonce qui contient tous les éléments d'entrée nécessaires au démarrage d'une négociation.

### 3.4.1 Envoi de l'annonce par modèle

Le participant désirant initialiser une négociation sur GNP dispose en premier lieu d'un choix de différents modèles de négociation (templates). À un modèle, correspond un formulaire d'annonce HTML unique. Le formulaire permet à l'usager de définir certains paramètres du modèle et ainsi de garantir une flexibilité d'implantation à l'intérieur de chaque modèle. Le formulaire complété et envoyé donnera naissance à l'annonce, document en format XML.

La séquence des actions d'initialisation d'une négociation se fait donc ainsi :

1. Affichage d'un choix de différents modèles d'annonce à un participant.
2. Le participant choisit un modèle d'annonce.
3. Affichage du formulaire d'annonce correspondant au modèle choisi.
4. Le participant remplit les champs éditables du formulaire d'annonce et l'envoie.
5. Le document XML d'annonce est créé.

### 3.4.2 Envoi de l'annonce directement

Pour l'exécution des scénario de test, nous procédons autrement. Le document de l'annonce est édité manuellement dans le format XML valide au sens GNP et lui est envoyé directement en invoquant les méthodes chargées de le faire. L'envoi du chemin complet de l'annonce à GNP remplace alors les 4 premières étapes de l'initialisation par modèle.

### 3.4.3 Séquence d'actions après l'envoi d'une annonce

L'annonce donne à GNP tous les éléments d'entrée nécessaires au démarrage d'une négociation. L'annonce apparaît à l'utilisateur dans un format HTML. Au niveau interne, l'annonce est un document XML. Lorsque l'annonce est remplie, elle est envoyée au système qui va déclencher une « session bean » chargée de :

- créer l'objet « ***rules*** » et son document XML à partir de l'annonce;

- créer l'objet « ***product Reference*** » et son document XML à partir de celui de l'annonce si celui-ci existe;

- créer l'objet « ***order*** » et son document XML à partir de celui de l'annonce s'il existe également;

- créer l'objet « ***negotiation*** » et son document vide;

- créer l'encanteur en utilisant l'instance de la classe définie dans le script en *Jpython*. Par la suite, il devrait être possible d'envoyer l'annonce d'un autre produit et ainsi de créer une autre négociation sur la base des règles établies par l'annonce initiale.

CHAPITRE 4

# SPECIFICATION ET TEST DE LOGICIEL

Le génie logiciel existe depuis de nombreuses années. Il a été introduit pour répondre aux problèmes de fiabilité des logiciels et du respect des spécifications du cahier de charges. Le génie logiciel [Naur 69] est donc l'Art de spécifier, de concevoir, de réaliser et de faire évoluer, dans des délais raisonnables, des logiciels, de la documentation, et des procédures de qualités en vue de l'exploitation dudit logiciel par un ordinateur. Le génie logiciel incorpore une stratégie de développement. Cette stratégie est souvent appelée modèle de processus ou paradigme du génie logiciel. Un modèle de processus, comme le modèle de cycle de vie par exemple, est choisi en fonction de la nature du projet et de l'application, des outils et méthodes à utiliser. Le modèle de cycle de vie est un modèle de phases ou activités qui commence quand le logiciel est conçu et se termine quand le logiciel n'est plus apte à être utilisé. Le cycle de vie comprend typiquement plusieurs phases :

- une phase de définition des besoins,
- une phase de conception,
- une phase d'implantation,
- une phase d'intégration,
- une phase d'installation et de test,
- une phase d'opération et de maintenance.

Plusieurs modèles de cycle de vie existent, parmi lesquels on cite le modèle en cascade, le prototypage rapide, le prototypage évolutif, réutilisation de logiciel et le développement incrémental.

Quels que soit la nature, le domaine, la taille et la complexité du système à développer, le cycle de vie passe par des phases qui amènent à la production d'un système vérifiant certaines caractéristiques et répondant à des besoins préalablement requis. Cependant, l'utilisation de la spécification écrite dans le langage naturel entraîne souvent des spécifications longues et informelles qui contiennent de l'ambiguïté et pour lesquelles il est difficile de vérifier l'exactitude et la complétude. Le pouvoir d'expression des techniques de description formelle (FDTs - Formal Description Techniques), ainsi que leur capacité à détecter les ambiguïtés dès les premières phases du développement en font des outils indispensables à tout développement rigoureux. Par ailleurs, l'usage de ces techniques favorise l'automatisation de plusieurs activités parmi lesquelles on cite la génération du code, la génération d'une suite de tests et la documentation systématique. De nombreuses FSTs existent, en particulier dans le domaine des télécommunications, dont les principaux sont les machines à états finis, les systèmes de transitions étiquetées, les machines à état finis étendus et les réseaux de Pétri. Ces différents modèles sont introduits dans la section suivante.

## 4.1 Spécification

### 4.1.1 Machines à états finis

Une machine à états finis (MEF) est un modèle abstrait pour la description des systèmes dont le comportement futur est affecté par leur comportement dans le passé. Ce comportement est décrit comme étant une séquence d'événements survenant à des moments discrets. Les systèmes sont modélisés par un graphe étiqueté dont les nœuds représentent l'état du système et les arcs représentent les transitions ou les événements internes ou externes qui, lorsqu'ils sont stimulés, peuvent changer l'état du système (voir

Figure 3.1). Ces événements sont classés en deux catégories : les entrées et les sorties. Les événements d'entrées sont la stimulation de l'environnement au système. Ceux de sorties, sont la réponse du système à ces stimulations.

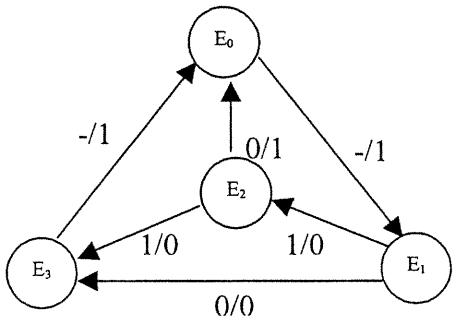


Figure 4.1 Exemple d'une MEF

Ce modèle a l'avantage d'avoir été beaucoup étudié et a largement bénéficié de la théorie des automates. Durant les vingt dernières années, les MEFs ont été le modèle le plus utilisé pour la génération automatique des suites de test [Gil 62, Koh 78].

### 4.1.2 Machines à états finis communicantes

Une machine à états finis communicante (MEFC) [Hol 91] est définie comme étant une machine abstraite qui accepte des données en entrée, produit des données en sortie et change son état interne selon un plan pré-défini. Dans ce cas, les MEFs communiquent à travers des files FIFO qui transforment les données produites par une machine en données d'entrée d'une autre machine. Ces files, appelées files à messages, sont formellement représentées par un triplet (S,N,C) où :

- S est un ensemble fini appelé vocabulaire de la file,
- N est un entier qui donne le nombre de places dans la file,
- C est le contenu de la file constitué d'un sous-ensemble ordonné de S.

Un système de MEFC est constitué d'un certain nombre de MEFCs avec les propriétés suivantes :

- Chaque MEFC est munie d'une file FIFO où les entrées de la machine sont stockées et d'où elles sont retirées. Ces entrées sont appelées des messages.
- Un canal de communication ne peut connecter que deux machines.
- Chaque couple de machines peut avoir deux canaux de communication, un canal par direction.
- La capacité des files et des canaux est théoriquement illimitée.

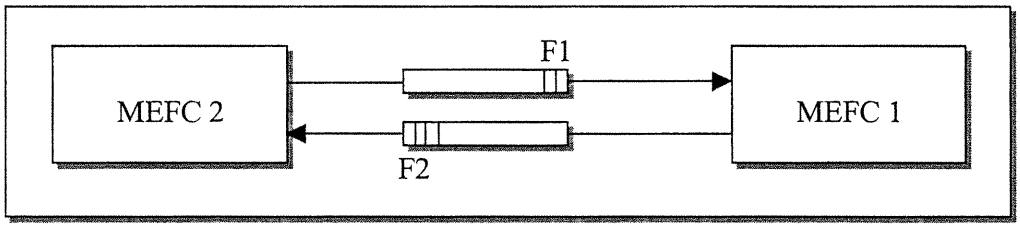


Figure 4.2 Système communicant à base de deux MEFCs

La sémantique d'un système de MEFCs est donnée par l'ensemble des états globaux où le système peut être. Un état global consiste en une collection d'états locaux, un état par MEFC et le nombre de messages dans chaque canal.

### 4.1.3 Machines à états finis étendus

Afin de décrire, à l'aide d'une MEF, un composant d'un protocole qui manipule des compteurs ou bien des structures de données complexes, le nombre d'états nécessaires devient vite très grand. Ce problème d'explosion des états, ou explosion combinatoire, est une limitation majeure des MEFs. Les MEFEs permettent, contrairement aux MEFs, de décrire aussi bien l'aspect contrôle que l'aspect de données d'un protocole de communication. Le modèle MEFE décris un processus comme étant une MEF étendue par les données d'entrée et de sortie ainsi qu'un certain nombre de variables locales [Dss

99]. Chaque transition est associée à un prédicat qui dépend aussi bien des paramètres effectifs de la données en entrée que des valeurs actuelles des variables locales. Une action est réalisée suite à l'exécution de la transition et peut modifier les variables locales.

Plusieurs langages de description formelle sont fondés sur le modèle des MEFEs parmi lesquels nous citons à titre d'exemple le langage SDL « Specification and Description Langages » [Bel 89] et le langage ESTELLE « Extended State Transition ModEL » [Bud 87].

### 4.1.4 Réseaux de Petri

Les réseaux de Petri [Hac 76] sont un formalisme graphique fondé sur la représentation des relations d'accessibilité entre états comme les graphes d'états. Il s'agit d'un outil graphique et mathématique applicable à plusieurs domaines où les notions d'événements et d'évolution simultanés sont importantes comme c'est le cas pour les protocoles de communication, le contrôle automatique, les systèmes discrets, l'évaluation des performances ...etc. Un réseau de Petri est un type particulier de graphe orienté avec un état initial appelé marquage initial $M_0$. Le graphe associé à un réseau de Petri est un graphe orienté qui possède deux types de nœuds : les places (conditions) et les transitions (événements). Un arc relie une place $p$ à une transition $t$ si et seulement si $Pre(p, t) \neq 0$. Un arc relie une transition $t$ à une place $p$ si et seulement si $Post(p, t) \neq 0$. Les applications *Pre* et *Post* sont représentées par des matrices dont le nombre de lignes et de colonnes sont respectivement égaux aux nombres de places et de transitions.

### 4.1.5 Langage Message Sequence Chart - MSC

MSC [Z.120] est un langage à deux syntaxes, graphique et textuelle, pour la spécification et la description des interactions entre les composants d'un système. Les diagrammes bidimensionnels graphiques donnent un aperçu sur l'échange des messages entre des instances d'un système et leur environnement. La forme textuelle, quant à elle, elle sert

essentiellement pour l'échange entre outils.

L'avantage principal d'un MSC est que sa représentation graphique permet de comprendre de façon intuitive le comportement du système décrit. Avant sa première standardisation en 92, MSC a été utilisé pendant longtemps par l'Union internationale des télécommunications (ITU) dans ses recommandations comme moyen non formel d'illustration. L'objectif derrière la standardisation de MSC est de permettre un support

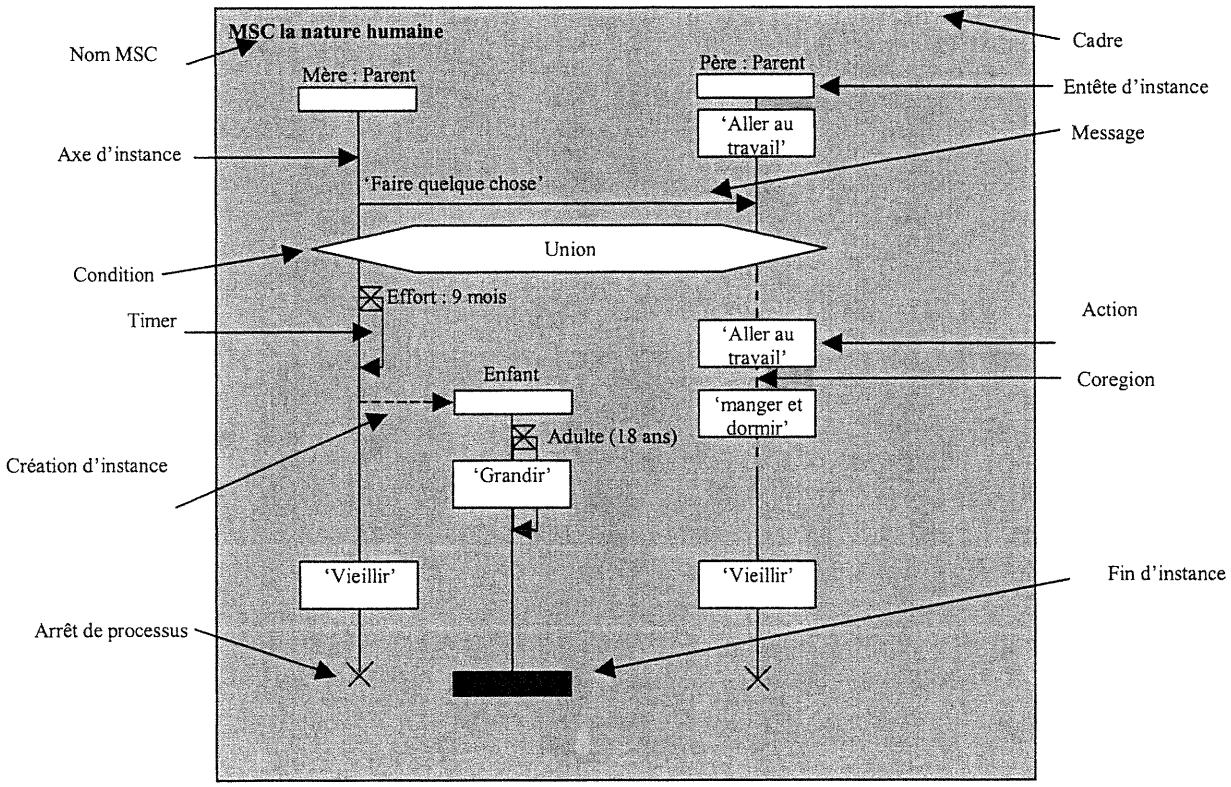


Figure 4.3 Les éléments de base de MSC

systématique du langage pour faciliter l'échange entre différents outils. Grâce à elle, l'importance du langage pour l'ingénierie des systèmes a considérablement augmentée.

En combinaison avec SDL ou d'autres langages, les diagrammes MSCs interviennent

pratiquement dans toutes les étapes du cycle de développement des systèmes. Ils sont pratiquement utilisés :

- Pour la spécification des besoins,
- Pour la spécification d'interfaces,
- Comme une spécification sommaire d'un processus de communication,
- Comme base pour la génération automatique d'un canevas d'une spécification SDL [Khe 00],
- Comme base pour la spécification et la génération des cas de test [Gra 94],
- Pour la documentation,
- Pour la conception orientée objet (interactions entre objets).

#### 4.5.1.1 Éléments de base du langage MSC

MSC supporte une conception structurée. Des scénarios simples décrit par le biais de bMSC (Basic MSC) (Figure 4.3) peuvent être combinés pour former des spécifications plus complètes par le biais de HMSC (High-level MSC). La Figure 4.3 montre ces éléments de base:

*Instances*

Les instances, avec les messages, sont les principaux éléments dans un MSC. Les instances représentent par exemple les blocs, les processus ou les services SDL. L'entête d'une instance possède un nom et, éventuellement, un type d'instance. Une instance peut aussi être décomposée et représentée par un MSC entier (avec plusieurs autres instances).

*Messages*

Les messages décrivent les événements de communication. Un message représente un événement asynchrone émis ou reçu.

*L'environnement (environment)*

Dans un MSC, l'environnement est représenté graphiquement par un cadre qui constitue la bordure du diagramme MSC.

*Action*

En plus de l'échange des messages, une action décrivant une activité interne d'une instance peut être spécifiée. Graphiquement, une action est représentée par un rectangle.

*Temporisateur (Timer)*

Le « timer » représente un message qui dépend du temps. Les manipulations des « timers » dans un MSC concernent l'initialisation (set), la réinitialisation (reset) et l'expiration. Quand un « timer » expire, un événement d'entrée se produit. La réinitialisation d'un temporisateur n'expire jamais et ne produit aucun événement.

*Création et fin d'instance*

La création et la fin des instances sont des événements très courants dans les systèmes de communication. Ceci est dû au fait que ces systèmes sont dynamiques, les instances apparaissant et disparaissant pendant l'exécution. Comme c'est le cas pour les messages, l'événement de création peut être étiqueté par une liste de paramètres. Le Symbole graphique pour la création d'une instance est une flèche discontinue partant de l'instance mère vers l'instance fille. La fin d'une instance détermine la fin de la description de l'instance et non la fin de l'instance elle même.

*Condition*

Une condition est utilisée pour marquer un état important dans un MSC ou pour la composition de plusieurs bMSC. Dans le premier cas, une condition décrit l'état d'un ensemble non vide d'instances spécifiées dans le MSC. Elle peut définir soit un état global du système qui se réfère à l'ensemble des instances et dans ce cas la condition est dite globale, soit un état non global se référant à un sous-ensemble des instances du

système. Quand la condition se réfère à une seule instance, elle est dite locale. Les conditions servent aussi pour la composition de plusieurs MSCs. De simples scénarios MSC peuvent être composés pour former des scénarios plus complexes. La composition des MSC doit obéir aux règles suivantes :

1. Composition via une condition globale : deux MSCs peuvent être décomposés séquentiellement si (i) ils possèdent les mêmes instances et si (ii) la condition initiale globale de l'un correspond à la condition finale globale de l'autre. La condition initiale indique l'état initial et la condition finale indique l'état final.
2. Composition via une condition non globale : deux MSCs peuvent être décomposés par le biais d'une condition non globale si pour chaque instance qu'ils ont en commun, le premier se termine avec une condition non globale et le second débute avec la même condition.

L'importance principal des MSCs dans le futur ne serait pas la description complète des systèmes, mais plutôt la spécification de certaines propriétés comme par exemple des traces d'exécution qui doivent être autorisées ou encore qui doivent être interdites. Dans tous les cas, la puissance des MSCs réside dans sa capacité à décrire clairement et intuitivement quelques traces d'exécution d'un système, alors que SDL, lui, est utilisé pour une spécification complète du système. Cependant, la limitation de MSC seulement à certaines traces ne veut pas dire qu'il a juste un caractère illustratif comme c'était le cas dans le passé. Loin de là, les MSCs peuvent servir pour décrire des objectifs de test pour la génération automatiques des cas de test, comme c'est le cas pour la méthode SAMSTAG [Gra 94] et l'outil objectGEODE [Obj 96].

## 4.2 Techniques de test

D'une manière générale, la conformité d'une implantation d'un système par rapport à sa spécification est assurée par les techniques de test. Il n'existe aucune définition

universelle de l'activité de test. Toutefois, cette activité tend principalement à juger le degré de conformité d'une implantation par rapport à sa spécification et à détecter d'éventuelles erreurs de programmation [Wass 77, whit 78]. Pour plusieurs organisations, la qualité du système logiciel prend de plus en plus de l'importance. Malgré les résultats encourageants obtenus ces dernières années, l'objectif « système avec zéro erreur » est encore à atteindre. Les causes des erreurs sont variées et imprévisibles et leur détection est une tâche immense en plus d'être coûteuse. Le test va donc continuer à être une activité importante dans le cycle de développement et de maintenance de logiciel, valant pas moins de 30 à 40 % du budget global [Cou 94]. Dans ce contexte, le test n'est plus vu comme une activité qui n'intervient qu'à la fin du cycle de développement. Loin de là, il est de plus en plus considéré comme indissociable de toute les étapes du cycle de développement, tel que montré dans la Figure 4.4.

Relativement récemment, le test a gagné beaucoup en maturité. Des méthodes de test ainsi que des techniques et des outils de test d'un niveau satisfaisant sont disponibles. Cependant, vu l'émergence des nouvelles technologies de l'information, la complexité des solutions rendues possibles par ces technologies va considérablement augmenter. L'intégration des processus d'affaire comme le e-commerce, les entreprises virtuelles, le traitement de données en vue de prise de décisions, etc. n'en sont que quelques exemples.

Plusieurs classifications de test existent dans la littérature. La plus connue est probablement celle qui distingue entre deux grandes stratégies : le test structurel et le test fonctionnel [Nat88].

