PARU DANS “CALCULATEURS PARALLÈLES”, VOLUME 9 - NO 2/1997 PAGES 183 - 207

Modélisation de la coopération et de la synchronisation dans les systèmes d’information

Une expérience de workflow basée sur les nouvelles technologies

Bruno Dillenseger — François Bourdon

CNET, 42 rue des Coutures, BP 6243, 14066 CAEN CEDEX.

RÉSUMÉ. La gestion des systèmes d’information des entreprises se doit de tenir compte de certaines technologies nouvelles, qui illustrent à la fois une ouverture de plus en plus grande de ces systèmes vers le monde extérieur, et une complexification de leur fonctionnement. Dans ce travail, nous proposons d’une part une modélisation de la synchronisation et de la coopération de processus administratifs (“workflow”), et, d’autre part, un cadre architectural et technologique capable de supporter la réalisation d’une telle modélisation. Enfin, nous illustrons cette approche en décrivant une expérience réalisée dans le cadre du projet PREVISIA[1].

ABSTRACT. Today’s management of the office information systems requires several new technologies to be taken into account. These technologies reflect the growing need in open systems, which results in an increasingly complex functionning. In this work, we introduce a synchronization and cooperation model for office tasks (so-called “workflow”). Then, we show the construction of an implementation architecture for this model, based on the integration of several advanced technologies. Finally, we describe an intranet workflow system[2], based on both the model and the technical approach.

MOTS CLÉS : système d’information ouvert, agents, workflow, Intranet, CORBA, Java, Prolog.

KEY WORDS : large scale open system, agents, workflow, Intranet, CORBA, Java, Prolog.

1. Plate-forme Répartie d’Évaluation et d’Intégration des Systèmes d’Information Avancés.
2. From the workflow module of the PREVISIA project (“evaluation and integration distributed platform of advanced information systems”).

1. Introduction

Les travaux menés au SEPT[3] depuis 1988 sur la bureautique communicante et plus généralement sur les systèmes d'information (SI) des entreprises, nous ont conduit à explorer deux pistes. D'une part, la piste applicative s'intéresse aux outils de gestion des SI, tels que *workflow*, rédaction collaborative de documents structurés, partage de ressources de l'entreprise, recherche d'information… (projet CIDRE[4] [BOU 89], projets ESPRIT II COMANDOS[5] [COM 91] et ISA[6] [ISA 90], projet CNET AMBIANCE[7] [MAU 90], projet PREVISIA). D'autre part, la piste de l'infrastructure de communication et d'exécution vise la mise en œuvre des solutions proposées.

C'est ainsi que nous avons établi une description fonctionnelle de la première version du système de communication à objets COOL v1[8] [LEA 89], dont la réalisation a été confiée à la société Chorus Systèmes[9]. Nous avons également proposé une architecture pour systèmes multi-agents basée sur l'introduction de la logique et du contrôle de l'activité qui traverse les objets. Ceci nous a permis de définir des protocoles de coopération basés sur les groupes fonctionnels. Ces travaux nous ont également conduits à travailler sur les problèmes de représentation des connaissances, de description formelle des comportements, de communication, ainsi que de gestion de la coopération et du partage de ressources.

Nous avons pu mettre en pratique et valider cette expérience acquise depuis plusieurs années, dans le cadre du projet PREVISIA à l'intérieur du module CIDRIA. Ce module vise le traitement coopératif des activités liées à l'échange et à la production d'information dans l'entreprise.

La gestion des systèmes d'information des entreprises se doit de prendre en compte certaines technologies nouvelles, qui illustrent une ouverture croissante de ces systèmes vers le monde extérieur, impliquant une complexification de leur fonctionnement. C'est dans ce contexte que l'un des objectifs de PREVISIA, qui rassemble des entreprises comme l'EDF/GDF et la BNP, associées à des centres de recherche comme l'INRIA et le CNET, est de tester les offres technologiques dans des situations proches des besoins exprimés par les membres du projet.

Dans ce travail, nous nous sommes intéressés aux applications qui gèrent la synchronisation de processus administratifs (*workflow*). Nous présentons une

3. Service d'Etudes communes de La Poste et de France Télécom — CNET-Caen depuis le 1er janvier 1997.
4. Circulation Intelligente de Dossiers REpartis.
5. COnstruction and MANagement of Distributed Open Systems.
6. Integrated Systems Architecture.
7. Aide à la Mise en oeuvre d'une Bureautique Intelligente et d'Applications Nouvelles pour la Communication d'Entreprise.
8. Chorus Object Oriented Layer.
9. Nous continuons la collaboration sur ce projet, afin d'étendre COOL-ORB (version conforme au standard CORBA 2 de l'OMG) : mobilité des objets, modèle d'exécution avancé (contrôle des activités compilées et/ou interprétées concurrentes), mécanisme d'observation et modèle des interactions entre objets.

modélisation de la coopération et de la synchronisation entre ces processus, qui est basée sur une approche multi-agents. Cette approche propose la construction d’un système adaptatif, dans lequel les acteurs (*agents*)[10] résolvent un but en coopérant (recherche du meilleur prestataire de service pour la réalisation d’une tâche donnée). Pour cela, ils adaptent leur comportement à l’évolution de leur environnement (e.g. prise en compte de nouveaux agents).

