IAG

Working paper 71/02

Développement de patrons de conception à ancrage social pour architectures multi-agent

Thi Thuy Hang Hoang, Manuel Kolp

UCL
Université catholique de Louvain

IAG
Institut d'administration et de gestion

DEVELOPPEMENT DE PATRONS DE CONCEPTION A ANCRAGE SOCIAL POUR ARCHITECTURES MULTI-AGENT

Developing Social Design Patterns for Multi-Agent Architectures

Thi Thuy Hang HOANG **Manuel KOLP**

IAG – Institut d'administration et de gestion,
ISYS – Unité de recherche « Systèmes d'information »
Université catholique de Louvain,
1 Place des Doyens, Louvain-La-Neuve, Belgique
tél.: +32 10 47 83 95, e-mail : {hoang, kolp@isys.ucl.ac.be}

Table des matières

Résumé

La croissance explosive des domaines d'application tels que le commerce électronique, la gestion des connaissances, les systèmes coopératifs, l'informatique point-à-point, mobile ou cognitive change profondément et irréversiblement nos vues sur l'ingénierie logicielle et les systèmes eux-mêmes.

Ceux-ci doivent maintenant supporter des architectures ouvertes pouvant évoluer continuellement afin d'intégrer, modifier ou retirer des composants nouveaux ou existants devant répondre aux exigences et conditions changeantes de l'environnement. Pour ces raisons – et d'autres – le paradigme orienté agent gagne en popularité sur les techniques traditionnelles de développement logiciel, y compris celles dites structurées et orienté objet.

Ce genre de systèmes et technologies demande des méthodologies adaptées. Tropos est une méthodologie d'analyse et de conception de systèmes agent. Elle propose une ontologie socio-intentionnelle basée sur trois niveaux : 1) éléments atomiques de base, 2) patron de conception à ancrage social, 3) styles organisationnels.

Ce travail se focalise sur le deuxième niveau. Il identifie des patrons de conception à ancrage social à utiliser dans Tropos, en développe certains (appel d'offre, mise au enchères et courtier) sur l'environnement de programmation agent JACK en utilisant le modèle théorique BDI et en tenant compte des spécifications FIPA.

Abstract

The explosive growth of application areas such as electronic commerce, knowledge management, cooperative systems, peer-to-peer, mobile or cognitive computing has profoundly and irreversibly changed our views on software and Software Engineering.

Software must now be based on open architectures that continuously change and evolve to accommodate new components and meet new requirements. For these reasons – and more – agent-oriented software development is gaining popularity over traditional software development techniques, including structured and object-oriented ones.

This kind of software and technologies requires specific methodologies. Tropos is an analysis and design methodology for agent systems. It proposes a socio-intentional ontology based on three levels: 1) basic atomic elements, 2) social design patterns, 3) organizational styles.

This work focuses on the second level. It identifies social design patterns to be used in Tropos and develops some of them (call for offers, bidding and broker) within the JACK agent programming environment following BDI and FIPA specifications.

1. Introduction

La croissance explosive des domaines d'application tels que le commerce électronique, la gestion des connaissances, les systèmes coopératifs, l'informatique point-à-point, mobile ou cognitive change profondément et irréversiblement nos vues sur l'ingénierie logicielle et les systèmes eux-mêmes. Ceux-ci doivent correspondre d'une façon plus adéquate à leurs environnements opérationnels et organisationnels afin de pouvoir s'adapter à leurs changements dynamiques [1]. Pour cela, ils doivent supporter des architectures ouvertes pouvant évoluer continuellement afin d'intégrer, modifier ou retirer des composants nouveaux ou existants devant répondre aux exigences et conditions changeantes de l'environnement. Les notions d'autonomie, d'interaction sociale, de négociation, de planification et de décision deviennent essentielles à de tels composants logiciels. Les systèmes doivent également être fonctionnels sur différents types de plateforme logicielle ou matérielle, sans devoir être recompilés, en ne possédant qu'une information minimale quant à l'environnement opérationnel et ses utilisateurs. Ces nouvelles donnes, à leur tour, nécessitent de nouveaux concepts, outils et technologies pour la conception et la gestion de ce type de systèmes.

1.1. Systèmes multi-agent

Pour ces raisons – et d'autres – le paradigme orienté agent gagne en popularité sur les techniques traditionnelles de développement logiciel, y compris celles dites structurées et orienté objet. Après tout, les architectures basées agent – connues sous le nom de systèmes multi-agent dans la communauté agent – sous-tendent des structures logicielles ouvertes et évoluantes pouvant changer dynamiquement à l'exécution afin d'exploiter les services de nouveaux agents, ou remplacer ceux devenus sous-performants. En outre, les agents logiciels peuvent, en principe, faire face aux circonstances imprévues : ils incluent dans leurs buts architecturaux des capacités de planification destinées à adapter ceux-ci aux conditions opérationnelles du système. Pour finir, les technologies agent ont mûri au point où des protocoles pour la communication, la négociation, les architectures système ont été normalisés notamment par la FIPA (Foundation for Intelligent Physical Agents) [20].

1.2. La méthodologie Tropos

Ce genre de systèmes et technologies demande des méthodologies adaptées. Tropos [1] est une méthodologie, en développement, d'analyse et conception de systèmes multi-agent. Elle se compose de quatre phases classiques : 1) analyse de cahiers de charge, 2) analyse conceptuelle, 3) conception architecturale et 4) conception détaillée. Elle adopte des idées des technologies multi-agent – tel que AUML [19] –, essentiellement pour la phase de conception détaillée du processus de développement. Elle reprend également et surtout des idées de l'ingénierie des besoins où les notions d'agent et de but sont largement utilisées pour l'analyse de cahiers de charges [4, 5].

L'idée principale qui distingue Tropos est l'utilisation de notions à ancrage social et intentionnel comme les agents et les buts en tant que concepts fondamentaux durant toutes les phases du développement logiciel et pas simplement pour l'analyse de cahiers de charge [15]. Pour ce faire, Tropos

définit une ontologie agent originale pour formaliser les systèmes agent comme structures sociales et intentionnelles [14]. De ce fait, les comportements cognitifs des entités logicielles d'une architecture système considérées comme agents sociaux, la modélisation de leur dynamique de même que le contexte organisationnel dans lequel le système devra opérer, pourront être approchés de manière cohérente [7]. Cette approche étend les propositions de développement de système multi-agent [18] dans le sens ou ces systèmes peuvent être considérés comme des organisations sociales d'entités autonomes et intelligentes – les agents – interagissant les unes avec les autres pour l'accomplissement de buts communs [13].

L'ontologie de Tropos est décrite à trois niveaux de granularité. Au niveau le plus bas (granularité la plus fine), Tropos adopte les concepts atomiques (agent, but, but qualitatif, dépendance, croyance, confiance, désir, connaissance, …) offerts par les modèles sociaux et/ou intentionnels d'analyse de cahiers de charge tels que i* [6], KAOS [4], NFR [3] ou ConGolog [5]. A un second niveau, cette ontologie définit des patrons de conception [8] à ancrage social et intentionnel tels que le médiateur, le courtier ou l'ambassadeur [13]. A un troisième niveau plus abstrait, l'ontologie offre un ensemble de styles architecturaux [2] de type organisationnel inspirés par la théorie des organisations et les alliances stratégiques comme la structure en cinq, le partenariat, la fusion, la *joint-venture*, … [16].

Chaque niveau supérieur de l'ontologie utilise les concepts définis dans les niveaux plus bas. Ainsi les patrons de conception utilisent et agrégent les notions atomiques sociales et intentionnelles définies au niveau de base afin de spécifier comment accomplir un but particulier du système et identifier les agents, dépendances sociales, rôles et responsabilités nécessaires à sa réalisation. A un niveau plus macroscopique, les styles organisationnels décrivent l'architecture globale du système sur base des patrons de conception et des éléments atomiques définis. Un style organisationnel représente une façon de structurer les multiples agents d'un système – logiciel, social ou physique – vu comme organisation afin de mettre en œuvre et accomplir ses buts stratégiques.

1.3. Développer des patrons de conception agent

Le présent travail se concentre sur le second niveau de l'ontologie, à savoir le développement de patrons de conception à ancrage social utilisés plus particulièrement lors la phase de conception détaillée de Tropos. Ces patrons de conception respecteront les spécifications FIPA et seront développés sur l'environnement agent JACK en se basant sur le modèle théorique BDI.

Le travail s'organise de la façon suivante. La section 2 introduit brièvement le paradigme orienté-agent, ses tenants et aboutissants. La notion d'agent et ses caractéristiques individuelles, sociales et de raisonnement, y compris le modèle BDI, y sont également présentées. La section 3 identifie des patrons de conception à ancrage social utilisés dans Tropos selon deux catégories: point-à-point et médiation. L'environnement JACK, plate forme JAVA orienté-agent, est décrit dans la section 4. Ces patrons respectent les spécifications de la FIPA (Foundation of Intelligent Physical Agents) que nous abordons dans la section 5. Les sections 6, 7 et 8 décrivent respectivement nos spécifications et

implémentations des patrons de conception appel d'offre, mise aux enchères et courtier. La section 9 conclut le travail et pointe vers des travaux ultérieurs.

2. Approche Agent

2.1. Un paradigme social et intelligent

L'approche orientée-agent est un paradigme résultant de recherches avancées dans différents domaines tels que l'intelligence artificielle distribuée, les systèmes coopératifs, les sciences cognitive, les théories de l'apprentissage ou encore la psycho-philosophie. En ingénierie logicielle, ce paradigme satisfait le besoin actuel de développer des systèmes ou parties de systèmes intelligents simulant le raisonnement humain ainsi que celui de modéliser leur domaine d'application. Les approches traditionnelles, y compris l'orientation objet, permettent difficilement la modélisation du raisonnement; souvent les applications développées suivant de telles approches présentent des limitations quant aux aspects du système nécessitant un certain raisonnement autonome.

Bien que ce paradigme soit encore dans une phase de recherche et développement quant aux standards, plate-formes, langages et protocoles, il a déjà montré des promesses particulières dans une variété de systèmes technico-sociaux: gestion du trafic aérien, simulation de combat, gestion de ressources, gestion du risque, systèmes de logistique distribuée, applications boursières, systèmes peer-to-peer de type Gnutella, e-marketplace, systèmes e-business, environnement de robotique cognitive ou systèmes de commande et de contrôle en temps réel, … Parce qu'elle offre une solution modulaire, flexible, dynamique et élégante à beaucoup de ces domaines nécessitant des systèmes impliquant à la fois des aspects pro-actifs/intelligents d'une part et socio-organisationnels d'autre part, l'approche orienté-agent convient idéalement à ces types d'environnements.

2.2. Agent et modèle BDI

Un agent représente une personne, une organisation, un système, une machine, une entité logicielle, … en somme toute entité caractérisée par la faculté d'agir en interaction avec son environnement. Trois types de caractéristiques peuvent être identifiés pour définir plus en détail un agent: individuelles et sociales et de raisonnement.

- Caractéristiques individuelles:
  - Autonomie: un agent est capable d'interagir avec l'environnement sans intervention extérieure.
  - Mobilité: un agent est capable de migrer de lui-même d'un environnement vers d'autre environnement.
  - Imprévision: un agent peut agir de manière non prévisible, quand bien même les conditions initiales de l'action sont connues.
  - Robustesse: un agent doit intégrer un certain degré de tolérance aux fautes et pouvoir gérer l'incomplétude des données.

- Caractéristiques sociales:
  - Interactivité: un agent doit être capable de communiquer avec d'autres agents et avec son environnement.
  - Adaptation: un agent doit être capable de répondre à d'autres agents et de s'adapter à l'environnement.
  - Sociabilité: un agent doit être capable de représenter d'autres agents et d'agir en leur nom.
  - Pro-activité: un agent a ses buts concrets qu'il accomplit de manière autonome et « consciente », il ne réagit pas seulement à l'environnement.
  - Coordination: un agent doit être capable de travailler dans un environnement qu'il partage avec d'autres agents.
  - Coopération: un agent est capable de se coordonner avec d'autres agents pour réaliser un but commun. Ils réussissent ou échouent ensemble.
  - Compétition: un agent peut être capable de toujours maximiser son bénéfice et éliminer ses concurrents à un coût minimal.
- Caractéristiques de raisonnement:
  - Un agent est caractérisé par une série de concepts pouvant simuler le raisonnement humain: des connaissances - croyances sur le monde extérieur et ressources (données et information) - conditionnant la mise en œuvre d'intentions (actions et plans précompilés) pour accomplir des désirs (objectifs déterminés de manière pro-active) ou des buts (objectifs assignés).
  - Le modèle BDI (Belief-Desire-Intention) est un modèle de raisonnement utilisé en intelligence artificielle et ayant ses racines en philosophie et sciences cognitives. Il est largement utilisé comme modèle de raisonnement agent notamment dans la plateforme JACK que nous décrivons plus loin.