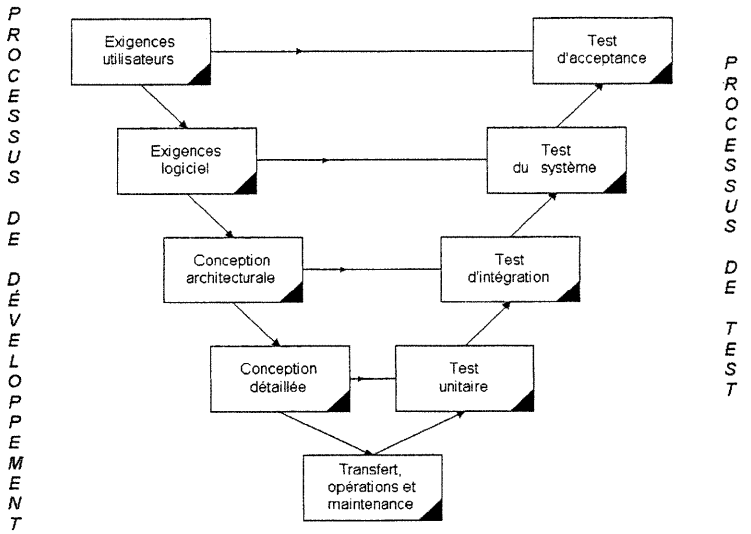


Figure 4.4 Processus de test dans le cycle de développement des logiciels

### 4.2.1 Test structurel

Dans cette stratégie de test, appelé aussi test de «boîte blanche» ou « white-box / glass-box / clear-box » pour le distinguer du test de «boîte noire», les cas de test sont dérivés à partir de la connaissance des détails de l'implantation. Cette dernière est vue comme une «boîte blanche» (ou transparente) [Mye 76]. Le test structurel est relativement appliqué à de petites unités de programmes, en général lors des tests unitaires. Le testeur se base principalement sur le code source et la connaissance de la structure du composant à tester pour dériver les données de test. L'analyse de la structure du code peut servir à déterminer le nombre des cas de test nécessaires pour garantir que chaque bout de code sera exécuté au moins une fois pendant le processus de test. Parmi les stratégie de test structurel, on retient essentiellement le test de branches/chemins, et le test de segments. Dans le test de branches/chemins, chaque branche indépendante du programme doit être exercée par au moins un cas de test. De cette façon, toutes les instructions du composant à tester sont exécutées au moins une fois. En plus, tous les branchements conditionnels

sont testés dans les deux cas : vrai et faux. Le test de segments exige quant à lui, que chaque instruction du programme soit exécutée par au moins un cas de test.

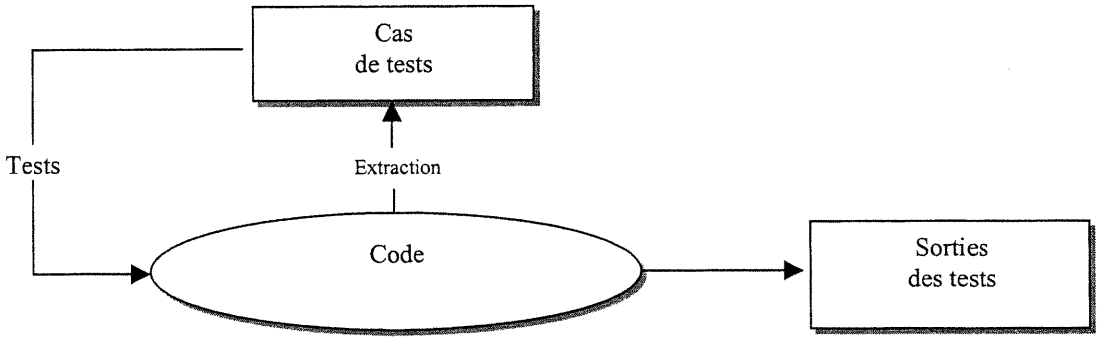


Figure 4.5 Architecture du test structurel

### 4.2.2 Test fonctionnel

Le test fonctionnel appelé aussi test «boîte noire» ou « black box » est une approche dans laquelle les cas de test sont dérivés à partir de la spécification de l'implantation. L'implantation est considérée comme une «boîte noire» dans la mesure où son comportement ne peut être perçu qu'à travers l'étude de ses entrées et des sorties conséquentes. Ce test est appelé « fonctionnel », car le testeur n'est intéressé que par les fonctionnalités du système et accorde peu ou pas d'importance à son implantation. L'implantation n'est alors accessible que via des interfaces appelées points de contrôle et d'observation (PCO). Voir la Figure 4.6.

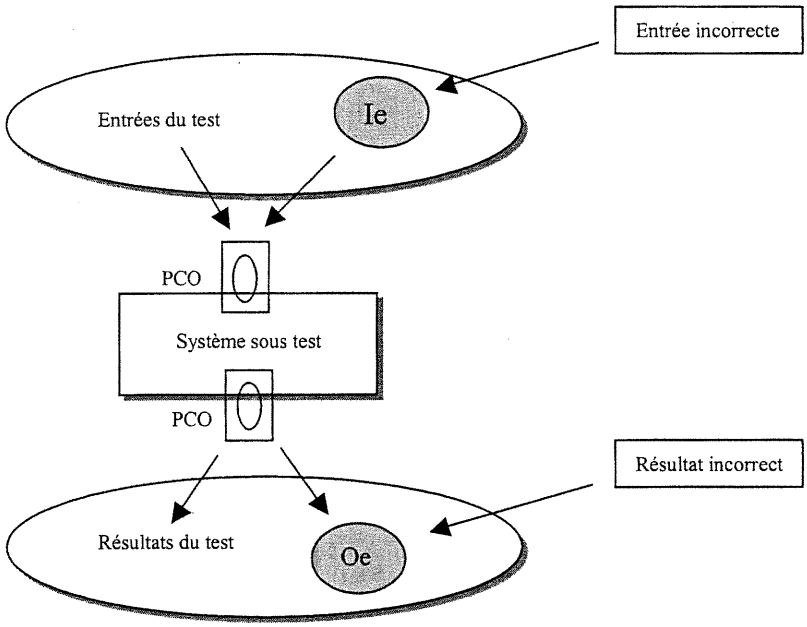


Figure 4.6 Architecture du test fonctionnel

Le test fonctionnel de «boîte noire» consiste à soumettre à l'implantation des entrées et de récupérer les sorties via les PCOs. Le verdict de test consiste quant à lui à comparer les résultats obtenus avec ceux attendus. Dans le cas ou les résultats attendus correspondent à ceux observés, l'implantation est dite conforme. Dans le cas contraire, une erreur est détectée et un processus de diagnostic est amorcé. Le test fonctionnel, appelé aussi test de conformité, est généralement réalisé en trois phases :

1- Extraction de la suite de test : Développer une méthode générale et efficace pour générer une suite de test à partir de la spécification. Une suite de test est un ensemble de cas de test visant chacun à vérifier une ou plusieurs propriétés du système. Pour ce faire, la spécification doit être exprimée dans un modèle non ambigu. La qualité de la suite de test générée est l'une des préoccupations majeures du test de conformité. Cette qualité, dite couverture de test, détermine l'efficacité de la méthode de génération utilisée. De même que pour les tests, plusieurs définitions ont été utilisées pour qualifier la couverture de test. Parfois, elle sert pour mesurer le pourcentage des cas de tests exécutés par rapport à ceux générés. Parfois, elle est utilisée pour mesurer la capacité des cas de test à détecter

les erreurs, on parle alors de couverture de fautes. Dans d'autres cas, elle est utilisée pour mesurer le pourcentage des instructions testées par rapport à l'ensemble des instructions du programme, et dans ce cas, on parle de couverture de programme. Dans tous les cas, la couverture de test est utilisée pour exprimer ce qui a été testé par rapport a ce qui devrait être testé dans un test exhaustif. Pratiquement, il est difficile, sinon impossible d'avoir une couverture complète de test. On a alors recours à un arbitrage entre l'assurance d'une bonne couverture et le coût ou l'effort requis pour tester l'implantation.

2- Exécution de la suite de test : Appliquer à l'implantation la suite de test générée dans l'étape précédente. La connaissance de l'architecture de test à appliquer dans cette étape est nécessaire. A un niveau d'abstraction élevé, l'architecture de test consiste en un testeur, une implantation sous test et un contexte de test (voir la Figure 4.7).

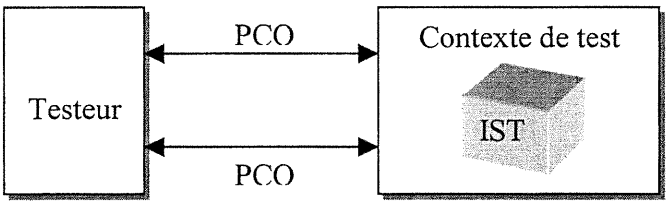


Figure 4.7 Architecture de test

Plusieurs architectures de test existent. Deux architectures de test ont été définies par l'ISO pour le test des protocoles de communication. Elles sont respectivement le test local et le test externe. Dans le test local, le testeur et l'implantation à tester s'exécutent dans la même machine. Ce test correspond au test classique des logiciels. Quant au test externe, il existe sous trois formes différentes: le test distribué, le test coordonné et le test à distance. Dans le test distribué, le testeur est dissocié en deux parties : le testeur supérieur (upper tester) et le testeur inférieur (lower tester). Chacun des deux testeurs accède à l'implantation à travers une des deux interfaces supérieure et inférieure (PCOs). Par ailleurs, l'interface inférieure n'est pas accessible directement. Le testeur inférieur y

accède via le service de communication sous-jacent (voir la Figure). Le test coordonné diffère du test distribué par la présence d'une procédure de coordination entre le testeur inférieur et le testeur supérieur selon un protocole de coordination de test ou TCP (Test Coordination Procedure). Le test à distance est un test distribué dans lequel seulement le testeur inférieur est utilisé.

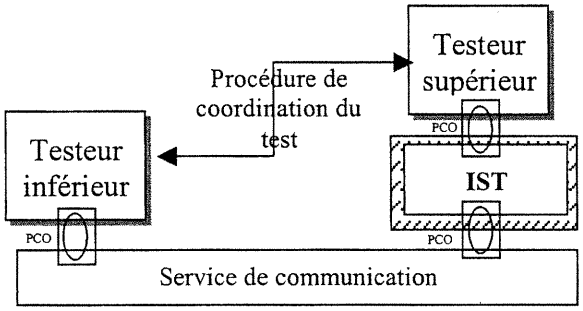


Figure 4.8 Architecture du test externe

3- Analyse du résultat du test : L'exécution d'un cas de test donne lieu à un verdict. La relation de conformité ou de non conformité est déduite à partir des verdicts obtenus. La conformité est une relation parmi plusieurs permettant de juger l'implantation par rapport à sa spécification. D'autres relations sont discutées dans [Led 91]. Un verdict peut être l'un des trois cas suivants : « passe », « échec », « non concluant ». Un verdict « passe » est annoncé quand les sorties observées satisfont le but de test et qu'elles sont complètement valides par rapport aux besoins de conformité. Le verdict « échec » est signalé lorsque les résultats observés sont invalides par rapport aux besoins de conformité. Finalement, un verdict est « non concluant » si les sorties observées sont valides du point de vue conformité, mais ne le sont pas par rapport au but du test.

## 4.3 Principaux types de test logiciel

Pour tester un logiciel de façon générale, y compris un système distribué, on a généralement recours au tests suivants:

### 4.3.1 Test de conformité

Tout système logiciel est conçu pour répondre à des objectifs déterminés et doit, par conséquent, être conforme à sa spécification. De manière générale, le test de conformité permet de vérifier cette relation. Ce genre de test occupe une place importante dans le domaine de l'ingénierie des logiciels. Il se base essentiellement sur le comportement extérieur de l'implantation. En général, ce genre de test ne peut être exhaustif vu le nombre important de cas à traiter. Les cas de test sont alors sélectionnées de façon à exécuter certaines parties du système et à détecter le maximum de fautes. Selon Myers [Mye 79], un bon cas de test est celui qui a une forte probabilité de trouver une erreur non encore découverte. L'objectif est alors de concevoir des cas de test qui découvrent systématiquement différentes classes d'erreurs avec un minimum de temps et d'effort.
Pour une génération efficace d'une suite de test, la définition de scénarios d'utilisation est un atout apprécié. Une approche pour atteindre cet objectif est de décrire, en se basant sur les spécifications du système, des fonctions reflétant chacune un besoin fonctionnel particulier. Comme exemple de fonctions relatives au commerce électronique à retenir, l'exploration, le login, la soumission d'une commande, le suivi de la commande, etc. [Kac 98]. A partir de chaque fonction retenue, on peut dériver plusieurs variantes qui décrivent chacune une façon possibles de réaliser cette fonction. Chaque variante donnera lieu à un sous-scénario légèrement différent. On peut alors distinguer entre plusieurs catégories de ces sous-scénarios: une catégorie qui regroupe les scénarios traitant les situations normales sans erreurs, autrement dit, la réalisation "normale" de la fonction. Les scénarios de cette catégorie sont appelés scénarios optimistes. Ils sont considérés comme étant des scénarios à faible rendement. Une autre catégorie regroupe les scénarios traitant avec des situations anormales et traitement avec erreur. Ces scénarios

peuvent inclure également des actions pour annuler, implicitement ou explicitement, la progression de la fonction après que le système a échoué à corriger l'erreur. Finalement une autre catégorie pour décrire les scénarios concurrentiels ou compétitifs. Bien que plus difficiles à construire, la description des scénarios concurrentiels est très utile pour détecter la présence des erreurs d'implantation à haut risque. Les scénarios des deux dernières catégorie sont considérés comme des scénarios à rendement important. Par "rendement", on désigne le nombre d'erreurs à grand risque susceptibles d'être détectées grâce à la simulation ou l'exécution de ces scénarios, ce qui caractérise la sévérité des fautes. Cependant, le test de conformité permet de prouver la présence de certaines erreurs, mais il ne peut jamais garantir leur absence.

### 4.3.2 Test de performance

Ce test a pour objectif de vérifier le comportement du système global dans des conditions d'usage intensif. En général, ce type de test nécessite une planification du moment ou il implique tout un laboratoire dans lequel le test doit être effectué. Typiquement, le test de performance est réalisé en simulant des navigateurs représentant des usagers virtuels entrain d'interagir avec le système. Ceci est rendu possible par le biais de scripts décrivant chacun un scénario d'usage particulier. Les scripts peuvent être crées manuellement, automatiquement ou semi-automatiquement. La plupart des outils de test de performance disponibles sur le marché offrent un assistant de génération de scripts plus ou moins évolué. Il existe globalement deux approches pour générer des scripts:

- L'approche la plus simple consiste à enregistrer le trafic HTTP généré pendant un usage normal, par les événements de la souris et du clavier. Pendant la simulation, ce trafic est rejoué tel que, sans même nécessiter un explorateur. Cette approche peut donner des cas de test non exécutables, car ils pourraient être liés aux dates ou au temps (voir la Figure 4.9).

- La deuxième approche, plus difficile à réaliser, consiste à enregistrer les événements de la souris et les entrées du clavier, et de rejouer, pendant la simulation, ces événements et entrées, tel que montré à la Figure 4.10.

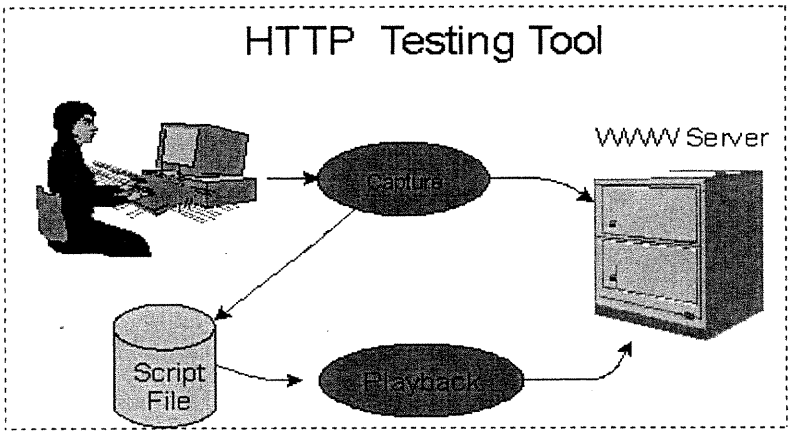


Figure 4.9 Capture et rejoue du trafic HTTP brut

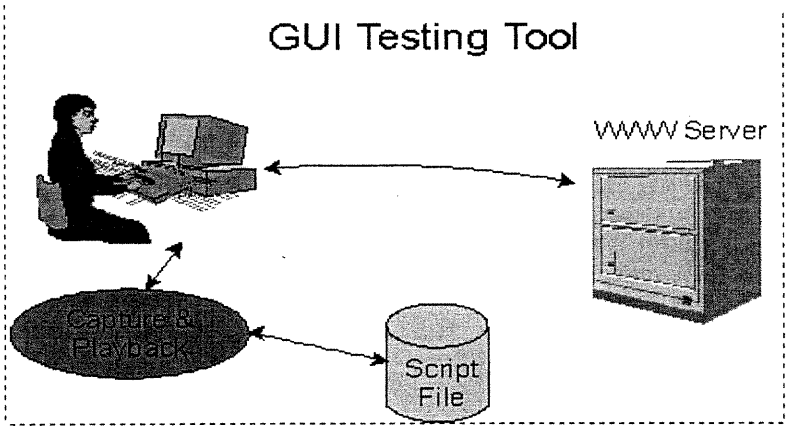


Figure 4.10 Capture et rejoue des événements de la souris et du clavier

La première approche, bien que plus facile à réaliser, peut provoquer des erreurs et des confusions produisant des résultats incorrects, car le trafic HTTP brut enregistré peut

contenir des informations dépendantes de la session en cours comme, par exemple, un identifiant de session SSL, qui n'auront pas de signification dans les autres sessions.
Une fois les scripts générés, la plus part des outils commerciaux mettent à la disposition des testeurs un outil de modification manuelle des scripts dont le but est de raffiner le contenu et de l'adapter à des scénarios bien spécifiques. Afin de simuler avec plus de fidélité le comportement d'un vrai usager, des "moments de réflexion" peuvent également être insérés dans différents endroits.

### 4.3.3 Test de sécurité

Le test de sécurité permet de détecter la capacité de l'implantation à résister à toute intrusion illégale et à toute altération illégale des données. Le problème de sécurité doit être soulevé à tous les niveaux du système, à savoir le navigateur, le réseau, le serveur web, le serveur d'application, le serveur de base de données, le système d'exploitation et même au niveau des systèmes partenaires. Il faut prévoir, en particulier, des cas de test visant à certifier que les échanges de messages financièrement critiques sont hautement sécurisés. Toute tentative de lecture ou de modification non autorisées de ces messages doit être détectée. Une bonne démarche à appliquer pour faciliter le test de sécurité des systèmes consiste à:

- Envisager, en premier, les risques de sécurité les plus récents.
- Vérifier la résistance du système aux menaces de sécurité les plus connues.
- Collaborer étroitement avec les administrateurs système qui, en général, possèdent des outils de détection d'intrusion.
- Identifier les différents utilisateurs du système et les privilèges attribués à chaque utilisateur.

Cependant, une étude plus élaborée des risques constitue en elle même un travail laborieux et se soldera nécessairement par d'autres recommandations à appliquer.

### 4.3.4 Test de configuration

Le test de configuration permet de savoir si le système peut bien fonctionner dans un environnement incluant une panoplie d'autres produits incluant des systèmes d'exploitation différents, des navigateurs différents, des machines différentes, etc., et de tester aussi son comportement vis-à-vis de versions anciennes de ces produits.

Tester le système sous ces différents environnements et garder trace des résultats des différentes combinaisons peut s'avérer délicat. Une approche efficace pour faciliter cette tâche serait d'utiliser des matrices qui montrent les différentes combinaisons utilisées. Par exemple, on peut mettre en ligne les systèmes d'exploitation et en colonne les différents fureteurs etc. Selon les besoins, il n'est pas toujours nécessaire de tester toutes les combinaisons possibles de ces logiciels.

### 4.3.5 Test de robustesse

Ce test s'intéresse au degré de résistance de l'implantation à des événements externes ou à des erreurs non prévus par la spécification. Autrement dis, la capacité de l'implantation à continuer à fonctionner même dans des situations anormales.