Nous proposons ensuite un cadre architectural et technologique destiné à la mise en œuvre d’un tel modèle, basé sur une intégration de technologies avancées (CORBA, Web, Prolog). Nous illustrons la faisabilité et l’opportunité de cette approche au travers d’une expérience réalisée au sein du projet PREVISIA.

2. Contexte/Problématique

2.1. Domaine général

Le contexte de ce travail est la gestion des informations dans les systèmes ouverts (“large-scale Open Systems”). Dès le début des années 90, certains auteurs [GAS 91] [HEW 91] décrivaient de tels systèmes comme des systèmes imprévisibles, susceptibles de recevoir de nouvelles informations de l’extérieur, et ce à tout moment. Plus récemment, d’autres auteurs [THI 93] comparent volontiers de tels systèmes à des systèmes chaotiques pour lesquels des modifications légères dans les conditions initiales de processus informationnels complexes, modifient profondément le devenir de ces processus. La grande complexité de ces systèmes vient de l’imprédictibilité des interactions entre les acteurs (ressources informatiques ou humaines) qui les constituent, mais également de l’hétérogénéité des outils, des pratiques et des conceptualisations rencontrés, ou encore de l’évolutivité des besoins, des moyens et de l’environnement dans lequel fonctionnent ces systèmes.

Les progrès récents en matière de techniques logicielles (le paradigme “objet” et plus récemment le paradigme “agent”) ont permis la mise en oeuvre de solutions architecturales (par exemple l’OMG/CORBA, voir 4.1.3.) élégantes pour gérer l’hétérogénéité des systèmes, des protocoles de coopération/communication et des langages. Ces résultats sont autant de points encourageants pour proposer des solutions aux problèmes posés.

10. Dans l’ensemble de cet article, le terme “acteur” est utilisé dans son acception générale, sans référence au modèle de calcul concurrent. En revanche le terme “agent” possède une signification propre à notre modèle, empruntée aux disciplines de l’Intelligence Artificielle Distribuée et des Systèmes Multi-Agents.

2.2. Le “workflow” bureautique étendu

Dans le domaine du *groupware* (applications de travail coopératif), le *workflow* se décline sous forme de circulation automatisée de formulaires électroniques, où l’on trouve les difficultés suivantes :

– la répartition inhérente aux procédures ;
– la formalisation a priori d’une procédure, souvent très complexe dans le détail ;
– la spécification des intervenants ;
– l’adaptation au contexte réel d’exécution ;
– le traitement d’exception face aux situations de blocage de la circulation.

Nous nous intéressons à une généralisation de ce type d’application dont l’objectif devient alors le traitement des flux d’information entre des processus administratifs. Comment gérer ces flux, les synchroniser, utiliser de nouvelles ressources, faire coopérer les acteurs pour mieux gérer ces flux ?

2.2.1. Diversité des temps de traitement

Quand on observe plus en détail le fonctionnement de ces flux, on s’aperçoit qu’il existe une grande diversité des temps de réaction entre les différents acteurs concernés. Lorsque des processus décisionnels sont mis en jeu, il apparaît des négociations entre acteurs de niveaux de compétence pas toujours homogènes, qui peuvent dans certains cas être accélérées lorsque certains acteurs fonctionnent par anticipation dans des cas de confiance (fondée sur des expériences antérieures) réciproque. Les acteurs qui produisent les informations échangées sont répartis organisationnellement et/ou géographiquement, certains sont nomades. Ils jouent un rôle à la fois dans l’organisation de l’entreprise (structure organisationnelle) et dans le traitement du processus (comportement/compétence/disponibilité). Dans certains cas, on observe des circuits parallèles utilisés pour des processus administratifs donnés. Certains sont formels et d’autres ne le sont pas, ce qui engendre plus ou moins de souplesse quant à la réalisation et à l’évolutivité des processus associés.

2.2.2. Autonomie des acteurs et contrôle des flux

Le rôle comportemental basé sur des notions de compétence (qui sait faire quoi ? comment ? quand ?) reflète une certaine autonomie des acteurs de l’entreprise en jeu dans ces processus. Cette superposition organisationnelle (qui a l’autorisation de faire quoi ?) et comportementale devrait bien s’accorder avec une approche technologique qui privilégie des entités “autonomes” en interaction, dont le contrôle opératoire est à la fois rigide dans les descriptions formelles (ossature des processus) et souple dans la reconfiguration possible en fonctionnement des comportements locaux des acteurs impliqués dans ces processus.

2.2.3. Intégration de l’existant

L’introduction de nouveaux outils ou couches logicielles ne peut se faire sans intégrer à moindre coût les outils et les couches logicielles existant dans l’entreprise.

2.2.4. Adaptativité aux fluctuations du contexte

Ces processus sont réactifs aux variations dues à la disponibilité et/ou la compétence des acteurs, à la configuration géographique de l’entreprise, à sa configuration organisationnelle ou encore opérationnelle (ajout de serveurs…). Ces variations doivent être perceptibles dans la description des processus eux-mêmes. L’approche “agent” que nous proposons autorise cette prise en compte contextuelle, conjointement avec une description plus rigide (appelée “ossature” d’un processus donné) formalisée à l’aide des réseaux de Petri.