Dans le modèle BDI, comme représenté dans la figure 1, un agent a des croyances sur le monde, il doit satisfaire des désirs en effectuant des intentions.

- ***Belief*** (croyance): l'agent a une série de propositions à propos du monde considérées comme des affirmations vraies.
- ***Desire*** (désir): l'agent a des objectifs qu'il voudrait accomplir.
- ***Intention*** (intention): l'agent souhaite déduire certaines propositions comme vraies. Il adopte une série d'intention pour former un plan d'action afin de transformer l'état du monde selon les objectifs qu'il a choisi d'accomplir.

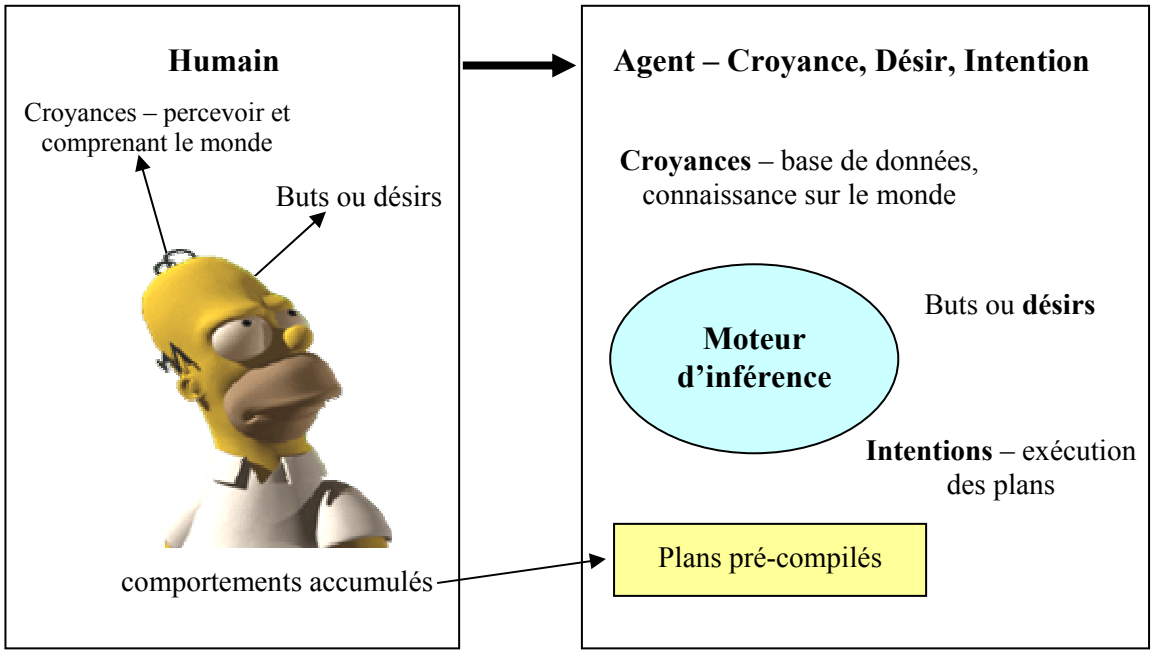

Figure 1: Modèle BDI

3. Patrons de conception à ancrage social

Dans cette nouvelle perspective de devoir développer de nos jours des systèmes agent, nous développons la notion de patron de conception à ancrage social. Un pattern (patron de conception) social définit les agents - tenant compte de leurs rôles et responsabilités – ainsi que les interactions sociales requises pour l'accomplissement des objectifs de ces agents. Un travail de recherche considérable a été effectuée en ingénierie logicielle ces quinze dernières années afin de définir des patrons de conception (par ex., [8]). Malheureusement, ceux-ci ne prennent pas en compte les aspects sociaux des entités logicielles (objets ou agents) qu'ils décrivent. D'un autre côté, les propositions de patrons de conception adressant des problèmes sociaux (par ex., [9], [10]) ne s'attaquent qu'à des considérations d'implémentation telles que les protocoles de communication, la collecte d'information à partir des sources de données ou les phases de connexion, certes importantes, mais n'insistent pas sur la description du côté social conceptuel des interactions entre agents, ni des caractéristiques organisationnelles du système multi-agent.

Nous identifions ci-dessous certains patrons de conception à ancrage social à utiliser dans Tropos récurrent dans la littérature sur les systèmes multi-agent ou coopératifs. En particulier, les structures suivantes sont inspirées des patrons de conception fédérés introduits dans [11], [12], [13], [14], [15], [16].

D'une manière générale, un patron de conception à ancrage social implique un agent offrant un ensemble de services de facilitation de communication avec d'autres agents en utilisant la connaissance qu'il possède concernant les requêtes adressées à et les capacités de ces agents. Un exemple typique est un agent qui demande à un courtier de trouver un ou plusieurs agents qui peuvent répondre à une requête. L'intermédiaire détermine alors un ensemble d'agents qui sont susceptibles de satisfaire cette requête, le leur envoie traite leurs réponses. L'utilisation d'agents intelligents simplifie ces

services de médiation/interaction étant donnée l'accès intrinsèque à leurs bases de connaissances et de plans pour optimiser l'accomplissement du but à savoir la réalisation efficace de l'accès à et de la diffusion de l'information dans un système multi-agent.

Deux types de patrons de conception peuvent être distingués: point-à-point et médiation.

3.1. Point-à-point

- **Appel d'offre**: Ce patron de conception implique un initiateur (contractant) et un nombre non défini de participants (clients). L'initiateur émet une requête de proposition pour un service particulier à destination de tous les participants et acceptent les propositions des participants de fournir le service à un coût particulier. L'initiateur sélectionne un participant qui fournira le service contracté et informera l'initiateur de l'achèvement de celui-ci. Le protocole ContractNet FIPA que nous étudierons et implémenterons par la suite assure des mécanismes de type appel d'offre entre agents.
- **Mise aux enchères**: Ce patron de conception implique un initiateur (commissaire priseur) et un nombre infini de participants assurant le rôle de parieurs. L'initiateur organise et conduit le processus. Il publie l'enchère ou le service demandé auprès des participants et reçoit les différentes propositions. A chaque itération, l'initiateur peut choisir une offre qu'il considère satisfaisante ou surenchérir l'offre à la baisse ou à la hausse. L'initiateur peut être un des parieurs ou peut aussi être dissocié de l'agent qui publie l'enchère. Dans ce dernier cas, il ne sera qu'un intermédiaire (intégrant des services de médiation comme définis ci-dessous) chargé de diriger la mise aux enchères et devra rendre compte au publieur de l'enchère. Le protocole Iterated ContractNet que nous étudierons et implémenterons par la suite assure des mécanismes de ce type entre agents. Il applique le ContractNet FIPA mais le processus de sélection se fait par itération impliquant une négociation de la proposition et du coût du service tant à la baisse qu'à la hausse.

3.2. Médiation

- **Courtier**: un courtier est un agent jouant le rôle d'arbitre et intermédiaire garantissant à un agent client (initiateur) des services d'accès à d'agents participants, fournisseurs de service afin de satisfaire les requêtes de l'agent client concernant la fourniture d'un service précis. Le courtier se charge d'identifier le fournisseur compétent, de médier la requête ainsi que la réponse au profit du client. Le protocole FIPA Broker que nous étudierons et implémenterons par la suite assure des mécanismes de ce type entre agents.
- **Matchmaker**: un matchmaker est un agent se chargeant de localiser un agent fournisseur de service pour un agent client et identifie le client auprès de l'agent fournisseur. A l'opposé du courtier qui gère toutes les interactions entre le client et le fournisseur, le matchmaker établit uniquement la connexion et laisse la responsabilité et l'arbitrage des interactions subséquentes aux agents intervenants.

- **Médiateur**: un médiateur médie les interactions entre différents agents appelés collègues. Un initiateur adresse au médiateur une requête de service au lieu de directement l'adresser à d'autres collègues pouvant jouer le rôle de fournisseur du service requis. Le médiateur possède des modèles d'accointance des collègues et coordonne la coopération entre eux. Inversement, chaque agent collègue possède un modèle d'accointance du médiateur. Alors qu'un courtier fait simplement la correspondance entre les participants fournisseurs de services et le client, initiateur de la requête, un médiateur encapsule les interactions et communication et maintient à jour continuellement les modèles d'accointance entre initiateurs et performeurs.
- **Moniteur**: un moniteur avertit un souscripteur d'événements qui lui sont pertinents. Il accepte des souscriptions, requiert des notifications à propos d'agents sujets présentant un intérêt, reçoit ces notifications et alerte les souscripteurs des événements pertinents. Les agents sujets fournissent des notifications de changement d'états lorsque demandé. Le souscripteur s'enregistre pour recevoir les notifications de changements d'état à propos d'agents sujets distribués, reçoit ces notifications accompagnées de l'information sur l'état courant du sujet et met à jour l'information concernant son état local.
- **Ambassadeur**: un ambassadeur médie un service requis par un acteur étranger à un agent local et gère la réponse. Si l'accès est autorisé, l'agent étranger peut soumettre des messages à l'ambassadeur pour traduction. Le contenu du message est traduit en accordance avec l'ontologie standard du système dont fait partie l'agent local. Les messages traduits sont transmis aux agents locaux cibles. Les résultats de la requête sont transmis en retour à l'agent étranger et traduits en sens inverse.
- **Wrapper**: un wrapper est un **ambassadeur** spécifique à un legacy system. Le wrapper interface les agents clients au legacy system assurant le rôle de traducteur et vérifie que les protocoles de communication sont respectés et que le legacy system reste indépendant des clients.

4. Environnement JACK de développement agent intelligent

4.1. Une plateforme BDI en Java

JACK Intelligent agent est un environnement de développement orienté agent intégrant entièrement le langage de programmation Java. Il se base fondamentalement sur le modèle agent BDI. La relation entre JACK et Java est analogue à celle entre les langages C++ et C. C a été développé comme un langage procédural et C++ fournit une extension orienté-objet à C. De même, JACK a été développé pour fournir une extension orientée-agent à Java. Le code source de JACK est d'abord compilé en code Java avant d'être exécuté.

De la même manière que la programmation orienté-objet présente un certain nombre de concepts spécifiques influençant la structure logique et physique d'un système, la programmation orienté-agent se réfère à des concepts spécifiques tournant autour de la notion d'**agent**. Ces agents sont autonomes et on la capacité de raisonner. Ils sont capables de prendre des décisions de manière pro-active tout en réagissant en temps réel aux événements de l'environnement.

L'architecture par défaut d'un agent JACK est de type BDI essentiellement basée sur des plans, des buts et des connaissances. Dans JACK, un agent est un composant logiciel qui peut raisonner confronté à un stimulus pro-actif (but dirigé) ou réactif (contrôle d'événement). Chaque agent a des composants fondamentaux:

- Un ensemble de **croyances** par rapport au monde dans lequel il évolue.
- Un ensemble des **événements** auxquels il répondra.
- Un ensemble de **buts** que l'agent peut réaliser suite à la demande d'un agent externe (événement) ou suite à un ou plusieurs changements de croyances.
- Un ensemble de **plans** décrivant comment l'agent peut manipuler les buts ou les événements qui surgissent.

Lorsqu'un agent est créé, il attend qu'il lui soit attribué un but à réaliser ou qu'il reçoive un événement auquel il doit répondre. Quand un tel but ou événement arrive, l'agent détermine les actions à entreprendre. Si l'agent déduit que le but ou l'événement a déjà été manipulé ou réalisé, il ne fait rien. Autrement, il examine ses plans pour déterminer ceux qui satisfont la demande et peuvent être applicables à cette situation en cours. Si un plan ne réussit pas, l'agent recherche d'autres plans qui pourraient accomplir ce but. Cette recherche se prolonge jusqu'à ce que l'agent réussisse ou qu'il n'y a plus des solutions.

4.2. Composants de JACK

JACK se compose des éléments principaux suivants. Les extraits de code réfèrent aux applications présentées plus loin:

a. **Agent**: le composant agent est utilisé pour définir le comportement d'une entité logicielle intelligente. Il inclut ses capacités, le type des messages et des événements auxquels il répond et les plans qu'il utilisera pour réaliser ses buts.