## 4.4 Automatisation du processus de test

De plus en plus d'entreprises s'intéressent au commerce électronique. Par ailleurs, ces systèmes doivent toujours prendre en considération l'évolution incessante des technologies de l'internet. Ceci a pour conséquence plusieurs cycles de développement et des tests répétés. L'automatisation des tests devient alors un besoin important. Complètement ou partiellement automatisée, l'exécution des tests doit tenir compte des cycles de développement de plus en plus courts qui se comptent parfois en quelques jours seulement. C'est dans cette perspective que nous avons développé l'environnement de test. A part les scénarios qui, dans une étape préliminaire, sont écrit manuellement en se basant sur les spécifications économiques, le reste du processus est automatisé.

## 4.5 Test des enchères électroniques

Les systèmes d'enchères électroniques sont des instances spécialisées des systèmes distribués. Ils sont composés de plusieurs entités de traitement distribuées et autonomes échangeant des messages à travers un réseau. Par ailleurs, Il sont réactifs, temps réel, concurrents et ouverts. Évidemment, les techniques traditionnels de test de type «boîte blanche» et «boîte noire» sont utiles pour tester la conformité de chaque composant par rapport à ses spécifications. Cependant, tester la qualité de l'assemblage et de l'intégration du système entier dans un environnement distribué est un défi nécessitant des outils, des techniques et un environnement de test plus élaborés. Dans un tel environnement, il faut considérer les problèmes de concurrence, de synchronisation et de communication. En plus, dans un contexte de commerce électronique, le test de la sécurité, de l'intégrité et de la vulnérabilité du système est d'une grande importance.

CHAPITRE 5

# DESCRIPTION DE L'ENVIRONNEMENT DE TEST

Les scénarios de test sont décrits avec le langage MSC (voir le paragraphe 4.1.5). Rappelons que l'objectif est de tester la conformité entre la spécification d'un modèle d'enchère sous forme d'algorithme économique et son implantation par le GNP. Les scénarios sont dérivés manuellement en se basant sur la spécification de chaque modèle d'enchère. La section suivante décrit, à travers un exemple, la méthodologie (guidelines) à suivre pour dériver des scénarios de test à partir des « sessions beans » du GNP. La même démarche peut être poursuivie pour dériver des scénarios de test pour d'autres systèmes similaires.

## 5.1 Dérivation des scénarios de test

Quel que soit le modèle d'une enchère, celle-ci représente la succession de deux types d'événements : les annonces et les mises. Les annonces expriment la volonté d'un négociateur à soumettre un bien ou un service à vendre (ou à acheter selon le côté du marché considéré). Les mises traduisent l'intérêt des autres participants envers ce bien ou ce service, en proposant un prix d'acquisition (ou de vente si l'annonce est un appel d'offres). Ce sont donc les opérations de base qui constitueront tout scénario de test.

Sous GNP, ces deux opérations sont réalisées respectivement par l'annonceur et le négociateur. Ceux-ci sont modélisés respectivement par les « session beans » AnnouncerSession et NegociatorSession. La description des méthodes de ces deux classes est donnée dans le tableau suivant :

| Méthode | Rôle | Objet retourné |
|---|---|---|
| **getTemplates**() | Récupère tout les modèles d'enchères disponibles | Document XML au format GNP (XMLDocument) contenant les modèles disponibles |
| **submitAnnounce**(String XMLDocument) | Soumet une annonce créant une négociation autonome | Document XML au format GNP (XMLDocument) contenant les clés primaires du produit, de la négociation et de l'annonce |

Table 5.1 Liste partielle des méthodes de « la session beans » AnnouncerSession

| Méthode | Rôle | Objet retourné |
|---|---|---|
| **submitOrder**(long prodRefGPK,<br>long negoGPK,<br>long QuoteGPK,<br>String orderData) | Permet de soumettre un mise relative à une annonce préalable | javax.jms.Message |
| **updateOrder**(long oldOrderGPK,<br>long negoGPK,<br>long previousQuoteGPK,<br>String newOrderData) | Permet la mise à jour d'une mise préalable | Long : L'identifiant GPK de la nouvelle mise |
| **deleteOrder**(long orderGPK) | Permet la suppression d'une mise | Aucun objet n'est retourné |
| **getMyOrders**() | Récupère l'ensemble des mises effectuées par un négociateur | Document XML au format GNP (XMLDocument) contenant les informations sur les différentes mises retrouvées |
| **getOrder**(long orderGPK) | Récupère les informations d'une mise donnée | Order : l'objet ordre |
| **getOrdersForProductReference**<br>(long productReferenceGPK) | Récupère l'ensemble des mises relatives à un produit donné | java.util.Enumeration : Liste des ordres retrouvés |

Table 5.2 Liste partielle des méthodes de la « session beans » NegotiatorSession

Afin d'exprimer des scénarios de négociations en MSC, nous avons fait les transformations suivantes :

- Nous avons modélisés les objets de base par des *types d'instances* MSC,
- Nous avons modélisés les références vers ces objets par des instances du type correspondant à ces objets. Ceci nous permet d'avoir plusieurs instances d'un même type. Par exemple, on peut avoir un vendeur et cinq acheteurs. Le vendeur et les acheteurs seront modélisés par des instances MSC de types, AnnouncerSession et NegociatorSession, respectivement.

Nous avons également modélisé les méthodes activées par chaque objet en des messages (MSG) au sens MSC. Pour cela, il est nécessaire de conserver la même syntaxe et de respecter la casse des noms de ces méthodes. Le tableau suivant présente les associations que nous avons fait entre ces composants de GNP et les éléments correspondant du langage MSC :

| GNP | MSC |
|---|---|
| AnnouncerSession | Type d'instance |
| NegotiatorSession | Type d'instance |
| Vendeur/Acheteur | Instance |
| SubmitAnnounce | MSG |
| SubmitOrder | MSG |

Table 5.3 Associations entre GNP et MSC

## 5.2 Exemple de scénarios

Deux scénarios simples constitués d'une annonce et d'une mise auront alors la forme suivante :

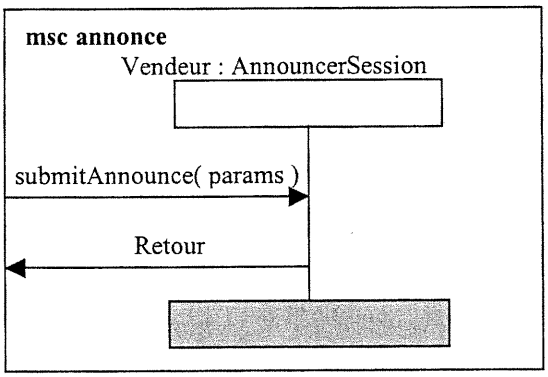


Figure 5.1 (a) scénario d'une annonce

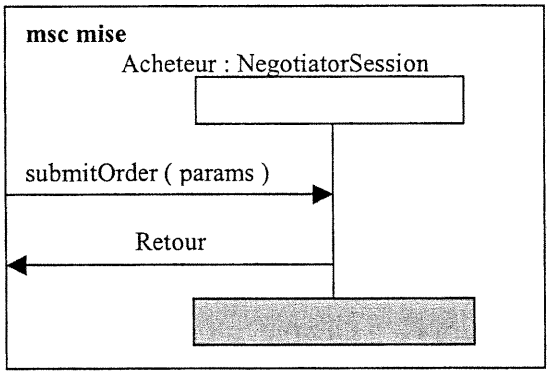


Figure 5.1 (b) scénario d'une mise

L'équivalent MSC/PR de ces deux scénarios est le suivant :

```
msc Annonce
ins ach process AnnouncerSession;
msg submitAnnounce;
msg retour;
ach
 in  submitAnnounce from env;
 out retour     to env;
ach endinstance;
endmsc Annonce;
```

Figure 5.2 (a) Scénario MSC/PR d'une annonce

```
msc Mise
ins vend  process NegotiatorSession;
msg submitOrder;
msg retour;
vend
 in  submitAnnounce from env;
 out retour     to env;
vend endinstance;
endmsc Annonce;
```

Figure 5.2 (b) Scénario MSC/PR d'une mise

Ce simple exemple montre comment, à partir de la connaissance des méthodes des « sessions beans » de GNP et de leur rôle, nous pouvons construire des scénarios d'interaction plus complexes faisant intervenir les différents acteurs du GNP. De façon

plus générale, l'association entre les éléments du GNP intervenant dans les scénarios de test et leur correspondants en formalisme MSC est résumée dans le tableau ci-dessous :

| Élément du GNP | Modélisé par |
|---|---|
| « Sessions/entités beans » | Type d'instance |
| Références aux « session beans » | Instances |
| Méthodes des « sessions/entités beans » | MSG |
| Paramètres des méthodes | Paramètres des messages |

Table 5.4 Équivalence entre les éléments du GNP et le langage MSC

Nous construisons les scénarios de test donc selon la forme suivante :

① **MSCDOCUMENT** fileName;

② **MSC** testAnnouncerSession;

③ # déclarations des messages(= méthodes)

**MSG** getTemplates;

**MSG** submitAnnounce(charstring);

**MSG** retour;

④ # déclaration des instances

**INS** ann01: **PROCESS** AnnouncerSession;

⑤ # Description de l'échange des messages

ann01:

**IN** getTemplates **from ENV;**

**out** retour **to ENV;**

**Condition** typematch (retour, GNPDocument)

**IN** submitAnnounce("announce.xml") **from ENV;**

**out** retour **to ENV;**

**Condition** typematch (retour, GNPDocument)

⑥ ann01: **ENDINSTANCE;**

⑦ **ENDMSC** testAnnouncerSession;

⑧ **ENDMSCDOCUMENT** test_GNP;

Avec:

1 Déclaration du nom du fichier MSC. Ce fichier peut contenir un ou plusieurs scénarios.

2 Déclaration du nom d'un scénario. Cette clause va de pair avec la clause 7.

3 Déclaration des messages d'interaction. Ces messages représentent les noms des méthodes des « sessions beans » intervenant dans le scénario.

4 Déclaration des instances et de leur type. On peut avoir plusieurs instances d'un même type au sein d'un même scénario.

5 C'est ici que se fait la description du scénario. Étant donné que l'interaction des acteurs d'une négociation se fait par le biais d'invocation de méthodes, ces dernières sont exprimées en MSC par des échanges de messages. Autrement dit, chaque fois qu'une méthode est invoquée, un événement (message) en entrée se produit. IN représente un message en entrée en provenance de l'environnement et OUT représente un message en sortie vers l'environnement. En fait, les messages destinés à l'environnement représentent les objets retournés suite à l'invocation des méthodes. Le nom de l'instance impliquée par les échanges précède chaque bloc d'échange. On signale ici que la grammaire textuelle retenue pour l'échange des messages est une grammaire *orientée événements*. Ainsi, il est plus facile d'exprimer un *ordre d'échange global*. La grammaire orientée instance permet d'exprimer un ordre d'échange uniquement au niveau de chaque instance.

6 Déclaration de la fin des instances déclarées dans (4). Cette clause met fin en même temps au scénario en cours.

7 Déclaration de la fin du scénario.

8 Déclaration de la fin du document MSC. D'autres scénarios peuvent être ajoutés entre les clauses (7) et (8).

Il est à signaler que les mots en gras sont le sous-ensemble MSC utilisé pour la description des scénarios et qui est actuellement reconnu par le traducteur. Ce sous-ensemble peut éventuellement être enrichi si de nouveaux besoins apparaissent.

## 5.3 Verdict de test

L'objectif derrière toute activité de test est d'évaluer la conformité du l'implantation sous test par rapport à une spécification de référence. Dans notre cas, nous avons utilisé les « conditions » pour modéliser le verdict de test. Dans MSC, les conditions sont utilisées,

soit pour exprimer un état du système, soit pour la composition de plusieurs MSCs dans un seul HMSC (High-Level MSC). Nous, nous avons exploité le rôle d'expression des états. Chaque fois que le traducteur rencontre une expression de « condition » (clause ⑤) dans un scénario, il la traduit en une structure booléenne « if » associée à un prédicat. L'évaluation de ce prédicat peut être dans l'un des deux état : vrai ou faux. Quand l'évaluation retourne vrai, alors le verdict est « passe ». Dans l'autre cas, le verdict est « échec ». Le nombre et le type de ces prédicats dépend des objectifs de test. Dans le paragraphe 5.4.6.2, nous allons voir que, dépendamment de notre objectif de test, nous avons utilisé trois prédicats.

## 5.4 Architecture de l'environnement de test

Nous avons réalisé l'implantation de l'environnement de test en grande partie par le langage *Jpython*. Avant d'entrer des les détails de l'implantation, nous présentons dans la section suivante un survol de ce langage.

### 5.4.1 *Jpython* : Survol technique

*Jpython* [08, 10] est un langage de programmation puissant et facile à utiliser. Il offre des structures de données de haut niveau puissantes et une approche simple, mais réelle de la programmation orientée-objet. La syntaxe élégante de *Jpython* et le typage dynamique, ajoutés à sa nature interprétée, en font un langage idéal pour écrire des scripts et pour le développement rapide d'applications dans de nombreux domaines et sur la plupart des plates-formes.

L'interpréteur *Jpython* est facilement extensible par de nouvelles fonctions et de nouveaux types de données implémentés en C ou en C++ (ou d'autres langages appelables depuis le C). *Jpython* convient également comme langage d'extension pour des logiciels configurables.

Étant un langage de très haut niveau, *Jpython* contient des types de données de haut

niveau intégrés, comme des tableaux redimensionnables et des dictionnaires qui, autrement, demanderaient des jours à implémenter efficacement. Grâce à ses types de données plus généraux, *Jpython* est applicable à un domaine de problèmes beaucoup plus large que *Awk* ou même *Perl*, et de nombreuses choses sont au moins aussi faciles en *Jpython* que dans ces langages.

*Jpython* permet de séparer les programmes en modules qui peuvent être réutilisés dans d'autres programmes. Il est fourni avec une vaste collection de modules standard qu'on peut utiliser comme base pour les programmes.

*Jpython* est également un langage interprété, ce qui permet de gagner un temps considérable pendant la réalisation de programmes, car aucune compilation ou édition de liens n'est nécessaire. L'interpréteur peut être utilisé de façon interactive, ce qui facilite l'expérimentation avec les possibilités du langage.

Les programmes écrits par *Jpython* sont très compacts et lisibles. Ils sont typiquement beaucoup plus courts que leurs équivalents en C, pour plusieurs raisons :

- les types de données de haut niveau permettent de réaliser des opérations complexes en une seule instruction;
- le regroupement des instructions se fait par indentation, sans accolades de début/fin;
- il n'est pas nécessaire de déclarer les variables ou les arguments.

*Jpython* est extensible: Il est facile d'ajouter une nouvelle fonction intégrée ou un module, écrit par exemple en C, dans l'interpréteur soit pour réaliser les opérations critiques à vitesse maximum, soit pour lier les programmes en *Jpython* à des bibliothèques qui ne sont disponibles que sous forme binaire (comme une bibliothèque graphique propriétaire). On peut aussi lier l'interpréteur *Jpython* dans une application écrite en C et l'utiliser comme langage d'extension ou de commande pour cette application.

La syntaxe de *Jpython* est très simple et, combinée avec des types de donnée évolués (listes, dico,...), conduit à des programmes à la fois compacts et lisibles. Il gère par lui-même ses ressources (mémoires, descripteurs de fichier,...).
La librairie standard de *Jpython* [09], et les paquetages inclus, donnent accès à une grande variété de services : chaînes de caractères et expressions régulières, services UNIX standard (fichiers, pipes, signaux, sockets, threads...), protocoles internet (Web, News, FTP, CGI, HTML...), persistances et bases de données, interfaces graphiques (Tcl/Tk) etc.

*Les types intégrés :*
*Jpython* permet l'utilisation de plusieurs sortes de types intégrés. Ce sont les entiers, les entiers longs, les nombres à virgule flottante et les complexes.

*Les chaînes :*
La chaîne est en fait une chaîne de caractères qui est utilisée pour stocker et représenter de l'information textuelle. L' indiçage et l'extraction sont importants pour la compréhension des types chaînes de caractères, listes et tuples. L'indiçage est le fait de sélectionner un élément par rapport à sa position dans l'objet: mot[5]. L'extraction est le fait de sélectionner une partie plus ou moins grande d'un objet: mot[2:5]. Une extraction impossible donnera un objet vide.

*Les listes:*
La liste est l'objet de collection ordonné le plus souple. Il permet de contenir toutes sortes d'objets : nombres, chaînes, et même d'autres listes: listeMixte = [0, 1, `hello']. Comme les chaînes, les objets d'une liste sont sélectionnables par indiçage. Le contenu des listes peut être modifier directement sur la liste (remplacement, destruction, etc.,). Les listes répondent aussi à l'appel de méthodes. Ces méthodes permettent l'ajout, la suppression, le tri et l'inversion.

*Les dictionnaires :*

Le dictionnaire est un système très souple pour intégrer des données. Un dictionnaire est une liste comportant à la place des indices, un mot servant de clé pour retrouver l'objet voulu. Un dictionnaire est affecté par une paire d'accolades ({}) : dico = {'japon' :'Tokyo', 'canada' : 'Ottawa'} . Les clés peuvent être de toutes les sortes d'objets non modifiables. Par contre, la valeur stockée peut être de n'importe quel type. Un tableau de l'annexe B résume les opérations possibles sur les dictionnaires.

*Les tuples :*

Le tuple est comme la liste, une collection ordonnée d'objets avec la différence que le tuple n'est pas modifiable sur place. Un tuple se déclare avec des valeurs entre parenthèses et non entre crochets comme la liste. tuple = (0, 1.4, `world'). Les tuples peuvent être indicés pour la recherche d'un élément. Une extraction est aussi possible. Ils supportent toutes sortes d'objets et même un tuple dans un tuple. Un tableau de l'annexe B résume les opérations possibles sur les tuples.

*Les fichiers :*

La commande open permet d'accéder au contenu d'un fichier texte: fichier = open(`mon_fichier', `w'). Dans cet exemple, "fichier" représente le contenu du fichier texte "mon_fichier" auquel on peut faire subir les traitements désirés grâce au paramètre d'ouverture "w". Les commandes associées au traitement des fichiers ne se résument que par des méthodes. L'annexe B contient un tableau sur les principales méthodes.

*Les fonctions :*

```
Def fibonacci(n):
    a, b = 0, 1
    while  b < n:
        print b,
        a, b = b, a+b
return
```

L'instruction "def" définis une nouvelle fonction. Le code se trouvant sous une instruction "def" devra être indenté au moins une fois. "return" permet de quitter la fonction et, éventuellement, de retourner une valeur.
Les fonctions acceptent des valeurs par défaut si ces dernières ont été déclarées lors de la définition de la fonction : def fonction(arg1, arg2, arg3="Hello").

*Les classes :*
*Jpython* offre également, mais sans l'imposer, la possibilité de programmer en orienté objet. La programmation orientée objet permet un niveau d'abstraction plus fin. Il est aussi possible, en plus des packages de base fournis avec *Jpython* [09], de charger des librairies Java.

Un mécanisme complet d'exception est également géré par *Jpython*. La structure de gestion des exceptions est faite par bloc de code.

### 5.4.2 Architecture de l'environnement de test

L'environnement de test tel que présenté par la Figure 5.3 se compose des modules suivants:

- un module de traduction,
- un module de paramètres,
- un module d'utilitaires,
- un module de prédicats.

