2000RP-01

Architecture informatique de GNP Version 1.0

Robert Gerin-Lajoie

Rapport de Projet
Project report

Avec la collaboration de Sophie Lamouroux, El Mostafa Ben Najim, Isabelle Therrien, Vincent Trussart

Montréal
Août 2000

CIRANO
Centre interuniversitaire de recherche en analyse des organisations

CIRANO

Le CIRANO est un organisme sans but lucratif constitué en vertu de la Loi des compagnies du Québec. Le financement de son infrastructure et de ses activités de recherche provient des cotisations de ses organisations-membres, d'une subvention d'infrastructure du ministère de la Recherche, de la Science et de la Technologie, de même que des subventions et mandats obtenus par ses équipes de recherche.

CIRANO is a private non-profit organization incorporated under the Québec Companies Act. Its infrastructure and research activities are funded through fees paid by member organizations, an infrastructure grant from the Ministère de la Recherche, de la Science et de la Technologie, and grants and research mandates obtained by its research teams.

Les organisations-partenaires / The Partner Organizations

- •École des Hautes Études Commerciales
- •École Polytechnique de Montréal
- •Université Concordia
- •Université de Montréal
- •Université du Québec à Montréal
- •Université Laval
- •Université McGill
- •Ministère des Finances du Québec
- •MRST
- •Alcan inc.
- •AXA Canada
- •Banque du Canada
- •Banque Laurentienne du Canada
- •Banque Nationale du Canada
- •Banque Royale du Canada
- •Bell Canada
- •Bombardier
- •Bourse de Montréal
- •Développement des ressources humaines Canada (DRHC)
- •Fédération des caisses Desjardins du Québec
- •Hydro-Québec
- •Industrie Canada
- •Pratt & Whitney Canada Inc.
- •Raymond Chabot Grant Thornton
- •Ville de Montréal

ISSN 1499-8629

TABLE DES MATIÈRES

1. Introduction

1.1. Historique du document

Version	*Date*	*Auteur*	*Commentaires*
0.1D	*Juin 2000*	*Robert Gérin-Lajoie,*	*En anglais, version de travail*
0.2 D	*Juillet 2000*	*Robert Gérin-Lajoie*	*En anglais, version de travail avec commentaires et texte de Sophie Lamouroux and Vincent Trussart*
0.3 D	*Août 2000*	*Robert Gérin-Lajoie*	*En français,et intégrant les documents produit l'été 2000 (SophieL., Vincent T., Mostapha N.)*
0.4D	*Février 2001*	*Robert Gérin-Lajoie*	*Corrections mineures*

1.2. Audience du document

[Décrire l'audience de ce document]

Ce document introduit l'aspect technique et informatique du système « Generic Negotiation Platform ». L'aspect économique de GNP et la définition formelle des algorithmes économique sont présentés dans le document « Enjeux de design des enchères ».

1.3. Objectifs du document d'architecture logicielle

[Expliquez le concept d'architecture logicielle et comment il est représenté dans ce document. Introduisez les termes et concepts des différentes vues du système présenté dans les prochaines sections.]

Ce document suit les conventions du « Unified Modeling Langage » ou « UML » pour présenter l'architecture du système GNP:

- Scénarios d'utilisations;
- Interactions humains-machines à partir des pages HTML et « Java Server Pages » (JSP);
- La vue des classes, avec diagrammes illustrants les relations entre les classes pour :
  - le modèle du domaine des enchères;
  - les interactions distribuées;
  - les enregistrements dans la base de données;

?? La vue du déploiement des processus sur les différents systèmes.

Ce document ne présente qu'un squelette de l'architecture. Ce document ne présentera pas une description détaillée de chaque méthode et attribut. Ces descriptions détaillées sont disponibles dans la documentation Web des développeurs. Le site Web des développeurs est accessible à http://comte.CIRANO.UMontreal.CA/~eree/devel/. Il contient la documentation Java selon le standard Javadoc et la documentation Python selon la convention du projet pydoc (Voir http://comte.CIRANO.UMontreal.CA/~eree/devel/pydoc.html). Un script régulier génère automatique ces pages de documentation.

1.4. Documents connexes

?? "Enjeux de design des enchères", V1.1 Yves Richelle, Jacques Robert et Robert Gérin-Lajoie, août 2000

?? "Présentation de GNP", Voir ..\documents\CRSNG_GNP\CRSNG_GNP.ppt

?? "Introduction à GNP pour les scripteurs", Sophie Lamouroux, mai 2000

?? "Manuel des participants de GNP", à venir, Yves Richelle

?? "Deployment & Operational description", à venir

?? "Plan de tests de GNP", à venir

?? Le site Web des développeurs de GNP (www.cirano.umontreal.ca/~eree/devel), protégé par un username et un mot de passe.

1.5. Références externes

?? "Introduction to Java 2, Enterprise Edition," voir http://developer.java.sun.com/developer/technicalArticles/J2EE/DesignEntApps/

?? "Manuals for WebLogic 5.1.0", voir http://www.weblogic.com/docs51/resources.html

?? "Manuals for Python and JPython", voir www.python.org

?? "Manual for XML and XML4J 3.0.1" (Xerces, DOM, ...), voir http://xml.apache.org/xerces-j/index.html

2. Survol de la place de marché

[Décrire brièvement le système et ses composantes majeures]

2.1. Niveau d'architecture du commerce électronique

Les portails de commerce électronique vont offrir une grande variété de services de commerce électronique. Les systèmes de commerce électronique sont complexes et distribués sur l'Internet parmi une grande variété d'ordinateurs et d'organisations. Ces systèmes doivent traverser des frontières très diverses : géographiques, politiques, compagnies, organisations. Fréquemment la coopération entre ces systèmes est minimale, et techniquement nous ne pouvons prendre pour acquis que l'échange de message entre les ordinateurs des compagnies participantes.

Pour exprimer la complexité, nous devons « diviser et conquérir », une vue à la fois. Prenez par exemple l'industrie de la construction : ils ont beaucoup de plans pour un édifice, chacun se concentrant sur un aspect.

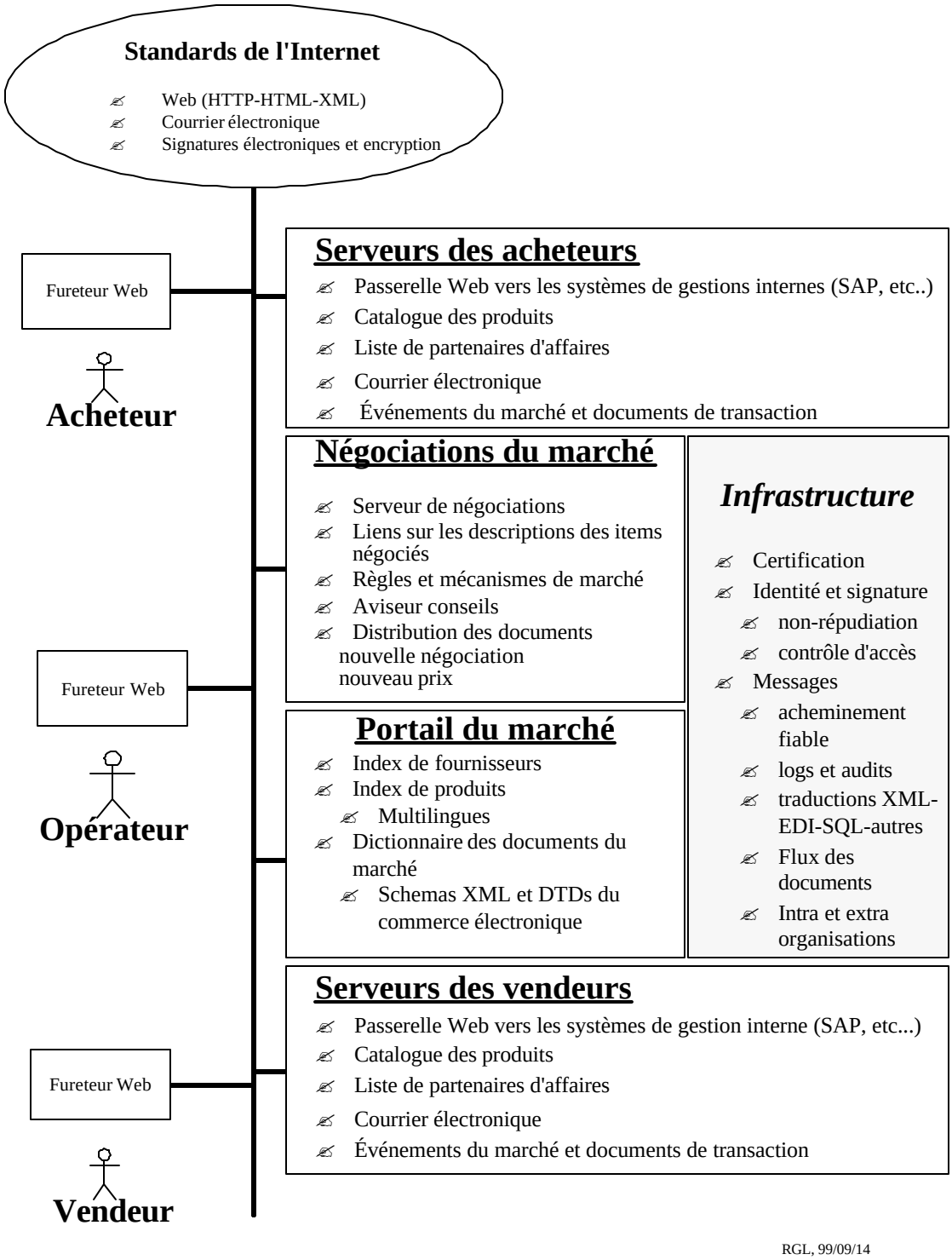
Pour voir et comprendre l'architecture informatique nécessaire pour remplir ces services de commerce électronique, nous devons itérer au travers de différents niveaux d'architecture : architecture d'affaire, architecture de l'information, architecture des fonctionnalités, architecture des composantes et finalement l'architecture du déploiement. Chaque niveau d'architecture focalise sur un aspect et ainsi nous pouvons saisir tous les aspects des systèmes.

1. Architecture d'affaire	Ce niveau modélise les solutions qu'un portail vend à ses participants.
2. Architecture fonctionnelle et des services	Ce niveau décrit les fonctions et les services un à un. Fréquemment un groupe de fonctions et de services seront assemblés pour produire une solution d'affaire.
3. Architecture de l'information et des données	Ce niveau décrit la sémantique des documents du commerce électronique. Il définit le domaine, les données et les objets d'affaire. Ce niveau est aussi appelé l' « ontologie » du commerce électronique. Le modèle de référence Eco et de ebXML divise l'ontologie des documents d'affaire en 6 couches : le marché, l'entreprise, le service, l'interaction, le document, et l'item d'information. La couche de document est déjà bien décrite dans les standards internationaux de l'EDI, X.12 ou EDIFACT et les standards futurs du XML.
4. Architecture technologique	Ce niveau décrit les composantes et les sous-systèmes assemblés pour construire une application, une fonction ou un service, ainsi que leurs interrelations.
5. Architecture de déploiement	Ce niveau est l'architecture du physique du réseau, des systèmes et des opérations. Ce niveau se

	préoccupe des questions de performances, de monté en charge, de fiabilité et de sécurité.

2.2. Architecture fonctionnelle d'une place de marché

Architecture d'une place de marché électronique



La figure « Architecture d'une place de marché électronique » illustre la vision d'une place de marché interentreprises et le rôle central que la négociation y jouera.