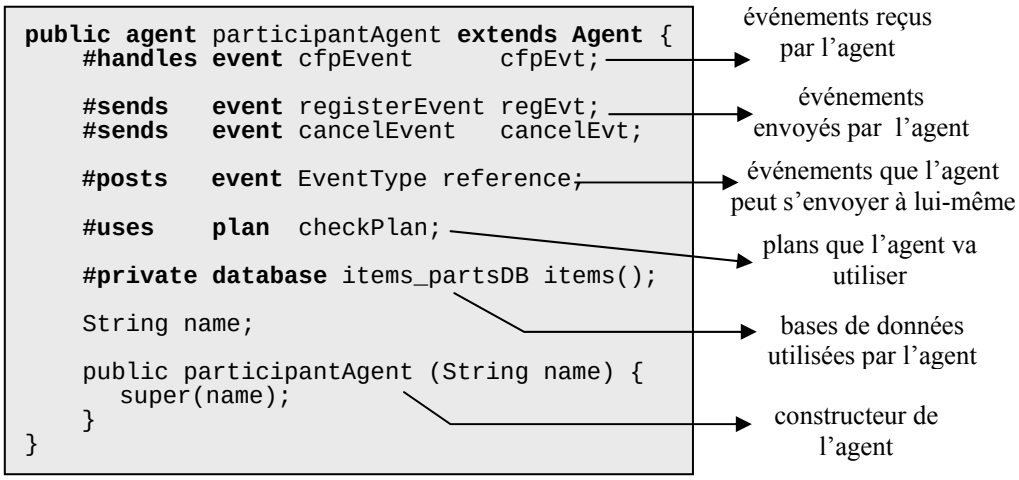

Figure 2: Déclaration d'un agent

b. **Capability**: Ce composant simplifie la conception du système agent, permet la réutilisation du code et l'encapsulation des fonctions propres à un agent. Une capacité peut se composer de plans, d'événements, d'un ensemble des croyances ou d'autres capacités. Un agent peut, alternativement, se composer d'un certain nombre de capacités, dont chacune a une fonction spécifique attribuée à l'agent. Les capacités sont construites de la même manière qu'un simple agent n'incluant que la déclaration de ses composants.

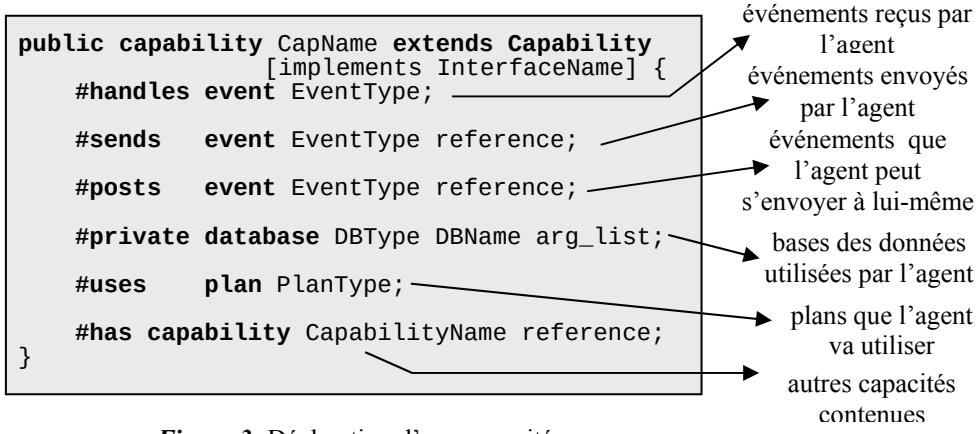

Figure 3: Déclaration d'une capacité

c. **Event**: Ce composant décrit une occurrence, en réponse à laquelle l'agent doit agir.

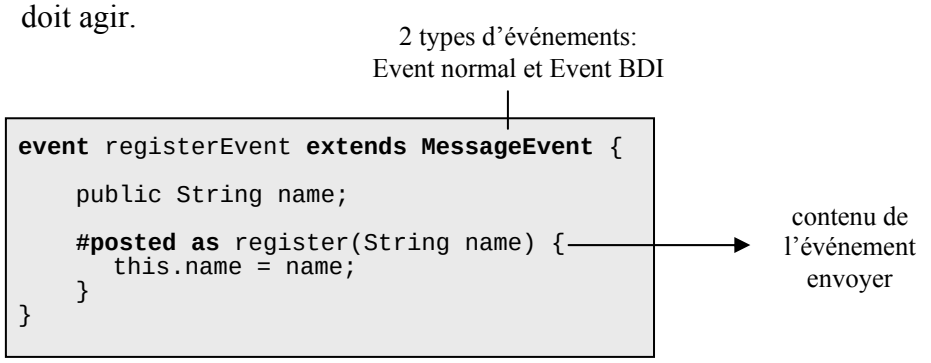

Figure 4: Déclaration d'un événement

JACK a 2 types d'événements:

- Normal Event: ce sont des événements de type transition qui sont déclenchés et auquel l'agent répond.
- BDI Event: ils incluent la dimension de raisonnement du modèle BDI : l'agent a un but et met tout en œuvre pour l'accomplir.

d. **Plan**: Les plans d'un agent sont les fonctions de cet agent. Ce sont les instructions que l'agent suit pour essayer d'accomplir ses buts et de manipuler ses événements indiqués.

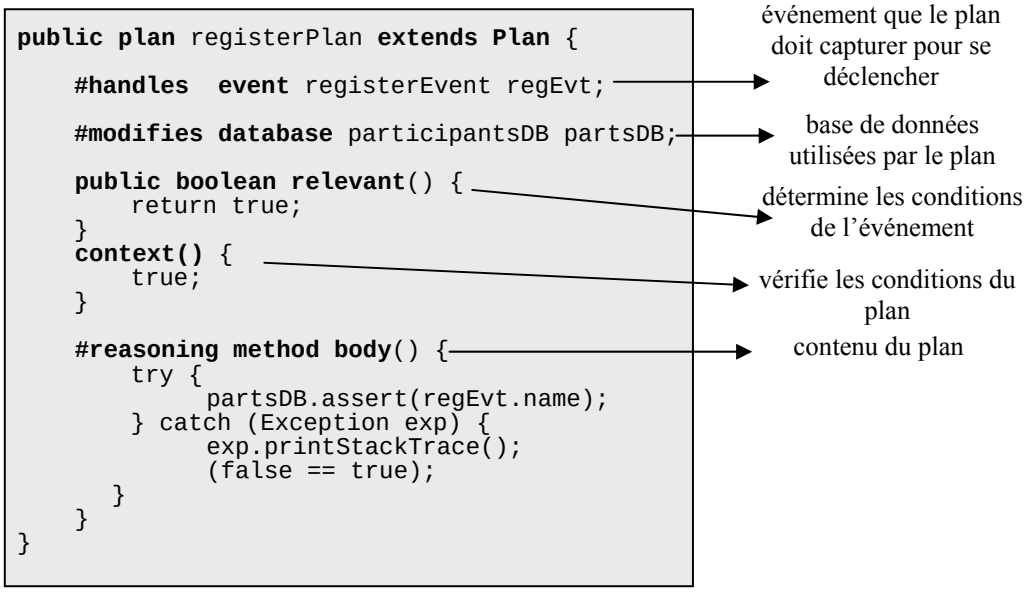

Figure 5: Déclaration d'un plan

e. **View**: Cette composante fournit une abstraction qui permet l'utilisation des requêtes complexes sur les bases de données, par exemple, les requêtes dont les résultats concernent plusieurs bases de données. Le modèle de données peut être mis en application en utilisant des croyances (bases de données) multiples ou les structures de données arbitraires de Java.

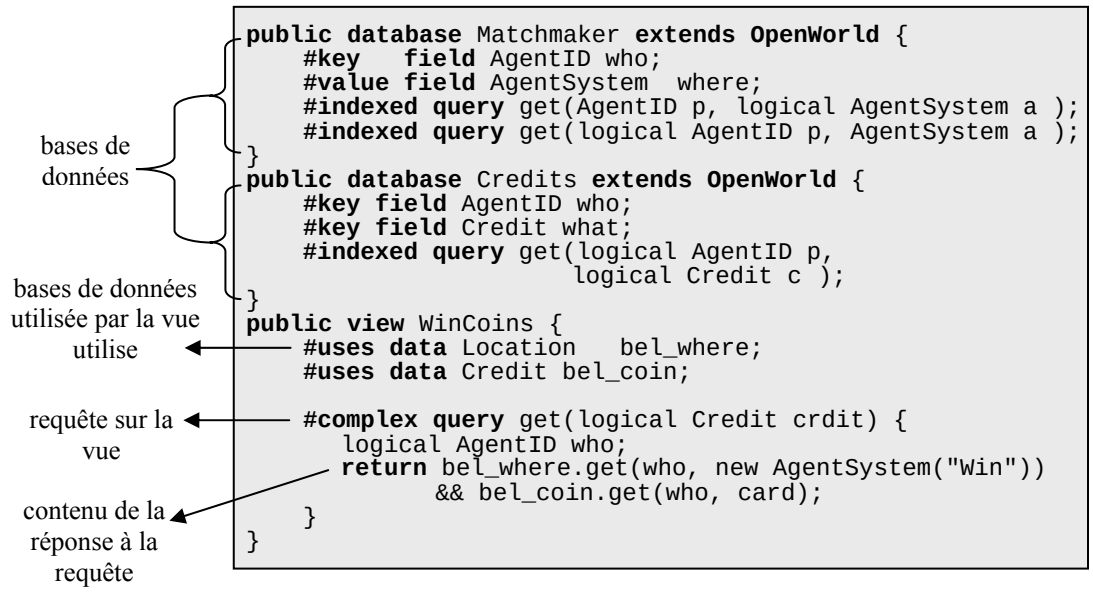

Figure 6: Déclaration d'une vue

f. **Database**: ce composant définit un ensemble de croyances de l'agent en utilisant une base de données générique. Cette base de données est spécifiquement conçue pour pouvoir être interrogée en utilisant des membres logiques. Les membres logiques sont des membres de données suivant les règles d'unification de la programmation logique (comme en prolog).

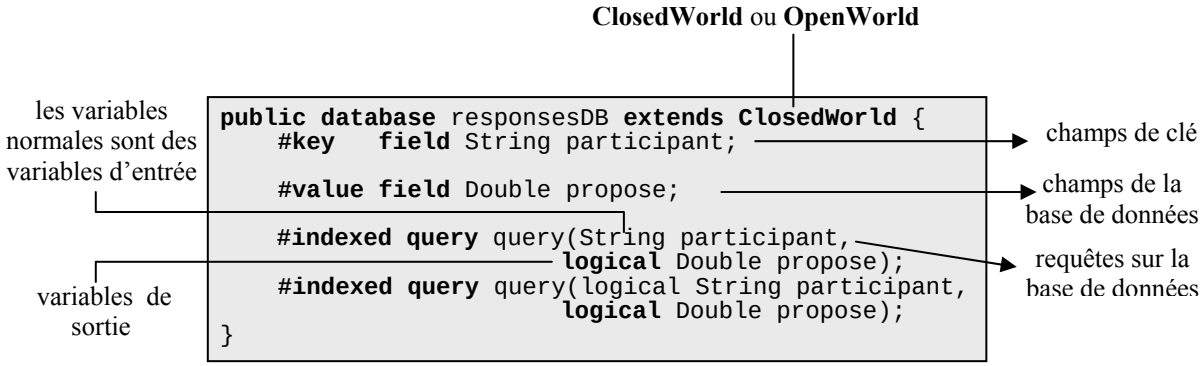

Figure 7: Déclaration d'une base de données

5. Spécifications agent de type social : les normes FIPA

La FIPA, fondation pour les agents physiques intelligents (Foundation for Intelligent Physical Agents), est un organisme international d'entreprises et d'institutions qui tentent de produire des spécifications utilisables sur les technologies agents intelligents. Le but principal de cette association est de normaliser et standardiser ces spécifications et de trouver des consensus entre différentes organisations impliquées, actuellement un cinquantaine tel que IBM, Siemens, France Telecom, Toshiba, …

Dans cette approche de normalisation et standardisation, la FIPA met l'accent sur et définit, entre autres standards, des protocoles de communication et d'interface

concernant l'interaction sociale d'agents logiciels d'un système entre eux ou avec les différents composants de l'environnement: humains, autres agents, entités logicielles non-agent (par ex. des objets) externe, monde physique, ... Dans cette perspective, la FIPA produit deux genres de spécifications: normative et informative.

- Une spécification est normative lorsqu'elle spécifie le comportement d'un sous-système afin d'assurer une interaction avec d'autres sous-systèmes respectant les normes FIPA.
- Une spécification est informative lorsque son but est de fournir des conseils à l'industrie, des spécifications de l'application sur certains aspects particuliers d'un sous-système de FIPA technologie.

Actuellement, la FIPA a publié quinze types de spécifications normatives et informatives concernant l'interaction sociale des agents se basant sur quatre secteurs de standardisation : interaction agent/humain, gestion d'agent, communication des agents et intégration agent-logiciel comme illustré par la figure 8.