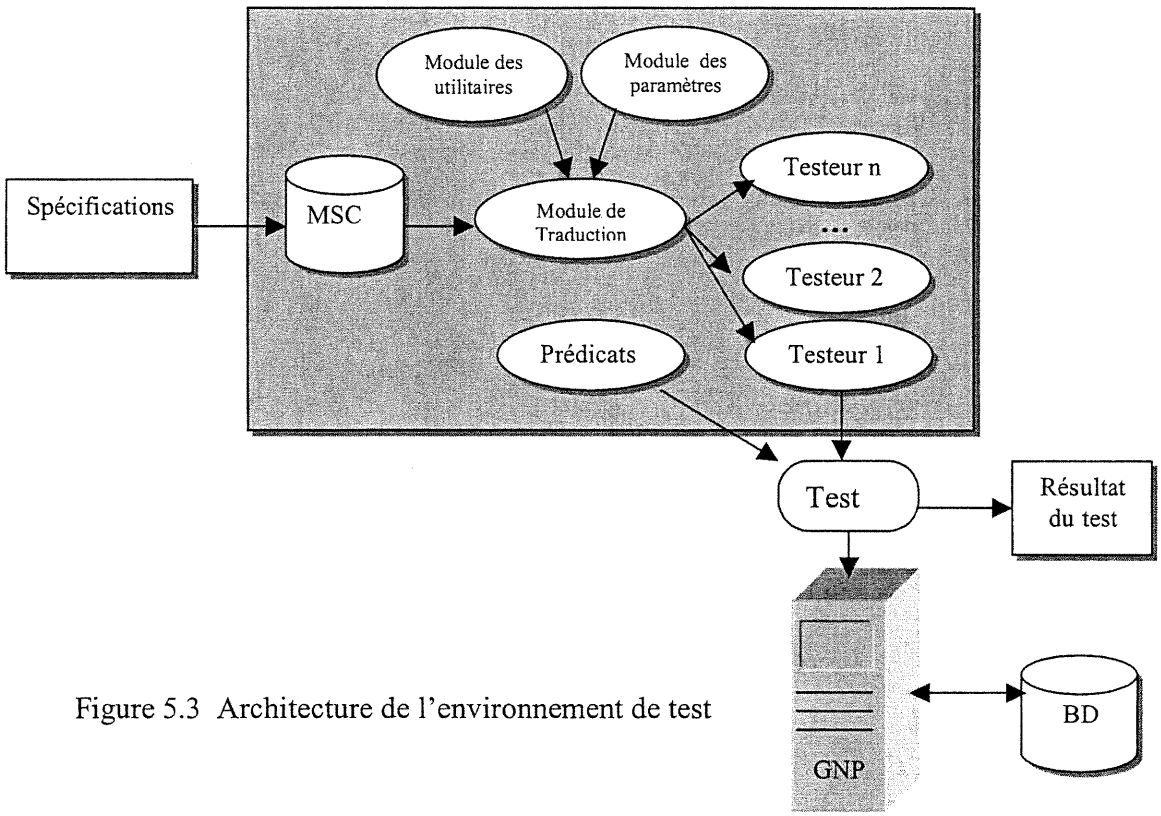


Figure 5.3 Architecture de l'environnement de test

### 5.4.3 Module de traduction

Il s'agit du module principal constituant l'environnement de test. Il permet de générer, à partir d'un fichier MSC, un ou plusieurs scripts *Jpython*. Le nombre de ces scripts est équivalent au nombre des scénarios décrits. Il importe de signaler que les valeurs de certains paramètres ne sont pas connues lors de l'écriture du scénario. Pour miser sur un produit, par exemple, il faut connaître l'identificateur de ce produit (son GPK ou Global Primary Key) et l'identifiant de la négociation concernée. Ces identificateurs ne sont effectivement disponibles que pendant l'exécution du test. Pour remédier à ce problème, nous insérons comme paramètre d'appel le nom de ce paramètre et non sa valeur. Pendant la traduction, ce nom est transformé en un appel de fonction qui, pendant

l'exécution du test, cherche la valeur effective du paramètre avant d'invoquer la méthode impliquée. Le module des paramètres contient la définition de ces fonctions-paramètres.

### 5.4.4 Module des paramètres

Ce module comporte la définition des fonctions-paramètres dont le rôle est de retourner, pendant le test, les valeurs effectives des différents paramètres. Il s'agit essentiellement des identificateurs des objets (GPK) dont les valeurs ne sont connus qu'au moment de l'exécution.

### 5.4.5 Module des utilitaires

Il s'agit d'un module d'utilitaires utilisés par le traducteur et par les testeurs pendant l'exécution du test. Les utilitaires définis dans ce module sont :

- Un utilitaire pour récupérer les informations pertinentes d'une annonce. Parmi ces informations, on trouve essentiellement l'identificateur d'une enchère donnée et l'identificateur de l'item objet de l'enchère.
- Un utilitaire pour ajouter au besoin un nouvel acteur à la liste des participants. Ceci permet d'éviter à se limiter à un nombre restreint de participants dans une négociation. Rappelons que le GNP, pour des considérations de sécurité, n'autorise que l'accès des participants déjà enregistrés .
- Un utilitaire pour construire des ordres de mise sous la forme XML définie par le GNP. Dans le cas ou le scénario de test manipule plusieurs mises, un seul document « order » est suffisant. Les autres seront construits à la volée par cet utilitaire. Pour cela, on se contente lors de la spécification du scénario d'indiquer juste la quantité et le prix misés.
- Un utilitaire pour la sauvegarde intermédiaire des résultats pour une ultérieure utilisation.
- Un utilitaire pour connaître le ou les gagnants d'une enchère donnée.

### 5.4.6 Module des prédicats

Ce module comporte des fonctions « prédicats » permettant de conclure un verdict de test. Deux types de prédicats existent. Le premier type est relatif à un test informatique. Le deuxième concerne le test économique. Le tableau suivant liste les modules que nous avons développé avec leur taille en nombre de lignes de code:

| **Nom du module** | **Taille du module** (*Jpython*) |
|---|---|
| Module de traduction | 631 lignes |
| Module des utilitaires | 213 lignes |
| Module des paramètres | 142 lignes |
| Module des prédicats | 115 lignes |

Table 5.5 Liste et taille des modules de test

#### 5.4.6.1 Test informatique

Bien que l'environnement de test est conçu pour tester les algorithmes économiques des enchères, nous l'utilisons également pour tester les composants informatiques de GNP. Ce test consiste à stimuler les méthodes des différents objets et de tester leur valeur retournée. Le verdict de test est conclut en comparant les valeurs retournées avec celles espérées. C'est ce que nous appelons test informatique.

#### 5.4.6.2 Test économique

Il s'agit du test fonctionnel des différents algorithmes économiques implantés par GNP. Plusieurs objectifs peuvent être définis dans ce test. Pour chaque objectif, il revient au testeur de définir les prédicats correspondants. Les différents prédicats définis dans ce modules sont :

- Un prédicat permettant de vérifier si l'objet en retour suite à l'appel d'une méthode correspond à une classe donnée.

- Un prédicat pour vérifier si une exception survenue suite à l'appel d'une méthode appartient à une classe d'exceptions déterminée. Les deux premiers prédicats sont utilisés dans les scénarios informatiques.
- Un prédicat permettant de vérifier si un « négociateur » donné est gagnant (ou parmi les gagnants) d'une enchère. Ce prédicat est utilisé dans les scénarios économiques. Ce prédicat permet également de préciser la quantité et le prix alloués à un éventuel gagnant.

L'exécution des différents testeurs résultants de la traduction des scénarios est faite de façon séquentielle. Chaque testeur s'exécute à la manière d'un client « lourd » (voir la Figure 3.1). Des rapports contenant les résultats d'exécution sont crées au fur et à mesure que le test est exécuté. On distingue entre deux niveaux de détail dans ces rapports: un niveau sommaire et un niveau détaillé. Le rôle du rapport sommaire est de renseigner brièvement sur l'exécution des scénarios de test sans être concerné par les détails. Dans ce rapport, chaque script de test possède une entrée indiquant si le verdict est « passe » ou « échec ». Dans le cas ou le verdict est « échec », le rapport détaillé du scénario en question apporte une assistance quant à la détection de l'origine de l'erreur. Autrement dit, le rapport détaillé constitue la première étape du diagnostic.

CHAPITRE 6

# EXPERIMENTATIONS ET RESULTATS

Dans le but d'évaluer l'outil de test développé, nous avons effectué des expérimentations sur diverses enchères. Nous présenterons dans la section suivante les résultats relatifs au test de l'enchère dite enchère de Vickrey [Vic 61], connue aussi sous le nom de l'enchère au deuxième prix progressif ( progressive second price).

## 6.1 Présentation de l'enchère PSP

Le mécanisme d'enchère au deuxième prix progressif [Rob 00], tel que mis en œuvre par le GNP, fonctionne théoriquement de la manière suivante. A chaque ronde, le système tient à jour l'état du marché E. A la fin de chaque ronde, une partie de l'information contenue dans l'état du marché est fournie à chacun des participants. Le participant peut alors soumettre de nouvelles mises (ordres) au marché. Après un délai fixé d'avance, le système met à jour l'état du marché en fonction des ordres reçus durant la ronde. L'enchère continue jusqu'à ce qu' une règle d'arrêt soit appelée.

### 6.1.1 Règles de l'enchère

Ce modèle propose les règles suivantes :

(i) Les enchères sont ascendantes pour la vente d'objets.

(ii) Les enchères fonctionnent par rondes. Chaque nouvelle ronde est déclenché après un délai fixe (la gestion du temps n'est pas décrite ici).

(iii) Plusieurs unités du même objet peuvent être vendues. Chaque unité peut être allouée à un acheteur différent. Le vendeur ne peut ajouter des unités supplémentaires durant la négociation.

(iv) Les participants peuvent soumettre des prix limites (mais ne sont pas obligés).

(v) La règle d'arrêt est basée sur une règle d'activité commune de la forme : la négociation s'arrête après x rondes d'inactivité. D'autres règles basées sur une activité individuelle peuvent également être incluses.

(vi) Aucune adjudication ne se fait durant la négociation.

### 6.1.2 Constantes du marché

Au-delà du choix des règles à utiliser, de la définition des produits à vendre, etc., certaines informations sont requises au moment de l'initialisation de l'enchère. Nous appellerons les constantes du marché les données qui sont utilisées par les scripts économiques et ne varient pas durant la négociation. Dans notre contexte, les constantes du marché sont :

(i) le nombre de rondes d'inactivité après lesquelles la négociation se termine,

(ii) l'incrément minimal pour la négociation,

(iii) la quantité totale offerte QT,

(iv) le prix limite fixé.

### 6.1.3 Mises de la ronde

Durant une ronde, les mises des participants sont enregistrées et placées dans une queue de messages. Chaque mise est identifiée à un acheteur et réfère à un produit. Ces mises sont extraites, identifiées et rangées dans un tableau. Chaque mise est identifiée par :

(a) le code (id) identifiant de l'acheteur,

(b) le numéro (ro) de la ronde,

(c) le prix limite (pl) soumis,

(d) la quantité demandée (qd),

(e) le prix alloué (pa) à,

(f) la quantité (qa) allouée à.

Les mises sont limitées de sorte que $\Sigma qd \leq QT$. Le tableau doit être construit de façon à ce que les mises inscrites avant soient placées en haut du tableau. À chaque ronde, l'état du marché est mis à jour. Il contient l'ensemble des informations pertinentes pour la suite de la négociation. Après la ronde r, l'état du marché contient les éléments suivants :

- le numéro de la ronde actuelle,
- le nombre de périodes d'inactivité dans la négociation,
- un tableau décrivant l'allocation intérimaire. Dans le cas où la négociation devrait se terminer sans de nouvelles mises, l'allocation intérimaire deviendrait l'adjudication finale.

### 6.1.4 Allocation intérimaire

L'allocation intérimaire (AI) prends la forme d'un tableau d'entrées. Plus spécifiquement, $AI = \{ id[i], ro[i], pl[i], qd[i], pa[i], qa[i]\}_{i=1,k}$ *(k est le nombre d'entrées dans le tableau).* Par exemple :

| *I* | *Id[i]* | *ro[i]* | *pl[i]* | *qd[i]* | *pa[i]* | *qa[i]* |
|---|---|---|---|---|---|---|
| 1 | Ach03 | 10 | 21 | 5 | 18 | 5 |
| 2 | Ach01 | 12 | 20 | 7 | 18 | 7 |
| 3 | Ach02 | 10 | 18 | 6 | 18 | 6 |
| 4 | Vend01 | 0 | 17 | 20 | 17 | 2 |

Table 6.1 exemple de tableau des allocations intérimaires

Une entrée peut prendre deux formes, Il peut s'agir d'une offre déposée par le vendeur lors de l'initialisation du marché ou d'une mise déposée par un acheteur. Dans les deux cas, l'acheteur et le vendeur déposent deux prix, le prix ferme et le prix limite. La

quantité demandée ne peut jamais excéder la quantité maximale disponible QT. Dans l'exemple ci-dessus, QT=20.

Les entrées sont classées par ordre de priorité. L'entrée *i=1* détenant l'ordre de priorité le plus élevé. Le classement se fait d'abord selon le critère de prix, les mises ayant un prix plus élevé ont un ordre de priorité supérieur. A prix égal, les mises introduites plus tôt dominent celle soumises plus tard. A prix et temps égals, les mises sont classées aléatoirement ou sur la base d'un autre critère à définir.

### 6.1.5 Calcul de l'allocation intérimaire

Une fois les nouvelles mises insérées dans la liste, il faut recalculer trois éléments : (i) les quantités allouées, qa, (ii) les prix alloués, pa, et (iii) le nouveau k correspondant au nombre d'entrées qui seront retenues dans le nouvel état du marché.

La règle d'allocation des quantités se confond avec la règle de priorité. Une fois l'ordre de priorité établi, il suffit d'allouer les unités en descendant l'ordre de priorité. La règle est fixée en commençant par le haut du tableau. Les quantités qa allouées sont calculées selon la formule :

$$qa[i] = Max\left[0, Min\left[qd[i], QT - \sum_{l=1}^{i-1} qa[l]\right]\right]$$

Le calcul de qa est utilisé pour déterminer k :

$$K = Min\{j | qa[j] < Min[qd[j], QT]\}$$

La suite de la procédure d'allocation consiste à calculer le prix. Les principes sous-jacents au calcul des prix alloués sont que: (i) Personne ne paie moins que son prix ferme; (ii) Personne ne paie plus que son prix limite; et (iii) entre ces deux bornes, le prix est un incrément au-dessus du prix du plus haut perdant si la mise est postérieure à la mise perdante, et égal au prix du plus haut perdant autrement.

## 6.2 Sélection des scénarios de test.

Les scénarios de test sélectionnés pour tester l'implantation de l'enchère PSP par le GNP ont pour but de répondre à deux objectifs : (i) s'assurer que les mises en dessous du prix limite fixé par le vendeur sont rejetées par le système, et (ii) vérifier que le choix des gagnants est établi en fonction du prix proposé dans la mise. D'autres objectifs peuvent être définis auquel cas d'autres cas de test doivent être sélectionnés.

Pour juger le premier objectif de test, nous retenons quatre scénarios de test. Dans les deux premiers scénarios, on trouve deux mises : une mise dont le prix est au dessus de la limite acceptable et une autre dont le prix est au dessous. La valeur des mises est identique dans le deux scénarios. Cependant leur ordonnancement est inversé à chaque fois. L'objectif est d'identifier une éventuelle influence de l'ordre des mises sur les résultats finals. Finalement, dans le troisième et le quatrième scénarios, des mises avec des valeurs limites sont testées. Les valeurs limites sont les valeurs qui délimitent deux classes d'équivalences : la classe des mises acceptées et la classe des mises refusées (voir la Figure 6.1). Comme c'est le cas pour les scénarios 1 et 2, le test de l'influence de l'ordre d'arrivée de ces mises est également testé.

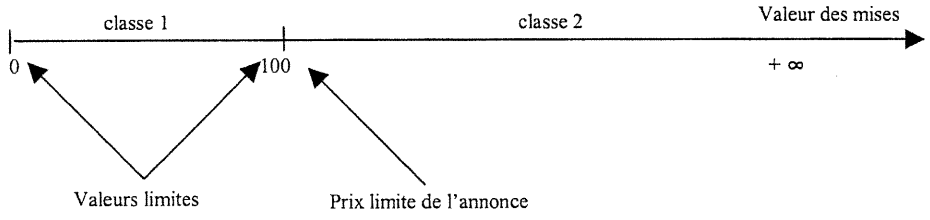


Figure 6.1 Valeur des mises possibles

Les résultats de ce test sont présentés dans le tableau suivant :

| | | *QT* | *Pl* | *Ordre de la mise* | *qd* | *pd* | *qa* | *Verdict* |
|---|---|---|---|---|---|---|---|---|
| Scénario 1 | Ach1 | 20 | 100 | 1ère | 15 | 110 | 15 | ***Passe*** |
| | Ach2 | | | 2ème | 15 | 90 | 0 | |
| Scénario 2 | Ach1 | | | 2ème | 15 | 110 | 15 | ***Passe*** |
| | Ach2 | | | 1ère | 15 | 90 | 0 | |
| Scénario 3 | Ach1 | 20 | 100 | 1ère | 15 | 0 | 0 | ***Passe*** |
| | Ach2 | | | 2ème | 15 | 100 | 15 | |
| Scénario 4 | Ach1 | | | 2ème | 15 | 0 | 0 | ***Passe*** |
| | Ach2 | | | 1ère | 15 | 100 | 15 | |

Table 6.2 Mises et résultats observés pour l'objectif (i)

Il est clair d'après les résultats des deux premiers scénarios que si la mise en dessous du *pl* avait été acceptée, l'acheteur 2 aurait gagné les cinq unités restantes de l'enchère. Dans ce cas particulier, les unités non vendues sont considérées comme « achetées » par le vendeur lui même. Dans le cas des deux derniers scénarios, la mise avec la valeur limite « 0 » a été rejetée dans les deux cas. Par contre, celle avec la valeur limite « 100 » a été acceptée. Ceci confirme que l'implantation est conforme par rapport à l'objectif (i).

Pour vérifier la conformité du deuxième objectif, à savoir que le choix du gagnant est déterminé en fonction de la valeur de la mise correspondante, nous sélectionnons également deux scénarios composés de deux mises chacun. L'ordre d'arrivée de ces mises est différent dans les deux scénarios afin de vérifier une éventuelle influence de l'ordre d'arrivée des mises sur les résultats finals. Étant donné que l'objectif (i) est vérifié séparément, nous n'auront pas besoin de traiter le cas des mises inférieures au prix limite. Les valeurs de mises de ces scénarios sont donc supérieures au prix limite.

Les valeurs et les résultats de ces mises sont présentées dans le tableau suivant :

| | | *QT* | *Pl* | *Ordre de la mise* | *qd* | *pd* | *qa* | *pa* | *Verdict* |
|---|---|---|---|---|---|---|---|---|---|
| Scénario 1 | Ach1 | 20 | 100 | 1ère | 20 | 110 | 0 | | *Passe* |
| | Ach2 | | | 2ème | 20 | 120 | 20 | 111 | |
| Scénario 2 | Ach1 | | | 2ème | 20 | 110 | 0 | | *Passe* |
| | Ach2 | | | 1ère | 20 | 120 | 20 | 110 | |

Tableau 6.3 Mises et résultats observés pour l'objectif (ii)

Nous remarquons que les prix alloués sont parfois légèrement supérieurs à celui de la mise perdante. Ceci est dû au fait que la mise gagnante est postérieure à la mise perdante. Dans ce cas, un incrément est ajouté au prix de la mise perdante. Revoir le paragraphe 6.1.5 pour plus d'explications.

Les scénarios MSC de ces cas de tests et les leurs résultas sont présentés dans l'annexe C.

## 6.3 Cas concret d'utilisation

Dans la section suivante, nous allons présenter, à travers l'exemple du premier scénario de l'objectif (i), l'utilisation effective du traducteur et les résultats obtenus.

### 6.3.1 Présentation du scénario

Le permier sénario de test, tel que présenté à l'annexe C, se présente comme suit:

```
mscdocument
msc     Scenario_01

msg  submitAnnounce (charstring)
msg  submitOrder (long, long, long, charstring)
msg  retour

ins vendeur:    process   AnnouncerSession
ins ach01:      process   NegotiatorSession
```

```
ins ach02:      process   NegotiatorSession

vendeur:
     in   submitAnnounce("announce.xml")     from env
     out  retour     to env

ach01 :
     in   submitOrder(productReferenceGPK(retour),
          negoGPK(retour), previousQuoteGPK(retour),
          Order("order.xml", 15, 110))  from env

ach02 :
     in   submitOrder(productReferenceGPK(retour),
          negoGPK(retour), previousQuoteGPK(retour),
          Order("order.xml", 15, 90))   from env

condition leader(retour, ach01)
condition leader(retour, ach02)

vendeur:   endinstance
ach01:          endinstance
ach02:          endinstance
endmsc Scenario_01
```

Note: Les mots en gras sont des mots réservés du langage MSC.

Selon le paradigme MSC le scénario commence par le mot-clé **MSCDOCUMENT**. Ensuite, vient la déclaration du scénario de test par le mot-clé **MSC** suivi du nom du scénario. C'est ce même nom qui sera attribué après la traduction au script de test relatif à ce scénario. Dans notre cas, il s'agit d'un seul scénario. On peut en avoir plusieurs dans le même document les uns à la suite des autres séparés par les mots-clés **MSC** et **ENDMSC**. La déclaration des messages vient immédiatement après. Deux messages sont retenus pour cet exemple: submitAnnounce et submitOrder. Les trois acteurs du scénario, à savoir vendeur, ach01 et ach02 sont déclarés en tant que types d'instances AnnouncerSession et NegotiatorSession. Le corps du scénario proprement dit commence ensuite. Selon le paradigme MSC, chaque suite d'événements est précédée de l'acteur responsable. Dans

l'exemple, ce sont vendeur qui commence par soumettre une annonce (**IN** submitAnnounce) et ach01 et ach02 qui terminent chacun par une mise sur cette annonce (**IN** submitOrder). Le premier paramètre de submitOrder est la clé primaire globale de la référence du produit objet de l'enchère (productReferenceGPK). Comme la valeur de ce paramètre n'est connue qu'après la soumission de l'annonce, nous avons substitué la valeur correspondante par le nom du paramètre. Pendant l'exécution du test, ce nom va provoquer l'appel d'une fonction du module des paramètres qui, en retour, donnera la valeur réelle de ce dernier, et ceci, évidemment, avant l'appel de la méthode sumbitOrder. C'est le cas également pour tous les autres paramètres dans la même situation.