2.2.1. Un bus commun : l'Internet et les standards internationaux

Les nouveaux systèmes pour le commerce électronique doivent utiliser un bus commun d'informations partagé entre tous les participants. Les standards classiques étaient EDI et EDIFACT, les nouveaux seront basés sur les nouveaux standards de l'Internet : HTTP, HTML et XML, le courrier électronique, les signatures électroniques et l'encryption.

2.2.2. Les fureteurs des participants au marché (vendeurs et acheteurs)

Tous les participants doivent pouvoir rejoindre le marché avec un fureteur agissant comme une interface universelle. Les participants accomplissent trois rôles différents dans le marché : acheteur, vendeur et opérateur de marché.

2.2.3. Les serveurs des vendeurs et des acheteurs.

Ces serveurs contiennent les catalogues de produits à vendre ou à acheter, les passerelles Web vers les systèmes de gestion interne et de courrier électronique. Les prix des biens selon les contracts, les disponibilités, les liens vers les systèmes de prise de commande, de livraison et de facturation sont parmi les fonctionnalité qu'ont y retrouve. Les serveurs de chaque côté du marché peuvent résider dans les locaux de la compagnie ou être gérés à forfait chez un fournisseur de services d'applications.

2.2.4. Le service du portail et les informations du marché.

Ce service génère l'information du marché utilisée par les participants et les programmes : les participants enregistrés, les entreprises, leur localisation et leurs notes de qualité, les schémas des documents du marché comme les ordres, les mises, les bons de connaissement.

2.2.5. Le service de négociation

Le serveur de négociation est au cœur de la place de marché. Il est spécialisé dans le pairage entre la demande des acheteurs, l'offre en biens et services des vendeurs et la fixation des prix. Il reçoit les annonces des acheteurs ou des vendeurs et les mises qui répondent à ces annonces.

2.2.6. Services d'infrastructure

Une place de marché utilise les standards de l'Internet : HTTP, HTML, SMTP, MIME et XML et échange les documents selon ces standards. Cependant, la grande variété syntaxique et sémantique des documents, de EDI aux divers sous-standards de XML demande un traducteur générique pour assurer la communication entre les partenaires de celle-ci.

Les services d'infrastructure robustes sont indispensables à la bonne qualité d'une place de marché :

- **La certification et l'identification des participants;**

  L'identification des participants au portail est essentielle à la confiance que chaque participant aura pour faire du commerce sur le portail. L'identité des participants est contrôlée par un système de contrôle d'accès basé sur les serveurs de répertoire LDAP ou par des tables SQL.

  Ce sous-système gère l'identité des participants, leur mot de passe et/ou leur clef de signature et d'encryption, et finalement les listes de contrôle d'accès aux services.

- **La messagerie asynchrone;**

La messagerie interordinateur garantie la livraison entre les systèmes, même si le réseau est bloqué ou certains des systèmes sont en panne. Un autre aspect crucial de la messagerie asynchrone est la journalisation des transactions disponibles pour les audits externes et l'arbitrage légal.

- **La traduction des documents;**

Les documents d'affaires sont dans un grand nombre de formats diversifié : de XML, EDI, SAP et fichiers plats avec des valeurs séparés par des virgules (CSV, Comma Separated Values). Ainsi un traducteur central est nécessaire entre les serveurs des participants, au-dessus du système de messagerie asynchrone. Si le portail ajoute et modifie ses partenaires fréquemment, la gestion des tables de traduction devra pouvoir se faire facilement et économiquement, sans reprogrammation.

- **La gestion et la facturation;**

La gestion des utilisateurs est responsable des options disponibles et de l'enregistrement des droits d'accès pour les participants.

Le sous-système de facturation reçoit les statistiques sur l'utilisation des services, après lequel il produit les coûts aux participants.

- **Les opérations.**

La console centrale d'opération surveille la disponibilité et la stabilité de tous les services du portail. Cette console est nécessaire pour assurer un haut niveau d' « Entente de Niveau de Service » ou « Service Level Agreement » (SLA).

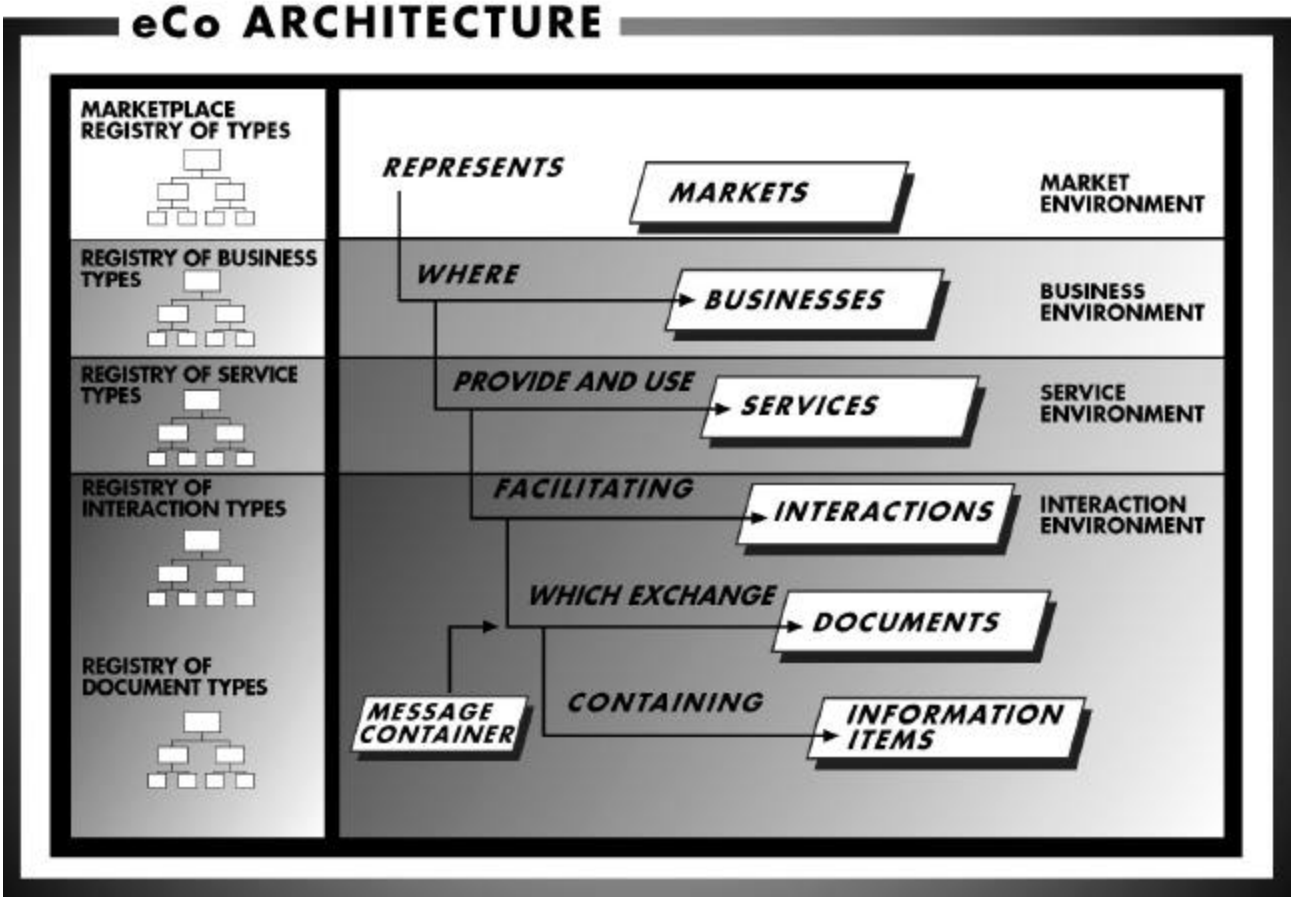
La console est basée sur le protocole standard « Simple Network Monitoring Agent » , SNMP.

2.3. L'architecture de l'information

Ce niveau définit la sémantique des messages échangés par le bus commun de la place de marché. Ce niveau décrit la sémantique des documents du commerce électronique. Il définit le domaine, les données et les objets d'affaire. Ce niveau est aussi appelé l' « ontologie » du commerce électronique.

Les modèles de références Eco et de ebXML divise l'ontologie des documents d'affaire en 6 couches : le marché, l'entreprise, le service, l'interaction, le document, et l'item d'information.

La couche de document est déjà bien décrite dans les standards internationaux de l'EDI, X.12 ou EDIFACT et les standards industriel de XML tels xCBL (par CommerceOne) et xCML (par Ariba).



2.4. Les modèles de données disponibles

Le système GNP focalisera sur l'ensemble minimum d'interactions et de documents XML nécessaires. Regardez la section 5.2 : « ONTOLOGIE DU DOMAINE ET DOCUMENTS XML »

3. Présentation du système : buts et contraintes

[Décrire les buts et les contraintes de l'architecture.]

3.1. La plate-forme de négociation générique

La plate-forme de négociation générique (GNP) est conçue pour s'insérer dans une place de marché tel que décrite précédemment. GNP est un système multi-tiers avec 5 tiers :

- Les participants : acheteurs, vendeurs et régisseurs de marché rejoignent les serveurs GNP avec leurs fureteurs, soient par des pages HTML, soient 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, utilisant des Servlets Java et des pages « Java Server Pages », JSP. Ce serveur est bâti sur l'infrastructure J 2EE fournit par le serveur BEA-Weblogic;

- Le serveur de requêtes GNP, utilisant les « Enterprises Java Beans » (EJB) pour les sessions et les entités;

- 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.

3.2. Les règles de marché

Le système GNP est conçu pour servir de base aux nouvelles règles économiques nécessaires aux systèmes de négociation d'entreprises à entreprises. Le système est bâti autour de ces concepts :

?? *Enchères directes, renversées et des deux côtés;*

?? *Négociations conduites avec des phases et des rondes;*

Les négociations se déroulent sur plusieurs phases lorsque les règles changent au cours de la négociation. Chaque ronde est une suite de trois événements de base : une cote émise par l'encanteur, une ou plusieurs mises des participants et finalement l'évaluation par l'encanteur des mises reçues, le tout suivi d'une allocation temporaire. La nouvelle ronde démarre avec une nouvelle cote illustrant le résultat de la ronde précédante.

?? *Des chorégraphies multiples*

- Enchère anglaise;

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

- Enchère hollande;

Une 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 modifié. L'enchère ferme dès la première mise reçue.

- Enchère cadencée;

Une 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

Une enchère synchronisée utilise la même chorégraphie pour piloter les négociations sur des items reliés.

- Des paramètres de temps variés

Pour chacune des types de chorégraphie énumérés plus haut, les paramètres de temps peuvent être variés, de la minute aux jours selon les besoins du marché.

?? *Marché avec des notes de qualité, score et ordonnancement complexe.*

Quand la qualité des items et des compagnies engagées dans une négociation varie, le système GNP de négociation intègre les facteurs de qualité dans le processus d'évaluation des mises et utilise une fonction de score paramétrisée pour ordonnancer les ordres.

?? *Enchères multi-unitaires*

Une enchère multi-unitaires négocie le prix et la quantité de plusieurs unités du même bien. Ce type d'enchères a la caractéristique unique que les unités d'achats ou de ventes peuvent être fragmentées à divers niveaux de prix et de quantité.

?? *Enchères multi-objets et enchères synchronisées*

Une enchère multi-objets négocie simultanément plusieurs objets reliés. Ces objets sont habituellement des biens ou des services complémentaires. L'annonceur accepte des mises partielles et il espère qu'une combinaison des mises partielles va couvrir sa demande initiale.

?? *Enchères multi-attributs*

Une enchère multi-attributs est une légère variation des précédants. Les attributs peuvent être des objets additionnels demandant une confirmation sur une propriété essentielle de l'annonce, comme le respect des dates de livraison ou autres.

