Une approche Multi-Agents des systèmes de bureautique communicante[1]

Bruno Dillenseger (31 75 91 39, dillenseger@sept.fr),
François Bourdon (31 75 91 19, bourdon@sept.fr),
SEPT SCE/ARC, 42 rue des Coutures, BP 6243, F-14066 Caen cedex (fax : 31 75 06 31).

MOTS-CLÉS : système multi-agents, agent cognitif, bureautique communicante, système réparti orienté objets, négociation de service, coopération, Prolog, COOL, C++.

RÉSUMÉ

Après avoir esquissé les besoins en répartition et en autonomie des systèmes d'information, cet article se propose de décrire une modélisation multi-agents des systèmes de bureautique communicante et une implémentation dans le cadre d'une application concrète. Cette mise en œuvre est réalisée avec PUMA, une plate-forme basée sur l'intégration d'un interpréteur Prolog à l'environnement réparti à objets COOL, que nous présentons comme un outil d'IAD. Nous concluons en discutant des ouvertures apportées par le modèle multi-agents et le principe de l'outil.

1. Introduction

1.1 Expression des besoins

L'autonomie devient de plus en plus un élément fondamental dans l'adaptation des systèmes informatiques à l'évolution imprévisible [14] de l'entreprise, dont l'environnement est géographiquement réparti. Cette évolution imprévisible des comportements réels de l'entreprise, contraint les systèmes informatiques à s'auto-adapter aux changements organisationnels et opérationnels induits.

Dans la période de transition que nous vivons [6], suivant le degré de visibilité offert par les applications sur leurs potentialités fonctionnelles, l'utilisateur ne voit plus systématiquement des applications monolithiques mais prend conscience d'objets interagissant. Le potentiel de chaque objet s'exprime en terme de services autonomes, susceptibles d'aider l'utilisateur dans les tâches qu'il doit accomplir.

Devant la profusion de services qui apparaissent, disparaissent et évoluent au sein de l'entreprise, l'utilisateur doit pouvoir être guidé pour trouver les plus pertinents vis-à-vis de ses besoins du moment. Cette assistance nécessite des échanges entre entités informatiques autonomes, qui ne peuvent être réalisées que par l'intermédiaire d'un langage d'interface normalisé susceptible, comme le disent Jean Erceau et Michel Barat à travers leur *Principe intégrateur* [3], "*de déterminer un univers sémantique minimum pouvant être partagé par l'émetteur et le récepteur d'un message et plus généralement par les différents agents du système amenés à communiquer*".

Outre une représentation fonctionnelle des services offerts et/ou demandés, cette recherche doit s'accompagner de possibilités de négociation pour adapter les requêtes aux offres du "marché", d'apprentissage (mémoire) pour ne pas recommencer systématiquement des recherches inutiles, et de communications sophistiquées offrant des possibilités d'intervention manuelle (intervenant humain) en cas de négociations délicates.

---

1. En collaboration avec le LAIAC, Université de Caen.

1.2 Des outils et des techniques

Les systèmes distribués alliés aux méthodologies objets permettent de construire des applications comme des collections d'entités indépendantes, en interaction, et pouvant être réparties physiquement. Ce mode de conception est particulièrement adapté pour traiter de nombreuses familles de problèmes qui sont répartis par nature. Il en est ainsi pour les études menées au SEPT[2] dans le domaine du "Computer Supported Cooperative Work" sur la circulation de document [5] ou la rédaction coopérative.

Dans un premier temps, la conception de telles applications a mis en évidence la nécessité d'un système réparti à objets. Ceci a conduit à l'adoption d'un système distribué basé sur la technologie micro-noyau Chorus et à la spécification puis au développement de la couche de communication orientée objets COOL [9], intégrée à C++. Mais dans un second temps, il paraît crucial d'enrichir les interactions et les comportements des entités afin de renforcer leur autonomie, leur capacité d'adaptation et d'évolution dans un système dynamique.

1.3 Vers une approche "agents"

Parmi les nombreuses questions étudiées en Intelligence Artificielle Distribuée telles que la coopération [13], la génération et l'exécution de plan distribuées [11], la représentation des connaissances et des croyances [10], figure en toile de fond un problème fondamental : lorsqu'un agent doit effectuer une tâche qu'il ne sait pas résoudre seul, par quel moyen peut-il trouver l'agent coopérant qui répondra au mieux à ses besoins?

Ce point est essentiel pour les systèmes répartis à objets, qui peuvent être vus comme des cas particuliers de systèmes multi-agents : lorsqu'un objet client cherche à invoquer une méthode sur un objet serveur distant, il doit utiliser des protocoles et des structures particulières dont le rôle est de mettre en relation les deux objets. Dans les architectures ODP[3], par exemple, un client peut trouver le serveur ad hoc en invoquant un courtier ("trader")[1], sorte d'annuaire de services auprès duquel les serveurs déclarent les services qu'ils exportent.