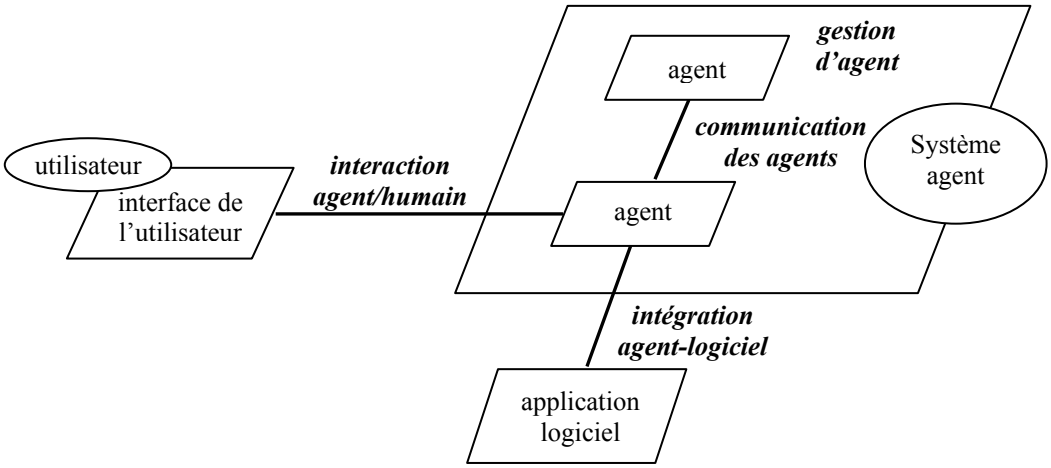

Figure 8: Les quatre types d'interface sociale FIPA

- ***Gestion d'agent*** [21]:

  L'objectif est de spécifier l'effort technologique minimum nécessaire à une gestion correcte et efficace des agents d'un système. La FIPA définit un modèle logique pour la création, l'enregistrement, la localisation, la communication, la migration et la destruction des agents.

- ***Communications des agents*** [22]:

  Cette standardisation définit les types de communications/interactions entre agents tels que négociation, coopération, échange d'information, ...

  Afin de supporter cet effort de normalisation, la FIPA a spécifié un langage de communication entre agents : le FIPA ACL (Agent Communication Language). Ce langage se base sur un modèle simple d'abstraction de la communication entre agents. Les agents sont supposés communiquer à un niveau conceptuel en ce sens que les contenus de la communication sont des

affirmations sur l'environnement ou sur la connaissance de l'agent. Cette caractéristique fondamentale permet de distinguer la communication entre agents, pouvant impliquer des contenus sémantiques se référant à des ontologies propres au domaine d'application ou au système de simples interactions entre entités comme le sont par exemple les interactions entre objets.

Le langage FIPA ACL est inspiré de deux langages: KQML[25] (Knowledge Query and Manipulation Language) et ARCOL. Il reprend d'ARCOL sa sémantique et de KQML sa convivialité et son utilisabilité.

Le langage FIPA se compose de cinq niveaux:

- **Protocoles:** règles sociales définissantes la structure des dialogues entre des agents.
- **Actes de communication:** types de communications effectuées.
- **Messages échangés entre agents:** méta-informations concernant le message que les agents s'échangent (réception, envoi, réponse, médiation, contexte, …).
- **Langage du contenu:** définitions de grammaire et de sémantique pour exprimer le contenu des messages.
- **Ontologie:** vocabulaire et sémantique des termes, concepts utilisés dans l'expression du contenu.

La figure 9 illustre comme exemple le message **inform** défini dans le protocole fipa-contract-net que nous détaillerons plus tard.

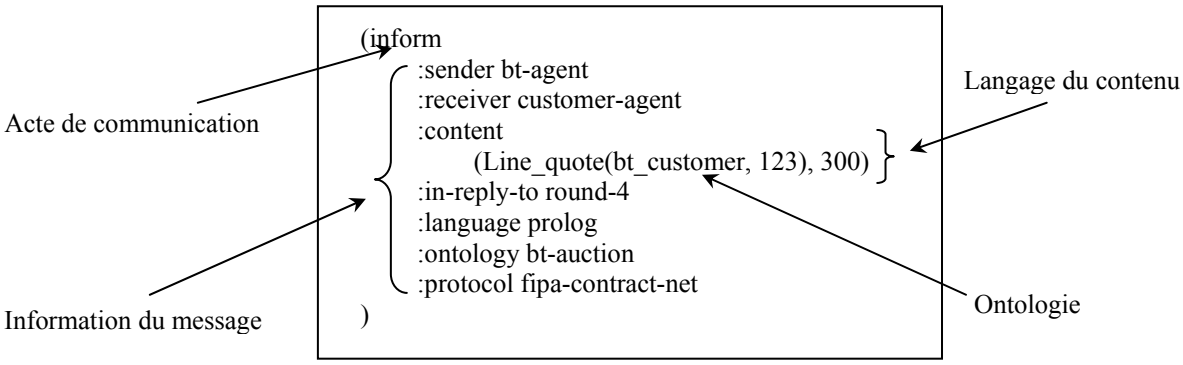

Figure 9: structure d'un message FIPA ACL

- ***Intégration agents- logiciels*** [23]:

  Les agents évoluent dans un monde isolé sans besoin d'intégrer un logiciel extérieur. Tous les services sont fournis par des agents à d'autres agents. Si un agent veut obtenir un service, il s'adresse à un autre agent pouvant fournir ce service. Les logiciels extérieurs qui ne sont pas des agents doivent être transparents aux agents natifs du système.

  Ces spécifications FIPA s'attachent à définir comment les ressources logicielles peuvent être décrites, partagées et contrôlées dynamiquement dans une communauté d'agents.

- ***Interaction agents- humains*** [24]:

  Ces spécifications normalisent les fonctionnalités de base et les interfaces des agents pouvant contrôler les dialogues avec les utilisateurs ou assurer la personnalisation de l'interaction humain-agent et du comportement agent en construisant et en maintenant des modèles utilisateur. Les modèles utilisateur peuvent être des profils d'utilisateur, c.-à-d. des bases d'informations explicitement fournies sur l'utilisateur ou être déduits de l'étude des comportements d'utilisation.

Parmi ces quatre axes de standardisation définis par la FIPA, le plus essentiel quant à la prise en compte des aspects sociaux des agents est celui concernant la spécification des communications entre des agents. Afin de normaliser ces communications, la FIPA a spécifié des protocoles d'interaction à respecter pour le développement d'une architecture multi-agent. Ces protocoles (ContractNet, Request, Query, Request-when, Broker, …), s'apparentent aux ou peuvent être utilisés pour développer et réaliser concrètement les patrons de conception à ancrage social. Dans la suite du travail nous nous focaliserons sur trois de ces protocoles : le ContractNet, l'Iterated Contract-Net et le Broker pour respectivement mettre en œuvre les patrons de conception appel d'offre, mise aux enchères et courtier dont nous avons parlé précédemment.

6. Appel d'offre

Cette section présente la réalisation concrète du patron de conception à ancrage social « Appel d'offre » implémenté en respectant les spécifications FIPA définies pour le protocole ContractNet. Nous commentons premièrement ce protocole, décrivons notre implémentation de celui-ci en agent BDI sur la plateforme JACK et expliquons finalement l'application JACK que nous avons développée pour l'utilisation concrète de ce pattern/protocole.

6.1. Protocole FIPA ContractNet

Le protocole ContractNet FIPA est inspiré du modèle original du même nom défini dans [26]. Il intègre en plus des actes de communications de type rejet et confirmation. Nous le définissons dans les grandes lignes ci-dessous et référons à [27] pour une spécification complète du protocole.

Dans ce protocole, un agent assume le rôle d'initiateur (initiator) qui souhaite voir un certain service accompli par un ou plusieurs autres agents participant de type performeur et souhaite optimiser ce service.

Pour ce faire, l'initiateur publie, à destination des agents participant, performeurs potentiels, un appel à propositions (call for proposal) qui spécifie le service à effectuer ainsi que toutes les conditions que l'initiateur souhaite. Les participants vérifient s'ils peuvent satisfaire aux conditions préalables. Si cela est possible, ils répondent à l'initiateur par un acte de proposition. Alternativement, un participant peut refuser de faire une proposition. Quand un moment-butoir (deadline) défini est dépassé, l'initiateur évalue toutes les propositions reçues et choisit l'agent qui accomplira le service ou peut toutes les refuser L'initiateur envoie alors un acte d'acceptation de proposition aux participants dont la proposition est choisie, les autres reçoivent un acte de rejet de proposition. Lorsque l'initiateur accepte la proposition, l'agent participant performeur acquière un engagement pour accomplir le service. Quand ce dernier a accompli le service, il envoie un message d'accomplissement à l'initiateur.

Il est à noter que ce protocole exige de l'initiateur de savoir quand il a reçu toutes les réponses des participants. Si un participant ne réussit pas à lancer un acte de proposition ou un acte de refus, l'initiateur peut potentiellement être mis en attente indéfiniment. Pour éviter ce problème, l'appel à proposition se compose d'un moment-butoir avant lequel les réponses doivent être reçues par l'initiateur. Les propositions reçues après le moment-butoir sont automatiquement rejetées avec comme justification que la proposition a été faite trop tardivement.

La figure 10 illustre en détails le protocole ContractNet FIPA en AUML [19]. Les termes restent en anglais pour des raisons de cohérence de vocabulaire par rapport aux spécifications FIPA. Il en sera de même pour la présentation des protocoles suivant ainsi que de leur implémentation.

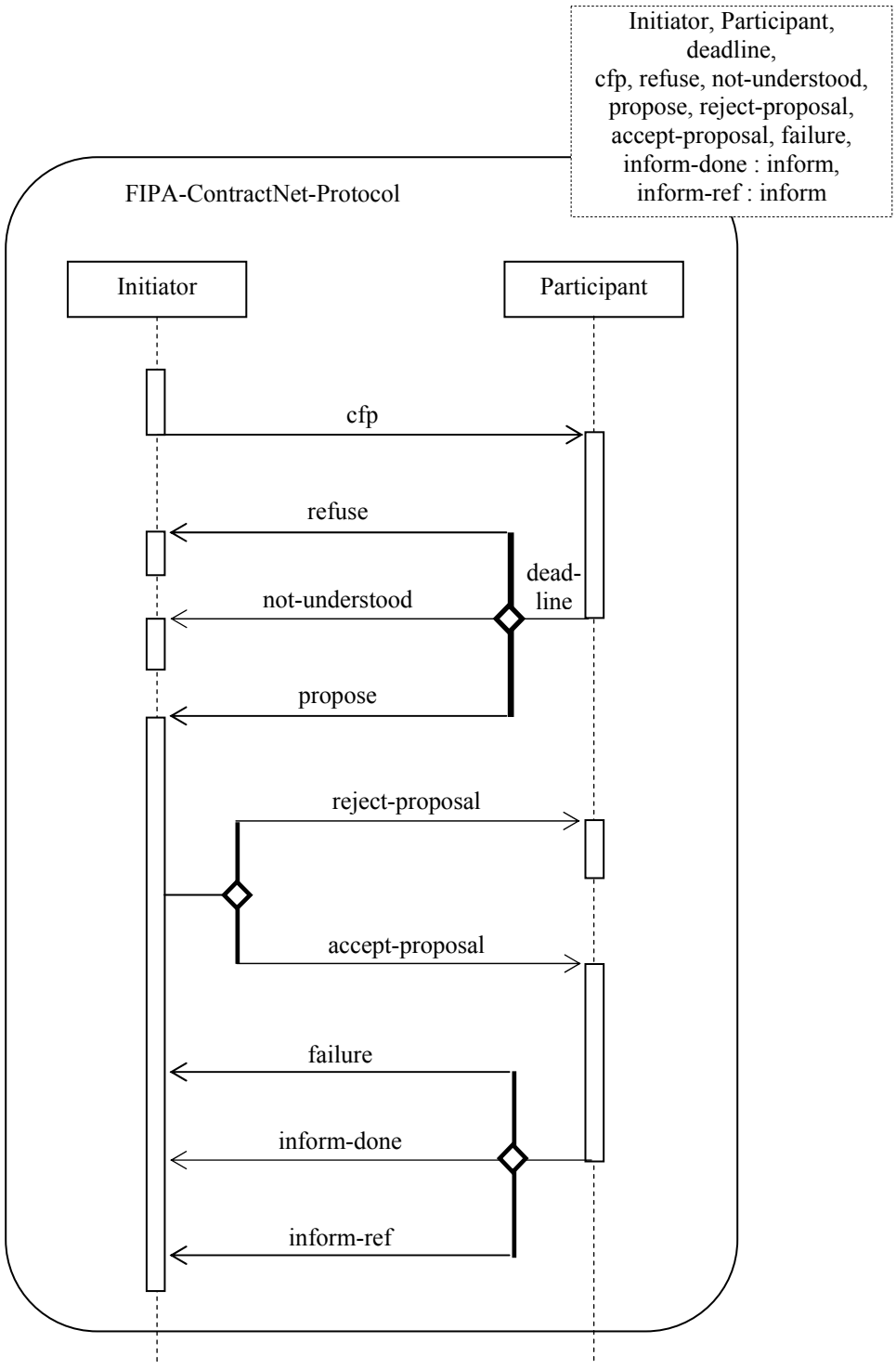

Figure 10: Protocole d'interaction Contract Net FIPA

6.2. Implémentation agent en JACK

Nous détaillons ici les différents composants agent de l'implémentation du protocole ContractNet FIPA en JACK à savoir successivement les agents, événements, plans, croyances et ressources java.