?? *Ordres simples et complexes*

Les ordres et les mises avec seulement un prix et une quantité sont la base de toute négociation. Des ordres plus complexes sont souvent requis dans une négociation d'affaire :

- Un ordre avec le prix courant et le prix de réserve (ou prix limite);

- Un ordre avec un vecteur de prix-quantité, exprimant une fonction d'offre ou de demande;

- Un ordre sur plusieurs objets combinés, dans une négociation combinée.

3.3. Flexibilité des règles économiques et d'affaires

Cette plate-forme de négociation générique a été conçue pour permettre une programmation rapide de n'importe quel type de négociation. Ainsi l'architecture de GNP est ciblée sur la flexibilité et une claire séparation des rôles entre d'un côté les documents d'affaires et les scripts économiques et d'un autre côté les sessions et les entités « Enterprises Java Beans » écrites en Java 1.2 .

Les documents économiques sont en XML, les scripts sont en JPython.

Cette table illustre les différents niveaux d'algorithme et scripts, ainsi que les documents d'affaire en XML et entités activées dans une négociation.

	Algorithmes et scripts	Documents et entités
Concepts économiques	**Étapes de réception (1ère):** « AnnonceValidator » « OrderValidator » **Étape d'évaluation (2ième):** « Auctioneer » Ou « MarketControler » (pour les activités du marché)	« Market description and state » « Party description and state » « Announce » « Quote » « Order » « Response » et « Allocation » « Adjudication »
Concepts de la plate-forme de négociation générique.	**Étapes de réception (1ère):** « Announcer session » « Reader session » « Negotiator session » « Administrator session » **Étape d'évaluation (2ième):** « Auctioneer dispatcher » « Auctioneer »	Entités « Enterprise JavaBean »
Systèmes génériques industriels	Java 2, Enterprise Edition, « BEA-Weblogic application server » Java 1.2/1.3 Virtual Machine	Base de données SQL; « PostgreSQL » ou « Oracle 8i »

3.4. La séquence d'appel aux règles

Le traitement des ordres et des mises par le système GNP est divisé en deux étapes : la réception et l'évaluation. L'étape de réception des ordres est faite en parallèle avec des « sessions beans », une pour chaque participant actif. À cette étape, les ordres sont validés par les objets « Validator », puis envoyés à la seconde étape par une queue asynchrone JMS. L'étape d'évaluation prend un ordre ou une alarme du cadran à la fois et appelle la méthode appropriée de l'encanteur. Une séquence stricte des appels aux règles économiques est ainsi réalisée.

Ce model d'entités asynchrones avec « Java Message Queue » sera traduit dans une future version de GNP comme des entités de messages selon le standard EJB 2.0. Ceci sera une base solide pour les flux de tâches dans les négociations.

3.5. Les propriétés du système

L'objectif des propriétés du système est d'avoir un environnement robuste assurant le bon fonctionnement des négociations :

1. *La performance : temps réponse et capacité en charge*

Un temps réponse court et une latence réduite lors de la réception des ordres des participants sont essentiels. Le temps réponse de la réception d'un ordre doit être au plus de 0.5 secondes sur le serveur, 1 seconde en incluant l'aller-retour d'un fureteur sur un réseau local. Ce temps réponse sera possible en utilisant les capacités de réception parallèle des mises sur le serveur d'application.

L'évaluation des mises et le changement de ronde ou de phase devra prendre au plus 1 seconde.

La capacité de monté en charge est bâtie avec les 2 axes suivants de l'architecture : plusieurs serveurs « frontaux » peuvent se répartir la charge de la phase de réception; la phase d'évaluation traite un message à la fois en conservant en mémoire vive le contexte d'une enchère si nécessaire.

Un serveur à 2 processeurs Intel ou Sparc pourra soutenir 100 participants simultanés produisant chacun une mise à la minute.

2. *Fiabilité transactionnelle;*

La fiabilité transactionnelle est requise. Après la réception par un participant de son accusé de réception, le système garanti que le serveur traitera éventuellement sa commande. Cela peut prendre un certain temps cependant si le système est surchargé ou en panne.

3. *Sécurité;*

La sécurité du système couvre quatre aspects :

?? L'identification

L'identification a pour but de reconnaître le participant. Elle se fait en demandant à un utilisateur de confirmer son identité avec une information privée ou un secret : un mot de passe, une clef privée, etc.… Pour la première version, un mot de passe encrypté et donc connu seulement du participant est amplement suffisant. Cependant, comme l'environnement J2EE que nous utilisons supporte les

systèmes à clef publique et l'identification par des serveurs LDAP, il sera aisé en mode déploiement de supporter ce deuxième type d'identification.

?? La confidentialité

La confidentialité a pour but de limiter aux seuls participants qui y ont droit l'information. L'encryption par https des informations sensibles sur le réseau permet d'éviter l'écoute du trafic et des messages. De plus les données emmagasinées dans les bases de données doivent être inaccessibles de l'extérieur, voir la section 7 sur le déploiement.

?? Le contrôle des rôles

Les listes de contrôle d'accès permettront de contrôler les rôles des participants ainsi que leurs actions. Le modèle de sécurité de « Java 2, Enterprise Edition » sera utilisé, en particulier les attributs de sécurité dans les fichiers de déploiements des WAR et des EJB.

?? L'intégrité

L'intégrité du système vise à s'assurer que le système reste intègre, sans virus et autres malversations. Cette intégrité repose essentiellement sur la qualité de la configuration de déploiement utilisée dans la phase opérationnelle.

4. Divers type de clients et d'accès réseaux au système

Plusieurs technologies d'accès peuvent éventuellement être utilisées par les logiciels clients: fureteur HTML, HTTP ou XML, agent distant écrit Java ou JPython, autre serveur d'entreprise.

En conclusion, ces objectifs et ces propriétés du système nous ont amené à prendre le modèle « Java 2, Enterprise Edition » et les pages « Java Server Page » pour desservir les clients HTML et XML.

Nous utilisons la plate-forme BEA-Weblogic 5.1 comme serveur HTTP et JSP et comme serveur d'application EJB, PostgreSQL ou Oracle8i comme base de données.

La « Java Virtual Machine » doit être configurée pour une utilisation en mode serveur, capable de supporter des centaines de fils d'exécution ou « threads », une centaine de session simultanée et un bon ramasse miette (« garbage collector »). La version 1.3 de Java avec Hotspot convient tout à fait.

4. Scénarios d'utilisation

Le diagramme qui suit illustre le scénario d'utilisation principal, les acteurs et leurs rôles dans le processus négociation.

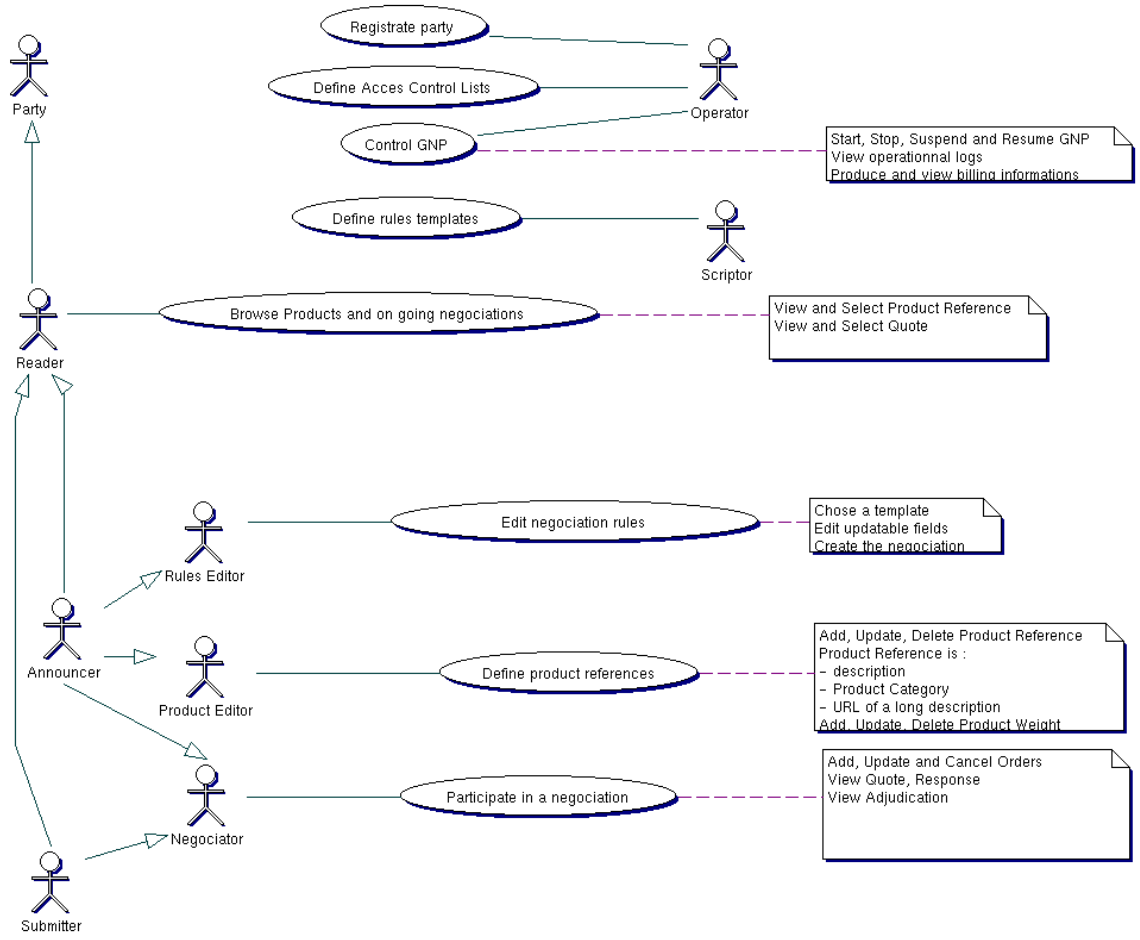


Diagramme 1 : Main Use Case

4.1. Lecteur - « Reader »

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

4.2. 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. Fixer les paramètres éditables comme l'heure de fermeture;

- 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.

4.3. Negociateur et soumissionnaire - « Negotiator and Submitter »

Un négociateur émet les ordres d'achats ou de vente. Un acheteur et un vendeur peuvent donc être un négociateur. 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.

4.4. Administrateur et operateur - « Administrator and Operator »

Les administrateurs et 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.

4.5. Le déroulement d'une négociation

L'initialisation d'une négociation sur GNP se fait par l'envoie d'un document XML initial : l'annonce, qui contient tous les éléments d'input nécessaire au démarrage d'une négociation.

4.5.1. Envoie 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 (template). 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

4.5.2. Envoie de l'annonce directement

Dans un objectif de tests, il est possible d'envoyer directement le document XML d'une annonce à GNP. Dans ce cas, le document de l'annonce doit être entièrement édité manuellement dans le format XML valide et attendu, dans un fichier.
L'envoie du chemin complet (en répertoires UNIX) à GNP remplace alors les 4 premières étapes de l'initialisation par modèle.

4.5.3. Séquence d'actions après l'envoie d'une annonce

L'annonce donne a GNP tous les éléments d'input nécessaire au démarrage d'une négociation. L'annonce apparaît à l'utilisateur dans un format HTML. Au niveau interne, l'annonce est un document XML. Elle contient toujours une partie *règles* (rules) et, selon le mécanisme choisi, une partie *produit* (productReference), et une partie *ordre* (order). La partie *règles* établit précisément les paramètres fixes de la négociation, telle 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 *produit* définit techniquement la nature du produit à négocier, renvoi à sa description précise. La partie *ordre* 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. Lorsque l'annonce est remplie, elle est envoyée au système qui va déclencher l'announceSession Bean.