Pour cet exemple, nous avons retenu deux verdicts de test: nous voulons savoir dans un premier temps si ach01 est le gagnant de l'enchère (ce qui devrait être le cas). Dans un deuxième temps, nous nous demandons si ach02 a, également, été déclaré gagnant (ce qui ne devrait pas se produire dans une situation normale).

### 6.3.2 Traduction et exécution du scénario

L'appel du traducteur se fait de la manière suivante:

```
C:> Translate scenario.msc
MSCDocument..................................  [ ok ]
MSC..........................................  [ ok ]

Reconnaissance des messages...
SubmitAnnounce ..............................  [ ok ]
SubmitOrder .................................  [ ok ]

Reconnaissance des instances...
vendeur .....................................[ ok ]
ach01   .....................................  [ ok ]
```

```
ach02    ..............................  [ ok ]
...
Le(s) script(s) suivant(s) a(ont) été généré(s):
Scenario_01.py ........................  [ ok ]
C:>
```

Avec `translate` est le nom de la commande de traduction (module de traduction) et `scenario.msc` est le nom du fichier contenant la description MSC du scénario de test. L'affichage sur écran indique l'évolution de l'analyse et de la traduction. Chaque fois que le traducteur reconnaît et traduit une section, il l'affiche sur écran permettant ainsi un suivi visuel de l'évolution de la traduction. Les éventuelles erreurs de syntaxe ou de logiques sont signalées avec le numéro de ligne correspondant.

Le code *Jpython* généré suite à la traduction de ce scénario est le suivant :

```
#  Section des Imports

import org.w3c.dom
import cirano.gnp.beans
import javax.jms
import java.util
from javax.naming import *
from cirano.gnp import ContextUtil, Log
from cirano.gnp import DOMUtil
from javax.ejb import *
from java.util import *
from java.lang import System
from time import sleep, strftime, localtime, clock
import os
import time

import testUtil, params, predicats
from testUtil import *
from predicats import *
from params import *

import sys
sys.path.insert(0, "/usagers/"+ System.getProperty ("user.name+"/Projets/gnp/msc/lib")
```

```
#Initialisation du serveur d'enchère : doit être modifier si nécessaire

server = \"t3://"+localServer+":"+localPort+"\"

if not os.path.isdir("/usagers/"+System.getProperty("user.name+"/TestsVerdicts/"+strftime
                                                    ("%a%d%b%Y", localtime(clock()))+"/:")
      os.makedirs("/usagers/"+System.getProperty("user.name+"/TestsVerdicts/"+strftime
                                                    ("%a%d%b%Y", localtime(clock()))+"/")

pathSom =  "/usagers/"+System.getProperty("user.name+"/TestsVerdicts/"')
pathDet = "/usagers/"+System.getProperty("user.name+"/TestsVerdicts/"+strftime
                                                    ("%a%d%b%Y", localtime(clock()))+"/"')

logSom = open(pathSom + strftime ("%a%d%b%Y", localtime(clock()))+".log", "a')
logDet =  Log.getNamedInstance(pathDet + ' + '"' + elem[1] + ".log")

logSom.write(">>>>> ---------------------------- DEBUT d'exécution du script: scenario_01.py ")
logDet.info  ("|--------------------  détail des verdicts de tests de scenario_1.py  ----------------|")

createPlayer("vendeur")
player = "vendeur"
pwd    = "vendeur"
ctx = ContextUtil.getNamingContext(server, player, pwd)

#
# Initial Context creating
#
try:
        announcerhome = ctx.lookup("beans.AnnouncerSessionHome")
except:
        print "Erreur!!: ne peut pas obtenir une home référence"

try:
        announcer = announcerhome.create()
except:
        print "Erreur: ne peut pas obtenir une session référence"

createPlayer("ach01")
player = "ach01"
pwd    = "ach01"
ctx = ContextUtil.getNamingContext(server, player, pwd)

#
# Initial Context creating
#
try:
        negotiatorhome = ctx.lookup("beans.NegotiatorSessionHome")
```

```
except:
        print "Erreur!!: ne peut pas obtenir une home référence"

try:
        negotiator = negotiatorhome.create()
except:
        print "Erreur: ne peut pas obtenir une session référence"

createPlayer("ach02")
player = "ach02"
pwd    = "ach02"
ctx = ContextUtil.getNamingContext(server, player, pwd)

#
# Initial Context creating
#
try:
        negotiatorhome = ctx.lookup("beans.NegotiatorSessionHome")
except:
        print "Erreur!!: ne peut pas obtenir une home référence"

try:
        negotiator = negotiatorhome.create()
except:
        print "Erreur: ne peut pas obtenir une session référence"

vendeur.submitAnnounce("announce.xml")
return_result(retour)

ach01.submitOrder(productReferenceGPK(retour), negoGPK(retour), previousQuoteGPK(retour),
Order("order.xml", 15, 110))

ach02.submitOrder(productReferenceGPK(retour), negoGPK(retour), previousQuoteGPK(retour),
Order("order.xml", 15, 90))

if leader(ach01)
        logDet.info ("Ligne: " + 26 + "  <--->    [ PASS ]")
else
        logDet.info ("Ligne: " + 26 + "  <--->    [ FAILL ] ")

if leader(ach02)
        logDet.info ("Ligne: " + 27 + "  <--->    [ PASS ]")
else
        logDet.info ("Ligne: " + 27 + "  <--->    [ FAILL ] ")

#  ----  Fin de la traduction du scénario en cours..~/~
```

L'éxecution de test se fait par l'appel *Jpython* du script généré:

```
C:> Jpython scenario_01.py
```

À la suite de l'éxecution du test, un nom de répertoire composé de la date du jour est crée et un fichier correspondant au script de test est crée dans ce répertoire. Ce fichier contient le résultat de tous les verdicts de test dans le scénario sous la forme suivante:

```
Ligne:xx (leader (result, ach01)  - - -  SUCCESS
```

Ou, dans le cas d'un échec:

```
Ligne:xx (leader (result, ach01)  - - -    FAIL
```

En plus de ces fichiers individualisé, un fichier global est crée dans ce meme répertoire. Ce fichier indique, pour chaque scénario s'il a complètement réussi, ou, par contre, comporte au moins un verdict avec la mention `- - FAIL.`

## 6.4 Architecture de test

L'architecture adoptée pour l'exécution des scénarios de test se présente comme à la figure 6.2.

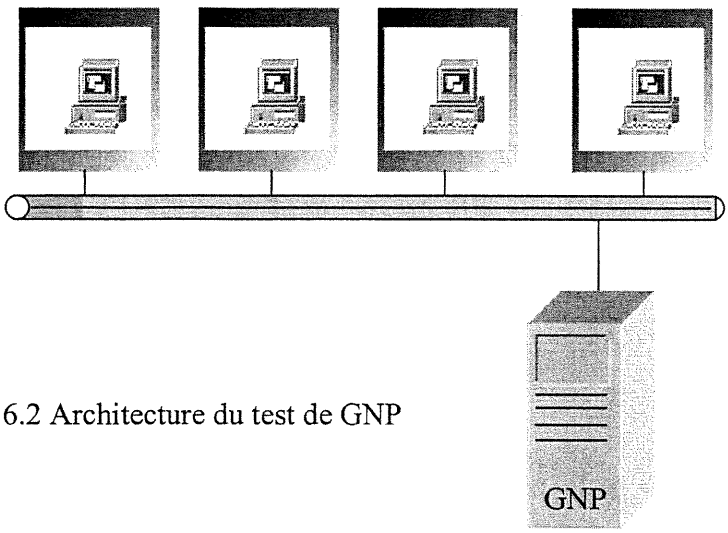


Figure 6.2 Architecture du test de GNP

Nous avons utilisé deux architectures pour exécuter les tests. Dans la première, dite test local, le testeur et GNP se trouve sur la même machine. La deuxième architecture utilisée correspond au test à distance. Dans cette architecture, le testeur et GNP se trouvent sur deux machines séparées. Le testeur accède aux PCOs de GNP via un service de communication. L'architecture de test des protocoles de communication la plus proche est présentée à la Figure 6.3. L'absence d'un testeur supérieur est justifiée par le fait que GNP ne possède aucune couche supérieure au dessus. Les deux architectures utilisées ont donné les mêmes résultats.

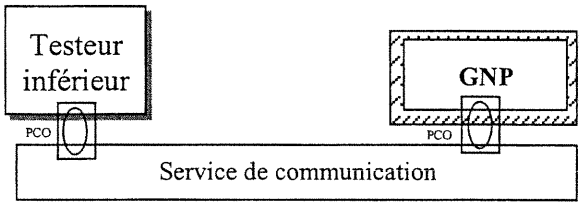


Figure 6.3 Architecture du test à distance de GNP

CHAPITRE 7

# CONCLUSION

L'activité de test est essentielle dans le développement de tout système logiciel. Elle est nécessaire afin d'évaluer ce que le système fait, et comment il le fera dans son environnement final.

Dans ce mémoire, nous avons défini ce qu'est un marché électronique et les moyens informatiques requis pour sa mise en place. Les enchères électroniques sont une application de ces marchés. Nous avons passé en revue différents modèles d'enchères et nous avons tardé davantage sur les enchères ouvertes. Ce modèle d'enchère se révèle être plus efficace que les enchères fermées, surtout dans un contexte où l'estimation précise de l'utilité du contrat ou de l'objet de l'enchère n'est pas facile. Notre contribution concerne le développement d'un environnement pour tester un serveur de négociations électronique, GNP. L'environnement de test développé se compose de :

- un formalisme de description des scénarios de test,
- une démarche pour la dérivation des scénarios,
- un traducteur.

Le formalisme de description des scénarios de test est conforme au langage MSC [Z.120]. La démarche à suivre pour la dérivation des scénarios apporte une assistance pour exprimer ces scénarios de façon conforme au formalisme. Le traducteur, écrit en *Jpython*, a pour fonction de transformer ces scénarios MSC en des robots de test. Ces scripts *Jpython* exécutables jouent le rôle de testeurs. Le verdict de test est posé automatiquement pendant l'exécution des tests.

Nous avons choisi MSC à cause, d'une part, de son aspect intuitif surtout dans sa forme graphique, et d'autre part, grâce à son support par plusieurs outils de test tel ObjectGEODE [Obj 96]. Les scénarios de test doivent être écrit en respectant les directives (guidelines) décrites dans le paragraphe 5.1.

Les expérimentations que nous avons réalisées ont montré qu'il était devenu plus facile de procéder à des tests systématiques des modèles d'enchères existants de GNP. Le développement de nouveaux modèles ne pose plus un grand problème pour s'assurer de leur validité et leur conformité au modèle de base.

Il nous a été donné de constater quelques « bugs » pendant les tests que nous avons effectué :

- Audelà de six annonces simultanées, la première annonce disparaît du système.
- Ceratines mises disparaisent mystérieusement du GNP. Après diagnostic, il s'est avéré que les mises dont le prix est égal au prix limite de l'annonce n'étaient pas accéptées.
- Au niveau du composant negotiatorSession, certaines méthodes qui devaient générer une exception quand les paramètres transmis ne sont pas correctes, se comportaient «normalement». C'etait le cas nottamment quand nous avons essayé d'annuler une commande qui n'existe pas, auquel cas, le GNP n'a généré aucune exception.

À pars ces anomalies légères, nous n'avons pas pu constater des erreurs de logique graves étant donné que l'algorithme de l'enchère au deuxième prix est relativement simple. Notre environnement de test sera d'autant plus utile que les modèles d'enchères sont plus complexes.

Comme extension à notre travail, nous proposons des recherches dans les directions suivantes:

- L'ajout d'un module pour tenir compte des scénarios concurrentiels. Ces derniers ont une grande capacité à détecter une classe d'erreurs qu'on ne peut détecter autrement.

- Développer une interface de connexion entre le traducteur et un générateur automatique de scénarios de test. Certes, l'environnement de test a été développé en vue d'un usage par le GNP, cependant, il peut être étendu facilement à une large gamme de logiciels d'autant plus que le traducteur est portable sur plusieurs plate-formes. Pour cela, une connexion avec un outil de génération de test serait d'une grande utilité.

## BIBLIOGRAPHIE


[**Bel 89**] Ferenc Belina, Dieter Hogrefe, « The CCITT Specification and Description Language SDL», Computer Networks and ISDN Systems, Vol. 16, 1989.

[**Bol 87**] T. Bolognesi. Introduction to the ISO Specification Language LOTOS. Computer Networks and ISDN Systems, Vol.14 (1), 1987, pp.25-59.

[**Bou 99**] C. Bourhfir, "Génération automatique de cas de test pour les systèmes modélisés par des machines à états finis communicantes.", thèse de PhD, DIRO, Université de Montréal, Avril 1999.

[**Bud 87**] S. Budkowski. An introduction to ESTELLE : A Specification Language for Distributed Systems. Computer Networks and ISDN Systems, Vol.14 (1), 1987, pp.03-23.

[**Cla 76**] L. Clarke, « A System to Generate Test data and Symbolically Execute Programs», IEEE Transactions on Software Engineering, 2(3), Septembre 1976.

[**Cla 86**] E.M. Clarke, E.A. Emerson, A.P. Sistla, « Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications », ACM. Transactions on Programming Languages and Systems, 8(2):244-263.

[**Cou 94**] Coulouris, G. Dollimore, J. and Kindberg, T. Chapter 18 of Distributed Systems - Concepts & Design (second edition), Addison-Wesley, (1994).

[**Dss 99**] R. Dssouli, K. Saleh, E. Aboulhamid, A. En-Nouaary and C. Bourhfir, « Test development for communication protocols : towards automation », Computer Networks, Vol. 31, No 17, juin 1999.

[**Gér 01**] Robert Gérin-Lajoie, Sophie Lamouroux, El Mostafa Ben Najim, Isabelle Therrien, Vincent Trussart « Architecture informatique de GNP V1.0 », document d'analyse interne au CIRANO, Février 2001.

[**Gil 62**] A.Gill. « Introduction to the Theory of Finite State Machines », McGraw-Hill, 1962.

[**Gra 94**] Jens Grabowski, « SDL and MSC Based Test Case Generation : An Overall View of the SaMsTag Method», Institut fur Informatik, Universitat Bern, May 1994.

[**Hac 76**] M. Hack. «Petri Net Languages», Technical Report 159, Laboratory for computer Science, MIT, 1976.

[**Hol 91**] Gerard J. Holzmann, « Design and Validation of Computer Protocols », Prentice Hall, 1991.

[**Kce 98**] Kacem Saleh, "Paper on Issues in Testing for E-Commerce", 1st IBM internet Workshop on E-Commerce - Toronto - September 1998.

[**Khe 00**] F. Khendek and D. Vincent, « Enriching SDL Specifications with MSCs», Proceedings of the SDL And MSC Workshop (SAM'2000), Grenoble, France, June 26-28, 2000.

[**Koh 78**] Z. Kohavi. « Switching and Finite Automata Theory », Mc Graw-Hill Computer sciences Series, 1987.

[**Kro 86**] Fred Kroeger, « Temporal Logic of Programs » Monographs on Theoretical Science. Springer-Verlag, 1986.

[**Led 91**] G. Leduc, « On the Role of Implementation Relations in the Design of Distribute Systems Using LOTOS», Thèse d'agrégation de l'enseignement Supérieur, Université de Liège, 1991.

[**Mye 76**] G. J. Myers, « Software Reliability », A Willey-Inter-Science Publication 1976.

[**Mye 79**] G. J. Myers, « The Art of Software Testing», John Willey & Sons, 1979.

[**Nau 69**] P. Naur, B. randall (eds) « Software engineering : A report on a conference sponsored by the NATO Science Commitee » NATO, Bruxelles, Belgique, 1969.

[**Obj 96**] ObjectGeode « ObjectGeode, MSC editor, User's Guide », Verilog 1996.

[**Rob 99**] Jacques Robert, «Vers une bourse électronique d'achats publics», Forum des marchés publics, Québec, 1999.

[**Rob 00**] Jacques Robert, «Description des règles d'enchères», Montréal, CIRANO, 2000.

[**Vic 61**] W. Vickrey, « Counter speculation, auctions and competitive sealed tenders», Journal of Finance, 16, 1961.

[**Was 77**] A. Wasserman, « On the Meanning of Descipline in Software Design and Development », Software Engineering techniques, Infotech state of the art report, 1977.

[**Whi 78**] C. H. White, « System reliability and integrity » in Infotech State of the art report, Infotech, 1978.

[**Wil 99**] Wiliam J. Pardi, « XML en Action », Microsoft Press, 1999.

[**Z.100** ] ITU-T, « Recommandation Z.120 – Specification and description Language (SDL) » 1996.

[**Z.120**] ITU-T, « Recommandation Z.120 – Message Sequence Chart (MSC) », 2000.

## Références en ligne:

[**01**] JOnAS (Java Open Application Server) : www.bullsoft.com/ejb

[**02**] Enterprise JavaBeans Specifications (JavaSoft) : www.javasoft.com/products/ejb/docs.html

[**03**] XML Specification (W3C) : www.w3.org/XML

[**04**] The Simple API for XML : www.megginson.com/SAX/

[**05**] Document Object Model : www.w3.org/XML

[**06**] Java Servlets API (Javasoft) : www.javasoft.com/products/servlet/index.html

[**07**] Java Apache Project : java.apache.org.