6.2.1 Agents:

- **Initiator Agent:** il s'agit de l'initiateur souhaitant recevoir ou faire se réaliser un service. Celui-ci enverra un appel à propositions aux participants. Il se compose des événements cfpEvent, reject_proposalEvent, accept_proposalEvent, des plans registerPlan, cfpPlan, respondPlan, inform_donePlan, inform_donePlan, failurePlan définis ci-dessous.
- **Participant Agent:** ce sont les performeurs potentiels qui reçoivent les appels à propositions des initiateurs. Ils vérifient dans leurs bases de données (croyances) s'ils peuvent répondre à ces demandes. Si c'est le cas, ils envoient leurs propositions à l'initiateur. Ils se composent des événements registerEvent, checkEvent, respondEvent, inform_doneEvent, inform_refEvent, failureEvent et des plans checkPlan, acceptPlan, rejectPlan définis ci-dessous.

6.2.2 Events (événements)

- **Register Event:** lorsqu'un agent souhaite être considéré comme performeur potentiel (participant) auprès d'un initiateur, il envoie un tel acte d'inscription à cet initiateur.
- **Cfp Event:** lorsque l'initiateur veut envoyer une demande à des participants enregistrés auprès de lui, il envoie un tel acte Cfp (Call for proposal) à ses participants.
- **Respond Event:** après avoir reçu un appel à propositions de type Cfp, les participants vérifient s'ils peuvent y répondre. Dans le cas positif, ils font une proposition au moyen de l'acte Propose Event, sinon, ils refusent le message Cfp.
- **Propose Event :** il s'agit de la proposition du participant en réponse à l'appel à proposition
- **Accept_proposal Event:** lorsque le moment-butoir est passé, l'initiateur choisit la meilleure proposition, et contacte l'agent participant choisi comme performeur pour la suite par un tel acte d'acceptation de proposition.
- **Reject_proposal Event:** l'initiateur envoie cet acte de rejet aux participants non choisis.
- **Failure Event:** si la proposition d'un participant est acceptée, celui-ci re-vérifie s'il est toujours en mesure de l'assumer. S'il n'est plus capable de le faire ou s'il change de décision de par les conditions changeantes de l'environnement, il le signifie à l'initiateur par cet acte d'échec.
- **Inform_done Event:** si le participant est toujours capable d'assumer sa proposition il signifie son accomplissement à l'initiateur lorsqu'elle est réalisée.
- **Inform_ref Event:** si le participant confirme sa capacité à accomplir le service, il envoie simultanément un tel acte référençant les

données et paramètres particuliers concernant ce service et sa réalisation.

6.2.3 Plans

- **Register Plan:** lorsque l'initiateur reçoit l'inscription d'un participant, il l'enregistre dans une base de données.
- **Cfp Plan:** suivant la relation entre désir et intention du modèle BDI, ce plan implémente concrètement, comme intention, le désir exprimé par l'événement Cfp Event. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Check Plan:** après avoir reçu un CfpEvent, un participant vérifie, par ce plan, ses bases de données de manière à déterminer s'il est capable de répondre à l'initiateur. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Respond Plan:** après avoir reçu les réponses des participants, l'initiateur, s'il y a plusieurs propositions choisit, par ce plan, celles qui satisfont les conditions requises et les empilent dans une base de données.
- **Accept Plan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'évenement Accept_proposal: le participant reçoit cet événement d'acceptation de sa proposition, il vérifie alors s'il peut ou il veut toujours effectuer ce service. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Reject Plan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Reject_proposal: le participant l'utilise pour recevoir le rejet de l'initiateur. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Failure Plan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Failure. Il est utilisé par l'initiateur pour signifier, en réponse au participant, que la proposition a bien été déclarée comme échouée. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Inform_done Plan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Inform_done. Il est utilisé par l'initiateur pour signifier, en réponse au participant, que la proposition a bien été déclarée comme réalisée. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Inform_ref Plan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Inform_ref. Il est utilisé par l'initiateur pour recevoir les informations concernant le service référencée par l'événement Inform_ref. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.

6.2.4 Base de données (croyances)

- **Items_Initiator DB:** base de données utilisée pour stocker les données de l'initiateur.
- **Items_Participant DB:** base de données utilisée pour stocker les données des participants.
- **Participants DB:** base de données utilisée pour stocker les participants qui se sont inscrits auprès l'initiateur.
- **Responses DB:** base de données utilisée pour stocker les propositions valides tant que le moment-butoir n'est pas dépassée.

6.2.5 Ressources Java

- **Initiator_Frame:** application GUI (Graphical User Interface) de l'initiateur.
- **Launch Java:** lance les ressources et démarre l'application Appel d'offre présentée ci-dessous.

6.3. L'application

Afin d'implémenter le protocole FIPA ContractNet, l'application développée dans ce travail suppose un initiateur proposant à l'achat ou à la vente différent items à des participants enregistrés auprès de lui. Un initiateur a une base de données stockant ses produits et leurs caractéristiques (quantité, prix de vente, prix d'achat).

La première interface de l'application, représentée à la figure 11, permet de charger la base de données de l'initiateur si celle-ci existe. L'utilisateur peut la créer ou la modifier (ajouter des items, modifier les valeurs existantes) et la sauver. L'utilisateur signifie le nombre des participants à enregistrer auprès de l'initiateur comme performeurs potentiels.

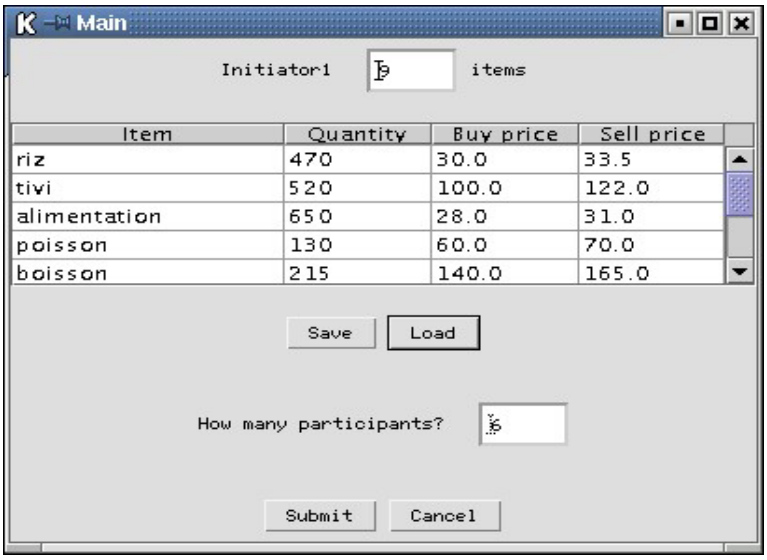

Figure 11: Interface Initiateur de l'Appel d'offre

Lorsque la base de données de l'initiateur et le nombre de participants sont corrects, l'utilisateur clique sur le bouton Submit de manière à activer l'interface principale de l'application ContractNet, représentée par la figure 12. Celle-ci permet à l'utilisateur de charger, modifier et sauvegarder les bases de données de chaque participant.

L'utilisateur joue le rôle de l'initiateur en cours de traitement (ici Initiator1). Chaque appel de proposition aux participants (par le bouton Send) nécessite de paramétrer chacun des champs suivants propre à la proposition (cf figure 12): type d'item (par sélection), quantité et moment-butoir (valeurs numérique) et achat/vente (par sélection). A chaque appel de proposition l'utilisateur peut toujours modifier les bases de données des participants, et reparamétrer les champs propres à la proposition.

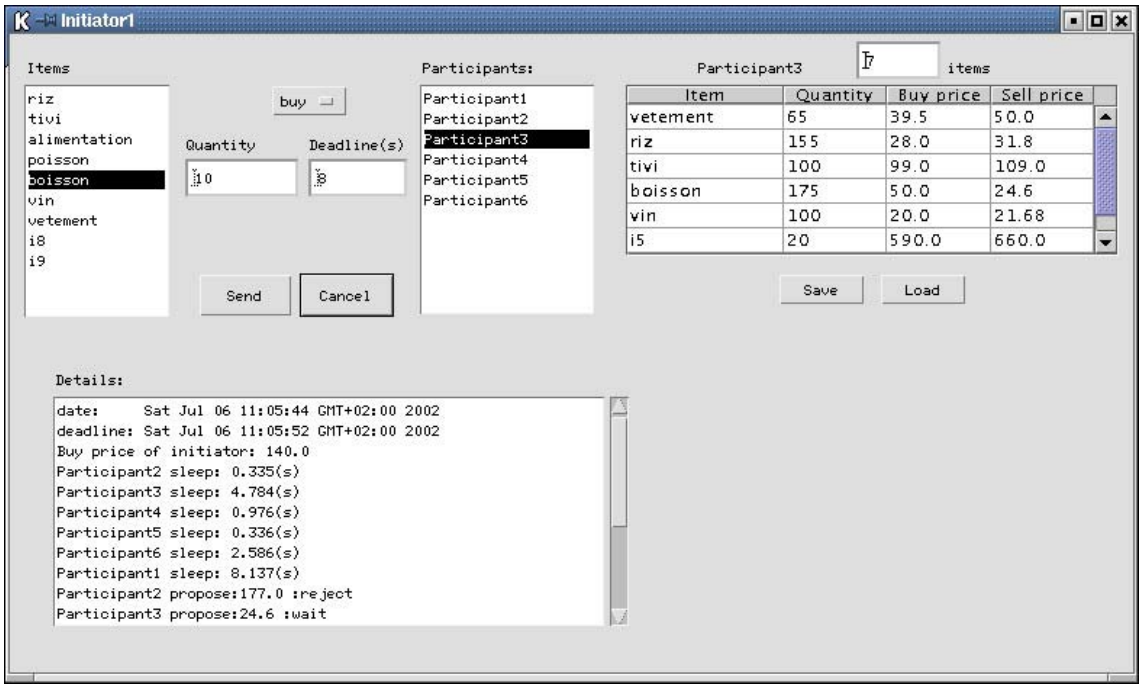

Figure 12: Interface principale de l'Appel d'offre

Le détail du processus, une fois le bouton Send cliqué défile dans la petite fenêtre de dialogue dans le coin inférieur gauche. S'affichent le moment de l'envoi de la proposition de l'initiateur, le moment-butoir et le laps de temps en secondes après lequel chaque participant répond. Si la proposition d'un participant n'est pas satisfaisante, l'initiateur la refuse, sinon il l'empile dans sa base de propositions jusqu'au dépassement du moment-butoir ; il choisit enfin la meilleure proposition.

7. Mise aux enchères

Cette section présente la réalisation concrète du patron de conception à ancrage social "Mise aux enchères" implémenté en respectant les spécifications FIPA définies pour le protocole Iterated ContractNet. Nous commentons premièrement ce protocole, décrivons notre implémentation de celui-ci en agent BDI sur la plateforme JACK et expliquons finalement l'application JACK que nous avons développée pour l'utilisation concrète de ce pattern/protocole.