L'announceSession va être en charge de :

- créer l'objet ***rules*** et son document XML ***gnpDocument:rules*** à partir de l'annonce.
- créer l'objet ***negotiation*** et son document vide
- créer l'objet ***productReference*** et son document XML à partir de celui de l'annonce s'il existe.
- créer l'objet ***order*** et son document XML à partir de celui de l'annonce s'il existe.
- 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, qu'une négociation sur un produit soit commencée ou non.

4.6. Différents concepts économiques utilisés lors de la programmation des scripts

Ronde : une ronde débute avec l'affichage d'une nouvelle cote et se termine par une réallocation des quantités en négociation. Si la réallocation ne donne lieu à aucun changement, la ronde est inactive.
Numérotation des rondes :
La négociation commence à la ronde 0, c'est-à-dire à l'affichage de la cote 0 et des réponses 0, établie par rapport aux ordres 0 déposés avant l'ouverture de la négociation, en même temps que le modèle. Ensuite, pour r 0, La cote r affichée à la ronde r prend en compte l'allocation déduite de l'ajout des ordres r déposés pendant la ronde r-1. Autrement dit, les nouveaux ordres reçus lors de la ronde r sont comptés comme des ordres de la ronde suivante r+1.
Activité : on considère qu'il y a eu une activité pendant une ronde si le ou les ordres reçus ont provoqué un changement dans l'allocation des quantités et/ou des prix. De manière équivalente, si le ou les derniers ordres alloués ont été déposés pendant la ronde courante.
Si un participant envoi un ordre éligible étant donné l'état de la ronde précédente, qui est rejeté parce qu'il a été dépassé par un autre ordre de la même ronde, ce participant est considéré actif.
2 approches de fin par inactivité :
- rondes de durée fixe - Fin de la négociation au bout de N rondes inactives / ou durée limite de la phase
- rondes changent à chaque nouvel envoie d'un ordre elligible ou lorsqu'un certain temps est dépassé - Fin de la négociation lorsque la durée limite d'une ronde est atteinte / ou durée limite de la phase

Elligibilité : Une mise est éligible si elle peut être déposée dans la table d'allocation. Elle doit respecter deux types de considération :
- le participant avait-il le droit de miser ? (Éligibilité du participant)
- le prix de la mise est-il acceptable par rapport aux mises allouées de la ronde précédente ? (Éligibilité de la mise)

Phase : Une phase correspond à des règles de négociations constantes, si les règles doivent être changées au cours d'une négociation, celle-ci sera découpée en plusieurs phases. Les différentes phases peuvent être séparées par un laps de temps et un nettoyage des ordres sur le marché peut être effectué en fonction de certaines règles d'activité.
Période d'annonce : Nous ne considérons pas les périodes d'annonce comme des phases, celles-ci ne changeant en rien les règles de négociation. Une période d'annonce donne la possibilité aux participants de l'un des cotés du marché d'envoyer un formulaire de référence à un produit et un formulaire d'ordre sur la base des même règles de négociation choisie dans le modèle. Une période d'annonce n'existe que dans les

marchés de type simple, les marchés doubles plaçant de manière symétrique les envois d'ordre des 2 cotés du marché.

2 possibilités pour les périodes d'annonce.

Cette période peut avoir lieu avant l'ouverture des négociations mais aussi pendant que les négociations sur les produits déjà annoncés se poursuivent. L'une ou l'autre de ces possibilités est choisie dans le modèle (dans les règles de négociation)

Marché simple : Type de marché dans lequel les mises des participants sont d'un côté seulement (achat ou vente) et répondent à une ou plusieurs ordres de type opposé (vente ou achat) contenu dans l'annonce initiale. Par exemple, les enchères à la criée pour la vente de biens ou les enchères renversées pour les appels d'offre; les enchères anglaises (traditionnelle à la criée, fermée), et les enchères hollandaises.

Marché double : Type de marché dans lequel l'annonce ne contient aucun ordre. Les mises envoyées par les participants sont indépendamment de type achat ou vente, constituant ainsi des ordres d'offre et de demande qui se compétitionnent mutuellement. (Exemple : marché boursier)

Coté : achat ou vente

Coté de référence du marché : Dans un contexte de marché simple, le coté de référence est le coté des ordres envoyés dans l'annonce. Les mises seront alors implicitement du coté opposé au coté de référence.

5. Architecture logique

5.1. Survol

[Survol de l'architecture logique]

Nous divisons la présentation de l'architecture de GNP et des classes de GNP en cinq facettes complémentaires : la logique du domaine, les interactions systèmes, le modèle de données, le modèle de sécurité et les interactions humain-machines.

La section 5.2 présente la logique du domaine et les documents XML qui représente les entités majeures du système. La section 5.3 présente les scripts économiques et leurs inter-réactions. La section 5.4 illustre les classes dynamiques et leurs inter-réactions. Le modèle d'inter-réaction dynamique est basé sur le modèle de référence « Enterprise Java Bean ». Toutes les inter-réactions des clients Web et des clients externes natifs Java se font avec des «Sessions Bean ». Cette section présente les appels entre les clients et le serveur d'application. La section 5.5 illustre la vue de la base de données tel que les entités sont conservées. La section 5.6 décrit le modèle de sécurité. La section 5.7 enfin présente les classes responsables de l'interface humain-machine, en particulier les classes Java utilisées dans les pages JSP et pour la librairie de tag JSP conforme au standard « taglib ».

5.2. Ontologie du domaine et documents XML

5.2.1. Définition XML des documents GNP

Les documents XML dans GNP ont été défini 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), sa date de création, son lien avec un autre document, etc. sont rajoutés par GNP au moment de la construction de l'objet.
exemple : format standard du document : negotiation.

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

Exemple avec un document de négociation:

```
<gnpDocument GPK="589886" announceGPK="589885" creationTime="957984417000"
state="OPENED">

 <negotiation> </negotiation>

</gnpDocument>
```

5.2.2. Le document de description et d'état d'un participant

a) Description d'un participant

Ce document n'a pas été utilisé jusqu'à maintenant. Il contiendrait les informations de description d'un participant, comme son nom, son adresse, et ainsi de suite.

· *Exemple de document "PartyDescription" minimal*

```
<gnpDocument>
    <party name="u13">
        <description/>
    </party>
</gnpDocument>
```

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 d'un participant et le formulaire initial.

· *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>
```

5.2.3. Le document de règles - « Rules »

Ce document contient l'ensemble des informations relatives à la négociation et communes à chacun des produits. Chaque phase y est décrite. Les noms des scripts encanteurs exécutant les règles économiques y sont fournis, ainsi que leurs paramètres comme la durée des rondes, l'heure de clôture des phases.

Ces informations sont non-éditables pendant la durée de la négociation.

?? *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/auctionneer032sl.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
l 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
```

5.2.4. La référence au produit - « Product Reference »;

La référence au 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, un par produit de l'enchère.

?? *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>
```

5.2.5. Un 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 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.

?? *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>
```

5.2.6. La négociation –« Negotiation »

L'encanteur construit et maintient à jour ce document interne. Ce document 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.

?? *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>
```

5.2.7. 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 ronde et de phase ainsi que le temps d'expiration en sont des composantes usuelles. Le domaine de mise général pour tous les participants y est aussi. Le domaine de mise contient l'intervalle de prix et de quantité dorénavant admissible.

La cote peut être personnalisée dans sa forme par la page JSP qui l'affiche et par une feuille de style.xsl

```
<!--    QUOTE  -->
<!-- a afficher:
- a 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>
```

5.2.8. 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 attribut à ce participant. Cette allocation la quantité allouée et son prix 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.

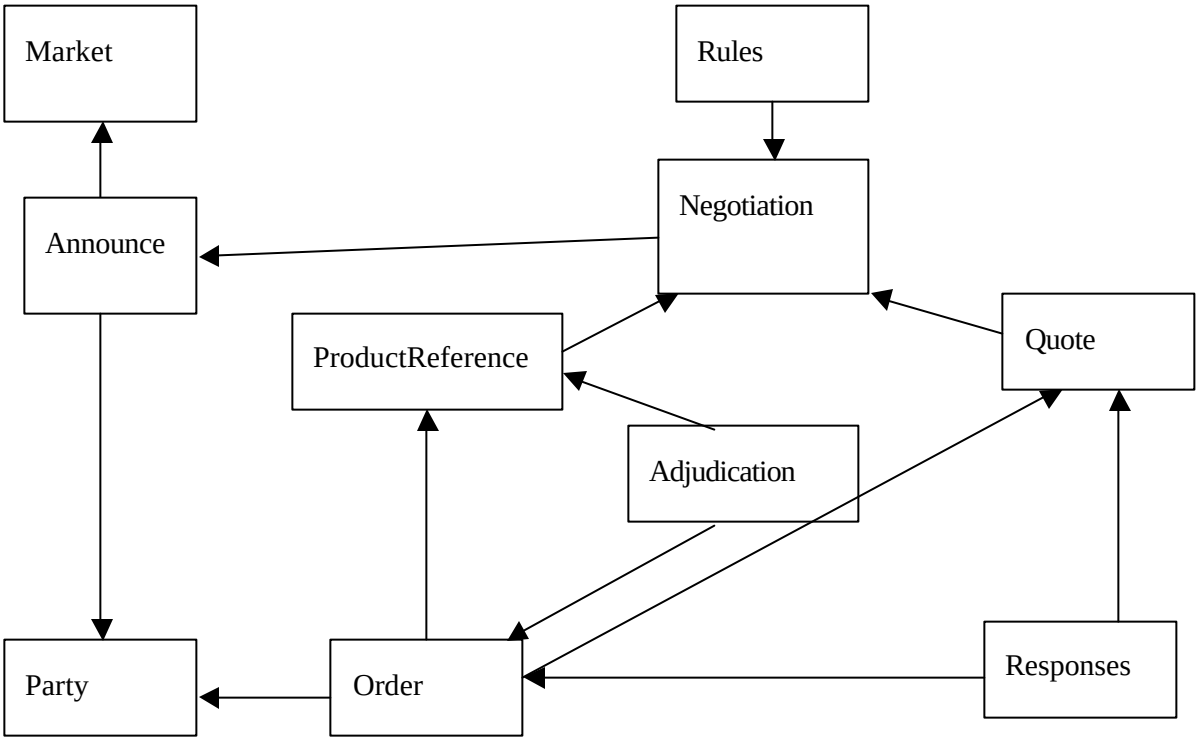
```
< --    RESPONSE   -->
<!-- a afficher:
- au joueur : PartyKey de l attribut orderGPK de response
- le dernier objet response concernant cette negociation et ce joueur
-->