La pertinence de l'approche multi-agents se justifie également par un aspect "génie logiciel", plus récent, avec la recherche d'une méthodologie orientée agents [7]. Le concept de *programmation orientée agents* est présenté dans [12] comme une spécialisation de la programmation orientée objets (prise en compte des notions de croyances, décision, obligation; types fondamentaux de communication inspirés de la théorie des actes de langages).

2. Une modélisation multi-agents

2.1 Obtenir un système uniforme

Pour apporter un premier niveau de réponse à la nécessité d'un principe intégrateur, nous adoptons une vision uniforme du système d'information de l'entreprise : le réseau relie un ensemble de machines (*sites*) qui disposent de certaines *ressources* susceptibles d'apparaître, disparaître ou évoluer : programmes d'application (e.g. base de données, éditeurs divers...) ou entités applicatives nomades (e.g. objet "document électronique" en circulation), périphériques

2. Services d'Etudes communes de La Poste et de France Télécom
3. Open Distributed Processing

(e.g. imprimantes), utilisateurs connectés. A chacune de ces ressources correspond un ensemble propre de *services* disponibles auxquels sont associées certaines *caractéristiques* :

- Les *caractéristiques structurelles*, statiques ou faiblement dynamiques en cas de mise à niveau, sont propres à l'implémentation particulière d'un service. Par exemple, pour un service d'impression, il peut s'agir de la marque de l'imprimante, du procédé utilisé, des formats reconnus, de la vitesse d'impression...
- Les *caractéristiques conjoncturelles*, typiquement dynamiques, dépendent du contexte d'invocation du service. Dans l'exemple du service d'impression, il peut s'agir du nombre d'impressions en attente, de l'indisponibilité pour cause de maintenance, de l'indice d'usure du dispositif d'impression, du nombre de feuilles disponibles...

La grande interopérabilité requise doit tirer partie d'une intégration des ressources dans une représentation uniforme du système. Pour cela, on associe un "agent" à chaque ressource, chargé de la représenter vis-à-vis des autres ressources. Ceci permet de créer une couche de coopération uniforme résultant de la projection des entités du système (cf figure 1). C'est dans cette couche qu'ont lieu les recherches de serveurs et les négociations de service.

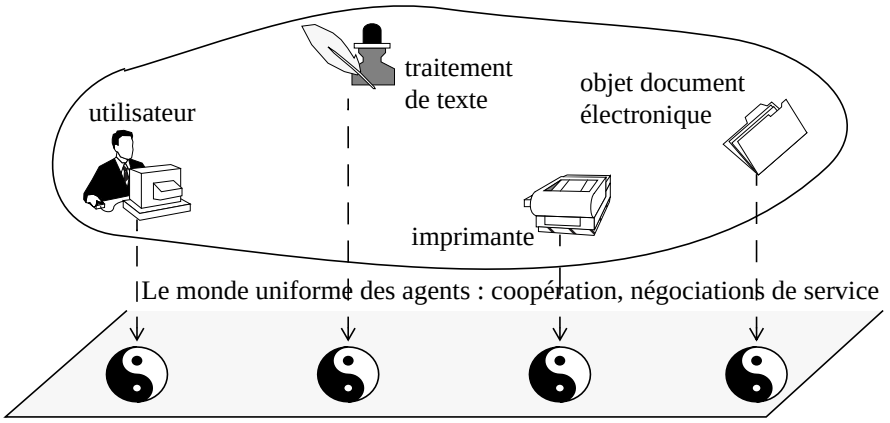

figure 1: représentation uniforme du système

Représentant et contrôleur unique d'une ressource et de ses services, l'agent constitue une double interface :

- il agit en *client* lorsqu'il recherche les serveurs et négocie les services;
- il se comporte en *serveur* quand il répond aux invocations de services d'agents clients.

2.2 Coopération multi-agents

L'agent est doté d'un comportement qui le met en permanence à l'écoute des sollicitations internes (i.e. émanant de la ressource qu'il représente) et externes (i.e. provenant des autres ressources du système). Ce rôle de négociateur implique une connaissance de certaines informations sur sa ressource et les services qu'elle fournit, sur le système en général et certaines ressources en particulier. Il possède également une activité propre qui consiste, par exemple, à surveiller le déroulement des tâches dont il est responsable, ou à réorganiser ses connaissances en vue d'un éventuel "apprentissage".

Lorsqu'une ressource a besoin d'un service proposé par une autre ressource, elle doit surmonter deux difficultés :

(1) entrer en relation avec les ressources ad hoc dans un système dynamique;

(2) exprimer de la façon la plus précise possible le service demandé.