2.2.5. Pérennité d’une information

La durée de validité d’une information produite peut être inférieure à la durée du processus qui utilise cette information. L’information est, suivant les cas, plus ou moins stable dans le temps. Cela peut être le cas d’un devis fourni par un fournisseur dans un processus de commande de matériel. Pour minimiser les risques d’engager des processus administratifs sur la base d’informations devenues obsolètes, nécessitant des retours en arrière ou des réessayages coûteux, il faut maîtriser les temps de traitement et d’acheminement des informations dans ces processus. Il faut également augmenter les mécanismes de détection des changements d’état des informations manipulées, ce qui nécessite une formalisation explicite de cette notion de validité dans le temps ; il faut aussi être capable de mesurer les conséquences de ces changements. La formalisation d’un processus administratif doit fournir des éléments pour expliciter les effets induits au sein d’un processus par une information devenue soudainement obsolète.

2.2.6. Résolution de problèmes locale et coopérative

Eviter les sous-utilisations et les sur-utilisations (entraînant des blocages) de ressources dont on dispose, nécessite soit un dimensionnement a priori suffisamment large pour supporter les surcharges, ou un système adaptatif capable de gérer localement (le plus près possible de la source des problèmes) des coopérations entre les acteurs concernés (ressources disponibles et demandeurs) par ces problèmes. Plutôt que de maintenir des bases d’information sur l’emploi du temps ou la charge des ressources du système, on peut stocker et gérer au plus près des acteurs concernés ce type d’information fortement liée aux ressources et généralement très instable.

Cette activité de résolution de problèmes permet de réagir en cas de blocage, de trouver la ressource/le service le plus adapté (coopération), ou encore de résoudre des conflits d’accès aux ressources (négociation).

2.2.7. Conclusion

Ce qui précède montre que la viabilité d’un système de “workflow étendu” est fonction de sa faculté à s’adapter à un environnement dynamique et hétérogène.

Le système que nous proposons se focalise plus particulièrement sur :

– la prise en compte de l’apparition de nouvelles ressources et de nouveaux types de services,

– la capacité, au cours de l’exécution d’une procédure, à choisir la ressource et le service les plus appropriés pour effectuer une tâche.

Pour répondre à ces besoins, notre système dispose de capacités de représentation des connaissances (compétences, état, responsabilités) et de résolution de problèmes.

3. Une approche multi-agents du workflow

3.1. Modèle général

3.1.1. Une vision uniforme du système

Les travaux menés par le SEPT sur le thème de l’introduction de raisonnement dans les systèmes de bureautique communicante nous ont conduits à une approche multi-agents[11] [DIL 96] [BEY 95]. Il s’agit d’intégrer chaque ressource de l’entreprise dans le système d’information (un utilisateur, un logiciel de correction orthographique, une imprimante, une salle de réunion…) par le biais d’une représentation permanente et exclusive. L’interopérabilité entre les ressources se traduit par une coopération entre les entités chargées de leur représentation (figure 1). Cette coopération implique une représentation de l’état des ressources et des services qu’elles peuvent rendre, ainsi que l’existence de protocoles et de structures permettant aux entités de trouver et négocier le service recherché.

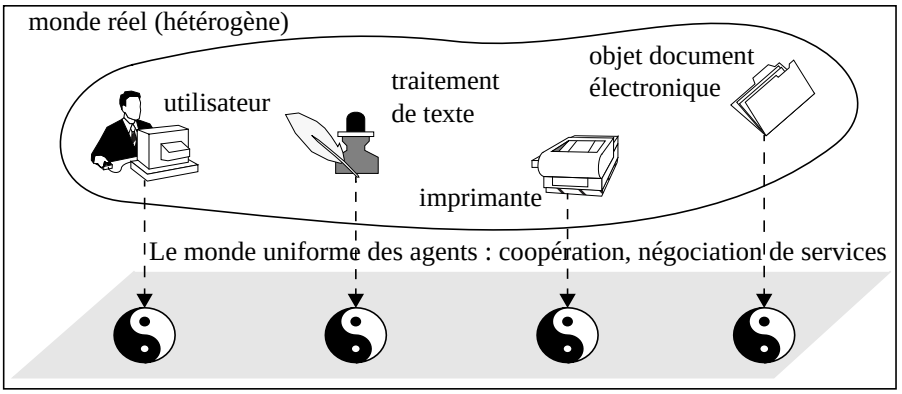


Figure 1. *Représentation uniforme du système*

Nous voyons alors que les entités représentant les ressources sont confrontées à des problématiques de représentation des connaissances, de résolution de problèmes, de coopération et de négociation dans un environnement réparti. En désignant ces entités par le terme *agent*, nous nous référons directement à l’Intelligence Artificielle

11. En collaboration avec l’équipe de Patrice Enjalbert et Jerzy Karczmarczuk de l’Université de Caen.

Distribuée. A la façon de Huhns dans [HUH 87], nous pouvons justifier notre approche multi-agents par six points-clés :

– l’avancée technologique et l’extension des réseaux, permettant une communication à grande échelle ;
– la confrontation à des problèmes dont la répartition est inhérente ;
– le besoin de modularité pour faciliter la conception et l’implémentation ;
– l’effet de synergie, qui permet de résoudre une complexité globale élevée par des entités de conception plus simple ;
– l’apport d’une unité de point de vue, ouvrant la voie à des modèles homogènes d’intégration des utilisateurs et des machines ;
– les théories sociologiques, intéressées notamment par la résolution collective de problèmes, trouvent un champ d’application et de simulation.