7.1. Protocole FIPA Iterated ContractNet

Le protocole d'interaction FIPA Iterated ContractNet [28] est une extension du protocole FIPA ContractNet que nous avons détaillé précédemment. Alors que, comme nous l'avons vu, le ContractNet FIPA simple n'inclut qu'un appel de proposition unique, l'Iterated ContractNet permet un surenchérissement à plusieurs tours de cet appel à proposition, ceci en fonction du résultat des propositions reçues au tour précédent. Ce modèle se rapproche plus d'une négociation de type « bidding » rencontrée dans le monde réel (mise aux enchères, vente publique, vente forcée, appel d'offre à plusieurs tours, …). Comme dans le FIPA ContractNet, l'initiateur envoie un acte d'appel à propositions (Call for proposal). Les participants enregistrés comme performeurs potentiels auprès de l'initiateur répondent par acte de proposition. L'initiateur peut accepter directement une proposition et rejeter les autres, ou bien réitérer le processus en publiant un appel à propositions modifié (par exemple prix à la baisse ou à la hausse). Le but est pour l'initiateur de chercher à obtenir de meilleures offres des performeurs en modifiant et en leur resoumettant l'appel à de nouvelles propositions. L'interaction s'achève si l'initiateur refuse toutes les propositions, s'il en accepte une ou s'il n'est plus capable à soumettre un nouvel appel à propositions au tour suivant (par exemple si le prix à la vente qu'il propose au tour suivant est inférieur au prix demandé pour l'achat). La figure 13 illustre en détail le protocole FIPA Iterated ContractNet en AUML.

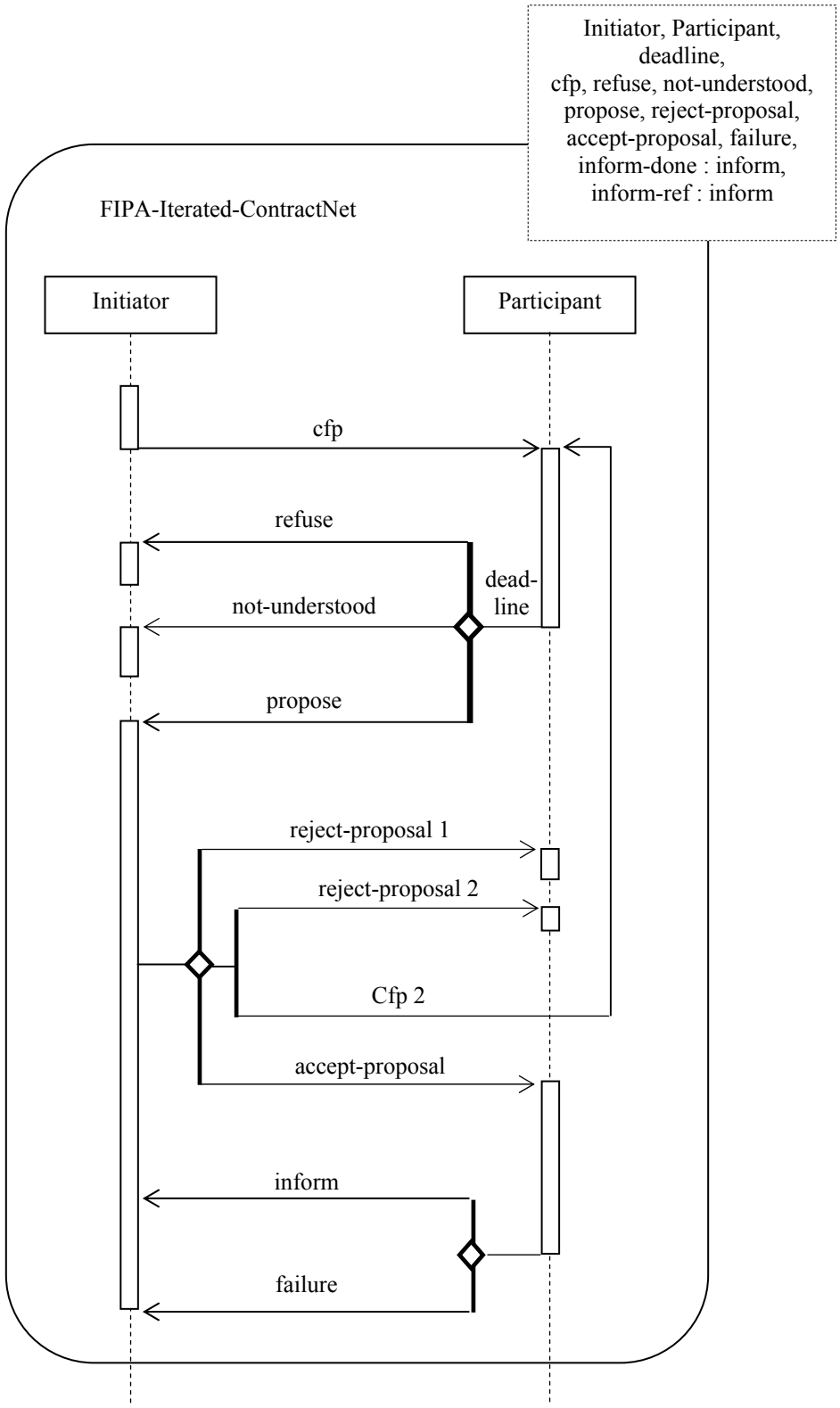

Figure 13: Protocole d'interaction FIPA Iterated ContractNet

7.2. Implémentation agent en JACK

Les composants JACK de l'Iterated ContractNet réutilisent ceux développés pour le ContractNet et que nous avons détaillés précédemment. Les composants suivants diffèrent cependant:

- **Cfp Plan:** lorsque l'initiateur envoie une demande Cfp aux participants enregistrés auprès de lui, il attend le dépassement du moment-butoir et choisit la meilleure proposition. S'il n'en reçoit aucune, il modifie la demande de proposition et la re-envoie. Il réitère l'opération jusqu'à recevoir une proposition satisfaisante ou n'être plus en mesure de surenchérir.
- **Times Belief:** base de données utilisé pour stocker le nombre d'itération d'une demande de propositions. L'initiateur se base sur cette base de données pour décider de la demande suivante.
- **Respond Plan:** après avoir reçu la proposition des participants, l'agent se base, par ce plan, sur la base de données Times pour décider d'accepter ou de refuser ces propositions.

7.3. L'application

La figure 14 propose un écran semblable que celui de l'application Appel d'offre. Cependant, si l'initiateur ne reçoit pas de proposition satisfaisante, il surenchérit sa demande.

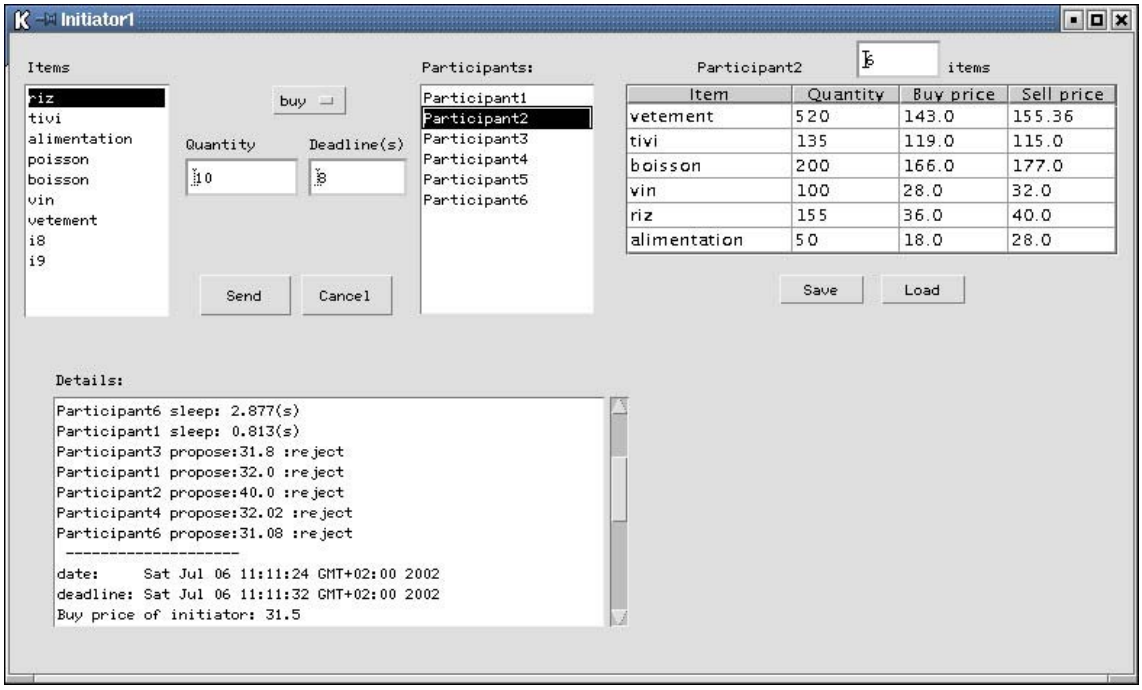

Figure 14: Interface principale de l'Appel d'offre

Dans la fenêtre de dialogue, on constate que par deux fois consécutives l'initiateur lance sa demande: en effet, au premier tour, toutes les propositions

des participants ayant été refusées, l'initiateur augmente son prix d'achat (31.5, initialement il était de 31) et fait une nouvelle offre.

8. Courtier

Cette section présente la réalisation concrète du patron de conception à ancrage social "Courtier" implémenté en respectant les spécifications FIPA définies pour le protocole Broker. Nous commentons premièrement ce protocole, décrivons notre implémentation de celui-ci en agent BDI sur la plateforme JACK et expliquons finalement l'application JACK que nous avons développée pour l'utilisation concrète de ce pattern/protocole.

8.1. Protocole FIPA Broker

Le concept d'intermédiaire d'information est largement répandu dans les systèmes nécessitant des mécanismes de négociation distribuée comme les systèmes multi-agent. Le protocole d'interaction FIPA Broker a été défini à des fins de support de ce type d'interaction sociale.

Le protocole FIPA Broker est un protocole d'interaction défini à un macro-niveau d'abstraction. En effet, l'acte de communication de type *proxy* (figure 15) pour le courtage [29] inclut un acte de communication comme argument. Par conséquent, le protocole d'interaction pour cet acte de communication imbriqué est également inclus dans ce macro protocole. Si l'acte de communication imbriqué inclut des actions à effectuer par les agents enregistrés auprès du courtier, le protocole d'interaction correspondant sera étendu pour informer du résultat de ces actions. De ce fait, le courtier devra tenir trace des paramètres du message *proxy* nécessaires à la médiation en retour du résultat à l'initiateur.

La représentation en AUML du protocole FIPA Broker est donnée par la figure 15.

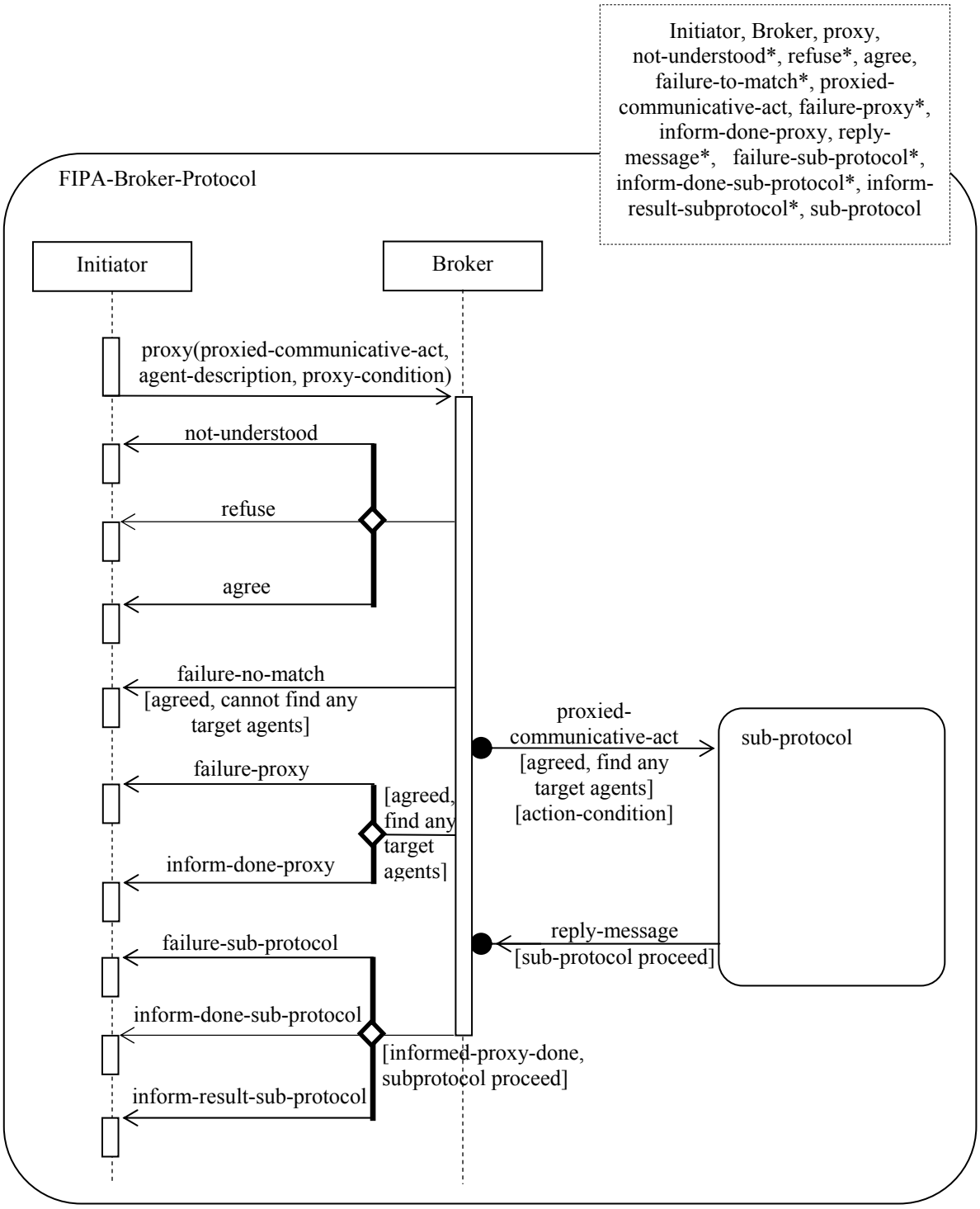

Figure 15: Protocole d'interaction FIPA Broker

8.2. Implémentation agent en JACK

Nous détaillons ici les différents composants agent de l'implémentation du protocole Broker FIPA en JACK à savoir successivement les agents, événements, plans, croyances et ressources java.