Le point (1) nécessite la mise en place de structures particulières et de protocoles associés, dont le rôle est de fournir des clients à un serveur et d'indiquer des serveurs à un client. On peut évidemment songer à l'annuaire de services et mettre en place des objets courtiers mais

d'autres métaphores sont envisageables (e.g. analogie avec la publicité, les petites annonces, les pages jaunes...). Nous décrivons le fonctionnement d'une structure de ce type en 4.2.

Quant au point (2), il implique le partage d'un certain niveau de langage et de sémantique entre les agents. Dans notre modèle, la coopération est basée sur la requête de service. Celle-ci s'exprime par un identificateur non ambigu de type de service (e.g. imprimer) et une liste de contraintes que doivent vérifier certaines caractéristiques (e.g. procédé laser, résolution supérieure à 400 ppp). Il y a donc partage de sémantique concernant les types de service et les caractéristiques. Cependant, il n'est pas supposé que les agents détiennent la connaissance globale de tous les services et de toutes les caractéristiques.

3. Des outils pour implémenter la modélisation

3.1 Un système réparti à objets : COOL

La couche COOL[(4)] a été spécifiée par le SEPT pour permettre la répartition effective d'applications telles que la Circulation Intelligente de Dossiers REpartis. COOL v1 encapsule les fonctionnalités de communication offertes par le micro-noyau de système réparti Chorus dans une couche orientée objets exploitable en C++ par l'utilisation d'une classe `COOL`.

Toute instance de cette classe possède des méthodes de communication bloquante (call/reply) ou non bloquante (send/receive), de gestion de groupes de communication, de migration entre sites du réseau. De plus, elle peut invoquer des méthodes sur d'autres objets (localement avec COOL v1 ou de façon répartie et transparente avec COOL v2, en cours de développement) et elle dispose d'objets sémaphores pour régler les problèmes d'accès concurrents. Par ailleurs, les objets COOL peuvent être passifs ou exécuter une activité propre. Enfin, le système de communication est complété par un service de nommage symbolique réparti grâce auquel les objets peuvent se désigner par un nom choisi.

3.2 Introduction d'un langage de haut niveau : Prolog

L'utilisation de C++ se justifie par son orientation objets et son efficacité issue de C, langage traditionnel des systèmes UNIX. Mais il se révèle limité ou inadapté à certains de nos besoins (représentation des connaissances, langage de haut niveau pour la négociation, comportements "intelligents" et adaptatifs). Le rapprochement effectué en direction de l'Intelligence Artificielle Distribuée, suite à la modélisation multi-agents, conduit à s'intéresser à d'autres langages couramment mis en œuvre dans ce domaine. Pourtant, on ne peut se passer des fonctionnalités offertes par COOL, d'où l'idée d'intégrer un langage de ce type dans un objet C++.

Le choix de Prolog découle de plusieurs considérations. D'un point de vue général, il s'agit d'un langage interprété, de haut niveau, disposant d'une représentation uniforme des données et des programmes. Par ajout de prédicats de communication, les programmes peuvent alors non seulement s'échanger du savoir mais aussi du savoir-faire. Dans le cadre de notre modélisation, cela permet de propager des procédures nouvelles, ou des modifications d'anciennes, aux agents qui peuvent se mettre à jour dynamiquement, sans interruption de leur fonctionnement. En outre, ce mécanisme fonctionne aussi en milieu hétérogène.

De plus, Prolog fournit une base de langage de communication commune à tous les agents. Conçu à l'origine comme un outil d'analyse du langage naturel, Prolog est un support particulièrement efficace pour définir et interpréter de nouveaux langages spécifiques aux

4. Chorus Object Oriented Layer

besoins rencontrés (e.g. définition d'un langage d'expression de contraintes pour la négociation de service). Enfin, les agents ne peuvent que profiter de la puissance de l'unification pour conduire des raisonnements plus ou moins élaborés.

Une formalisation de type multi-agents du "Human Computer Cooperative Work" a déjà été l'objet du projet IMAGINE [8]. Il a donné lieu à un environnement et une boîte à outils "multi-agents" basés sur un Prolog parallèle. L'approche est similaire à la nôtre : mêler les travaux en matière de CSCW, dont le but est d'assister la coopération entre agents humains par des ressources informatiques, avec les modèles anthropomorphiques de coopération développés en IAD, afin de faciliter les interactions et la coopération entre utilisateurs et applications, en les intégrant au sein d'une même modélisation. Mais ces recherches ne bénéficient pas du support d'un système réparti à objets tel que COOL.

3.3 Prolog pour Univers Multi-Agents

3.3.1 Un interpréteur Prolog dans un objet COOL

PUMA est le résultat de l'intégration d'un Prolog de type Edimbourg dans un objet COOL. Notre objectif principal n'était pas de choisir la "meilleure" implémentation de Prolog mais de prouver la faisabilité et de montrer les atouts d'un telle approche. Néanmoins, il sera nécessaire dans une étape ultérieure de faire le point sur les différentes versions de Prolog et également sur les plate-formes d'IAD pour perfectionner notre maquette. L'expérience acquise nous éclairera dans ces investigations.