3.1.2. Le modèle de coopération

Chaque agent possède un **nom unique**, une **boîte aux lettres** pour recevoir des messages, et peut **envoyer des messages** aux agents dont il connaît le nom. L’agent est caractérisé par ses **compétences** et ses **caractéristiques**. Les caractéristiques définissent un ensemble d’attributs propres à la ressource, et sur lesquelles on peut exprimer des **contraintes**. Par exemple, un agent représentant une imprimante aura une caractéristique “résolution” pouvant prendre certaines valeurs (e.g. 600 ou 1200 dpi). Ces contraintes sur les caractéristiques permettent de spécialiser le type de service rendu par la ressource. Il existe en effet une **sémantique partagée** concernant les types de service (e.g. imprimer, corriger…) et les caractéristiques que l’on peut y associer. Ainsi, une imprimante offrira l’impression couleur dans telle résolution, une autre noir et blanc mais en recto-verso. C’est ainsi que les agents définissent leurs compétences.

Lorsqu’un agent a besoin d’un service, il envoie une **requête** vers un autre agent qu’il connaît. L’agent récepteur peut alors **rejeter** la requête ou y **répondre**. Cela pose évidemment le problème de la recherche de l’agent serveur ad hoc. De nombreux travaux visent à proposer des organisations, des structures et des protocoles permettant aux agents d’entrer en relation pour coopérer de façon optimale (cf. [DIL 96]). Le protocole le plus connu est celui du réseau de contrat [SMI 80], qui consiste à diffuser un appel d’offres à ses *accointances*[12], puis à recueillir et évaluer les offres après un certain délai, avant de passer un contrat avec l’agent élu. Des extensions [KUW 95] ont été proposées, consistant par exemple à propager un appel d’offres d’agent en agent.

3.1.3. Gérer l’hétérogénéité des ressources : les adaptateurs

L’effet d’uniformité apporté par le modèle multi-agents suppose qu’une couche intervienne pour connecter chaque ressource à son agent. Cette couche est réalisée par les **adaptateurs**, entités où l’hétérogénéité du système d’information sera

12. Les accointances d’un agent est l’ensemble des agents qu’il connaît.

traduite en abstractions multi-agents. Ces adaptateurs permettent d’intégrer des ressources pré-existantes (matériel, logiciel, utilisateurs…), mais il est aussi possible de créer des ressources logicielles intégrant d’origine cette partie “adaptateur”.

Par exemple, l’adaptateur d’un utilisateur sera typiquement une interface graphique par laquelle il se connectera à son agent pour émettre des requêtes d’une part, et répondre aux requêtes reçues d’autre part. Si l’on souhaite intégrer une salle de réunion au système, en vue de réaliser des réservations automatiques, on créera un agent pour la représenter. Il aura pour compétences la réservation et l’annulation de réunion. Ses caractéristiques décriront l’équipement et la situation géographique de la salle. Un utilisateur sera alors chargé de maintenir la cohérence entre l’état de la salle réelle et l’état de l’agent correspondant (e.g. si l’équipement de la salle est modifié), via une interface graphique. Ici, on peut considérer que l’adaptateur est constitué par la combinaison de l’utilisateur et de l’interface graphique.

3.1.4. Vue d’ensemble (figure 2)

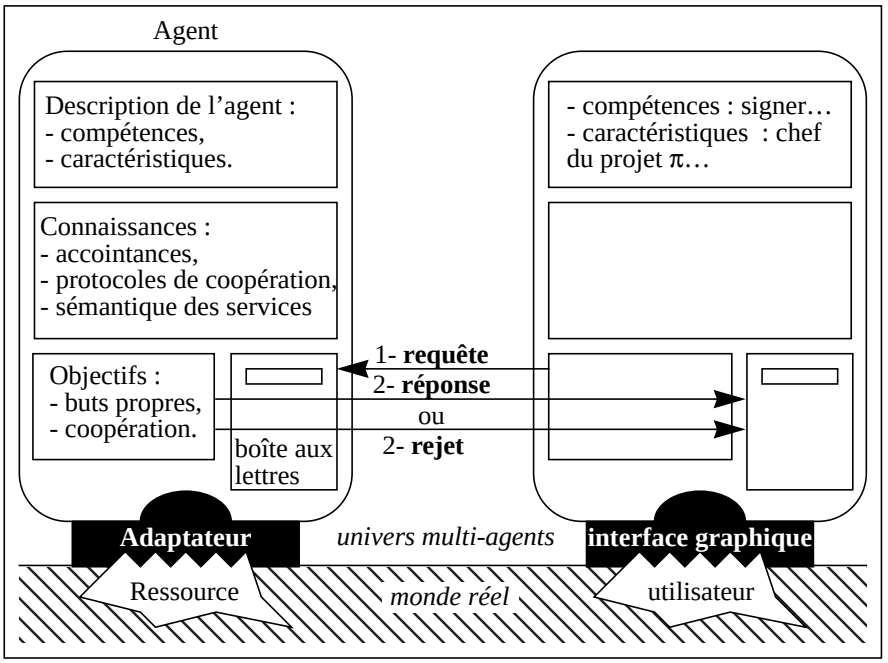


Figure 2. *Agents et coopération dans le système d’information*

3.2. Application au workflow

3.2.1. Les abstractions

La spécification d’une procédure de workflow consiste à décrire un scénario de lancement et de synchronisation d’un ensemble de **tâches**. Au cours de l’exécution

de ce scénario, des **éléments d’information** seront consommés ou produits par ces tâches : il s’agit des paramètres nécessaires à l’exécution de la tâche, ou des résultats de cette exécution. Il existe deux types de tâche :

– l’**action élémentaire**, qui correspond directement à une compétence (i.e. association d’un type de service et de contraintes sur ses caractéristiques) ;

– la **macro-action**, qui représente un sous-scénario.