[**08**] JPython : www.*Jpython*.org

[**09**] Python Library Reference : marcelin.cirano.qc.ca/devel/python/lib/lib.html

[**10**] Python Tutorial : marcelin.cirano.qc.ca/devel/python/tut/tut.html

# ANNEXE A: EXEMPLE DE DOCUMENTS GNP EN XML

*Exemple de document "PartyDescription" minimal*

```
<gnpDocument>
    <party name="vendeur72">
        <description/>
    </party>
</gnpDocument>
```

*Exemple de document "PartyState"*

```
<gnpDocument>
 <party>
      <allowedNego>2686977</allowedNego>
      <state>
      <partyKey>u13</partyKey>
            <hiddenInfo><status>active</status></hiddenInfo>
            <displayedInfo>
                  <portfolio>
                        <priceQuantity><price>30</price><quantity/>
                        </priceQuantity></portfolio>
                  <costs/><lastCosts/><profit/><lowestCost/><note/>
                  </displayedInfo>
      <privateOrderDomain>
       <gnpForm>
            <input name="/gnpDocument/order/timeOut" type="hidden"
            value="0"/>
            <input name="/gnpDocument/order/sentToRound" type="hidden"
            value="0"/>
            <input name="/gnpDocument/order/sentPhase" type="hidden"
            value="setupA"/>
            <input name="/gnpDocument/order/side" type="hidden"
            value="sell"/>
      <input name="/gnpDocument/order/sentPQList/priceQuantity/price"
      type="hidden" value="0"/>
            <input checked="" descCout="113" descNote="71"
            name="/gnpDocument/order/sentPQList/priceQuantity/quantity"
                  type="radio" value="113;71"/>
            <input descCout="115" descNote="72"
            name="/gnpDocument/order/sentPQList/priceQuantity/quantity"
            type="radio" value="115;72"/>
            <input descCout="131" descNote="87"
            name="/gnpDocument/order/sentPQList/priceQuantity/quantity"
            type="radio" value="131;87"/>
            <input name="submitButtons" type="submit"
      value="Soumettre"/>
       </gnpForm>
```

```
      <message>
      Veuillez choisir vos coûts et votre note de qualité. Vous
      disposez de 60 secondes pour prendre votre décision.
      </message>
      </privateOrderDomain></state></party>
</gnpDocument>
```

***Exemple de document "Rules"***

```
<!--   RULES    -->
<!-- Ne pas afficher nécessairement mais disponible a tous les
joueurs  -->

<?xml version="1.0" encoding="UTF-8"?>
<gnpDocument GPK="2490398" creationTime="960218160000"
negotiationGPK="2490397">
    <rules>
     <economicAlgorithm>

<orderValidator>cirano.gnp.NOOPOrderValidator</orderValidator>
            <auctionneer
class="cirano.gnp.services.PythonAuctionneer">
                <param name="scriptFile"
value="/usagers/lamouros/Projets/gnp/auctionneers/auctionneer032s
1.py"/>
            </auctionneer>
     </economicAlgorithm>
     <rulesByPhaseList>
            <rulesByPhase final="1" name="continuous" number="1">
                    <allocationParameters>
                <increment><price>1</price></increment>
                <!-- Le cote reference du marche est celui de
1 ordre initial, il est important de verifier qu ils sont
coherents -->
                     <referenceSide>sell</referenceSide>
                    </allocationParameters>
               <phaseStartCondition>
                        <instantTimeout>inactif</instantTimeout>
               </phaseStartCondition>
                    <phaseEndCondition>
               <!-- fin de la phase par delai ou* fin d'une
certaine ronde -->
                <!-- delai en ms ou s , "-1" pour aucun delai-->
                        <durationTimeout
unit="ms">180000</durationTimeout>
                <!-- numero de la ronde au bout de laquelle finit
la phase -->
                <endOfRound>1</endOfRound>
                    </phaseEndCondition>
```

```
                <roundEndCondition>
                <!-- fin des rondes par delai ou* reception d'un
certain nombre d'ordre -->
                 <!-- delai en ms ou s , "-1" pour aucun delai-->
                 <durationTimeout
unit="ms">180000</durationTimeout>
                 <!-- nombre d'ordres a attendre -->
                 <waitforOrder>1</waitforOrder>
                     </roundEndCondition>
                     <clearingCondition>
                    <endOfPhase>inactif</endOfPhase>
                </clearingCondition>
              </rulesByPhase>
      </rulesByPhaseList>
    </rules>
</gnpDocument>

* : la premiere condition atteinte sera executee
```

***Exemple de document "ProductReference"***

```
<!--   PRODUCT REFERENCE    -->
<!-- a afficher a tous les joueurs  -->

 <?xml version="1.0" encoding="UTF-8"?>
<gnpDocument GPK="2490399" creationTime="960218160000"
    negotiationGPK="2490397">     <productReference>
        <productDescription>testproduct</productDescription>
        <productURL>http://www.testproduct.com</productURL>

  </productReference>

 </gnpDocument>
```

***Exemple de document "Order"***

```
<! --      ORDER   -- >
<!-- ne pas afficher
 La liste des ordres de meme partyKey peut etre disponible a ce
joueur -->

 <?xml version="1.0" encoding="UTF-8"?>
<gnpDocument GPK="131146" creationTime="959635079000"
overridden="false"
    overrideOrderGPK="" partyKey="1" productReferenceGPK="131135"
    quoteGPK="131145">
     <order>
          <sentToRound>2</sentToRound>
          <side>buy</side>
          <sentPQList>
```

```
                    <priceQuantity>
                        <price>24</price>
                        <quantity>200</quantity>
                        </priceQuantity>
                </sentPQList>
        </order>
</gnpDocument>
```

***Exemple de document " Annonce":***

```
<gnpDocument>
<rules>
<economicAlgorithm>
        <orderValidator>cirano.gnp.NOOPOrderValidator</orderValidat
or>
        <auctionneer class="cirano.gnp.services.PythonAuctionneer">
                <param                                  name="scriptFile"
                value="/usagers/bennajim/Projets/gnp/tests/auctionneer
                .py" />
        </auctionneer>
</economicAlgorithm>
<rulesByPhaseList>
        <rulesByPhase name="continuous" number="1" final="true">
        <allocationParameters>
                <increment>
                        <price>1</price>
                </increment>
                <referenceSide>sell</referenceSide>
        </allocationParameters>
        <phaseStartCondition>
                <instantTimeout>2000-02-11T12:00:00-
        05:00</instantTimeout>
        </phaseStartCondition>
        <phaseEndCondition>
                <instantTimeout>
                        <iso>2000-02-11T16:30:00-05:00</iso>
                </instantTimeout>
        </phaseEndCondition>
        <roundEndCondition>
                <waitforOrder>1</waitforOrder>
        </roundEndCondition>
        <clearingCondition>endOfPhase</clearingCondition>
        </rulesByPhase>
</rulesByPhaseList>
</rules>
<productReference>
        <productDescription>testproduct</productDescription>
        <productURL>http://www.testproduct.com</productURL>
</productReference>
<order>
```

```
        <side>by</side>
        <sentToRound>0</sentToRound>
        <priceQuantityList>
              <priceQuantity>
                    <price>100</price>
                    <quantity>20</quantity>
              </priceQuantity>
        </priceQuantityList>
</order>
</ gnpDocument >
```

***Exemple de document "Negotiation"***

```
<!--   NEGOTIATION    -->
<!-- Ne pas afficher et non disponible (sauf au scripteur) -->

<?xml version="1.0" encoding="UTF-8"?>
<gnpDocument GPK="131133" announceGPK="131132"
    creationTime="959634878000" state="OPENED">
    <negotiation>
        <questions>
            <canStart>1</canStart>
            <isRoundEnd>1</isRoundEnd>
            <canClose>0</canClose>
            <clearMarket/>
        </questions>
        <information>
            <roundNumber>3</roundNumber>
            <phaseNumber>1</phaseNumber>
            <phaseEnd>
                <instantTimeout>
                    <iso>2000-02-11T16:30:00-05:00</iso>
</instantTimeout>
                <durationTimeout
unit="ms">10000</durationTimeout>
            </phaseEnd>
            <roundEnd>
                <waitforOrderCounter>0</waitforOrderCounter>
            </roundEnd>
        </information>
        <infoByProductList>
            <infoByProduct productReferenceGPK="131135">
                <parameters>

<refSideTotalQuantity>200</refSideTotalQuantity>
                    <lastAllocatedIndex>3</lastAllocatedIndex>
                </parameters>
                <allocTable>
                    <extendedOrder orderGPK="131146">
```

```
            <sentRound>3</sentRound>
            <receivedAtRound>3</receivedAtRound>
            <orderSide>buy</orderSide>
            <sentPQList>
                <priceQuantity>
                    <price>24</price>
                    <quantity>200</quantity>
                </priceQuantity>
            </sentPQList>
            <allocatedPQList>
                <priceQuantity>
                    <price>21.0</price>
                    <quantity>0</quantity>
                </priceQuantity>
            </allocatedPQList>
        </extendedOrder>
        <extendedOrder orderGPK="131140">
            <sentRound>1</sentRound>
            <receivedAtRound>1</receivedAtRound>
            <orderSide>buy</orderSide>
            <sentPQList>
                <priceQuantity>
                    <price>20</price>
                    <quantity>110</quantity>
                </priceQuantity>
            </sentPQList>
            <allocatedPQList>
                <priceQuantity>
                    <price>20.0</price>
                    <quantity>0</quantity>
                </priceQuantity>
            </allocatedPQList>
        </extendedOrder>
    </allocTable>
    <adjudicatedTable>
        <extendedOrder orderGPK="131137">
            <sentRound>0</sentRound>
            <receivedAtRound>0</receivedAtRound>
            <orderSide>sell</orderSide>
            <sentPQList>
                <priceQuantity>
                    <price>18</price>
                    <quantity>200</quantity>
                </priceQuantity>
            </sentPQList>
            <allocatedPQList>
                <priceQuantity>
                    <price>18.0</price>
                    <quantity>0.0</quantity>
```

```
                        </priceQuantity>
                    </allocatedPQList>
                </extendedOrder>
            </adjudicatedTable>
        </infoByProduct>
    </infoByProductList>
</negotiation>
</gnpDocument>
```

***Exemple de document "quote"***

```
<!--    QUOTE  -->
<!-- à afficher:
- à tous les joueurs
- le dernier objet quote concernant cette negociation -->

<?xml version="1.0" encoding="UTF-8"?>
<gnpDocument>
    <quote>
        <information>
            <roundNumber>0</roundNumber>
            <phaseNumber>1</phaseNumber>
            <phaseEnd>
                <instantTimeout>
                    <iso>2000-02-11T16:30:00-05:00</iso>
</instantTimeout>
                <durationTimeout
unit="ms">180000</durationTimeout>
            </phaseEnd>
            <roundEnd>
                <waitforOrder>1</waitforOrder>
            </roundEnd>
        </information>
        <infoByProductList>
            <infoByProduct productReferenceGPK="2490399">
                <publicAllocTable>
                    <publicOrder orderGPK="2490401">
                        <orderSide>sell</orderSide>
                        <allocatedPQList>
                            <priceQuantity>
                                <price>18.0</price>
                                <quantity>200.0</quantity>
                            </priceQuantity>
                        </allocatedPQList>
                    </publicOrder>
                </publicAllocTable>
                <publicOrderDomain>
                    <gnpForm>
```

```
                        <hiddenentry
                            name="/gnpDocument/order/sentToRound"
value="0"/>
                        <hiddenentry
name="/gnpDocument/order/side" value="buy"/>
                        <fieldentry description="Prix"

name="/gnpDocument/order/sentPQList/priceQuantity/price"
value=""/>
                        <fieldentry description="Quantité"

name="/gnpDocument/order/sentPQList/priceQuantity/quantity"
value=""/>
                    </gnpForm>
                </publicOrderDomain>
            </infoByProduct>
        </infoByProductList>
    </quote>
</gnpDocument>
```

***Exemple de document "response"***

```
< --    RESPONSE   -->
<!-- a afficher:
- au joueur : PartyKey de l attribut orderGPK de response
- le dernier objet response concernant cette negociation et ce
joueur -->

<?xml version="1.0" encoding="UTF-8"?>
<gnpDocument orderGPK="131146" quoteGPK="131145">
    <response>
        <validatorValidation key="353" valid="1">l ordre 131146 a
ete
            accepte par le validator</validatorValidation>
        <onOrderValidation key="999" valid="1">l ordre 131146 a
ete
            ajoute au doc de negociation</onOrderValidation>
        <displayNote></displayNote>
    </response>
</gnpDocument>
```

***Exemple de document "adjudication"***

```
< --    ADJUDICATION   -->
<!-- a afficher:
-aux seuls deux joueurs concernes : buyerPartyKey et
sellerPartyKey
```

```
-les adjudications existantes pour cette negociation et ces
joueurs -->

<?xml version="1.0" encoding="UTF-8"?>
<gnpDocument GPK="1966420" buyOrderGPK="1966411"
    creationTime="959974763000" productReferenceGPK="1966406"
sellOrderGPK="1966408">
    <adjudication>
        <buyerPartyKey>ach01</buyerPartyKey>
        <sellerPartyKey>vendeur</sellerPartyKey>
        <adjudicatedPQList>
            <priceQuantity>
                <price>100.0</price>
                <quantity>15.0</quantity>
            </priceQuantity>
        </adjudicatedPQList>
    </adjudication>
</gnpDocument>
```

# ANNEXE B: TYPES SUPPORTES PAR *JPYTHON* ET OPERATIONS ASSOCIEES

Les types numériques

| Constantes | Interprétation |
|---|---|
| | |
| 314 / -2 / 0 | Entiers normaux |
| 314314314L | Entiers longs (taille illimitée) |
| 1.23 /3.14e-10 / 4E210 | Virgules flottantes |
| 0177 / 0x9ff | Constantes Octales et hexadécimales |
| 3+4j / 3.0-4.0J | Constantes complexes |

Les opérateurs principaux pour les types numériques:

| Opérateur | Déscription |
|---|---|
| | |
| x or y | ou logique |
| x and y | et logique |
| | |
| not x | négation logique |
| <, <=, >, >=, ==, <>, != | opérateurs de comparaison |
| is, is not | Test d'identité |
| in, not in | Appartenance à une séquence |
| x \| y | ou bits-à-bits |
| x ^ y | ou exclusif bits-à-bits |
| x & y | et bits-à-bits |
| x << y, x >> y | Décalage de x par y bits |
| | |
| x + y, x -y | addition ou concaténation / soustarction |
| x * y | multiplication ou répetition |
| x / y, x % y | division / reste de la div. (modulo) |
| | |
| -x | négation unaire |

Les opérateurs principaux pour les chaines

| Opération | Intérprétation |
|---|---|
| | |
| s1="" | chaine vide |
| s2="l' uf" | double guillemets |
| bloc="""  ."""" | bloc à triple guillemet |
| s1+s2, s2*3 | concaténation, répétition |
| s2[i], s2[i:j], len(s2) | indice, extraction, longueur |
| "Hello %s" % 'World' | formatage de chaine |
| for x in s2, 'o' in s2 | itération, appartenance |

L'opérateur surchargé %

| | | | |
|---|---|---|---|
| %s | chaîne ( ou autre objet) | %X | entier héxa. Majuscules |
| %c | caractère | %e | format de réels no 1 |
| %d | entier décimal | %E | format de réels no 2 |
| %u | entier non-signé | %f | format de réels no 3 |
| %o | entier octal | %g | format de réels no 4 |
| %x | entier hexadécimal | %% | % littéral no 5 |

Variation sur les chaînes

| | | | |
|---|---|---|---|
| \[entrée] | ingnoré | \n | fin de ligne |
| \\ | antislash (garde le 2eme) | \v | tabulation verticale |
| \' | guillemet simple | \t | tabulation horizontale |
| \" | guillemet double | \r | retour chariot |
| \a | sonnerie | \f | formfeed (saut de page) |
| \b | espace arrière | \0XX | valeur octale XX |
| \e | échappement | \xXX | valeur Hexa XX |
| \000 | nulle (termine pas la chaîne) | \autre | autre caractère (retenu) |

Opérations sur les listes

| **Opération** | **Interprétation** |
|---|---|
| | |
| L1=[] | liste vide |
| L2=[0, 1, 2, 3] | 4 élément indicé de 0 à 3 |
| L3=['abc', ['def', 'ghi']] | liste incluses |
| L2[i], L3[i][j] | indice |
| L2[i:j] | tranche |
| len(L2) | longueur |
| L1+L2 | concaténation |
| L1*3 | répétition |
| for x in L2 | parcours |
| 3 in L2 | appartenance |
| L2.append(4) | méthodes : agrandissement |
| L2.sort() | tri |
| L2.index() | recherche |
| L2.reverse() | inversion |
| del L2[k], L2[i:j]=[] | effacement |
| L2[i]=1 | affectation par indice |
| L2[i:j]=[4, 5, 6] | affectation par tranche |
| range(4), xrange(0,4) | céation de listes / tuples d'entiers |

Opérations sur les dictionnaires

| Opération | Interprétation |
|---|---|
| | |
| d1 = {} | dictionnaire vide |
| d2={'one' : 1, 'two' : 2} | dictionnaire à duex éléments |
| d3={'count': {'one': 1, 'two': 2}} | inclusion |
| d2['one'], d3['count']['one'] | indiçage par clé |
| d2.has_keys('one') | methodes : test d'appartenance |
| d2.keys() | liste des clés |
| d2.values() | liste des valeurs |
| len(d1) | longueur (nombre d'entrée) |
| d2[cle] = [nouveau] | ajout / modification |
| del d2[cle] | destruction |

Opérations sur les typles

| Opération | Interprétation |
|---|---|
| | |
| () | un tuple vide |
| n1 = (0,) | un tuple à un élément (et non une expression) |
| n2 = (0,1,2,3) | un tuple à quatre éléments |
| n2 = 0,1,2,3 | un autre tuple à quatre éléments |
| n3 = ('abc', ('def', 'ghi')) | tuple avec inclusion |
| t[i], n3[i][j] | indiçage |
| n1[i:j] | tranche |
| len(n1) | longueur |
| n1+n2 | concaténation |
| n2 * 3 | répétition |
| for x in n2 | itération |
| 3 in s2 | test d'appartenance |

Opérations sur les fichiers

| Opération | Interprétation |
|---|---|
| | |
| sortie = open('/tmp/spam', 'w') | crée un fichier de sortie ('w' => écriture) |
| entre = open('donnee', 'r') | ouvre un fichier en entrée ('r' => lecture) |
| s = entre.read() | lit le fichier entier dans une chaine |
| s = entre.read(N) | lit N octets (1 ou plus) |
| s = entre.readline() | lit la ligne suivante |
| L = entre.readlines() | lit la fichier dans une liste de lignes |
| sortie.write(s) | écrit s dans le fichier |
| sortie.writelines(L) | écrit toutes les lignes contenues pas L |
| sortie.close() | fermeture manuelle |

Les mots réservés.

| | |
|---|---|
| and | "et" logique |
| assert | insértion d'un test d'exception pour le débogage |
| break | instruction pour quitter une boucle |
| class | déclaration d'une classe |
| continue | instruction pour continuer une boucle |
| def | définition d'une fonction |
| del | supression d'un objet |
| elif | condition supplémentaire pour une sélection avec "if" |
| else | bloc optionnel d'exécution de code |
| except | définition d'une gestion d'exception |
| exec | execution d'une chaine de caractères |
| finally | bloc final pour une gestion d'exceptions |
| for | commande d'itération |
| from | appel d'objet pour une importation |
| global | déclaration pour une variable visible de partout |
| if | commande de séléction |
| import | importation de module |
| in | recherche d'un objet cible dans un objet séquence |
| is | comparaison |
| lambda | expression de déclaration d'une fonction simple |
| not | "non" logique |
| or | "ou" logique |
| pass | expression équivalent à ne rien faire |
| print | commande d'affichage |
| raise | levée d'exception |
| return | expression pour le retour depuis une fonction |
| try | début de la gestion d'exception |
| while | commande d'itération |

Quelques librairies:

| | |
|---|---|
| cgi | librairie pour le traitement de cgi |
| ftplib | librairie en rapport avec le service ftp |
| gdbm | création de base de données par dictionnaire |
| math | librairie mathématique |
| os | fonction et commande par rapport à l'OS |
| poplib | gestion de courriers éléctroniques |
| random | gestion de choix aléatoires |
| re / regex | librairie d'expressions régulières |
| shelve | gestion de base de données |
| smtplib | gestion de courrier éléctronique |
| socket | librairie concernant les sockets |
| string | gestion et transformation par rapport aux chaines de caractères |
| sys | librairie contenant des fonctions Python en rapport avec le système |
| telnet | fonctions telnet pour Python |
| time | librairie concernant le temps, et la date |
| Tkinter | création d'un environnemet graphique |

# ANNEXE C: SCENARIOS DE TEST

**Scénarios relatifs a l'objectif (i)**

```
mscdocument
msc    Scenario_01

msg    submitAnnounce (charstring)
msg    submitOrder (long, long, long, charstring)
msg    retour

ins   vendeur:        process      AnnouncerSession
ins   ach01:          process      NegotiatorSession
ins   ach02:          process      NegotiatorSession

vendeur:
        in    submitAnnounce("announce.xml")          from env
        out   retour                                  to env

ach01 :
        in    submitOrder(productReferenceGPK(retour), negoGPK(retour),
        previousQuoteGPK(retour), Order("order.xml", 15, 110))  from   env

ach02 :
        in    submitOrder(productReferenceGPK(retour), negoGPK(retour),
        previousQuoteGPK(retour), Order("order.xml", 15, 90))   from   env

condition leader(retour, ach01)
condition leader(retour, ach02)

vendeur:      endinstance
ach01:        endinstance
ach02:        endinstance
endmsc Scenario_01
```

**Les résultats du test confirment les attentes:**

leader(retour, ach01) [ passe ]
leader(retour, ach02) [ échec ]

```
msc    Scénario_02

msg    submitAnnounce (charstring)
msg    submitOrder (long, long, long, charstring)
msg    retour

ins    vendeur:        process        AnnouncerSession
ins    ach01:          process        NegotiatorSession
ins    ach02:          process        NegotiatorSession

vendeur:
        in     submitAnnounce("announce.xml")              from env
        out    retour                                      to env

ach02 :
        in     submitOrder(productReferenceGPK(retour), negoGPK(retour),
        previousQuoteGPK(retour), Order("order.xml", 15, 90))    from    env

ach01 :
        in     submitOrder(productReferenceGPK(retour), negoGPK(retour),
        previousQuoteGPK(retour), Order("order.xml", 15, 110))   from    env

condition leader(retour, ach01)
condition leader(retour, ach02)

vendeur:        endinstance
ach01:          endinstance
ach02:          endinstance
endmsc Scenario_02
```

**Les résultats du test confirment les attentes:**

leader(retour, ach01) [ passe ]
leader(retour, ach02) [ échec ]