8.2.1 Agents:

- **Broker Agent:** il s'agit de l'agent courtier assurant les services de médiation entre initiateurs et participants. Il possède les événements cfpEvent, refuseEvent, agreeEvent, failure_no_matchEvent, reject_proposalEvent, proposalEvent, failure_proxyEvent, failure_sub_protocolEvent, inform_done_proxyEvent, inform_result_subEvent et les plans registerPlan, cancelPlan, proxyPlan, respondPlan expliqués plus ci-dessous.
- **Initiator Agent:** il s'agit de l'initiateur souhaitant recevoir ou faire se réaliser un service. Celui-ci enverra au courtier un acte de communication contenant un appel à propositions destiné aux participants. Il possède les événements proxyEvent, accept_proposalEvent et les plans refusePlan, agreePlan, failure_no_matchPlan, failure_proxyPlan, inform_done_proxyPlan, failure_sub_protocolPlan, proposalPlan, inform_result_subPlan expliqués plus ci-dessous.
- **Participant Agent:** ce sont les performateurs potentiels qui reçoivent les appels à propositions du courtier agissant pour le compte de l'initiateur. Ils vérifient dans leurs bases de données (croyances) s'ils peuvent répondre à ces demandes. Si c'est le cas, ils envoient leurs propositions au courtier. Ils se composent des événements registerEvent, cancelEvent, respondEvent, inform_doneEvent et des plans checkPlan, accept_proposalPlan, reject_ProposalPlan expliqués plus ci-dessous.

8.2.2 Events (événements)

- **Register Event:** lorsqu'un agent souhaite être considéré comme performeur potentiel (participant) auprès d'un courtier, il envoie un tel acte d'inscription à courtier.
- **Cancel Event:** lorsqu'un participant souhaite ne plus être considéré comme performeur potentiel, il envoie un tel acte d'annulation au courtier.
- **ProxyEvent:** lorsque l'initiateur veut envoyer une demande de service à des participants enregistrés auprès d'un courtier, il envoie à ce courtier un tel acte imbriquant un acte de communication tel qu'un Cfp (Call for proposal) destinés aux participants de ce courtier.
- **RefuseEvent:** si l'initiateur envoie une demande de service à médier à un courtier qui ne peut le faire, celui-ci refuse la demande par un tel acte.

- **AgreeEvent:** au contraire, si le courtier est capable de médier cette demande, il l'accepte à l'initiateur avant d'en assurer l'envoi aux participants enregistrés auprès de lui.
- **Failure_no_matchEvent:** après acceptation (événement précédent), le courtier vérifie le contenu du message *proxy* (ProxyEvent) c'est-à-dire l'acte de communication lui-même, et cherche des participants enregistrés susceptibles de pouvoir y répondre pour le leur envoyer. Si le courtier n'en trouve pas, il enverra un tel acte à l'initiateur.
- **Cfp Event:** si le courtier trouve des participants adéquats, il leur envoie l'acte de communication qui est contenu dans le *proxy* envoyé par l'initiateur. Si cet acte de communication correspond au protocole FIPA ContractNet, le courtier enverra l'appel à proposition aux participants correspondants.
- **Respond Event:** après avoir reçu l'appel à proposition, un participant vérifie s'il peut y répondre. Si c'est le cas, il propose une solution, sinon, il refuse.
- **Proposal Event:** lorsque le moment-butoir contenu dans le message *proxy* est atteint, le courtier choisit la meilleure proposition pour l'envoyer en retour à l'initiateur.
- **Reject_proposal Event:** le courtier envoie cet acte de rejet aux participants dont la proposition n'a pas été choisie.
- **Accept_proposal Event:** l'initiateur contacte l'agent participant dont la proposition est choisie comme performeur pour la suite par un tel acte d'acceptation de proposition.
- **Inform_done Event:** si le participant est toujours capable d'assumer sa proposition il signifie son accomplissement à l'initiateur lorsqu'elle est réalisée.
- **Failure_proxyEvent:** si la médiation du message proxy échoue, le courtier le signifie à l'initiateur par cet acte.
- **Inform_done_proxyEvent:** si le courtier réussit à effectuer le sous-protocole imbriqué dans le message *proxy*, il le signifie à l'initiateur par cet acte.
- **Failure_sub_protocolEvent:** si le courtier ne reçoit que des refus de tous les participants, il le signifie à l'initiateur par cet acte.
- **Inform_result_subEvent:** le courtier envoie à l'initiateur les informations concernant le participant qui a fait la meilleure proposition par cet acte.

8.2.3 Plans

- **Register Plan:** lorsque le courtier reçoit l'inscription d'un participant, il l'enregistre dans une base de données.
- **CancelPlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Cancel: lorsque le participant annule

sa participation, le courtier l'élimine de sa base de données de participants. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.

- **Proxy Plan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Proxy: après avoir reçu le message *proxy*, le courtier vérifie qu'il peut médier cette demande de service. Si c'est possible, il envoie l'acte de communication *Agree* (Agree Event) à l'initiateur, et fait suivre la demande, par exemple un appel à proposition aux participants. Il attendra le dépassement du moment-butoir pour choisir la meilleure proposition. Cet événement Proxy pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **AgreePlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Agree: l'initiateur s'en sert pour recevoir du courtier l'acceptation de médier la demande de service. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **RefusePlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement RefuseEvent: l'initiateur s'en sert pour recevoir du courtier le refus de médier la demande de service. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Failure_no_matchPlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Failure_no_matchEvent: l'initiateur s'en sert pour recevoir du courtier cet acte d'échec quant à trouver des participants susceptibles de répondre à la demande. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Check Plan:** ce plan participe concrètement, comme intention, à l'accomplissement du désir exprimé en termes BDI par l'événement CfpEvent: après avoir reçu un CfpEvent, un participant vérifie, par ce plan, ses bases de données de manière à déterminer s'il est capable de répondre au courtier. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **RespondPlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Respond. Après avoir reçu la réponse des participants, s'il y a plusieurs propositions, le courtier choisit les propositions satisfaisantes et les stocke dans sa base de données. Quand le moment-butoir passe, le *proxyPlan* du courtier utilise ces données pour prendre une décision. Le courtier choisira la meilleure proposition et l'enverra à l'initiateur. Il refusera les autres propositions. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Reject_ProposalPlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Reject_proposal: le participant l'utilise pour recevoir le rejet du courtier. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.

- **ProposalPlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Proposal: l'initiateur l'utilise pour recevoir la meilleure proposition choisie par le courtier. L'initiateur enverra alors directement l'acte d'acceptation au participant. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Accept_proposalPlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'évenement Accept_proposal: le participant reçoit cet événement d'acceptation de sa proposition, il vérifie alors s'il peut ou il veut toujours effectuer ce service. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Failure_proxyPlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Failure_Proxy. L'initiateur l'utilise pour recevoir cet événement envoyé par le courtier s'il y a échec quant à la médiation de service aux participants. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Inform_done_proxyPlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Inform_done_proxy. L'initiateur l'utilise pour recevoir cet événement envoyé par le courtier lorsqu'il a effectivement choisit la meilleure proposition. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Failure_sub_protocolPlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Failure_sub_protocol. L'initiateur l'utilise pour recevoir cet événement envoyé par le courtier lorsque le sous-protocole échoue ou qu'aucun participant ne répond. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.
- **Inform_result_subPlan:** ce plan implémente comme intention le désir exprimé en termes BDI par l'événement Inform_result_sub. L'initiateur l'utilise pour recevoir les informations concernant le sous-protocole. Cet événement pourrait être réalisé par d'autres plans ajoutés ultérieurement.

8.2.4 Base de données (croyances)

- **Condition BD:** base de données utilisée pour stocker les conditions du message *proxy* envoyé par l'initiateur.
- **Items_Participant BD:** base de données utilisée pour stocker les données des participants.
- **Participants BD:** base de données utilisée pour stocker les participants inscrits auprès du courtier.
- **Reponses BD:** base de données utilisée pour stocker les propositions avant que le moment-butoir soit dépassé.

8.2.5 Java Ressources

- **Broker_Frame:** application GUI pilotant les interactions entre le courtier et les participants d'une part et le courtier et l'initiateur d'autre part.
- **Act Java:** l'application Courtier développée peut faire le courtage impliquant n'importe quel protocole FIPA (ou autre). Lorsque l'initiateur envoie une demande de proposition au courtier, il peut imbriquer, dans son message poxy au courtier, n'importe quel protocole d'interaction comme ceux vus précédemment - FIPA ContractNet, FIPA Iterated ContractNet – ou même récursivement un FIPA Broker. Afin de proposer une application générique, lorsque l'initiateur envoie une demande "proxy" - condition, description, et type d'acte (Act) -, Act est le contenu de la demande de l'initiateur. Ce contenu dépend du protocole d'interaction utilisé. Le courtier médiera la demande selon ce protocole.
- **Cfp Java:** dans le cadre de notre application Courtier, nous utilisons comme protocole d'interaction imbriqué le FIPA ContractNet. L'appel à proposition Cfp constitue la demande dans ce protocole. Celle-ci est implémentée comme sous type de **Act** expliqué ci-dessus. Lorsque le courtier reçoit un **Act**, il utilisera son type (en l'occurrence ici Cfp du FIPA ContractNet) pour médier la demande aux participants correspondants par rapport à la spécification du protocole imbriqué. Dans notre application, lorsque le courtier reçoit la demande de service imbriquant un Cfp ContractNet, il le déduit par late-binding et médie l'appel à proposition aux participants adéquats par rapport à la spécification du protocole imbriqué.
- **Launch Java:** lance les ressources et démarre l'application du Courtier présentée ci-dessous.

8.3. L'application

Le premier écran (figure 16) gère la relation entre le courtier et les participants. Les participants peuvent s'inscrire auprès d'un courtier et se désinscrire. L'utilisateur peut charger, modifier et sauvegarder les base de données des participants.

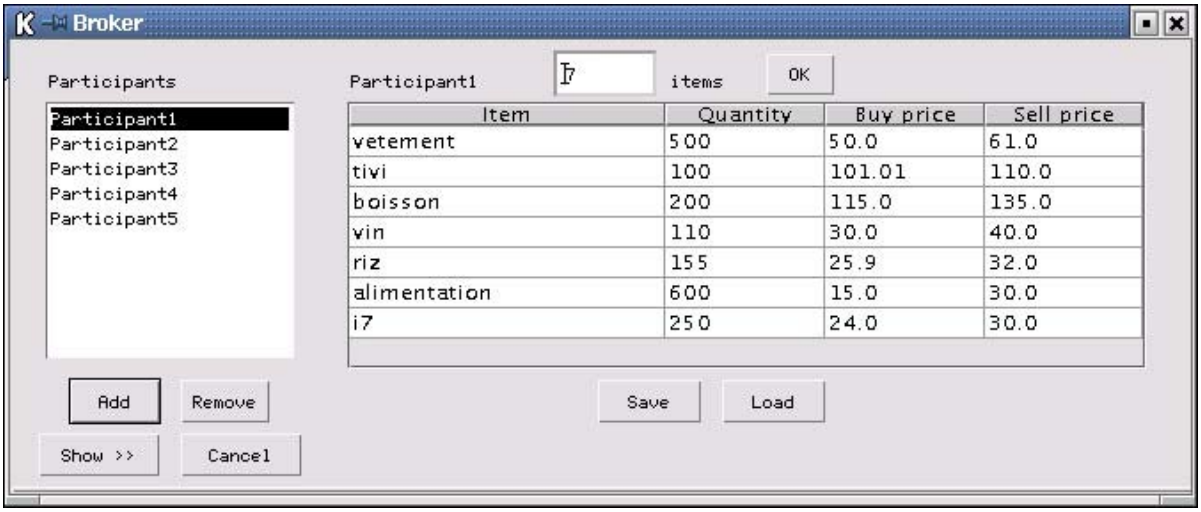

Figure 16: Interface de l'initiateur du Courtier

L'utilisateur clique sur le bouton Show pour afficher l'écran gérant l'interaction entre le courtier et l'initiateur.