Chaque action élémentaire est représentée par un nom unique, dont la sémantique est partagée au sein du système. De plus, elle est associée à une liste spécifique d’éléments d’information produits ou consommés, dont la sémantique est également partagée (figure 3).

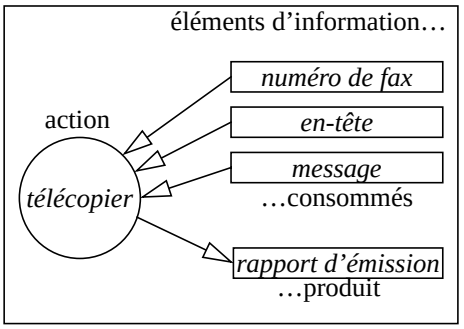


Figure 3. *Un exemple d’action élémentaire*

Une macro-action décrit une sous-procédure de workflow. De même que l’action, une macro est identifiée par un nom unique et caractérisée par des paramètres et des résultats. En revanche, la macro-action ne correspond pas à une compétence, et son exécution sera typiquement non atomique et répartie sur plusieurs agents.

3.2.2. Le formalisme de workflow

Contrairement à certaines approches qui consistent à créer des procédures bureautiques de circulation de formulaires électroniques officiels, nous souhaitons mettre à disposition un formalisme simple et attractif pour que tout utilisateur puisse décrire un enchaînement simple de tâches variées à automatiser. Toutefois, cette démarche complémentaire n’empêche pas que certaines procédures plus complexes ou officielles soient définies par un administrateur.

Le formalisme choisi est directement issu des *Information Control Nets* (ICN), qui ont été spécifiés au Xerox PARC pour modéliser des flux d’information dans une administration [ELL 79]. Dérivés des réseaux de Petri, ils en possèdent les propriétés, tout en simplifiant la représentation graphique. De plus, ils laissent apparaître explicitement des productions et consommations d’éléments d’information. Les ICN ont été adoptés dans plusieurs applications de workflow ([BOU 89]), y compris commerciales.

Toute procédure de workflow est représentée sous forme d’un graphe orienté ayant une source — “point de départ” — et un puits — “point d’arrivée” — uniques. Les nœuds de ce graphe sont :

– soit des tâches (actions élémentaires ou macro-actions),

– soit des connecteurs logiques.

Lorsqu’un nœud de tâche est rencontré, l’action ou la macro-action correspondante est déclenchée, et le nœud ne pourra être franchi que lorsque la tâche sera terminée. Quant aux connecteurs, ils consistent à introduire des points de choix,

de parallélisme et de synchronisation dans la procédure. Les éléments du formalisme sont décrits par la figure 4, et des exemples de procédure seront donnés en 5.

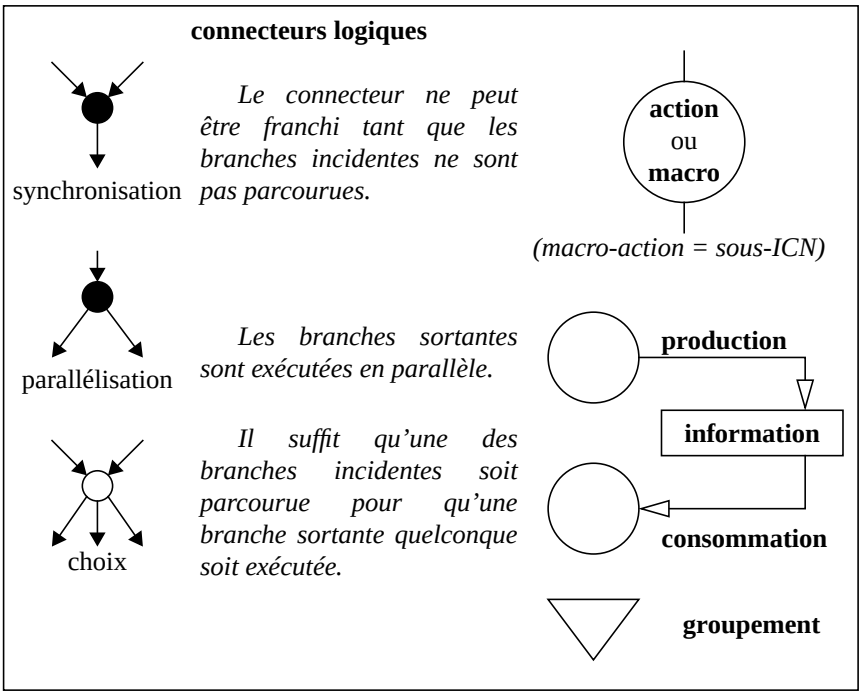


Figure 4. *Eléments du formalisme de type ICN*

En l’absence de notion de formulaire, la gestion des éléments d’information est assurée par des mécanismes particuliers. L’opération de **fusion** permet d’indiquer que différents éléments d’information, consommés et/ou produits par certaines tâches de la procédure représentent en réalité une seule et même information. C’est ainsi que l’on relie les résultats d’une tâche aux paramètres d’une autre, ou que l’on peut modifier la valeur d’un élément d’information en cours de procédure. Le **groupement** consiste à concaténer un nombre quelconque d’éléments d’information pour en créer un nouveau. Il s’agit presque d’une action élémentaire, à la différence près qu’elle prend un nombre variable de paramètres et qu’elle s’exécute de façon purement interne.