---

```
msc    Scenario_03

msg    submitAnnounce (charstring)
msg    submitOrder (long, long, long, charstring)
msg    retour

ins    vendeur:        process        AnnouncerSession
ins    ach01:          process        NegotiatorSession
```

```
ins   ach02:          process        NegotiatorSession

vendeur:
        in    submitAnnounce("announce.xml")              from env
        out   retour                                      to env

ach01 :
        in    submitOrder(productReferenceGPK(retour), negoGPK(retour),
        previousQuoteGPK(retour), Order("order.xml", 15, 0))     from   env

ach02 :
        in    submitOrder(productReferenceGPK(retour), negoGPK(retour),
        previousQuoteGPK(retour), Order("order.xml", 15, 100))  from   env

condition leader(retour, ach01)
condition leader(retour, ach02)

vendeur:        endinstance
ach01:          endinstance
ach02:          endinstance
endmsc Scenario_03
```

**Les résultats du test confirment les attentes:**
leader(retour, ach01) [ échec]
leader(retour, ach02) [ passe ]

---

```
msc   Scenario_04

msg   submitAnnounce (charstring)
msg   submitOrder (long, long, long, charstring)
msg   retour

ins   vendeur:        process        AnnouncerSession
ins   ach01:          process        NegotiatorSession
ins   ach02:          process        NegotiatorSession

vendeur:
        in    submitAnnounce("announce.xml")              from env
        out   retour                                      to env

ach02 :
        in    submitOrder(productReferenceGPK(retour), negoGPK(retour),
        previousQuoteGPK(retour), Order("order.xml", 15, 100))  from   env
```

ach01 :

in submitOrder(productReferenceGPK(retour), negoGPK(retour), previousQuoteGPK(retour), Order("order.xml", 15, 0)) from env

condition leader(retour, ach01)
condition leader(retour, ach02)

vendeur: endinstance
ach01: endinstance
ach02: endinstance
endmsc Scenario_04

**Les résultats du test confirment les attentes:**
leader(retour, ach01) [ échec]
leader(retour, ach02) [ passe ]

---

**Scénarios relatifs a l'objectif (ii)**

```
msc   Scenario_01

msg   submitAnnounce (charstring)
msg   submitOrder (long, long, long, charstring)
msg   retour

ins   vendeur:      process     AnnouncerSession
ins   ach01:        process     NegotiatorSession
ins   ach02:        process     NegotiatorSession

vendeur:
      in    submitAnnounce("announce.xml")          from env
      out   retour                                  to env

ach01 :
      in    submitOrder(productReferenceGPK(retour), negoGPK(retour),
      previousQuoteGPK(retour), Order("order.xml", 20, 110))  from   env

ach02 :
      in    submitOrder(productReferenceGPK(retour), negoGPK(retour),
      previousQuoteGPK(retour), Order("order.xml", 20, 120))  from   env

condition leader(retour, ach01)
condition leader(retour, ach02)

vendeur:      endinstance
ach01:        endinstance
ach02:        endinstance
endmsc Scenario_01
```

**Les résultats du test confirment les attentes:**

leader(retour, ach01) [ échec]
leader(retour, ach02) [ passe ]

---

```
msc   Scenario_02

msg   submitAnnounce (charstring)
msg   submitOrder (long, long, long, charstring)
msg   retour
```

```
ins   vendeur:        process        AnnouncerSession
ins   ach01:          process        NegotiatorSession
ins   ach02:          process        NegotiatorSession

vendeur:
        in    submitAnnounce("announce.xml")              from env
        out   retour                                      to env

ach02 :
        in    submitOrder(productReferenceGPK(retour), negoGPK(retour),
        previousQuoteGPK(retour), Order("order.xml", 20, 120))  from   env

ach01 :
        in    submitOrder(productReferenceGPK(retour), negoGPK(retour),
        previousQuoteGPK(retour), Order("order.xml", 20, 110))  from   env

condition leader(retour, ach01)
condition leader(retour, ach02)

vendeur:        endinstance
ach01:          endinstance
ach02:          endinstance
endmsc Scenario_01
```

**Les résultats du test confirment les attentes:**

leader(retour, ach01) [ échec]
leader(retour, ach02) [ passe ]

# ANNEXE D: CODE SOURCE JPYTHON DES MODULES

**Module de traduction:**

```
# Author: El Mostafa Ben Najim
##
### imports
##
#
from java.lang import System
from string import *
import string
from sys import argv, exit
import sys
sys.path.insert(0, '/usagers/'+System.getProperty("user.name")+'/Projets/gnp/msc/scenarios')
outputPath = "/usagers/"+System.getProperty('user.name')+"/Projets/gnp/msc/robots/"
bFile     = outputPath+"robotsExec"+".sh"
bachFile   = open(bFile, 'w')
bachFile.write("#!/bin/sh\n\n")
#
## Declaration des variables
#
localServer   = "dev05"
localPort     = "7001"
ds_mscdocument= 0
ds_msc        = 0
ds_inst       = 0
start_msg     = 0
inst_defini   = 0
nbr_inst      = 0
nbr_msg       = 0
actual_player = 0
actual_inst   = ''
inst          = []
messages      = []
no_ligne      = 0
erreur        = 0
listScript    = []
announcers    = []

if len(argv) != 2:
        print "\nUsage: ./run generate nomFichierMSC\n"
        exit(1)
else:
        try:
                f = open(argv[1], 'r')
        except:
                print "\n\tLe fichier ", argv[1]," n'existe pas \n"
```

```
                    exit(1)

outPutFile = ""

ligne = f.readline()
while (ligne != ''):
        no_ligne=no_ligne+1

        # remove empty lines
        while split(ligne)==[]:
                ligne = f.readline()
                no_ligne=no_ligne+1

        # delete  ';' at the end of each line
        if (find(upper(ligne),"CONDITION")!=-1) and (find(ligne, ';')== -1):
                ligne = ligne[:find(ligne,'\n')]+";"

        if find(ligne, ';')!= -1:
                ligne=ligne[0:find(ligne, ';')]

        elem=split(ligne)
    args = []
    params = ""
        #print "no ligne: ", no_ligne  # enlever ce commentaire pour debogguer
        if (upper(elem[0]) == "MSCDOCUMENT"):
                if (ds_mscdocument==1):
                        erreur=1
                        print "  Ligne: ",no_ligne,"## double declaration de
MSCDOCUMENT dans le meme scenario.."

                else:
                        ds_mscdocument=1
                        print "\nMSCDocument..... [ ok ]"

        elif (upper(elem[0]) == "MSC"):
                if ((ds_mscdocument == 0) or (ds_msc == 1)):
                        erreur=1
                        print "  Ligne: ",no_ligne,"## Erreur de déclaration.."
                else:
                        ds_msc=1
                        print "MSC..... [ ok ]"
                        print "Reconnaissance des messages....       "
                        #Création et initialisation du fichier script .py
                        outPutFile=outputPath+elem[1]+".py"
                        scriptFile=open(outPutFile, 'w')
                        scriptFile.write("#\n#  Import\n#\n\n")
                        scriptFile.write("import org.w3c.dom\n")
                        scriptFile.write("import cirano.gnp.beans\n")
                        scriptFile.write("import javax.jms\n")
                        scriptFile.write("import java.util\n")
```

```
                                scriptFile.write("from javax.naming import *\n")
                                scriptFile.write("from cirano.gnp import ContextUtil, Log\n")
                                scriptFile.write("from cirano.gnp import DOMUtil\n")
                                scriptFile.write("from javax.ejb import *\n")
                                scriptFile.write("from java.util import *\n")
                                scriptFile.write("from java.lang import System\n")
                                scriptFile.write("import os\n")
                                scriptFile.write("import time \n")
                                scriptFile.write("from time import sleep, strftime, localtime, clock
\n")
                                scriptFile.write("import sys\n")
                                scriptFile.write("sys.path.insert(0, '/usagers/"+
                                                        System.getProperty
                                                        ("user.name")+"/Projets/gnp/msc/lib')\n")
                                scriptFile.write("import testUtil, params, predicats \n")
                                scriptFile.write("from testUtil import *\n")
                                scriptFile.write("from predicats import *\n")
                                scriptFile.write("from params import *\n")
                                scriptFile.write( "#Initialisation du serveur: doit être modifier si
                                                                                nécessaire\n")
                                scriptFile.write("server =
\"t3://"+localServer+":"+localPort+"\"\n\n")
                        scriptFile.write('if not
os.path.isdir("/usagers/"+System.getProperty("user.name")+"/TestsVerdicts/"+strftime
("%a%d%b%Y", localtime(clock()))+"/"):\n')

scriptFile.write('\tos.makedirs("/usagers/"+System.getProperty("user.name")+"/TestsVerdicts/"+
strftime ("%a%d%b%Y", localtime(clock()))+"/")\n\n')
                        scriptFile.write('pathSom =
"/usagers/"+System.getProperty("user.name")+"/TestsVerdicts/"\n')
                        scriptFile.write('pathDet =
"/usagers/"+System.getProperty("user.name")+"/TestsVerdicts/"+strftime ("%a%d%b%Y",
localtime(clock()))+"/"\n\n')
                        scriptFile.write('logSom = open(pathSom + strftime ("%a%d%b%Y",
localtime(clock()))+".log", "a")\n')
                        scriptFile.write('logDet =  Log.getNamedInstance(pathDet + ' + "'" + elem[1] +
'.log")\n\n')
                        scriptFile.write('logSom.write(">>>>> ----------------------------------------
DEBUT d\'execution du script:'+elem[1]+'.py\\n")\n')
                        scriptFile.write('logDet.info("|----------------  deltail des verdicts de tests de
'+elem[1]+'.py  ----------------|\\n\\n\\n")\n\n\n')
                        scriptFile.write('ok = 1\n')

                elif (upper(elem[0]) == "MSG"):
                        if (ds_inst == 1) or (ds_msc == 0) or (ds_mscdocument == 0):
                                erreur=1
                                print "  Ligne: ",no_ligne,"## MSG ne peut etre declare a cette
endroit.."
                        else:
                                if (find(elem[1], '(') == -1):
```

```
                                        if elem[1][-1]== ";":

                                                messages.append(elem[1][:-1])
                                        else:
                                                messages.append(elem[1])

                                else:

                                        messages.append(elem[1][0:find(elem[1], '(')])
                                nbr_msg = nbr_msg +1


        elif (upper(elem[0]) == "INS"):
                if (elem[1] in inst):
                                erreur = 1
                                print "  Ligne: ",no_ligne,"## L'instance, ",elem[1]," est deja
declaree.."
                else:
                                if (start_msg!=0):
                                        erreur=1
                                        print "  Ligne: ",no_ligne,"## L'instance, ",elem[1]," ne
peut pas etre declare a cet endroit.."
                                if (ds_inst!=1):
                                        print "Reconnaissance des messages.... [ ok ]"
                                        print "Reconnaissance des instances..."
                                        scriptFile.write("ctx =
ContextUtil.getNamingContext()\n\n")
                                ds_inst=1


                                inst.append(elem[1])
                                nbr_inst      = nbr_inst+1

                                if elem[1][-1]==":":
                                        elem[1]=elem[1][:-1]

                                if (upper(elem[3]) == "ANNOUNCERSESSION") and (elem[1] not
in announcers):
                                        announcers.append(elem[1])
                                        #print "les annonceurs actuels ", announcers

                                scriptFile.write("createParty(\""+elem[1]+"\")\n")
                                scriptFile.write("try:\n")
                                scriptFile.write("\t"+elem[1]+" = ctx.lookup(\"beans." + elem[3]+
"Home\").create()\n")
                                scriptFile.write("except:\n")
                                scriptFile.write("\tprint \"Erreur!!: ne peut pas obtenir une home
session reference\"\n")
                scriptFile.write("\tlogSom.write('une Erreur est survenue lors de la creation d\\'
une session reference: " + elem[3] + "Home!! arret immediat du script\\n')\n")
```

```
                scriptFile.write('\tsys.exit(1)\n\n')
                                #scriptFile.write("try:\n")
                                #scriptFile.write("\t"+elem[1]+" = "+elem[1]+"home.create()\n")
                                #scriptFile.write("except:\n")
                                #scriptFile.write("\tprint \"Erreur: ne peut pas obtenir une session
reference\"\n\n")
                #scriptFile.write("\tlogSom.write('une Erreur est survenue lors de la creation d\\'
une Session Reference: " + elem[3]+ "!! arret immediat du script\\n')\n")
                                #scriptFile.write('\tsys.exit(1)\n\n')


        elif (upper(elem[0]) == "ENDMSC"):

                if (ds_msc == 0):
                                erreur=1
                                print "  Ligne: ",no_ligne,"## Aucune clause MSC ne correspond a
ENDMSC.."
                elif (nbr_inst!=0):
                                erreur=1
                                print "  Ligne: ",no_ligne,"## ENDMSC ne peut etre declare a cet
endroit.."


                else:
                                print "Fin du MSC en cours..... [ ok ]"
                                listScript.append(outPutFile)
                                # reinitialisation de l'environnement
                scriptFile.write("if ok:\n")
                scriptFile.write("\tlogSom.write('script: "+elem[1]+".py <---------> [ PASS
]\\n')\n")
                scriptFile.write("else:\n")
                scriptFile.write("\tlogSom.write('script: "+elem[1]+".py <---------> un ou
plusieurs cas de tests ont echoues. voir le Log detail pour plus d\\'info\\n')\n")
                scriptFile.write('logSom.write(">>>>> ---------------------------------------- FIN
d\'execution du script:'+elem[1]+'.py\\n\\n\\n")\n')
                scriptFile.write('logDet.info("FIN  d\'execution")\n')
                                scriptFile.write("#  ----  Fin de la traduction du scenario en
cours..~/~")
                                scriptFile.close()
                bachFile.write("./run "+elem[1] + ".py\n")
                                outPutFile=""
                                ds_msc=0
                                ds_inst=0
                                start_msg=0
                                inst_defini=0
                                nbr_inst=0
                                nbr_msg=0
                                actual_player=0
                                actual_inst="
                                inst = []
```

```
                        messages =  []
                        announcers=[]


        elif (upper(elem[0]) == "ENDMSCDOCUMENT"):
                if ((ds_mscdocument == 0) or (ds_msc == 1)):
                        erreur=1
                        print "  Ligne: ",no_ligne,"## ENDMSCDOCUMENT ne peut etre
declare a cet endroit.."
                else:
                        print "Fin du traitement..... [ ok ]\n"
                        ds_mscdocument =0
                        ds_msc=0
                        ds_inst=0
                        start_msg=0


        elif (elem[0] in inst ):

                if (start_msg==0):
                        print "Reconnaissance des instances... [ ok ]"
              scriptFile.write("\n#\n###\n##### Executing ...\n###\n#\n")
                start_msg=1
                if (elem[1]!="ENDINSTANCE"):
                        if elem[0][-1]==':':
                                actual_inst = elem[0][:-1]
                        else:
                                actual_inst = elem[0]

                else:
                        nbr_inst= nbr_inst -1


                if (elem[1]== "IN"):
                        scriptFile.write("\n#\n### new test case ..\n#\n")
                        scriptFile.write("\nctx = ContextUtil.getNamingContext(server,
\""+elem[0][:-1]+"\", \""+elem[0][:-1]+"\")\n")

                        # make sur the message is had been declared..
                        if ((find(elem[2], '(')!= -1) and (elem[2][0:find(elem[2], '(')] not in
messages)
                                or (find(elem[2], '(') == -1) and (elem[2] not in
messages )):
                                print "  Ligne: ",no_ligne,"## Aucun message de type
"+ elem[2]+".. n'a ete declare"
                                print messages
                                erreur = 1

                        # fetches params..
                        if (find(ligne, '(') != -1) and (find(ligne, ')') != -1):
```

```
                                        params = ligne[find(ligne, '(') +1 : rfind(ligne, ')')]
                else:       params = ""



                # Add quotes to args..
                if strip(params) != "":
                        args = split(params, ",")

                params = ""
                for i in range (0, len(args)):

                        if strip(args[i][:find(args[i],"(")]) in ["negoGPK",
"productReferenceGPK","quoteGPK", "previousQuoteGPK"]:
                                param = strip(args[i][find(args[i], '(') :find(args[i], ')') ])
                                args[i] = args[i][ : find(args[i], param)+1] +""+  args[i][find(args[i],
param)+1:find(args[i], param)+len(param)]+""+   args[i][find(args[i], param)+len(param):]
                        elif  strip(args[i][:find(args[i],"(")]) in ["orderGPK", "oldOrderGPK"]:
                                param = strip(args[i][find(args[i], '(') :find(args[i], ')') ])
                                args[i] = args[i][ : find(args[i], param)+1] + actual_inst +', "'+
args[i][find(args[i], param)+1:find(args[i], param)+len(param)]+""+   args[i][find(args[i],
param)+len(param):]

                        params = params +args[i] + ","
                params=params[:-1]



                                #preparer le terrain pour executer la methode..
                                if (find(elem[2],'(')!= -1):
                                        scriptFile.write("testUtil.last_method = \"" +
strip(elem[2][:find(elem[2],'(')])+"\"\n")
                                else:
                                        scriptFile.write("testUtil.last_method = \"" +
strip(elem[2])+"\"\n")

                                scriptFile.write("testUtil.exception = 0\n")
                                scriptFile.write("try:\n")



                                if ((find(elem[2], 'submitAnnounce'))!= -1):
                        scriptFile.write("\tfann = open("+args[0]+")\n")

                        scriptFile.write("\ttestUtil.doc = ")
                        #if (len(args) == 1):
                        #       scriptFile.write(actual_inst+".submitAnnounce(fann.read())\n")
                        #else:
                        scriptFile.write(actual_inst+".submitAnnounce(fann.read() " +
params[find(params,args[0]) +len(args[0]):]+")\n")
```

```
                        scriptFile.write("\tannGPK   = DOMUtil.get(testUtil.doc,
'/gnpDocument/announceResult@announceGPK')\n")
                        scriptFile.write("\tnegGPK  = DOMUtil.get(testUtil.doc,
'/gnpDocument/announceResult@negotiationGPK')\n")
                        scriptFile.write("\tquotGPK = DOMUtil.get(testUtil.doc,
'/gnpDocument/announceResult@quoteGPK')\n")
                        scriptFile.write("\tprefGPK  = DOMUtil.get(testUtil.doc,
'/gnpDocument/announceResult@productReferenceGPK')\n")
                        scriptFile.write("\ttestUtil.annList.append([\""+actual_inst+"\", annGPK,
negGPK, quotGPK, prefGPK])\n")
                        scriptFile.write("\tprint \"---------- negoGPK\", negGPK\n")
                elif  ((find(elem[2], 'submitOrder'))!= -1) or ((find(elem[2], 'updateOrder'))!= -
1):


                        if (find(params,"Order") != -1):
                                scriptFile.write("\tford =
open("+params[find(params,"Order"):]+")\n")
                                params=params[:find(params,"Order" )-1]
                        else:
                                scriptFile.write("\tford = open("+params[rfind(params, ',')
+1:]+")\n")
                                params=params[:rfind(params,"," ) +1]

                        scriptFile.write("\ttestUtil.doc = ")
                        scriptFile.write(actual_inst+"."+elem[2][0:find(elem[2], '(')+1]+params+"
ford.read())\n")

                                else:
                                        scriptFile.write("\ttestUtil.doc = ")
                        if find(elem[2], '(') != -1:

scriptFile.write(actual_inst+"."+elem[2][0:find(elem[2], '(')]+"("+params+")\n")
                        else:

scriptFile.write(actual_inst+"."+elem[2][0:len(elem[2])]+"("+params+")\n")


                                scriptFile.write("except:\n")
                                scriptFile.write("\ttestUtil.exception_name = sys.exc_type;\n")
                                scriptFile.write("\ttestUtil.exception = 1;\n\n")

                elif (elem[1] == "OUT"):
                                scriptFile.write("return_result('"+elem[2]+"')\n")

                elif (elem[1] == "CONDITION"):

                                scriptFile.write("\nprint '\\nexecution de: " + ligne+ "\\n\\n'\n")
                                ligne=lstrip      (ligne[find(upper(ligne), 'CONDITION')+9   :
max(len(ligne),find(ligne,'\n'))])
```

```
                                arg1 = strip(ligne[find(ligne,'(')+1:find(ligne,',')])

                        if len(split(ligne[find(ligne, '(') : find(ligne, ')')], ",")) == 2:
                                arg2 = strip(ligne[find(ligne, ',')+1:find(ligne, ')')])
                        else:
                                arg2 = strip(ligne[find(ligne, ',')+1:find(ligne, ',')+1 +find(ligne[find(ligne,
',')+1:], ',' )])

                        if find(upper(ligne), 'LEADER') == -1:
                                        scriptFile.write("if " +
ligne[:find(ligne,"(")+1]+"\""+arg1+"\""+ligne[find(ligne,arg1)+len(arg1):]+":\n")
                        else:
                                scriptFile.write("if " + ligne[:find(ligne,"(")+1]+"\""+arg1+"\""+",
\""+arg2+"\""+ligne[find(ligne,arg2)+len(arg2):]+":\n")

                                        scriptFile.write("\t"+ "print \"------------------------------ OK !!!\"\n")
                        scriptFile.write("\tlogDet.info ('Ligne: " + `no_ligne` + "  <---> " +  ligne + "
[ PASS ]')\n")
                                        scriptFile.write("else:\n")
                        scriptFile.write("\tlogDet.error('Ligne: " + `no_ligne` + "  <---> " +  ligne + "
[ERREUR]')\n")
                        scriptFile.write('\tok = 0\n')
                                        scriptFile.write("\t"+ "print \"------------------------------ KO
!!!\"\n\n\n")


                elif (upper(elem[0]) in ['IN', 'OUT', 'CONDITION']):

                        if (upper(elem[0])== "IN"):
                                        scriptFile.write("\n#\n### new test case ..\n#\n")
                                        # make sur the message  had been  declared..
                                        if ((find(elem[1], '(')!= -1) and (elem[1][0:find(elem[1], '(')] not in
messages)
                                                or (find(elem[1], '(') == -1) and (elem[1] not in
messages )):
                                                print "  Ligne: ",no_ligne,"## Aucun message de type
"+ elem[1]+".. n'a ete declare"
                                                print "          Les messages declares sont: ", messages
                                                erreur = 1

                                        # fetches params..
                                        if (find(ligne, '(') != -1) and (find(ligne, ')') != -1):
                                                params = ligne[find(ligne, '(') +1 : rfind(ligne, ')')]

                        # Add quotes to args..
                        if strip(params) != "":
                                args = split(params, ",")
```

```
                params=""
                for i in range (0, len(args)):
                        #print "argument actuel: ", args[i][:find(args[i],"(")]
                        if strip(args[i][:find(args[i],"(")]) in ["negoGPK",
"productReferenceGPK","quoteGPK", "previousQuoteGPK"]:
                                param = strip(args[i][find(args[i], '(') :find(args[i], ')') ])
                                args[i] = args[i][ : find(args[i], param)+1] +'"'+  args[i][find(args[i],
param)+1:find(args[i], param)+len(param)]+'"'+   args[i][find(args[i], param)+len(param):]
                        elif  strip(args[i][:find(args[i],"(")]) in ["orderGPK", "oldOrderGPK"]:
                                param = strip(args[i][find(args[i], '(') :find(args[i], ')') ])
                                args[i] = args[i][ : find(args[i], param)+1] + actual_inst +', "'+
args[i][find(args[i], param)+1:find(args[i], param)+len(param)]+'"'+   args[i][find(args[i],
param)+len(param):]

                        params = params +args[i] + ","
                params=params[:-1]

                                #preparer le terrain pour l'execution de la methode..
                                if (find(elem[1],'(')!= -1):
                                        scriptFile.write("testUtil.last_method = \"" +
strip(elem[1][:find(elem[1],'(')])+"\"\n")
                                else:
                                        scriptFile.write("testUtil.last_method = \"" +
strip(elem[1])+"\"\n")

                                scriptFile.write("testUtil.exception = 0\n")
                                scriptFile.write("try:\n")



                                if ((find(elem[1], 'submitAnnounce'))!= -1): # and ((find(elem[2],
'submitOrder'))== -1):
                        scriptFile.write("\tfann = open("+args[0]+")\n")
                        scriptFile.write("\ttestUtil.doc = ")
                        scriptFile.write(actual_inst+".submitAnnounce(fann.read() " +
params[find(params,args[0]) +len(args[0] ):]+")\n")
                        scriptFile.write("\tannGPK   = DOMUtil.get(testUtil.doc,
'/gnpDocument/announceResult@announceGPK')\n")
                        scriptFile.write("\tnegGPK  = DOMUtil.get(testUtil.doc,
'/gnpDocument/announceResult@negotiationGPK')\n")
                        scriptFile.write("\tquotGPK = DOMUtil.get(testUtil.doc,
'/gnpDocument/announceResult@quoteGPK')\n")
                        scriptFile.write("\tprefGPK  = DOMUtil.get(testUtil.doc,
'/gnpDocument/announceResult@productReferenceGPK')\n")
                        scriptFile.write("\ttestUtil.annList.append([\""+actual_inst+"\", annGPK,
negGPK, quotGPK, prefGPK])\n")
                        scriptFile.write("\tprint \"---------- negoGPK\", negGPK\n")
                elif  ((find(elem[1], 'submitOrder'))!= -1) or ((find(elem[1], 'updateOrder'))!= -
1):
```

```
                        if (find(params,"Order") != -1):
                            scriptFile.write("\tford =
open("+params[find(params,"Order"):]+")\n")
                            params=params[:find(params,"Order" )-1]
                        else:
                            scriptFile.write("\tford = open("+params[rfind(params, ','
+1:]+")\n")
                            params=params[:rfind(params,"," )+1]

                        scriptFile.write("\ttestUtil.doc = ")
                        scriptFile.write(actual_inst+"."+elem[1][0:find(elem[1], '(')+1]+params+"
ford.read())\n")

                            else:
                        scriptFile.write("\ttestUtil.doc = ")
                        if find(elem[1], '(') != -1:

scriptFile.write(actual_inst+"."+elem[1][0:find(elem[1], '(')]+"("+params+")\n")
                        else:

scriptFile.write(actual_inst+"."+elem[1][0:len(elem[1])]+"("+params+")\n")

                            scriptFile.write("except:\n")
                            scriptFile.write("\ttestUtil.exception_name = sys.exc_type;\n")
                            scriptFile.write("\ttestUtil.exception = 1;\n\n")

                    elif (upper(elem[0])== "OUT"):
                            scriptFile.write("return_result('"+elem[1]+"')\n")

                    elif (upper(elem[0])== "CONDITION"):
                            scriptFile.write("\nprint '\\nexecution de: " + ligne+ "\\n\\n'\n")
                            ligne=lstrip      (ligne[find(upper(ligne), 'CONDITION')+9   :
max(len(ligne),find(ligne,'\n'))])
                            arg1 = strip(ligne[find(ligne,'(')+1:find(ligne,',')])

                if len(split(ligne[find(ligne, '(') : find(ligne, ')')], ",")) == 2:
                    arg2 = strip(ligne[find(ligne, ',')+1:find(ligne, ')')])
                else:
                    arg2 = strip(ligne[find(ligne, ',')+1:find(ligne, ',')+1 +find(ligne[find(ligne,
',')+1:], ',' )])

                if find(upper(ligne), 'LEADER') == -1:
                                    scriptFile.write("if " +
ligne[:find(ligne,"(")+1]+"\""+arg1+"\""+ligne[find(ligne,arg1)+len(arg1):]+":\n")
                else:
                    scriptFile.write("if " + ligne[:find(ligne,"(")+1]+"\""+arg1+"\""+",
\""+arg2+"\""+ligne[find(ligne,arg2)+len(arg2):]+":\n")

                            scriptFile.write("\t"+ "print \"---------------------------- OK !!!\"\n")
```

```
                scriptFile.write("\tlogDet.info('Ligne: " + `no_ligne` + "  <---> " +  ligne + "  [
PASS ]')\n")
                                scriptFile.write("else:\n")
                scriptFile.write("\tlogDet.error('Ligne: " + `no_ligne` + "  <---> " +  ligne + "
[ERREUR]')\n")
                scriptFile.write('\tok = 0\n')
                                scriptFile.write("\t"+ "print \"----------------------------  KO
!!!\"\n\n\n")



        elif(upper(elem[0]) == "COMMENT"):
                scriptFile.write("###"+ligne[8:]+"\n")


        else:
                print "  Ligne: ",no_ligne,"##",elem[0],"     IDENTIFIANT INCONNU"
                erreur=1


        ligne = f.readline()


if ds_mscdocument==1:
        print "Fin de fichier non expectee.."
        erreur=1

f.close()
bachFile.close()
print "Le(s) script(s) suivant(s) a(ont) ete genere(s):\n"
for script in listScript:
                print "\t", script,"\n"

if (erreur == 1):
    print "\nAttention: une (ou plusieurs) erreur grave a ete detectee.."
    print "le script jpython genere peut ne pas s'executer correctement"
    print "verifiez votre fichier MSC et re-essayez de nouveau \n"
else:
    print "\nL'execution sera faite, par defaut sur le serveur: "+localServer+", port: "+localPort
    print "Si necessaire,changer ces deux params au niveau de chaque script"
    print "ou, mieux encore, au niveau du traducteur une fois pour toute...\n\n"


#------- Fin de l'excution du scenario ~~/:~
```