<?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>
```

5.2.9. Adjudication

L’adjudication est produite par l’encanteur à la fin d’une 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.

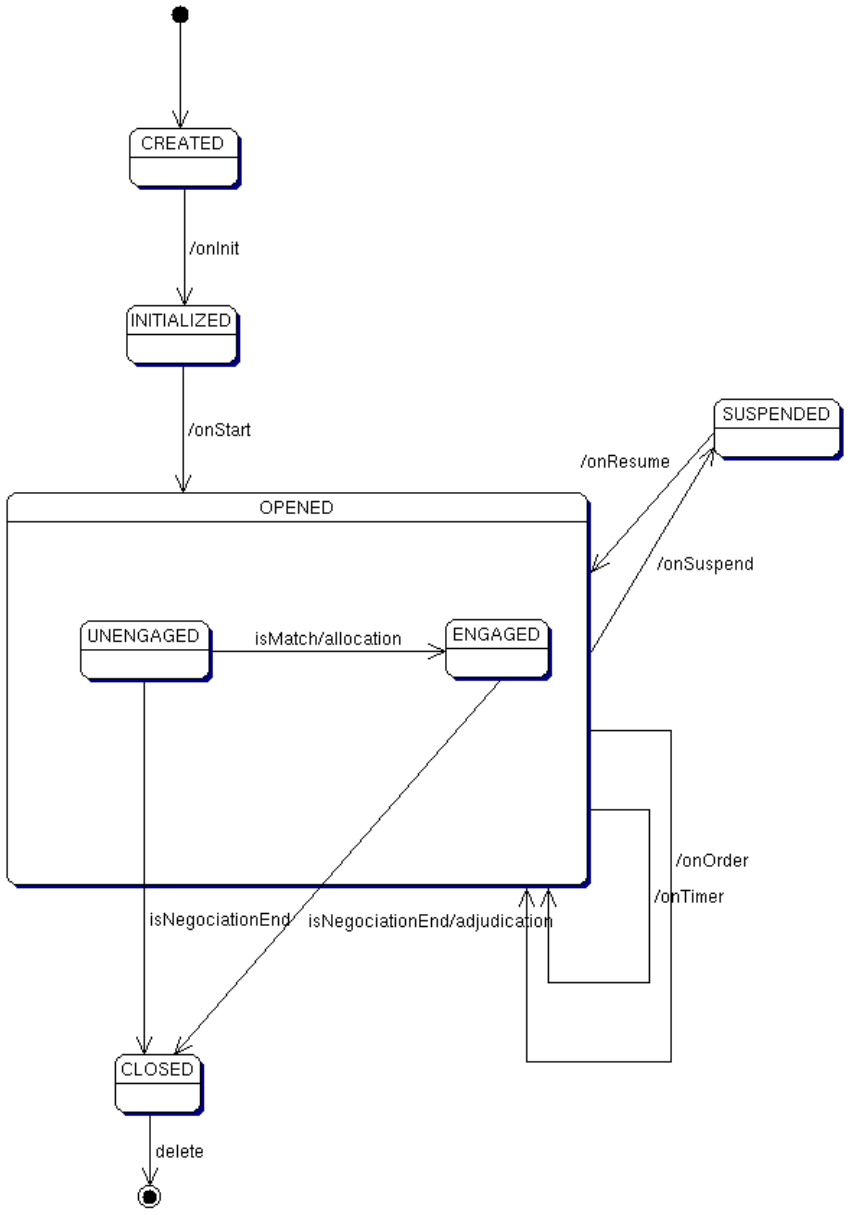
```
< --    ADJUDICATION   -->
<!-- a afficher:
-aux seuls deux joueurs concernes : buyerPartyKey et sellerPartyKey
-les adjudications existantes pour cette negociation et ces joueurs -
->

<?xml version="1.0" encoding="UTF-8"?>
<gnpDocument GPK="1966420" buyOrderGPK="1966411"
    creationTime="959974763000" productReferenceGPK="1966406"
sellOrderGPK="1966408">
    <adjudication>
        <buyerPartyKey>1</buyerPartyKey>
        <sellerPartyKey>1</sellerPartyKey>
        <adjudicatedPQList>
            <priceQuantity>
                <price>25.0</price>
                <quantity>100.0</quantity>
            </priceQuantity>
        </adjudicatedPQList>
    </adjudication>
</gnpDocument>
```

5.2.10. Relation entre les documents et les entités

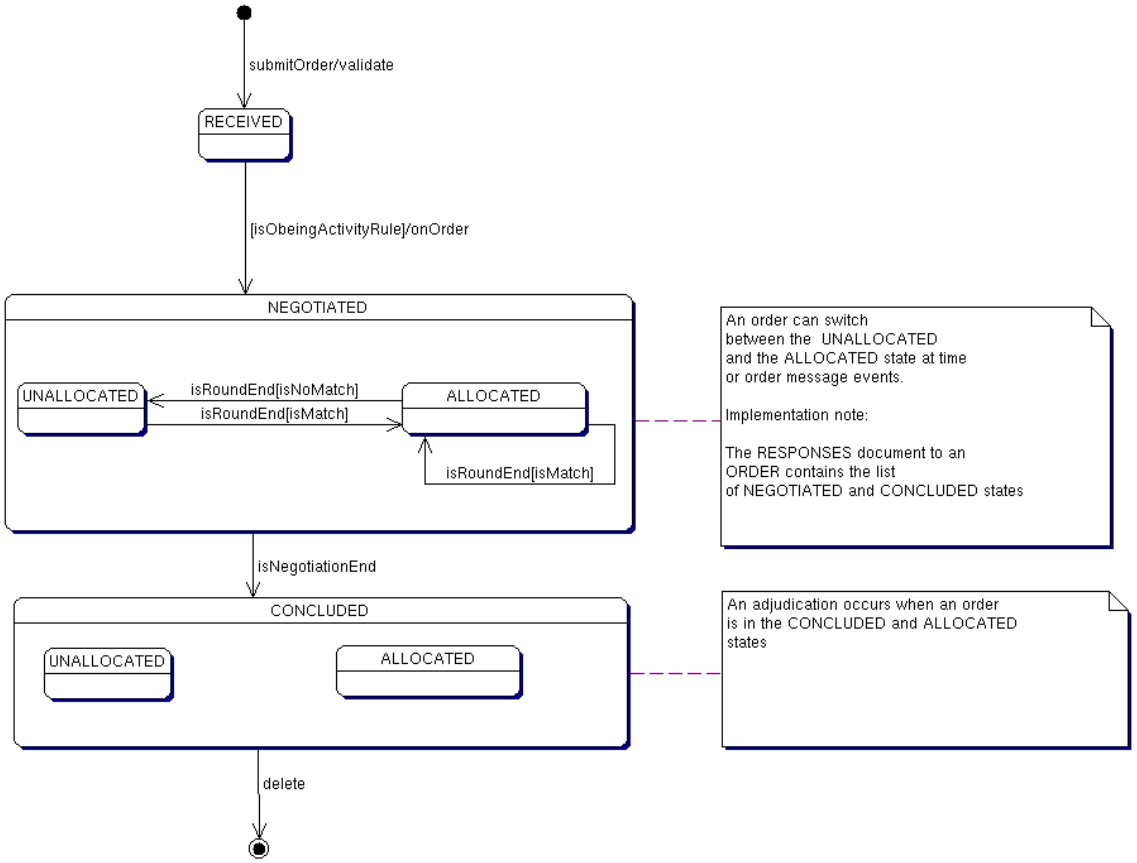


5.2.11. États majeurs des entités

a) État des négociations



b) États des ordres



5.3. Les règles économiques

Les règles économiques sont contenues dans des scripts. Les règles peuvent être écrites en JPython ou en Java. Chacune de ces classes de script est définie dans un fichier dont l'adresse sera indiquée dans le document de règles “rules”. Son nom n'est pas requis mais une instance de celle-ci doit être créer sous un nom précis, reconnu par GNP. À titre d'exemple, la classe auctionneer peut être nommée de manière indifférente, mais doit hériter de la classe Java : Auctionneer. À la fin de ce fichier, une instance de cette classe doit être créer impérativement sous le nom : __auctioneer__. Chacune des méthodes de l'instance __auctionneer__ définies dans ce fichier vont être appelées suite à certains événements, les lignes de code JPython définies dans les méthodes seront alors exécutées.

Les arguments des méthodes sont les documents tels qu'ils existent au moment de l'appel de la méthode, lorsqu'il s'agit de documents d'entrée, ils sont construits manuellement ou par le biais de l'interface usager, tel que décidé a priori par le scripteur. Lorsqu'il s'agit de <negotiation/ ou des documents de sortie, ils sont vides lors de leur premier appel pour être ensuite construits par les méthodes successives elles-mêmes (en fonction des choix du scripteur). La construction d'un document et de son arborescence se fait essentiellement par les classes Java : Document, Element (et Node) définies par W3C et la classe DOMUtil, localement définie. La documentation de ces classes se trouve sur : [http://edmond.CIRANO.UMontreal.CA/~eree/devel/xmldocs/], [http://edmond.CIRANO.UMontreal.CA/~eree/devel/gnp/javadoc/cirano/gnp/DOMUtil.html]

Les scripts doivent fournir quatre types d'objets :

5.3.1. Le contrôleur de marché – « MarketControler »

Le système GNP a un contrôleur de marché par entité de marché. Les événements de marché lui sont envoyés : « onStart », « onClose », « onSuspend », « onResume ».

5.3.2. Le validateur d'annonce – « AnnounceValidator »

Ce script contient une seule méthode appelée lors de la construction et la validation d'une annonce.

5.3.3. Le validateur d'ordre - « OrderValidator »

Ce script contient une seule méthode appelée lors de la construction et la validation d'un ordre.

5.3.4. L'encanteur – « Auctioneer »

Le système GNP a une instance d'encanteur par type de négociation activé. Les événements d'une enchère appellent les méthodes sur cet objet : « onOrder », « onTimeOut », « onRoundEnd », « onPhaseEnd » sont parmi les plus importantes.

?? *Les principales méthodes de la classe auctionneer*

onInit (negotiation, rules)	appelée après la création d'une annonce
onProductRef (negotiation, rules, productReference, quote)	appelée après onInit, si un document productReference existe dans l'annonce appelée après la création d'un document ProductReference
onOrder (negotiation, rules, order, response)	appelée après onProductRef sur réception de order contenu dans l'annonce appelée après chaque order pendant la négociation
onRoundEnd (negotiation, rules, response, quote, adjudication)	appelée si isRoundEnd retourne 1 (Cette méthode est appellée conditionnellement au résultat de la méthode isRoundEnd, voir 3-2)
onTimeout (negotiation, rules, java-lang-string)	appelée sur un message du timer à une heure précisée antérieurement

À ces méthodes, s'ajoutent principalement onStartPhase et onClosePhase, qui marquent respectivement le démarrage et la clôture d'une phase de la négociation (onStartPhase et onClosePhase sont appelées conditionnellement à canStartPhase et canClosePhase)

?? *Les séquences des appels de méthodes*

Les différentes méthodes sont appelées en séquence à la suite de 3 événements :
- la réception d'une annonce en début de négociation
- la réception d'un ordre (l'ordre initial de l'annonce ou/et les mises envoyées par les participants en

cours de négociation)
- la réception d'un message du Timer indiquant une durée écoulée ou l'échéance d'une date fixe.

La séquence des méthodes est conditionnée par certaines méthodes qui peuvent être appelées ou ne pas l'être. En particulier, les méthodes onStartPhase, onRoundEnd et onClosePhase sont appelées conditionnellement au booléen (0 ou 1) retourné par les méthodes respectives canStartPhase, isRoundEnd et canClosePhase.

<table>
<tr><td colspan="4">Réception d'une annonce</td><td>Fermeture</td></tr>
<tr>
<td>- onInit(nego,rules)
- canStartPhase --1
- onStartPhase(nego,rules)
- onProductRef(nego, rules,quote,productRef)

- onOrder(nego,rules, response,quote)

- isRoundEnd -- 1
- onRoundEnd(
- canClosePhase -- 0</td>
<td>onInit
- canStartPhase -- 0

- onProductRef
- canStartPhase --1
- onStartPhase
- onOrder

- isRoundEnd -- 1
- onRoundEnd
- canClosePhase -- 0</td>
<td>- onInit
- canStartPhase -- 0

- onProductRef
- canStartPhase -- 0

- onOrder
- canStartPhase --1
- onStartPhase
- isRoundEnd -- 1
- onRoundEnd
- canClosePhase -- 0</td>
<td>- onInit
- canStartPhase -- 0

- onProductRef
- canStartPhase -- 0

- onOrder
- canStartPhase -- 0

- canClosePhase -- 0</td>
<td>- canClosePhase -- 1
- onClosePhase</td>
</tr>
<tr><td colspan="4">Réception d'une mise</td><td></td></tr>
<tr>
<td>- onOrder
- isRoundEnd --1
- onRoundEnd
- canClosePhase -- 0</td>
<td>- onOrder
- isRoundEnd -- 0

- canClosePhase -- 0</td>
<td>-onOrder</td>
<td></td>
<td>- canClosePhase -- 1
- onClosePhase</td>
</tr>
<tr><td colspan="4">Réception d'un message du Timer</td><td></td></tr>
</table>

- onTimeout - isRoundEnd --1 - onRoundEnd - canClosePhase --0	onTimeout - isRoundEnd --0 - canClosePhase --0		-onTimeout - canStartPhase --1 - onStartPhase	- canClosePhase -- 1 - onClosePhase

?? *Exemple*

Un exemple du script "auctionneer" dans lequel sont définies les méthodes en python est donné en annexe. Cet exemple montre toutes les méthodes appelées dans GNP.

5.4. Interactions du système GNP

La section précédante a présenté les interactions entre les documents et les scripts économiques. Maintenant, illustrons les interactions au niveau des composantes informatiques de GNP.

Les composantes obtiennent des références et interagissent dans cet ordre :

?? Les pages « JSP »;

?? Les « Session Beans », ReaderSession, AnnouncerSession et NegotiatorSession;

?? La queue de message asynchrone « JMS »;

?? La phase d'évaluation de l'encan et l'encanteur réparti entre l'aiguilleur, « Auctioneer dispatcher », la façade générique « Auctioneer » et la façade d'encanteur Python « PythonAuctionner », et enfin l'encanteur lui-même

?? Les données gérées par les « EntityBean » contenants les documents XML;

?? Puis, dans une version future, le distributeur d'événements de négociation à un groupe de participant. Ce distributeur utilisera le modèle « publish and subscribe » de JMS.

5.4.1. Diagramme de collaboration et de séquence.

Trois diagrammes de séquence illustre la dynamique d'une enchère, du cas simple aux cas complexes.

Interactions entre les composantes	http://comte.CIRANO.UMontreal.CA/~eree/devel/gnp/tjdoc/doc-files/rgl_aRound_InteractionDiagram.html
Soumettre un ordre (cas simple)	http://comte.CIRANO.UMontreal.CA/~eree/devel/gnp/tjdoc/doc-files/rgl_submitOrder_InteractionDiagram.html
Soumettre une	http://comte.CIRANO.UMontreal.CA/~eree/devel/gnp/tjdoc/doc-

annonce	files/rgl_annonce_InteractionDiagram.html

5.5. Modèle de données

Le modèle des entités EJB et des documents XML est traduit en tables SQL selon ces patrons. Chaque type de document XML GNP a une table correspondante, et la clef primaire est un compteur unique appelé «Generic Primary Key » ou GPK. Un GPK est un entier long (64 bits) construit par la concaténation d'une partie haute (48 bits) et d'une partie basse (16 bits). La partie haute est obtenue et incrémentée à chaque démarrage du serveur, la partie basse est incrémentée à la création de chaque nouvel enregistrement. De plus le temps de création de chaque enregistrement est mis dans la table.

Voici les 7 tables contenant les données du système GNP. Chaque document XML dans les tables est de type « gnpDocument », avec au moins deux attributs obligatoires pour tous les documents: la GPK et la date de création.

Nom de la table SQL	Description	**Type de document XML attendu (Ces fragments sont sous la balise commune GNPDocument)**
GPK	Prochaine valeur la partie haute de la clef.	**aucun**
ANNOUNCES		**announce**
MARKETS		**market**
NEGOTIATIONS		**negotiation**
JOURNAL		
PARTIES		**party**
PRODREFS		**productReference**
QUOTES		**quote**
ORDERS		**order**
ADJUDICATIONS		**adjudication**
RESPONSES		**response**
RULES		**rules**

Dans cette première version, les enregistrements contiennent presque uniquement la clef GPK et le document XML mis dans un champ SQL de type varchar. De plus des procédures d'extraction automatique des attributs XML vers des colonnes de tables « étendues » ont été faites en PL/SQL sur Oracle pour permettre des requêtes arbitre et rapidement sur le contenu des documents XML.

Voir le fichier gnp/sql/postgres.sql définissant ces tables.

5.6. Sous-système d'identification de contrôle d'accès

GNP utilise le sous-système d'identification et de sécurité fournit par BEA-Weblogic, voir la documentation sur les royaumes « Realms ». Cette version de GNP utilise le gestionnaire de royaume pour base de données « RDBMSRealm ». Les noms d'utilisateurs, mots de passe et listes de contrôle d'accès sont tous conservés dans la base de données SQL. Ce modèle d'opération est simple et convient bien au démarrage. Une version future de GNP pourra utiliser le gestionnaire de royaume basé sur LDAP fournit par BEA-Weblogic. Celui-ci gère les contrôles d'accès à partir des données fournit par un serveur LDAP externe.

Consultez le fichier gnp/sql/realm_postgres.sql décrivant ces tables. Le « package » cirano.util.realm contient les classes Java qui interface le gestionnaire de royaume RDBMSRealm.

5.7. Interaction des participants

Les interactions avec les participants sont réalisées avec pages JSP. Le patron « Model-View-Controler » est utilisé. Les sessions « EJB » instancient la partie modèle de ce patron. Les pages et les « taglib » JSP font la partie de visualisation. Le contrôle est fournit par le contrôle d'activation des pages JSP.

Toutes les pages JSP sont sous gnp/pages, un répertoire par modèle de négociation.

5.7.1. Taglib

Les Taglib définies dans les spécifications de « Java Server Page », V1.1, sont utilisés dans GNP 1.0 et servent à définir les composantes des interactions génériques.

Consultez le package « cirano.jsp.taglib »

5.7.2. Formulaires de saisie : GNPXForm

Le système GNP v1.0 utilise un système de formulaire très simplifié, GNPXForm, basé sur le futur standard Xforms du W3C. Les systèmes de formulaires disponibles sur le marché par les compagnie JetFrom et autres se sont avérés beaucoup trop lourd et coûteux en temps pour être utilent dans une phase d'expérimentation et d'essais pilotes. Les formulaires de saisie sont pour les essais pilotes encodés par SSL sur https, guarantissant ainsi la confidentialité de l'information lors de son transit sur le réseau. L'identification de l'utilisateur de la session au préalable par un mot de passe encodé et l'envoie des mises par https permet un niveau premier de confiance dans la sécurité du système. Ce niveau de confiance pourra être augmenté par l'utilisation du système à clef publique et la signature des messages qui sera déployé dans les places de marchés gouvernementales. Le système GNP pourra utiliser ces systèmes s'ils sont présents.

?? *Validation de formulaires de saisie HTML : gnpXForm*

gnpXForm est un sous-ensemble très restreint du futur standard XForms du w3c. Plusieurs éléments n'y figurent pas : les booléens, les choix parmis des options et les valeurs par défaut, entre autre. Ces éléments ont été omis pour garder l'implantation de gnpXForm la plus simple possible et parce ils sont implicitement présents dans les formulaires HTML (il est possible de spécifier des valeurs par défaut, des énumérations... Ces valeurs sont donc "pré validées" par le fureteur.)

Voici les types reconnus et leurs attributs

number	**Ce type permet de s'assurer qu'un element est de type numérique (entier ou décimal).**	
Nom de l'attribut	Description :	Valeur par défaut
min	valeur minimale du nombre	0
max	valeur maximale du nombre	plus l'infini
decimals	nombre de décimales significatives. Une valeur de 0 implique le nombre est un entier	0
date	**Ce type permet de s'assurer qu'un element est de type date tel que supporté par la classe java.text.DateFormat selon la "locale" spécifiée.**	
Nom de l'attribut	Description :	Défaut
min	valeur minimale de la date ou NOW (qui est la date au moment de la validation du formulaire)	très loin dans le passé
max	valeur maximale de la date	très loin dans le futur
locale	Locale à utiliser (format iso. Par exemple, fr_CA, en_US...)	en_US
duration	**Ce type spécifie une durée temporelle**	
Attributs :	Description :	Nom
unit	DAYS, SECONDS, MINUTES, HOURS,	MINUTES

?? *Exemple d'utilisation*

Voici un exemple de document gnpXForm permettant de valider un formulaire HTML et qui affiche le DOM résultant.

Document à produire

Le document à produire doit être de la forme :