4. Opportunités offertes par certaines technologies

4.1. Des technologies adaptées aux différents besoins

4.1.1. Intranet… HTML et compagnie

Le langage HTML a permis de tisser le *World Wide Web* par ses liens hypertextes, et a rendu l’accès à Internet attrayant, facile et uniforme :

– attrayant par son caractère multimédia (texte, image, vidéo et son) ;
– facile du fait de l’approche hypertexte et de la convivialité des visualisateurs ;
– uniforme car les liens permettent d’accéder à des serveurs autres que HTTP, à travers la notion d’URL (*Uniform Resource Locator*) : FTP, NNTP (news), SMTP (e-mail).

L’utilisation interne à l’entreprise des outils d’Internet pour la communication — voire, dans une certaine mesure, le travail coopératif — est à l’origine du concept d’intranet. L’accès à la messagerie, aux forums de discussion, à des formulaires électroniques, à la documentation et à l’information interne de l’entreprise peut se faire à travers une interface banalisée offrant les propriétés énumérées ci-dessus. Les postes clients ne nécessitent aucune installation particulière, si ce n’est un accès réseau, un visualisateur Web… et de préférence une configuration matérielle et système “au goût du jour”.

4.1.2. Les applets Java

Désormais, les visualisateurs Web ne se limitent pas à la consultation d’information, et à une interactivité limitée à des formulaires électroniques pouvant déclencher des traitements sur un serveur. La notion d’*applet* désigne une mini-application écrite dans un langage interprété, pouvant être téléchargée et exécutée au sein d’une page HTML. Ce principe a popularisé le langage Java, faisant parfois oublier qu’il s’agit d’un langage de programmation à part entière.

Si les premières utilisations étaient essentiellement décoratives ou ludiques, des applications plus ambitieuses sont aujourd’hui entrevues (cf. version Java du traitement de texte *WordPerfect*). A cet égard, les techniques de compilation à la volée (*just-in-time compiler*) et l’apparition de processeurs Java réduisent considérablement les problèmes de performances à l’exécution.

Un langage de programmation moderne tel que Java offre une approche orientée objets élégante, associée à des classes standard encapsulant les fonctions génériques courantes (interface graphique, réseau, entrées-sorties, utilitaires divers…). Parmi les fonctionnalités les plus avancées, on note la multi-activité (*threads*), ainsi que le *garbage collector* qui décharge le programmeur de la gestion mémoire. Enfin, le principe de compilation vers une machine virtuelle et la disponibilité des classes standard apportent la portabilité mise à profit dans le cas des applets.

Au sein de l’entreprise, ces propriétés ouvrent la voie à une approche de type *Net Computer* qui simplifie :

– le développement des applications (indépendance à l’hétérogénéité matérielle et système) ;
– le déploiement des applications et de leurs versions successives.

4.1.3. Les systèmes répartis à objets et le standard CORBA

L’un des points clés de nos travaux dans le domaine de la bureautique communicante et du *groupware* consiste à montrer la nécessité de concevoir les applications du système d’information comme des collections d’entités indépendantes et réparties en interaction. Cette approche par “composants autonomes” permet de prendre en compte la répartition naturelle des problèmes et favorise l’adaptation au changement sans remise en cause globale du système d’information. Si l’approche basée sur les systèmes répartis à objets s’avérait prometteuse, nous étions cependant confrontés à l’absence de standard ou de normes effectives, ce qui posait les problèmes de pérennité, de portabilité et d’interopérabilité. De plus, ces systèmes représentaient alors des sujets de recherche à part entière, et nécessitaient des environnements très particuliers, très différents des postes de travail habituels des utilisateurs. Face à cette situation, deux actions majeures visent à produire normes et standards.

Côté normalisation, plusieurs organismes (ISO, IEC, AFNOR…) travaillent à la définition de normes concernant la spécification des systèmes répartis ouverts, désignées sous le vocable d’ODP[13] [ODP]. Développé par l’ISO et l’ITU-T (anciennement CCITT), le modèle de référence de la norme (“RM-ODP”), profondément orienté objet, offre à tout type d’application un cadre conceptuel pour définir une architecture répartie favorisant l’interopérabilité et la portabilité.

Sur le plan des standards, l’OMG[14] a été créé en 1989 par onze organisations afin de spécifier une infrastructure de communication véritablement ouverte et non propriétaire. A travers le standard CORBA [OMG 95], c’est désormais plus de 500 membres[15] qui contribuent à la promotion d’une approche objet susceptible d’améliorer la réutilisabilité, la portabilité et l’interopérabilité dans un environnement réparti et hétérogène. Il existe de multiples implémentations du standard, qui commencent à pouvoir interopérer et qui fonctionnent dans des environnements informatiques standard.

L’OMG définit un modèle objet de base et un modèle de référence d’architecture. Au cœur de l’*Object Management Architecture* (OMA, cf. figure 5), on trouve l’ORB (*Object Request Broker*), qui fournit les mécanismes d’échange transparent de requêtes et de résultats (cf. répartition et hétérogénéité). CORBA (*Common ORB Architecture*) désigne la spécification de l’architecture de l’ORB lui-même.