D'un premier point de vue, on peut considérer PUMA comme l'intégration d'un interpréteur Prolog dans un objet C++ en environnement réparti. La classe de base `puma` possède l'interface complète d'accès et de contrôle du noyau Prolog mais ne définit pas d'activité. Son constructeur prend en argument le nom symbolique de l'objet et le nom du fichier Prolog initial à charger. Cette classe est essentiellement destinée à être utilisée par des sous-classes opérationnelles via héritage. Elle peut aussi être exploitée directement pour créer un objet serveur Prolog qui répond aux invocations de méthodes ou exécute son réflexe de création s'il est défini dans la base initiale.

Deux classes sont dérivées de `puma.` La classe `interp` définit une activité d'interpréteur interactif et ses instances se comportent donc exactement comme des interpréteurs Prolog disposant de tous les mécanismes de COOL intégrés dans de nouveaux prédicats. Cette classe est particulièrement utile pour tester, observer, mettre au point d'autres objets dérivant de `puma`, voire, dans une moindre mesure, tout objet `COOL`. La classe `agent` définit une activité générique d'agent qui consiste en l'invocation répétée et continue d'un prédicat représentant une étape élémentaire de son activité, ce prédicat devant être programmé dans la base Prolog chargée à la création de l'agent. Typiquement, il consiste à lire un message dans la boîte aux lettres et à le traiter. Eventuellement, il peut y avoir une tâche propre à mener mais il est recommandé que chaque étape élémentaire soit de courte durée afin de garder l'agent suffisamment disponible et d'éviter un débordement de la boîte aux lettres. Pour assurer le suivi d'une tâche longue dans le temps, l'agent peut recourir à la création d'un autre agent qui s'occupera spécifiquement de cette tâche.

Enfin, toute classe dérivant de COOL peut s'adjoindre de façon quasi-transparente un agent PUMA en héritant de la classe `objetSMA`. L'instance d'une classe dérivant de `objetSMA` possède les attributs nécessaires et suffisants afin d'invoquer cet agent de façon synchrone ou asynchrone. De plus, l'agent suit automatiquement l'objet auquel il est associé au cours de ses migrations éventuelles et il dispose de réflexes programmables (réflexes de création, de migration et de disparition). L'utilisation de cette classe correspond au principe de "projection" des ressources d'un système dans l'univers homogène des agents.

3.3.2 Un dialecte Prolog étendu

On peut également voir PUMA comme l'intégration de COOL dans un langage Prolog, puisque la majorité des fonctionnalités de COOL sont encapsulées dans de nouveaux prédicats, dont la plupart est dédiée à la communication : nommage symbolique, communication asynchrone avec fonctionnalités de groupe, gestion des groupes, communication synchrone, migration, création d'objets COOL (donc en particulier d'objets PUMA).

En outre, un avantage particulier apparaît dès que l'on désire utiliser des objets nomades actifs. En effet, la migration d'activité est un problème complexe (e.g. problème de la migration de la pile d'invocations qui peut concerner plusieurs objets) qui n'est pas réglé dans COOL : l'activité d'un objet COOL est redémarrée à zéro après chaque migration. En revanche, la migration est transparente dans tout programme PUMA qui invoque le prédicat migrate/1, grâce au mécanisme de sauvegarde/restauration de l'état Prolog.

3.3.3 Interopérabilité C++ - Prolog

La dualité de l'intégration permet de concevoir des objets composites dont l'activité est un programme C++ faisant des requêtes Prolog, un programme Prolog invoquant des méthodes C++ ou bien une combinaison des deux. Le tableau 1 présente les prédicats et méthodes d'invocation entre C++ et Prolog. D'un point de vue général, cette possibilité est intéressante

↱	C++/COOL		PUMA	
invocation	*synchrone*	*asynchrone*	*synchrone*	*asynchrone*
C++	invocation standard de méthode	`objectSend()`	`puma_call()`	`objectSend()`
Prolog	interrupt/1	send/2	phone/1	send/2

tableau 1 : interopérabilité entre des programmes C++ et Prolog

simplement du fait que ces langages sont adaptés à des types de tâches distincts.

4. Un exemple d'application

4.1 Réservation de salle multi-agents

4.1.1 Présentation générale

Il s'agit de mettre en œuvre la modélisation exposée jusqu'ici, à l'aide des outils présentés, dans un cas concret et simple. Nous allons donc décrire la réalisation multi-agents d'un système de réservation de salles. Ce système met en jeu deux types principaux d'agents.

Chaque *agent salle* représente de façon exclusive une salle et maintient un ensemble d'informations sur ses caractéristiques structurelles (équipements divers, lieu, dimensions...) ou conjoncturelles (planning des réservations, travaux...). L'agent sert d'interface complète pour l'accès aux services typiques d'une salle qui sont : réservation et annulation.