```
    <gnpDocument>
        <price>??</price>
        <quantity>??</quantity>
        <expiration>??</expiration>
    </gnpDocument>
```

Les ?? dénotent les champs à remplir. Nous désirons que le prix soit plus grand ou égal à 5, la quantité soit entre 100 et 200 (inclusivement).

?? *Document de validation gnpXForm*

```
<gnpXForm>
    <number name="/gnpDocument/price">
        <min>5</min>
        <decimals>2</decimals>
    </number>
    <number name="/gnpDocument/quantity">
        <min>100</min>
        <max>200</max>
        <decimals>0</decimals>
    </number>
    <date name="/gnpDocument/expiration">
    </date>
</gnpXForm>
```

?? *La sélection du "bid domain" à utiliser se fait de cette façon :*

Si le OrderResponse existe et qu'il contient un "bid domain", on l'utilise.
Sinon, si le PartyResponse existe et qu'il contient un "bid domain", on l'utilise.
Sinon, utilise le "bid domain" contenu dans la Quote la plus récente

?? *Formulaire HTML complet :*

```
<FORM NAME="A_Form" METHOD=POST ACTION="urlToInvoke.jsp">
  <P>Price
  <INPUT TYPE="text" NAME="/gnpDocument/price" SIZE="10" VALUE="5">
  <P>Quantity
   <INPUT TYPE="text" NAME="/gnpDocument/quantity" SIZE="10"
VALUE="100">
   <INPUT TYPE="text" NAME="/gnpDocument/name" SIZE="10">
   <INPUT TYPE="hidden" NAME="gnpXForm"
VALUE="&lt;gnpXForm&gt;&lt;numbername=&quot;/gnpDocument/price&quot;&
gt;&lt;min&gt;5&lt;/min&gt;&lt;decimals&gt;2&lt;/decimals&gt;&lt;/num
ber&gt;&lt;numbername=&quot;/gnpDocument/quantity&quot;&gt;&lt;min&gt
;100&lt;/min&gt;&lt;max&gt;200&lt;/max&gt;&lt;decimals&gt;0&lt;/decim
```

```
als&gt;&lt;/number&gt;&lt;datename=&quot;/gnpDocument/expiration&quot
;&gt;&lt;/date&gt;&lt;/gnpXForm&gt;">
  <INPUT TYPE="submit" VALUE="Submit">
</FORM>
```

?? *Ajout de nouveaux types*

Il est possible d'ajouter de nouveaux types qui "dérivent" les types de base. Pour l'instant, uniquement l'héritage simple à un niveau est supporté. Par exemple, il est possible de définir un type "price" qui dérive le type de base "number" avec les attributs min=0 et decimals=2. Lors de la spécification d'un nouveau type, si un attribut n'est pas spécifié, la valeur par défaut de l'attribut du type parent est utilisée. De plus, il est présentement impossible d'ajouter un nouvel attribut. Le fichier "GNPXFormsTypes.xml" permet la configuration des types. Voici un exemple de spécification de nouveaux types qui pourraient être utilisés dans une négotiations "hypothétique", où un prix est valide entre 0 et 100000 (sans décimales) et la quantité est fixée à 1.