Lorsqu’un programmeur réalise une application, il peut associer ces propres objets à d’autres objets proposant des fonctionnalités standard définies par l’OMG.

13. Open Distributed Processing
14. Object Management Group
15. L’OMG rassemble notamment les ténors de l’industrie informatique, tels Sun, IBM, Digital, HP, Novell, Microsoft…

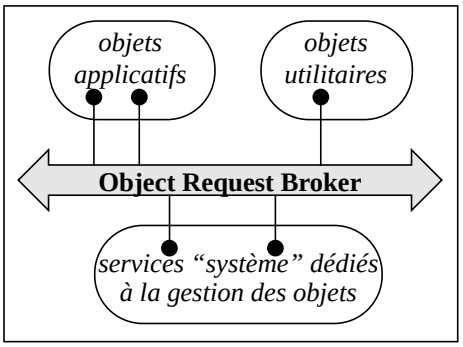


Figure 5. *Le modèle OMA*

Les services objets (*object services*) fournissent des services de transparence dans la gestion des objets répartis : cycle de vie, nommage, persistance, notification d'événements, transactions... Les fonctions communes (*common facilities*) et les objets de métier (*business objects*) apportent une factorisation de fonctionnalités génériques de niveau plus élevé que les services objets.

Tout serveur est vu comme un objet, au travers de son *interface*, i.e. d'un ensemble d'*opérations* qu'un client peut invoquer. Ces opérations et leurs paramètres sont décrits de façon abstraite, dans un langage de description d'interface appelé IDL (*Interface Definition Language*). Des compilateurs spécifiques au langage d'implémentation d'un client ou d'un serveur utilisent cette définition pour générer les composants réalisant le lien avec l'ORB. On parle de *mapping* IDL-C, IDL-C++, IDL-Java…

4.1.4. Les moteurs de résolution de problèmes

Comme nous l'avons montré en 2., le caractère dynamique et l'ouverture des systèmes d'information imposent à leurs composants des propriétés d'autonomie et de coopération. Pour s'adapter à leur environnement, les composants ont besoin de capacités de résolution de problèmes, basées sur une représentation des connaissances de haut niveau.

Or, il existe des noyaux logiciels adaptés à ce type de besoin. Disponibles sous forme de bibliothèques de fonctions ou de classes, ils peuvent être intégrés aux composants et servir avantageusement de support à une algorithmique avancée : moteurs d'inférences ou de résolution de contraintes, logique… Nous avons porté une attention toute particulière au langage Prolog, car il constitue une base à la fois générique et standard, dont les implémentations sont aujourd'hui assez nombreuses et bien optimisées. En outre, son caractère interprété introduit des propriétés hautement dynamiques qui se sont avérées précieuses (cf. 4.2.3. et 5.2.2.).

4.2. Combinaison de ces technologies

4.2.1. JAVA et intranet

Si l’on se place dans un contexte intranet, l’utilisation d’applets Java permet d’enrichir considérablement l’approche intranet. Aux formulaires HTML, on peut substituer de véritables interfaces graphiques écrites en Java, possédant des capacités de traitement local et communicant suivant des protocoles quelconques avec le serveur. On entre alors véritablement dans un modèle client-serveur au sens de la programmation répartie, avec les avantages spécifiques à Java côté client, en termes de facilité de développement et de déploiement. Par ailleurs, les problèmes liés à la sécurité sont *a priori* moins sensibles et/ou mieux maîtrisés dans un contexte intranet.

Par son intégration au Web, cette approche fait du visualisateur Web l’interface universelle entre l’utilisateur et le système global.

4.2.2. CORBA et Java

La disponibilité d’un *mapping* IDL-Java permet de réaliser des applets pouvant invoquer des serveurs CORBA. Le modèle client-serveur que nous suggérions pour les applications intranet peut donc se baser sur une architecture CORBA. On bénéficie alors d’un modèle de communication de haut niveau et de services standard (nommage, événements…).

4.2.3. CORBA et Prolog

L’intégration d’un noyau de résolution de problèmes dans un serveur CORBA est une voie qui mérite véritablement d’être explorée. Dans [DIL 95], nous avions introduit ce type d’approche en intégrant un interpréteur Prolog au système réparti à objets COOLv1 [HAB 89]. L’intégration était double, puisque non seulement les objets du système disposaient d’un noyau Prolog propre, mais le dialecte Prolog était lui-même étendu par des prédicats reprenant les fonctionnalités du système.

En ce qui concerne l’expérience que nous décrivons ici, il s’agit simplement d’intégrer un noyau Prolog, sans le modifier, à un serveur CORBA. En effet, notre objectif est d’avoir un outil de raisonnement standard. Le surcoût introduit par un langage interprété mais bien optimisé n’est pas nécessairement sensible en regard des performances réseau.

En dehors de ce support à l’algorithmique avancée, l’introduction de Prolog offre une propriété rare de mise-à-jour dynamique des serveurs CORBA. Pour de nombreuses raisons, le langage le plus couramment utilisé pour implémenter des opérations définies en IDL est le C++. Lorsqu’il s’agit de mettre à jour un tel serveur, il faut l’interrompre, le recompiler et le relancer, ce qui est toujours délicat dans une architecture répartie. Or, dans notre approche, la partie développée en C++ se limite à la mise en forme d’un but Prolog, à sa résolution et à la récupération des résultats (il n’existe pas encore de mapping IDL-Prolog). Ainsi, le comportement de chaque opération correspond en définitive à des procédures Prolog, que l’on peut

dynamiquement modifier, sans interruption du serveur et en conservant son état. Cette propriété va dans le sens d’une augmentation de la capacité d’adaptation.