Lorsqu'un utilisateur désire réserver une salle, il a accès au service correspondant via son *agent utilisateur* qui se charge de contacter les serveurs ad hoc (i.e les agents salles). Cet agent est donc non seulement chargé de représenter l'utilisateur auprès du système, mais il constitue également le point d'entrée vers l'ensemble des services accessibles par l'utilisateur.

4.1.2 Les agents

Chaque agent utilisateur et chaque agent salle est un agent *permanent* car il représente une ressource. Il doit offrir une interface de contrôle minimale qui permet à son administrateur de le déplacer sur un autre site ou de le supprimer. L'agent et l'interface de contrôle associée forment une entité composite, implémentée par la classe `interface` qui hérite de `objetSMA`.

Deux activités parallèles sont ainsi définies : l'activité de l'agent (typiquement codée dans un programme Prolog), et l'activité de l'interface qui capte les évènements provenant de l'utilisateur. L'interface de contrôle permet également à l'utilisateur d'invoquer l'agent sous-jacent. Chaque agent, en fonction du type de la ressource associée, propose l'accès à un certain nombre de services disponibles localement ou via le système global. Un service est *interne* ou *externe* selon que l'agent peut l'exécuter de façon autonome ou qu'il doit invoquer un autre agent.

Tout agent permanent charge une base Prolog particulière qui défini deux services internes : la visualisation et l'édition des caractéristiques de l'agent. Les agent salles, en consultant une base spécifique, apprennent des services internes supplémentaires tels que la réservation de salle, l'annulation de réservation ou la consultation du planning. Quant aux agents utilisateurs, une base spécifique leur fournit un service interne de mémento, mais aussi deux services externes : la réservation de salle et l'annulation. Grâce à la programmation déclarative des services en Prolog, les agents sont capables de mettre à jour leurs services ou d'en apprendre de nouveaux de façon dynamique, et de les proposer à l'interface.

Le système met également en jeu des *agents temporaires*, qui sont créés pour accomplir des tâches spécifiques lorsqu'un service est long à exécuter ou qu'il nécessite une migration. Ce type d'agent n'a nul besoin d'une interface de contrôle et consiste simplement en une instance de la classe `agent` associée à une suite spécifique de bases Prolog. Comme un agent temporaire est spécialisé dans le domaine de la tâche qu'il doit réaliser, il existe beaucoup de types dérivés.

Par exemple, le service de mémento, service interne offert par tout agent utilisateur, nécessite la création d'un agent auxiliaire. En effet, l'exécution de ce service, qui est interactif avec l'utilisateur, ne doit pas perturber le fonctionnement de l'agent utilisateur. Le service est donc assuré par un agent temporaire qui récupère les informations dont il a besoin en accédant directement au noyau Prolog de son créateur (invocation synchrone Prolog-Prolog) à chaque fois qu'il en a besoin. Ce mécanisme permet d'avoir de nombreux services locaux en parallèles, sans duplication des informations dont la redondance entraîne perte de place mémoire et difficultés de maintient de cohérence.

4.1.3 Le langage de négociation

Ni la représentation uniforme des ressources du système par des agents potentiellement clients et serveurs, ni la disponibilité d'un langage commun (Prolog) à tous les agents ne constitue un *principe intégrateur* à elles-seules. Il est nécessaire d'y adjoindre une sémantique qui doit être partagée par les agents du système, sans quoi toute tentative de communication et donc de coopération est illusoire.

Par conséquent, la notion de service étant l'abstraction de base de notre conception de la coopération, il est indispensable que les agents puissent décrire sans ambiguïté le service qu'ils cherchent. Pour cela, ils disposent :

- d'un ensemble de symboles, chacun représentant un service particulier (e.g imprimer, réserver-salle) auquel est associé un type de serveur (e.g. imprimante, salle).
- d'un ensemble de symboles représentant des caractéristiques (e.g. résolution, surface), de façon unique dans le domaine du serveur considéré.
- d'un langage d'expression de contraintes sur ces caractéristiques.

Notre langage d'expression de contraintes permet d'exprimer l'appartenance ou la non-appartenance d'une caractéristique à un intervalle ou à une énumération de valeurs. Le moteur de résolution de contraintes réalisé est générique mais peut être enrichi par chaque agent pour résoudre certains cas en introduisant des contraintes par défaut ou de nouvelles règles spécifiques à son domaine.

De plus, le langage intègre la notion de satisfaction, qui permet d'enrichir le "ou" par des préférences. La satisfaction est le résultat de l'évaluation d'un programme Prolog qui est fonction des valeurs des caractéristiques après résolution. Il en résulte une grande expressivité du langage, grâce auquel on peut formuler une requête du genre : "réserver une salle pour 10 personnes, avec un rétroprojecteur, de 8h30 à 11h, le 1er mardi de janvier 94 de préférence ou un autre mardi sinon."