```
<gnpXFormsTypes>
    <type name="hypoprice" extends="number">
        <attribute name="decimals" value="0"/>
        <attribute name="max" value="100000"/>
    </type>
    <type name="hypoqty" extends="number">
        <attribute name="decimals" value="0"/>
        <attribute name="min" value="1"/>
        <attribute name="max" value="1"/>
    </type>
</gnpXFormsTypes>
```

Un formulaire de validation gnpXForms utilisant ces nouveaux types serait :

```
<gnpXForm>
    <hypoprice name="/gnpDocument/price">
        <min>100</min>
    </hypoprice>
    <hypoqty name="/gnpDocument/quantity"/>
</gnpXForm>
```

5.7.3. Interfaces de programmation

?? *Validation d'un formulaire*

La méthode statique "cirano.gno.GNPXForms.isValid(Document htmlForm, Document validationForm)" permet de valider facilement les données reçues d'un formulaire HTML (auparavant converties en document XML en utilisant les services de "cirano.gnp.jsp.FormTool" pour restructurer le DOM à partir de la liste "plate" d'attributs HTML) avec un DOM contenant le formulaire de validation.

?? *Gestion des erreurs et codes de retour*

La méthode isValid lance une exception de type cirano.gnp.GNPXFormsException s'il y a une ou plusieurs règles de violées (ie: si le formulaire HTML soumit est invalide). L'exception lancée offre des méthodes permettant de connaitre quels champs sont invalides et la raison de leur invalidité.

Une façon simple d'utiliser cette capacité dans des pages JSP est de définir une "error page" et de gérer cette exception à partir de l'error page. Une approche de gestion

de l'erreur serait de réafficher le formulaire HTML avec les valeurs erronnées dans une autre couleur.

Voir la javadoc de la classe cirano.gnp.GNPXFormsException pour plus d'informations.

5.8. Interactions de l'administrateur

Les interactions de l'administrateur sont faites par des pages HTML et JSP. Ces pages sont sous le répertoire gnp/pages/admin.

6. Architecture de déploiement

[Décrivez les processus déployés. Décrivez le lien avec les systèmes matériels et les périphériques.]

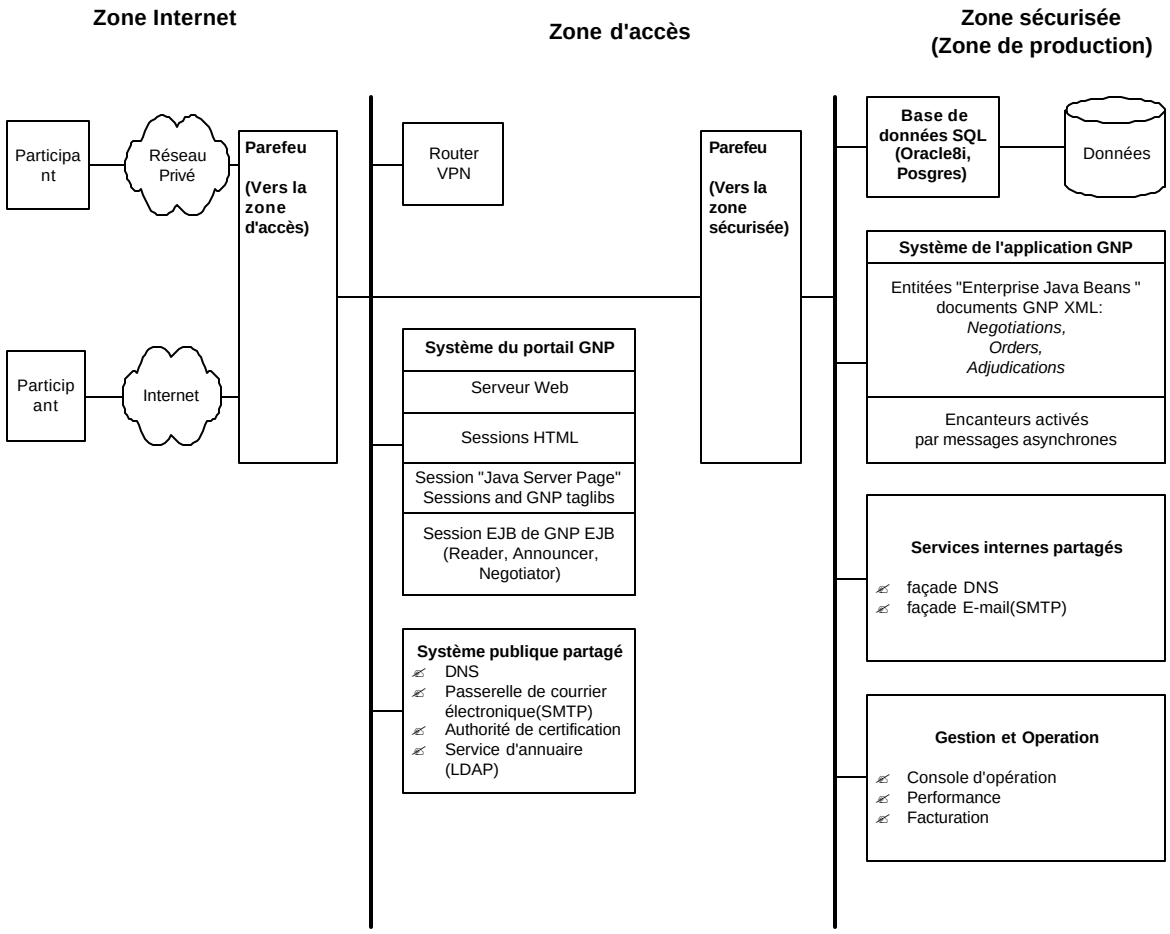


Diagramme : Architecture de déploiement

6.1. Configuration du matériel

En mode minimal, suffisant pour les tests et les essais pilotes, un seul serveur matériel avec 2 processeurs peut exécuter les logiciels suivant sur 3 processus différents: les portail d'accès HTTP avec des pages JSP; application Enterprise Java Bean Weblogic et enfin le serveur SQL.

6.1.1. Serveur GNP

serveur Sun *Enterprise 250* , 2 processeur UltraSparc 350 Mhz et plus

ou serveur Linux RedHat, 2 processeurs Intel 500 Mhz et plus

mémoire vive 512 Mo

2 disques 4 Go

dérouleur de bande (facultatif)

console

carte réseau Éthernet

6.1.2. Réseau

lignes 1-800 ou réseau Internet privé (pour l`essai pilote)

Accès par modem 28.8 au minimum (56Kbs recommendé)

Serveur de terminal PPP, avec configuration dynamique DHCP

Double garde-barrière (2 routeurs) * (pas nécessaire pour l'essai pilote)

Accès Internet pour le serveur GNP

6.2. Configuration du logiciel

6.2.1. Serveur GNP

Logiciel Java développé par l'équipe TEM du CIRANO

Linux Redhat, 2.2 sur Intel ou Solaris 8, version serveur sur Sparc

Java 1.3, en mode serveur

Weblogic Server 5.1, voir http://www.bea.com

6.2.2. Serveur SQL

Oracle 8i (préférré), ou PosGRES 7.0, au moins 8 connexions simultanées,

sur la même machine que le serveur GNP (pour les essai pilote) ou distant

6.2.3. Serveur HTTP et JSP pour le portail d`accès

Bea-Weblogic 5.1, voir http://www.weblogic.com

6.2.4. Gestion commune des usagers

- Sur la base de données SQL (par défaut)

- Netscape Directory Server V3.0 (LDAP V3)

6.3. Services connexes requis

Service de temps réseau NTP fiable, synchronisé sur une horloge "atomique"

Hébergement de page Web « minimale » pour les participants

Service de courrier électronique SMTP de et vers l`Internet

Compte POP3 ou IMAP pour les participants et le gestionnaire du marché

Serveur de nom, DNS

7. Tests et qualité

[Identifiez les outils et la méthodologie utilisés pour réaliser les tests de qualité.]

Pour automatiser ces tests, nous avons développé des robots en Java et JPython qui exécutent les scénarios de tests. Un scénario est un « succès » si les robots dans les scénarios valident une ou plusieurs assertions. Une assertion peut autant être la présence d'une condition ou d'un événement ou au contraire l'absence d'une condition ou d'un événement.

7.1. Introduction

Ce document décrit l'environnement de test de GNP base sur trois composantes:

?? *Language de description de scenarios de test*

?? *Robots de test*

?? *Controleur de coordination*

L'objectif recherche est de pouvoir soumettre a GNP des séquences de test écrites dans un language formel (éventuellement générées par un outil basé sur ce language) et de pouvoir traduire ces sequences en un script executable.

La suite de ce document permettra de décrire et de montrer le role de chaque element de cet environnement.

7.2. Language de description des scénarios

Parmi les alternatives disponibles, le diagramme de sequence de messages (MSC) a été retenu. Contrairement aux autres languages de specification et de description (comme SDL) qui mettent l'accent sur la specification interne des composants d'un systeme, MSC met plutot l'accent sur l'aspect communication entre les differentes composantes du systeme par le biais d'echange de messages.

MSC a été standardisé en 86, 92, 96 et finalement en 2000. En comparant juste les deux derniers standards, l'apport essentiel a la version 2000 est le support des donnees. Bien que la notion des donnees existait deja dans le standard precedent, celle ci n'avait aucune signification semantique. Autrement dit, les parametres d'un message, Mess(param1, …, paramn), ne constituent qu'une extension du nom de ce message. Les choses ont donc evolue avec le standard 2000.

Afin de permettre a MSC d'etre utilise avec un language de donnees au choix , aucun language de donnees specifique n'as été retenu. Ainsi, le language de donnees est un parametre pour MSC que l'on defini au niveau du MSCDOCUMENT par les deux mots cles :

language <language_de_donnees>;

data <definition_de_donnees>;

Par exemple, on aura dans un document MSCDOCUMENT les declarations suivantes :

```
MSCDOCUMENT nom_mscdocument
```

```
LANGUAGE    C;

DATA              #Include mes_c_defs.h;

..

MSC   nom_msc;

..

ENDMSC   nom_msc;

ENDMSCDOCUMENT;
```

Tous les types de variables utilises dans les differentes declarations a l'interieur d'un document MSCDOCUMENT doivent etre declares dans «mes_c_defs.h » conformement au language choisi, qui est la language C dans l'exemple precedent. C'est cette convention qui sera retenu dans la suite de ce document.

Le standard actuel du language MSC a ete enrichi pour supporter plusieurs nouvelles possibilites, qui etaient des limitations dans les precedents standards. Cependant, et pour pouvoir decrire les scenarios conformement aux objectifs de test de GNP, un sous ensemble de ce language sera suffisant. Afin de degager ce sous-ensemble, nous allons decrire un scénario representatif de l'ensemble des scenarios de test, et nous allons nous limiter aux mots-cles apparaisant dans ce modele, quitte a en ajouter d'autres si de nouveaux besoins surgissent.

7.3. Contrôleurs et robots de test

?? **Robots de test:**

Chacun de ces robots jouera le role d'un acteur externe a GNP. Chaque scenario de test sera joué par un ou plusieurs robots . Le role de ces acteurs consistera à stimuler le comportement des composants de GNP à tester. Ainsi, chaque fois qu'un composant GNP (AdministratorSession, Announcersession, NegotiatorSession ou ReaderSession) recoi un message de l'ntité ENV ou lui envoi un message, c'est avec un acteur externe qu'il est entrain d'interagir.

Le mot ROBOT prend toute sa signifiacation lors d'une execution distibuee des tests

?? **CONTROLEUR DE COORDINATION:**

Le role du controleur et de coordonner les taches des differents robots et de reporter les resultats de test. Ainsi, lors d'une execution sequentielle, les resultats sont reportes au fur et a mesure; alors que pendant une execution distribuee, le controleur attend la fin de l'execution de chaque robot avant de reporter ses resultats.

7.4. Exemple d'un scénario modèle

```
MSCDOCUMENT     testGNP;

MSC   scenario01;
```

```
MSG   getTemplates;

MSG   submitAnnounce(charstring);

MSG   submitOrder(long, long, long, charstring);

MSG   getMyOrders;