**Module des utilitaires:**

```
from javax.naming import *
from cirano.gnp.beans import AnnouncerSessionHome
from cirano.gnp.beans import AnnouncerSession
from cirano.gnp import ContextUtil
from cirano.gnp import DOMUtil
from javax.ejb import *
from java.util import *
from java.lang import System
from sys import argv, exit
import sys
import time
import random
from re import *
from string import *

#-------------------------------------------------------------
## global variables
#-------------------------------------------------------------
global doc
global exception
global exception_name
global last_method
global last_announce
global annList
global result
global results

results=[]
annList=[]
result=""
doc=""
exception=""
exception_name=""
last_method=""
last_announce=0

#-------------------------------------------------------------
## fetches the indices for a specific announce from list of announces..
#-------------------------------------------------------------
def fetch_annDocument(annresult):
        """
        returns an indice that specifies the announce document in the list of announces
        arg: the name of the announce result document
        It defaults to the first indice if the announce result document is not found
        """

          global  last_announce
          if (annresult!=""):
                  isFound = 0
```

```
                for enum in range (0, len(annList)):
                        #print "les annonces disp ", enum[0]," - ", enum[1]
                        if annList[enum][0]== annresult:
                                last_announce = enum
                       #print "dans fetch anndocument, returned: ", enum
                                isFound=1
                if not isFound:
                        print "warning !!\n\t L'annonce désignée n'a pas été trouvée. l'annonce première sera
considérée à sa place\n"
                        last_announce=0
        else:
                last_announce=0


#  ------------------------------------------------------------
## Add a new player to the database. If the player exists, it simply continu..
#  ------------------------------------------------------------

def createParty(player):
      """
      Adds a new party. If the party exists, it simply "pass"
      It waits 60 sec every time a new party is created
      """
        ctx = ContextUtil.getNamingContext()
        home = ctx.lookup("beans.PartyHome")
        state ="<gnpDocument><party><state></state></party></gnpDocument>"
      desc = "<gnpDocument> <description> <party
name="+player+"></party></description></gnpDocument>"

        try:
                bean = home.create(player, desc, state)
                print "New player: must wait 60 sec.. "
                time.sleep(60)
        except:
                pass

#------------------------------------------------------------
## sets on the fly, if necessary, the price and the quantity of an order document
#------------------------------------------------------------

def Order(file,quantity=1, price=0 ):
      """
      sets on the fly, if necessary, the price and the quantity of an order document
      """
        if (price!=0):
                setOrder(file, price,quantity)

      return file
```

```
#-------------------------------------------------------------
## parses an order document to change the price and the quantity
#-------------------------------------------------------------

def setOrder(orderFile, price, quantity):
      """
      parses an existant order document to change the price and the quantity
      """

        ford = open(orderFile,'r+')
        domOrder = DOMUtil.parse(ford.read())
        ford.close()
        ford = open(orderFile,'w+')

        elem1 = DOMUtil.getElement(domOrder, "/gnpDocument/order/sentPQList/priceQuantity/price")
        elem2 = DOMUtil.getElement(domOrder, "/gnpDocument/order/sentPQList/priceQuantity/quantity")

        elem1.getFirstChild().setNodeValue(str(price))
        elem2.getFirstChild().setNodeValue(str(quantity))

        ford.write(DOMUtil.toXMLString(domOrder))
        ford.close()



#-------------------------------------------------------------
## append all returned results in a list for later use
#-------------------------------------------------------------

def return_result(res):
      """
      append all returned results in a list for later use
      """

        global doc;
        global exception;
        global exception_name;
        global last_method;
        global result;
        if exception:
                result = exception_name
                results.append([res,0, result])
        else:
                result = doc
                if last_method == "submitAnnounce":
                        annList[len(annList) -1][0] = res
                results.append([res, 1, result])

#-------------------------------------------------------------
## gets and parses adjudication document to extract all winners
```

```
#-------------------------------------------------------------

def getLeaders(announceRes, party, prodRefGPK):
      """
      returns list of [leader name, quantity allocated, price] of a negotiation
      """
        global last_announce

        fetch_annDocument(announceRes)
        ctx     = ContextUtil.getNamingContext()
        negohome = ctx.lookup("beans.NegotiationHome")
        negoXML  = negohome.findByAnnounceGPK(long( annList[last_announce][1])).getDocument()

        print "waiting end of negotiation.."
        while (DOMUtil.get(negoXML, "/gnpDocument@state")!= "CLOSED"):
                time.sleep(5)
                negoXML = negohome.findByAnnounceGPK(long( annList[last_announce][1])
).getDocument()

        ctx = ContextUtil.getNamingContext()
        try:
                nego = ctx.lookup("beans.NegotiatorSessionHome").create()
        except:
                print "Erreur, GetLeaders: ne peut pas obtenir une session (home) reference"

        enum = nego.getAdjudicationsForProductReference(prodRefGPK)

      list = []
        while enum.hasMoreElements():
                elem = enum.nextElement()
                adjuXML = elem.getXMLDocument()
                adjuDom = DOMUtil.parse(adjuXML)
                buyer = DOMUtil.get(adjuDom, "gnpDocument/adjudication/buyerPartyKey")
                seller = DOMUtil.get(adjuDom, "gnpDocument/adjudication/sellerPartyKey")
                price = DOMUtil.get(adjuDom,
"gnpDocument/adjudication/adjudicatedPQList/priceQuantity/price")
                quantity = DOMUtil.get(adjuDom,
"gnpDocument/adjudication/adjudicatedPQList/priceQuantity/quantity")
                list.append([buyer,quantity,price])

        return list
```

**Module des paramètres:**

```
# Author: El Mostafa Ben Najim
##
### imports
##
#
from java.lang import System
import sys
sys.path.insert(0, '/usagers/'+System.getProperty("user.name")+'/Projets/gnp/msc/lib')
import testUtil
from testUtil import *

#-------------------------------------------------------------
## getter for productReferenceGPK given an announce result document
#-------------------------------------------------------------
def productReferenceGPK(announce=""):
      """
        Returns the product reference GPK of an announce result document
      It defaults to the first announce result document
        """

        try:
            fetch_annDocument(announce)
            return long(annList[testUtil.last_announce][4] )
      except:
            return long(0)


#-------------------------------------------------------------
## getter for negoGPK given an announce result document
#-------------------------------------------------------------
def negoGPK(announce=""):
      """
        Returns the negotiation GPK of an announce result document
      It defaults to the first announce result document
        """

      try:
            fetch_annDocument(announce)
            return long(annList[testUtil.last_announce][2] )
      except:
            return long(0)

#-------------------------------------------------------------
## getter for quoteGPK given an announce result document
#-------------------------------------------------------------
def quoteGPK(announce=""):
      """
        Returns the quote GPK of an announce result document
```

```
        It defaults to the first announce result document
          """
        try:
              fetch_annDocument(announce)
                return long(annList[testUtil.last_announce][3] )
        except:
              return long(0)

#-------------------------------------------------------------
## getter for previousQuoteGPK given an announce result document
#-------------------------------------------------------------
def previousQuoteGPK(announcer=""):

        """
          Returns the previous quote GPK of a negotiation
        It defaults to the first negotiation
          """


        try:
              ctx       = ContextUtil.getNamingContext()
                readerHome = ctx.lookup("beans.ReaderSessionHome")
                reader     = readerHome.create()
                lastQuote  = reader.getLastQuoteForNegotiation(negoGPK(announcer))
                return long(lastQuote.getGPK())
        except:
              return (0)

#-------------------------------------------------------------
## getter for an orderGPK given an announce result document and a negotiator
#-------------------------------------------------------------
def orderGPK(negotiator, announcer=""):
        """
          Returns an order GPK submited to a negotiation by a party (the last one submitted)
        It defaults to the first negotiation ..
        If no order is found, it returns long(0)
          """
          try:
              ordGPK   = long(0)
                enum     = negotiator.getOrdersForProductReference(productReferenceGPK(announcer))
                for elem in enum:
                        ordGPK = elem.getGPK()
                return ordGPK
        except:
              return long(0)

#-------------------------------------------------------------
## getter for oldOrderGPK given an announce result document and a negotiator
#-------------------------------------------------------------
def oldOrderGPK(negotiator, announcer=""):
        """
```

```
        Returns the most recent order GPK submited to a negotiation by a party
    In fact, it's just a call to orderGPK :-))
        """
        try:
            return long(orderGPK(negotiator, announcer))
    except:
            return long(0)


#-------------------------------------------------------------
## getter for a market name: the first one found by findAllMarkets()
#-------------------------------------------------------------
def marketName():
    """
        Returns the first market name found
    If there is no markets, it returns ""
    """
    ctx      = ContextUtil.getNamingContext()
    adm      = ctx.lookup("beans.AdministratorSessionHome").create()
    try:
        markets = adm.findAllMarkets()
        return markets[0].getName()
    except:
        return ""

#-------------------------------------------------------------
## getter for a market name: the first one found by findAllMarkets()
#-------------------------------------------------------------
def partyName():
    """
        Returns the first party name found
    If there is no parties, it returns ""
    """
    ctx      = ContextUtil.getNamingContext()
    adm      = ctx.lookup("beans.AdministratorSessionHome").create()
    try:
        parties = adm.findAllParties()
        return parties[0].getName()
    except:
        return ""
```

**Module des prédicats:**

```
# Author: El Mostafa Ben Najim
##
### imports
##
#
from java.lang import System
from string import *
import sys
sys.path.insert(0, '/usagers/'+System.getProperty("user.name")+'/Projets/gnp/msc/lib')
import testUtil
import params
from testUtil import *
from params import *


#-------------------------------------------------------------
##
#-------------------------------------------------------------
def typeOf(excep, type):
        """
      returns true if the exception name is "type"
      """
         for indice in range (0, len(testUtil.results)):
                  if testUtil.results[indice][0] == excep:
                           result = testUtil.results[indice][2]
      try:
            return find(upper(result.getName()), upper(type))!= -1
      except:
            return 0

#-------------------------------------------------------------
##
#-------------------------------------------------------------
def typeMatch(res, type):
        """
      returns true if the two arguments are of the same type
      """
         void_methods = ["purgeDeletedNegotiations", "setNegotiationState", "deleteOrder"]
      long_methods = ["updateOrder", "getNegotiationGPKforQuoteGPK",
"getProductReferenceGPKforQuoteGPK"]

         isFound=0
         if (testUtil.exception!=1):
                  for indice in range (0, len(testUtil.results)):
                           if testUtil.results[indice][0] == res:
                                    result = testUtil.results[indice][2]
                                isFound=1

                  if (testUtil.last_method in void_methods) and isFound:
```

```
                try:
                        return isinstance(result, type)

                except:
                        try:
                                return ("VOID" == upper(type))

                        except:
                                return 0

        elif    (testUtil.last_method in long_methods) and isFound:
                try:
                   return (result == long(0)) or (result > long(0))
                except:
                   return 0

                elif isFound:

                 return isinstance(result, type)

           else: pass
        else:
                #print "Warnning: Exception was NOT expected.."
                return 0

#-------------------------------------------------------------
##
#-------------------------------------------------------------
def leader(announceRes, party, quantity=0, price=0):
      """
      returns list of [leader name, quantity allocated, price] of a negotiation
      """

        match = 0 # A priori not leader
        leadersList  = getLeaders(announceRes, party, productReferenceGPK(announceRes))

      if leadersList != []:
                if (quantity == 0) or (price == 0):

                        indice=0
                        while (indice<len(leadersList)) and (match == 0):

                                if (party ==leadersList[indice][0]):
                                        match=1
                                indice = indice +1
                else:
                        indice=0
                        while (indice<len(leadersList)) and (match == 0):
```

```
                                        if (party == leadersList[indice][0]) and (quantity ==
atof(leadersList[indice][1])) and (price==atof(leadersList[indice][2])):
                                                match=1
                                        if (party == leadersList[indice][0]):

                                                print "Leader - quantity - price : ",leadersList[indice][0]," -
",leadersList[indice][1]," - ",leadersList[indice][2]
                                        indice=indice+1
        else:
                if upper(party) == "NONE":
                        match = 1
        return match
```