4.2 Un exemple de protocole de coopération

4.2.1 Objectifs

Dans le système présenté, la coopération est basée sur une structure de groupe particulière qui s'appuie sur les fonctionnalités de communication de groupe Chorus. Nous décrivons ici les protocoles mis en œuvre pour la gestion et l'invocation du groupe. Ceux-ci ont été conçus dans l'optique de réduire les inconvénients de protocoles tels que le réseau de contrat [13] qui, par des flux importants de messages, engendrent :

- l'encombrement du réseau de communication,
- l'encombrement des boîtes aux lettres des agents,
- le ralentissement de l'activité des agents à cause la surcharge des messages à traiter.

4.2.2 La gestion interne du groupe

La structure mise en place consiste à ce que tous les agents salle maintiennent une sorte d'association d'artisans d'un même domaine, dont le but n'est pas la concurrence mais la satisfaction des clients. Lorsqu'un agent salle est créé dans le système, il diffuse un message au groupe des salles pour déclarer son apparition. En retour, chaque membre du groupe répond par un message de déclaration d'appartenance à ce groupe[5]. Comme ces messages de déclaration contiennent les caractéristiques structurelles (surface, équipement, emplacement...) de l'agent émetteur, il en résulte que toute salle connaît en permanence les autres salles et leurs caractéristiques structurelles. Les caractéristiques conjoncturelles (e.g. planning) ne sont pas transmises afin de ne pas abuser de la diffusion de groupe. Néanmoins, le bon fonctionnement de cette structure repose sur deux hypothèses :

(1) le groupe est lentement dynamique, i.e. la création, la disparition d'un membre ou l'évolution de ces caractéristiques structurelles sont des phénomènes rares;
(2) le nombre de membres est réduit.

En effet, lorsqu'un groupe comporte n membres, la déclaration d'un nouveau membre provoque l'émission de 2n messages, dont n dans une même diffusion. Même si le coût de diffusion est difficilement estimable a priori puisqu'il dépend du type de réseau, il reste quand même n messages envoyés individuellement qui, de plus, parviennent dans la même boîte aux lettres. D'un point de vue général, puisque nous procédons à des duplications d'information, les deux hypothèses permettent de considérer qu'il n'est pas trop coûteux de maintenir un niveau de cohérence satisfaisant.

4.2.3 L'invocation du groupe

Puisque les membres se connaissent les uns les autres, il est inutile de diffuser une requête de service au groupe entier. Lorsqu'un agent utilisateur désire réserver une salle, il lui suffit de contacter un seul agent salle. Comme le client n'est pas supposé connaître de serveur a priori,

---

5. Signalons qu'un protocole similaire est exploité par le micro-noyau Chorus lors de l'apparition (boot) d'une machine sur le réseau, afin de déterminer son numéro d'incarnation indispensable à la génération d'identificateurs uniques ("hello protocol").

il invoque le groupe de communication en utilisant le mode *fonctionnel*. Ce mode consiste à envoyer un seul message à un membre quelconque du groupe.

L'agent utilisateur n'étant pas un expert dans le domaine des réservations de salle, il est incapable de formuler une requête de service complète. Par conséquent, l'agent salle qui reçoit l'invocation répond en créant un agent temporaire, sorte d'*agent commercial*, chargé de s'occuper de la requête du client. L'agent commercial recopie la liste des agents salles ainsi que leurs caractéristiques structurelles auprès de son créateur, en accédant directement à son noyau Prolog. Ensuite, il migre vers le site du client et établit de façon interactive la requête de service complète. Il interroge l'utilisateur pour déterminer les principales contraintes, mais il accède aussi directement à la base de son agent pour récupérer certaines informations. Ainsi, la répartition logique et physique du problème est respectée : la requête de service complète est établie par un expert du domaine en provenance d'un site distant.

Enfin, l'agent commercial fait la liste des serveurs potentiels en éliminant les salles structurellement incompatibles et l'ordonne en fonction d'un critère de satisfaction(6). Il contacte successivement les serveurs de cette liste tant qu'il reçoit des refus pour cause d'indisponibilité. Lorsqu'un serveur accepte, l'agent commercial transmet le contrat au client qui prévient l'utilisateur et enregistre l'information en vue d'une consultation ultérieure.

Ainsi, notre protocole de recherche de serveur est clairement optimisé : peu de messages, étalés dans le temps et pas de diffusion. S'il est vrai que la migration de l'agent commercial est une opération coûteuse, il faut néanmoins souligner qu'elle n'a lieu qu'une fois et qu'elle possèdent des vertus essentielles. En premier lieu, elle permet d'éviter les nombreuses communications qui auraient été nécessaires pour dialoguer avec l'utilisateur à distance. En second lieu, la co-résidence du commercial et du client permet des interactions rapides, non tributaires des performances du réseau de communication.