MSG   getResponsesDocumentForOrder(long);

MSG   getOrdersForProductReference(long);

MSG   updateOrder(long, long, long, charstring);

MSG   getOrder(long);

MSG   deleteOrder(long);

MSG   getLastQuoteForNegotiation(long);

MSG   getNegotiationDocument(long);

MSG   getNegotiations;

MSG   getNextGPK;

MSG   getQuoteDocument(long);

MSG   getQuotes;


INS   ann01: PROCESS    AnnouncerSession;

INS   neg01: PROCESS    NegotiatorSession;

INS   neg02: PROCESS    NegotiatorSession;

INS   rdr01: PROCESS    ReaderSession;


COMMENT Les sequences  suivantes ont pour rôle de tester quelques messages
          GNP;


ann01: IN getTemplates() from    ENV;

       OUT tmpltRes   to      ENV;

       CONDITION TypeMatch(tmpltres, "Document");


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

```
       OUT annRes     to      ENV;

       CONDITION TypeMatch(annRes, "Document");



   neg01:  IN submitOrder(productReferenceGPK(annRes), negoGPK(annRes),
   previousQuoteGPK(annRes), "badOrder.xml") from ENV;

          OUT resultSO1 to ENV;

        CONDITION TypeOf(resultSO1, "GNPException");


neg02:  IN submitOrder(productReferenceGPK(annRes), negoGPK(annRes),
                       previousQuoteGPK(annRes), "goodOrder.xml") from ENV;

        OUT resultSO2 to ENV;

        CONDITION TypeMatch(resultSO2, "Message");


        IN  getMyOrders()     FROM    ENV;

        OUT resultGMO   to    ENV;

        CONDITION typeMatch(resultGMO, " Document");


rdr01:  IN getLastQuoteForNegotiation(negoGPK(annRes)) FROM ENV;

        OUT resultGLQFN       to    ENV;

        CONDITION  typeMatch(resultGLQFN, "Quote");

        IN getQuoteDocument( previousQuoteGPK(annRes)) FROM ENV;

        OUT resultGQD   to    ENV;

        CONDITION typeMatch( resultGQD, "Document");


ann01: ENDINSTANCE;

neg01: ENDINSTANCE;

rdr01: ENDINSTANCE;

neg02: ENDINSTANCE;
```

```
ENDMSC  test_annoucerSession;

ENDMSCDOCUMENT test_GNP;
```

Comme cela apparaît sur ce scénario modèle, les taches séquentielles a exécuter par un scénario sont groupées par bloc de trois actions :

Action No 1 :

Identifiée par le mot-clé **IN.** Elle consiste a recevoir un message a partir d'un robot, representé ici par le mot-clé **ENV**. Ce message provoquera la reaction du composant GNP a tester, en executant sa methode ayant le meme nom que celui du messge. Dans le permier cas de test de l'exemple precedent, le composant a stimuler est l'annouceurSession, et la methode a executer est getTemplates().

L'ensemble de tous les messages a utiliser dans les differents scenarios de test est donc limite et bien connu. Le recensement de ces messages permettera d'eviter l'usage de messages non comprehensibles par GNP. La liste exhaustive de ces messages sera donnee en annexe de ce document.

Action No 2 :

Identifiee par le mot-clé **OUT.** Elle consiste à récupérer le resultat produit par GNP suite a la stimulation précédente. Il est a noter que ce resultat peut bien etre une exception. Cependant, cela ne veut pas dire pour autant que l'execution du test a echoue.

Action No 3 :

Identifiée par le mot-clé **CONDITION**. Permet de vérifier la **conformité** entre le résultat retournée par GNP lors de l'action No 2 et celui auquel on s'attend dans un fonctionnement correct. Le resultat de cette comparaison permettera de conclure le verdict de test.

Il est a noter, cependant, que les actions 2 et 3 sont optionnelles. On a recours a l'action 3 (CONDITION) pour verifier que le resultat retourne par l'action 2 (OUT) est conforme a certaines conditions. Si par contre l'execution d'un message est juste necessaire pour l'execution des restes des messages, on peut ne pas utiliser l'action 3 avec eventuellement l'action 2. Afin d'illustrer ceci par un exemple, prenons le cas d'un scenario dont l'objectif est de tester le negotiatorSession. Pour ce test, la soumission d'une annonce est necessaire. Cependant, verifier le resultat retourne n'est pas utile. Donc, on peut avoir la sequence de test suivante:

```
announcer:          IN  submitAnnounce(..)  FROM  ENV

                    OUT  resultSA   TO ENV


negotiator:         IN submitOrder(....)  FROM   ENV

                    OUT  resultSO   to ENV

                    CONDITION (resultSO, "Message")
```

...etc

Partant de ces exemples, le sous ensemble du language MSC dont on aura besoin pour la description des differents scenarios de test est le suivant :

COMMENT	***CONDITION***	***ENDINSTANCE ENDMSC***
	ENDMSCDOCUMENT	***ENV***
FROM	***IN***	***INST***
MSC	***MSCDOCUMENT***	***MSG***
OUT	*PROCESS*	*TO*

7.5. Traduction de MSC vers jpython

Le traducteur de scenarios est un programme (script) jpython qui traduit un scenario ecrit en MSC en un script jpython equivalent et executable. C'est ce script qui executera les sequences de test decrites dans le fichier MSC.

En plus de la traduction, le traducteur verifie que le fichier recu en input est conforme au formalisme MSC. Ainsi, il permet, entre autre, de s'assurer que :

L'ordre des clauses MSC est bien respecte

Une instance ne peut etre utilisee pour echanger des messages si celle-ci n'a pas été declaree au prealable

Une meme instance ne peut etre declaree plus d'une fois

Chaque instance declaree doit avoir une clause ENDINSTANCE correspondante

Chaque MSC declaree au sein d'un MSCDOCUMENT doit avoir une clause ENDMSC correspondante avant une eventuelle clause MSC ou avant la clause ENDMSCDOCUMENT

Toutes les instances declarees dans un MSC doivent avoir une clause ENDINSTANCE avant la clause ENDMSC correpondante au MSC dans lequel elles ont été declarees.

Tous les messages utilises dans le corps des sequences de test doivent etre declares au prealables avec leurs parametres eventuels.

Il est egalement possible d'enrichir ces scenarios par des commentaires explicatifs a l'aide de la clause MSC « COMMENT » qui seront traduits en des commentaires equivalents en jpython.

Pour que le script genere soit compatible avec GNP, il faut respecter certaines regles lors de l'ecriture des scenarios MSC.

Ainsi, pour chaque **methode** des differentes session de GNP a tester a savoir AdministratorSession, AnnoucnerSession, NegotiatorSession et ReaderSession, il faut utiliser au niveau MSC un **message** avec le **meme nom** de la methode. Si la methode a des **parametres**, le message doit en avoir autant. Le **nom de la sessionbean** est declare en MSC comme etant le **type d'une instance** MSC.

ex :

```
MSCDOCUMENT          testGNP;

MSC      scenario01;

MSG      submitAnnounce(charstring);

INS        announcer :  INSTANCE    AnnouncerSession;


announcer : IN    submitAnnounce( "announce.xml")  FROM  ENV;

            OUT    annResult  to ENV;

            CONDITION   TypeMatch(annResult,"Document");

announcer: ENDINSTANCE;

ENDMSC      scenario01;

ENDMSCDOUMENT testGNP;
```

Ce scenario a pour objectif de tester la methode **submitAnnounce()** de la session **AnnouncerSession**.

Il existe actuellement trois fonctions (**perdicats**) permettant de conclure le verdict du test:

TypeOf: est utilisee dans le cas ou l'on s'attend a recevoir une exception. Dans ce cas, le nom de l'exception attendue est transmis comme parametre (string) a la fonction.

ex: **TypeOf(result, "GNPException")**

TypeMatch: est utilisee dans les autres cas pour verifier le type de l'objet retourne suite a l'appel de la methode a tester. Le type attendu est transmis egallement en parametre a la fonction.

ex: TypeMatch(result, "Document")

Leader: est utilisee pour connaitre l'identite de joueur ayant "gagne" l'enchere. Elle peut avoir plusieurs formes selon les cas:

ex: leader(annResult, nego01), ou leader(annResult, nego01, 20, 50)

avec 20 et 50 representant la quantite alloue a nego01 et le prix correpondant lors de l'adjudication .

Lorsequ'une methode a tester possede des parametres dont la valeur ne peut etre determinee que pendant l'execution, comme le negoGPK par exemple, ces parametres sont remplaces par des fonctions qui portent le meme nom que celui du parametre et dont le role est de fournir la valeur effective du parametre pendant l'execution. Dans certains cas, il faut preciser a quel instance cette fonction-parametre va-t-elle etre appliquee. soit l'exemple suivant:

```
ann01: IN submitAnnounce("announceDoc1.xml")  from ENV

       OUT ann1Res  to ENV

       CONDITION ...

ann02: IN submitAnnounce("announceDoc2.xml")  from ENV

       OUT ann2Res  to ENV

       CONDITION ...

neg01: IN submitOrder(productReferenceGPK(ann2Res), ...)
```

La precision de l'argument **ann2Res** pour la fonction **productReferenceGPK(),** permet de retourner la valeur du productReferenceGPK correspondant a la deuxieme annonce etant donne que **ann2Res** est le document resultat de la deuxieme annonce.

La liste des (fonctions) parametres a utiliser dans les divers appels est la suivante (Peut etre amenee a evoluer):

Nom de la fonction-parametre	retour	arg optionnel
marketName()	string	sans arg
negoGPK(annResult)	long	oui
oldOrderGPK(annResult)	long	oui
orderGPK(annResult)	long	oui
playerName()	string	sang arg
previousQuoteGPK(annResult)	long	oui
productReferenceGPK(annResult)	long	oui
quoteGPK(annResult)	long	oui

En résumé, l'ecriture en MSC suivante:

```
negotiator01: in submitOrder(productReferenceGPK(),negoGPK(),
                                 previousQuoteGPK(),
                                 Order("order.xml") )  FROM ENV
```

sera traduite en jpython par:

```
negotiator01.submitOrder(productReferenceGPK(), negoGPK(),
             previousQuoteGPK(), Order("order.xml"))
```

l'execution par jpython de cette fonction provoquera les appels successifs suivants:

?? *productReferenceGPK()*

?? *negoGPK()*

?? *previousQuoteGPK()*

?? *Order("order.xml")*

?? *negotiator01.submitOrder(long, long, long, string)*

Il est à noter que pour la fonction-argument Order(), elle peut recevoir deux autres arguments optionnels: le prix offert et la quantite demandee. Dans ce cas, un meme document (order) XML peut etre utilise pour soumettre plusieurs mises diferentes, le prix et la quantite etant adaptes a la volée.

7.6. Liste des messages supportés

Liste des messages auquels il faut se limiter lors de l'écriture des scenario en MSC

(Cette liste est amenee a changer au fur et a mesure que GNP évolue)

Messages pour l'instance NegotiatorSession :

```
submitOrder
getOrdersForProductReference
deleteOrder
getOrder
getMyOrders
updateOrder
```

getResponsesDocumentForOrder

getResponsesForOrder

Messages pour l'instance AdministratorSession :

setNegotiationState

purgeDeletedNegotiations

startMarket

submitMarketCommand

submitNegotiationCommand

createParty

findParty

findAllParties

getPartyStateBatch

createMarket

findMarket

findNegotiation

findAllMarkets

findAllNegotiations

findAllNegotiationsByMarket

Messages pour l'instance AnnouncerSession :

submitAnnounce

getTemplates

Messages pour l'instance ReaderSession :

getNextGPK

getQuotes

getQuoteDocument

getNegotiations

getNegotiationDocument

getLastQuoteForNegotiation