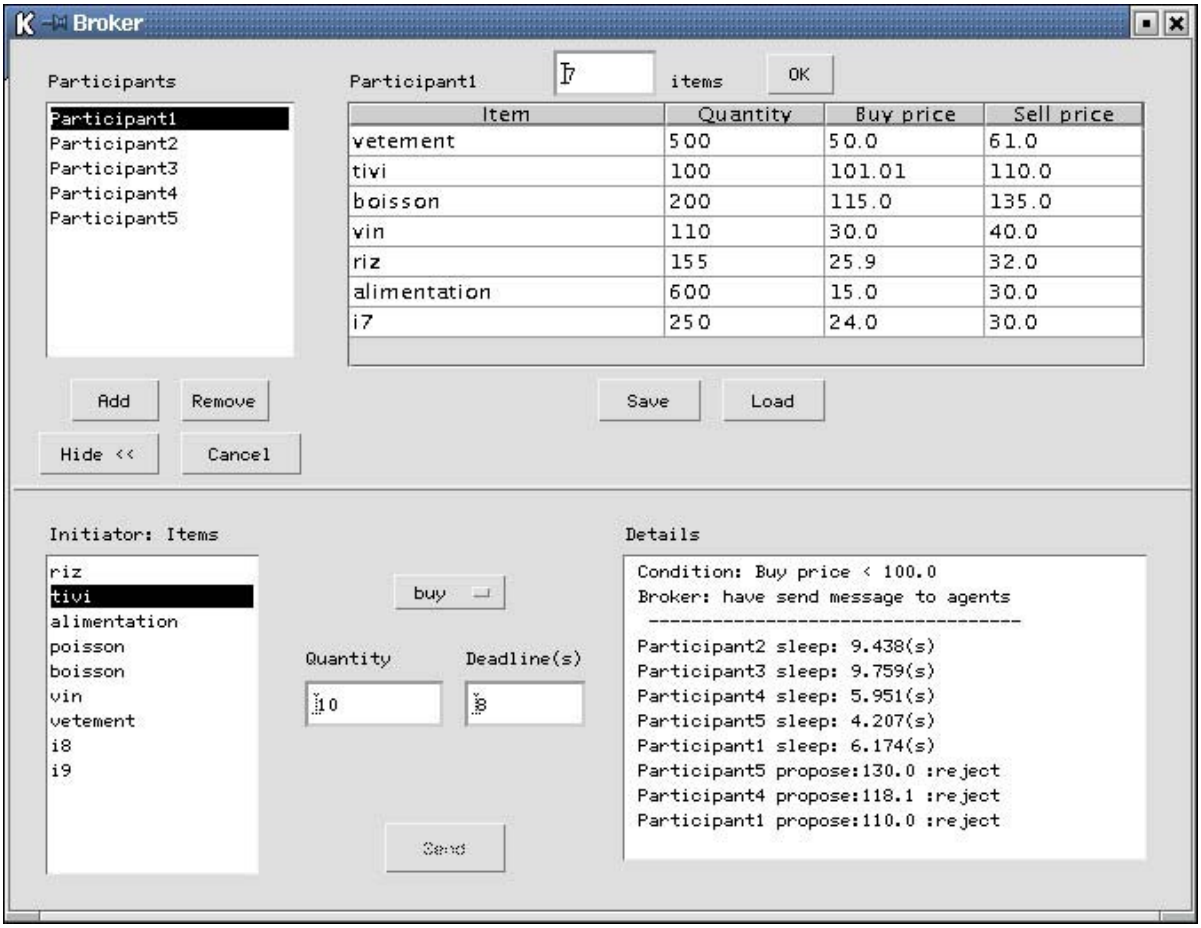

Figure 17: Interface principale du Courtier

L'utilisateur joue le rôle de l'initiateur en cours de traitement. Chaque envoi d'une demande de service à médier nécessite de paramétrer chacun des champs

suivants propres à cette demande (cf figure 17): type d'item (par sélection), quantité et moment-butoir (valeurs numériques) et achat/vente (par sélection). A chaque soumission d'une demande à médier, l'utilisateur peut toujours modifier les bases de données des participants, et reparamétrer les différents champs. Les informations concernant l'interaction s'affichent dans la fenêtre de dialogue dans le coin inférieur droit.

9. Conclusion

Tout concepteur (de logiciel ou autre) se réfère à des patrons, styles ou idiomes pour décrire les modèles et structures qu'il construit. Une des idées majeures de la méthodologie Tropos est de proposer de développer des architectures multi-agent comme des organisations sociales et intentionnelles d'agents interagissant entre eux pour accomplir les buts du système à développer. Dans ce contexte, Tropos défini une ontologie agent à trois niveaux : 1) éléments atomiques de base, 2) patron de conception à ancrage social, 3) styles organisationnels. Ce travail se focalise sur le deuxième niveau. Il identifie des patrons de conception à ancrage social à utiliser dans Tropos, en propose certains (appel d'offre, mise au enchères et courtier) développés sur l'environnement agent JACK en utilisant le modèle théorique BDI et en tenant compte des spécifications FIPA.

Les directions de recherche futures incluent la formalisation avec précision comme meta-structures de ces patrons de conception à ancrage social ainsi que la définition d'un catalogue structuré les inventoriant. Ces patrons devront aussi être comparés et contrastés par rapport aux patrons de conception classiques (voir par ex. [8]). Le rapport entre patrons de conception à ancrage social et styles architecturaux organisationnels [16] ou même classique [17] au sein de l'ontologie propre à Tropos devra également être formalisé de manière à spécifier comment les premiers expriment les seconds, notamment lors du passage de la phase de conception architecturale à la phase de conception détaillée.

Une autre piste de recherche future est le développement à partir du présent travail d'une plateforme agent point-à-point de type Gnutella pouvant assurer des services e-business et basé sur les normes FIPA décrites dans la section 5. Plus particulièrement le développement d'agents ACC (Agent Communication Channel) pour la communication agent, AMS (Agent Management System) pour la gestion agent et DF (Directory Facilitator) pour la fourniture de services répertoriés à d'autres agents.

10. Références

[1]. J. Castro, M. Kolp, et J. Mylopoulos. “Towards Requirements-Driven Information Systems Engineering: the Tropos Project”. Dans *Information Systems* (27), Elsevier, Amsterdam, Pay-Bas, 2002.

[2]. L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 1998.

[3]. L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Publishing, 2000.

[4]. A. Dardenne, A. van Lamsweerde, and S. Fickas. “Goal-directed requirements acquisition”. Dans *Science of Computer Programming*, 20(1–2):3–50, 1993.

[5]. E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, University of Toronto, Department of Computer Science, Canada, 1995.

[6]. Y. Lespérance, T. Kelley, J. Mylopoulos, and E. Yu. « Modeling Dynamic Domains with ConGolog”. Dans *Proceedings of the 11th International Conference on Advanced Information Systems Engineering* (CAiSE’99), pages 108–123, Heidelberg, Allemagne, Juin 1999.

[7]. M. Kolp, P. Giorgini, et J. Mylopoulos. “Organizational Multi-Agent Architecture: A Mobile Robot Example”. Dans *Proceedings of the 1st International Conference on Autonomous Agent and Multi Agent Systems* (AAMAS'02), Bologna, Italie, Juillet 2002.

[8]. E. Gamma, R. Helm, R. Johnson, et J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*, Addison-Wesley, 1995.

[9]. Y. Aridor, et D. B. Lange “Agent Design Patterns: Element of Agent Application Design”. Dans *Proceeding of 2nd International Conference on Autonomous Agents* (Agents’98), ACM Press, St. Paul, Minnepolis, USA, 1998.

[10]. E. Kendall, P.V. Murali Krishna, C. V. Pathak, et C. B. Suersh “Patterns of Intelligent and Mobile Agents”. Dans *Proceedings of the 2nd International Conference on Autonomous Agents* (Agent’98), pages 92-99, ACM Press, St. Paul, Minnepolis, USA, 1998.

[11]. S. Hayden, C. Carrick, et Q. Yang. “Architectural Design Patterns for Multiagent Coordination”. Dans *Proceedings of the 3rd International Conference on Agent Systems*, (Agents’99), Seattle, WA, USA, Mai 1999.

[12]. S. G. Woods et M. Barbacci. “Architectural Evaluation of Collaborative Agent-Based Systems”. Rapport Technique, CMU/SEI-99-TR-025, Software Engineering Institute, Carnegie Mellon University, PA, USA, 1999.

[13]. M. Kolp, P. Giorgini, et J. Mylopoulos. “A Goal-Based Organizational Perspective on Multi-Agent Architectures”. Dans J.-J. C Meyer et M. Tambe (Eds.) *Intelligent Agents VIII – Agent Theories, Architectures, and Languages*, pages 128-140, Springer, 2002.

[14]. A. Fuxman, P. Giorgini, M. Kolp, et J. Mylopoulos. “Information Systems as Social Structures”. Dans *Proceedings of the 2nd International Conference on Formal Ontologies for Information Systems* (FOIS’01), Ogunquit, WA, USA, Octobre 2001.

[15]. M. Kolp, P. Giorgini, et J. Mylopoulos. “Information Systems Development through Social Structures”. Dans *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering* (SEKE’02), Ishia, Italie, Juillet 2002.

[16]. P. Giorgini, M. Kolp, et J. Mylopoulos. “Multi-Agent Architecture as Organizational Structures”. A paraître dans *International Journal of Cooperative Information Systems* (IJCIS), World Scientific, 2002.

[17]. M. Shaw et D. Garlan. “Software Architecture: Perspectives on an Emerging Discipline”, Prentice Hall, 1996.

[18]. C. Iglesias, M. Garrijo, et J. Gonzalez. “A survey of agent-oriented methodologies”. Dans *Proceeding of the 5th Int. Workshop on Intelligent Agents: Agent Theories, Architectures, and Languages, ATAL’98*, page 317-330, Paris, France, Octobre 1999.

[19]. James Odell, H. Van Dyke Parunak, Bernhard Bauer, "Extending UML for Agents". Dans *Proceedings of the 2nd International Bi-Conference Workshop on Agent-Oriented Information Systems,* (AOIS'00), Austin, TX, USA, Juillet 2000

[20]. FIPA - Foundation for Intelligent Physical Agents (voir http://fipa.org/).

[21]. FIPA97 Specification Part 1 Agent Management, Issue 1 (voir http://www.iit.upco.es/~luisf/programacion/fipa/Spec.www/fipa97/f8a21.doc).

[22]. FIPA97 Specification Part 2 Agent Communication, Issue 1 (voir http://www.iit.upco.es/~luisf/programacion/fipa/Spec.www/fipa97/fipa8a22.doc)

[23]. FIPA97 Specification Part 3 Agent Software Integration, Issue 1 (voir http://www.iit.upco.es/~luisf/programacion/fipa/Spec.www/fipa97/f7a13.doc).

[24]. FIPA97 Specification Part 8 Human-Agent Interaction, Issue 1 (voir http://www.iit.upco.es/~luisf/programacion/fipa/Spec.www/fipa98/fipa8a24.doc)

[25]. Finin T. et al: “KQML as an agent communication language”, Dans Bradshaw J (Ed.), *Software agents*, MIT Press, 1997.

[26]. Smith, R. G. “The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver”. Dans *IEEE Transactions on Computers,* 29 (12): 1104-1113, Décembre 1980.

[27]. FIPA Contract Net Interaction Protocol Specification http://www.fipa.org/specs/fipa00029/XC00029F.pdf

[28]. FIPA Iterated Contract Net Interaction Protocol Specification http://www.fipa.org/specs/fipa00030/PC00030D.pdf

[29]. FIPA Brokering Interaction Protocol Specification http://www.fipa.org/specs/fipa00033/XC00033E.pdf

[30]. P. D. O’Brien, R. C. Nicol: “FIPA – Towards a standard for software agents”, Dans *BT Technol J* 16 (3) Juillet 1998.

[31]. Agent Oriented Software Pty. Ltd., “JACK Intelligent Agents™ User Guide”, Release 3.5, 05 Mars 2002.