A titre d'illustration, le coût de recherche d'un service en termes de messages est comparé avec le protocole du réseau de contrat (CNet) dans le tableau 2, où n représente le nombre de

	réseau de contrat		groupe	
au mieux	- n appels d'offres - 1 offre - 1 contrat - 1 rapport d'exécution	**n+3 messages**	- 1 appel d'offres - 1 migration - 1 requête de service - 1 rapport d'exécution	**3 messages** 1 migration
au pire	- n appels d'offres - n offres - n contrats - n rejets[a]	**4n messages**	- 1 appel d'offres - 1 migration - n requêtes de service - n rejets	**2n+1 messages** 1 migration
cas "moyen"	- n appels d'offres - n/2 offres - n/4 contrats - n/4 messages (rejets + 1 rapport)	**2n messages**	- 1 appel d'offres - 1 migration - n/4 requêtes de service - n/4 messages (rejets + 1 rapport)	**n/2+1 messages** 1 migration

tableau 2 : évaluation et comparaison du réseau de contrat et du protocole de groupe

a.Entre le moment où le serveur établit son offre et où il reçoit le contrat, il est possible qu'il ne puisse plus l'honorer, suite à de nouveaux contrats acceptés.

6. Le critère de satisfaction, qui pourrait être exprimé par le client, consiste par défaut à favoriser les salles dont l'équipement est juste suffisant.

serveurs potentiels. Bien sûr, cette comparaison n'est valable que dans le cadre des hypothèses pré-citées qui permettent de négliger les communications ponctuelles dues à la gestion du groupe.

4.3 Considérations pratiques

Testé de façon répartie sur 3 PC avec micro-noyau Chorus[7], ce système a permis d'apprécier la rapidité des communications, qui sont intégrées au cœur même du micro-noyau Chorus, ainsi que l'efficacité du protocole de groupe. Le système fonctionne également sur station SPARC 10 avec un simulateur Chorus, mais un goulet d'étranglement apparaît rapidement à cause de l'unique processus du simulateur.

En ce qui concerne l'encombrement des objets PUMA, on note que les classes `agent` et `interp` (cf 3.3.1) ne font que 17K. Cette taille réduite résulte de l'intégration de la partie binaire de l'interpréteur Prolog dans chaque contexte d'adressage, sur chaque machine. Chaque contexte d'adressage, qui contient aussi la bibliothèque COOL, totalise 120K. L'allocation dynamique est essentiellement due à l'interpréteur Prolog. Lorsque les programmes Prolog sont chargés, la place mémoire occupée avoisine les 100K. Ces chiffres permettent d'estimer le nombre d'agents que l'on peut créer et le coût de la migration par rapport au simple envoi de message, en fonction de la configuration machine et réseau. En ce qui concerne la migration, rappelons qu'elle présente l'avantage de conserver le déroulement de l'activité.

5. Conclusion

5.1 Bilan

Dans cet article, nous avons proposé un modèle d'intégration et de coopération orienté agents pour toutes les ressources (logicielles, matérielles, humaines) accessibles via le système d'information de l'entreprise. Dans le cadre du "Computer-Supported Cooperative Work" et du "Human-Computer Cooperative Work", notre objectif est de faciliter les rencontres entre prestataires (serveurs) et consommateurs (clients) de services et d'enrichir la phase de négociation qui s'en suit :

- pour que le client trouve rapidement le service le plus adapté à ses besoins,
- malgré une offre de services dynamique,
- sans engorger le système de communication.

Après avoir exposé les atouts d'une technologie répartie orientée objets, nous avons présenté différentes couches et outils (COOL, PUMA et classes dérivées, bases Prolog, résolution de contraintes) pour mettre en œuvre la modélisation. Une réalisation pratique nous a permis d'illustrer et de discuter nos propos, tout en introduisant une structure de groupe particulière et les protocoles associés qui visent à optimiser la mise en relation d'un client avec le serveur ad hoc. Cette structure, qui permet de diminuer l'espace de recherche grâce à la distinction entre caractéristiques structurelles et conjoncturelles des agents, est destinée aux services fréquemment invoqués mais dont la famille des prestataires est faiblement dynamique et peu nombreuse.

---

7. Il s'agit de PC 486 à 66 MHz, avec le système UNIX SCO/Fusion

5.2 Perspectives

L'approche multi-agents paraît riche en possibilités et nous envisageons la réalisation d'autres applications, nécessitant la mise en œuvre d'autres structures et d'autres protocoles de coopération. Une première application est une messagerie intelligente qui permet de contacter une personne ou un ensemble de personnes en fonction de certains critères. Dans notre modélisation, il s'agit de contacter un (les) agent(s) dont les caractéristiques vérifient certaines contraintes. La structure de coopération envisagée met en œuvre des agents annuaire organisés en groupe comme décrit précédemment. Chacun de ces annuaires connaît les agents vérifiant certaines contraintes (le *filtre* propre de l'annuaire) ainsi que leurs caractéristiques structurelles. La configuration physique (localisation) et logique (contraintes exprimées dans les filtres) peut être émergente, pour peu que les agents annuaires puissent se dupliquer, fusionner et se déplacer en fonction du contexte. Ceci nécessite des raisonnements sur le type des agents enregistrés et/ou recherchés (taxinomie), et sur la provenance des invocations. Un modèle de placement automatique basé sur l'observation des relations entre les entités d'un système est exposé dans [4].

Une autre application intéressante concerne CIDRE, qui est d'ailleurs historiquement à l'origine de notre approche multi-agents. Cette application de circulation de documents électroniques, distribuée logiquement et physiquement par nature, peut tirer profit de notre modélisation, de nos outils, de nos structures et des protocoles associés :

- description déclarative du schéma de circulation, rendant possible l'intervention d'un administrateur en cours de circulation;
- support pour la conception d'un moteur "intelligent" d'exécution des schémas de circulation (optimisation du coût, du délai ou de la qualité), comprenant un système expert de résolution des problèmes de blocage (traitement des "exceptions");
- utilisation des agents pour proposer des réponses automatiques lorsqu'un utilisateur doit remplir certains champs (état civil, adresse...) ou pour indiquer les disponibilités, les délégations...
- intégration du système global pour un accès transparent aux différentes ressources.

Par ailleurs, il faudra mener une réflexion approfondie à propos des systèmes à grande échelle. La modélisation, les structures de coopération et les protocoles devront être étendus, modifiés ou complétés pour prendre en compte l'interconnexion de plusieurs systèmes d'information. L'interopérabilité entre différents "domaines administratifs" pose, en particulier, des problèmes supplémentaires concernant le langage et la sémantique.

En ce qui concerne l'outil lui-même, il est appelé à évoluer fortement avec la mise en œuvre de la version 2 de COOL qui propose notamment un modèle rénové d'invocation et d'exécution. De plus, d'autres versions de Prolog et d'autres langages interprétés seront examinés à la lumière des forces et faiblesses de PUMA, conjointement à une étude sur les langages de scripts et les 'shells' pour système réparti à objets.

BIBLIOGRAPHIE

[1] *Basic Reference Model of Open Distributed Processing : non-normative (descriptive) specification of trader.* ISO/IEC JTC 1/SC21 WG7 N, Standards Australia 1993.

[2] Gul Agha. *Actors, A Model of Concurrent Computation in Distributed Systems.* MIT Press 1988.

[3] Michel Barat, Jean Erceau. *Utilité et utilisation d'un principe intégrateur dans un outil de conception de systèmes complexes multi-experts.* Actes du 2ème congrès européen de systémique, vol. III pp 860-869. Prague, octobre 1993.

[4] François Bourdon. *Un modèle de dérive des connaissances; applications en bureautique*. Thèse de doctorat de l'Université du Maine, juillet 92.

[5] Jean-Marc Deshayes, Vadim Abrossimov, Rodger Lea. *The CIDRE distributed object system based on Chorus.* Proc. of TOOLS'89.

[6] Marc Desreumaux. *Pourquoi les entreprises ont-elles besoin de systèmes d'information flexibles ?* Tome 7, 1er congrès biennal AFCET, Versailles, juin 1993.

[7] Jacques Ferber. *BRIC : essai de mise en perspective d'une méthodologie multi-agent* journée d'étude AFCET "méthodes orientées agents", Paris, 14 septembre 94.

[8] Hans Haugeneder. *IMAGINE Final Project Report.* projet ESPRIT 5362, IMAGINE Consortium.

[9] Rodger Lea, Christian Jacquemot, Eric Pillevesse. *COOL : system support for distributed programming.* Communications of the ACM, Vol.36, No.9, 1993.

[10] Claire Lefèvre, Claire Beyssade. *Système multi-agents et modalités épistémiques.* Premières journées francophones IAD & SMA, Toulouse 1993.

[11] V. Lesser, D. Corkhill. *Distributed problem solving.* Encyclopedia of Artificial Intelligence, Vol. 2 (1987), 245-251.

[12] Yoav Shoham. *Agent-oriented programming.* Artificial Intelligence 60 (1993), Elsevier, 51-92.

[13] R.G. Smith. *The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver.* IEEE transactions on computers Vol. C-29, No. 12 (december 1980), 1104-1113.

[14] R.A. Thiétart. *Ordre et Chaos dans les Organisations.* Journée d'étude AFCET "Les Systèmes d'Information, Autonomie et Chaos", Paris, 24 novembre 1993.