Une autre application du caractère interprété et déclaratif de Prolog a trait à la sauvegarde et à la restauration de l’état d’un serveur CORBA. Puisque dans notre système, toutes les opérations IDL sont implémentées par des procédures Prolog, il est assez naturel que l’état du serveur soit conservé sous forme d’un ensemble de clauses Prolog. Dès lors, il est immédiat de sauvegarder ces clauses dans un fichier afin de pouvoir interrompre le fonctionnement du serveur (e.g. pour cause de maintenance matérielle) et le relancer ultérieurement. Le serveur pourra même être relancé sur une architecture complètement différente, car cet état est totalement portable. On peut aussi facilement conserver des états, les examiner pour une mise au point, ou faire des sauvegardes régulières pour conférer un niveau élémentaire de “persistance” (tolérance aux pannes).

5. L’expérience “CIDRIA générique”

5.1. Présentation générale

5.1.1. Un workflow “générique”

L’étude “CIDRIA générique”, issue du module *workflow* du projet PRÉVISIA, a été l’occasion pour nous de concrétiser l’approche multi-agents décrite en 3. Nous avons voulu explorer une approche complémentaire de celle que nous avions toujours adoptée, qui consistait en une circulation automatisée de formulaires électroniques. L’approche que nous décrivons ici consiste à appréhender le worfklow de façon “générique”, i.e. comme une application permettant d’organiser le déclenchement et la synchronisation d’un ensemble de tâches mettant en jeu différentes ressources du système d’information. Le caractère générique est d’autant plus marqué que ces ressources sont hétérogènes : applications métier ou logiciel “transversal”, utilisateurs, matériels divers… Le workflow apparaît alors comme une structuration de haut niveau de la coopération entre les agents du système.

5.1.2. L’intégration de technologies avancées

La spécificité de CIDRIA générique tient aussi à ces choix technologiques. La plate-forme est construite sur une architecture CORBA, sur laquelle nous avons greffé trois types d’outils, conformément aux arguments du 4.

– des outils pour construire des clients CORBA multi-plates-formes : visualisateur Web, langage Java avec mapping IDL ;
– des outils pour manipuler des documents électroniques : langage HTML, serveur HTTP, visualisateur Web ;
– un outil pour la représentation des connaissances et la résolution de problèmes : interpréteur Prolog.

5.2. Le serveur et les agents

5.2.1. Principe

Le processus serveur CORBA manipule deux types d’objets serveurs CORBA. D’une part, il contient un certain nombre d’objets “agent”, serveurs CORBA auxquels se connectent les adaptateurs de chaque ressource. L’interface (au sens IDL) des objets “agent” permet de gérer entièrement l’utilisation du système, et en particulier la coopération multi-agents. D’autre part, le serveur contient un (et un seul) objet “coopérateur” qui propose deux interfaces (figure 6) :

– une interface d’administration permettant de gérer le serveur ;
– une interface de connexion, de type *agent factory* (i.e. création d’objets “agent” à la demande), qui permet aux adaptateurs d’obtenir une référence d’objet CORBA correspondant à leur agent.

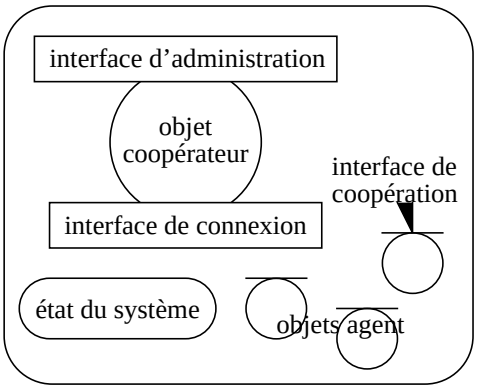


Figure 6. *Constitution du serveur*

Le processus serveur CORBA contient l’état global du système : les agents, les coopérations en cours, les types d’action et les macro-actions existantes.

Concrètement, le regroupement d’agents sur un même serveur est une nécessité opérationnelle. Cela permet d’avoir des propriétés de permanence de la représentation des ressources et de persistance de l’état des agents. De plus, par rapport à l’architecture client-serveur, il y a indépendance à la localisation du client (i.e. de l’adaptateur) et maintien de la cohérence des coopérations en cours quoi qu’il arrive côté client.

5.2.2. L’administration du serveur

Les opérations d’administration permettent de sauvegarder l’état du serveur, de restaurer un état sauvegardé précédemment et de mettre à jour l’implémentation des opérations. Elles permettent aussi d’ajouter ou de retirer un agent, une action élémentaire ou une macro-action dite *système*[16]. Une interface graphique d’administration, écrite en Java, permet de contrôler le serveur à distance par des invocations CORBA. L’ajout d’une macro-action consiste à utiliser un éditeur graphique d’ICN écrit en Java (cf. 5.3.).

Conformément à l’approche décrite en 4.2.3., l’état du serveur et les opérations sont codés en Prolog. En effet, le serveur CORBA est un programme écrit en C++, auquel on a intégré un objet résultant de l’encapsulation de la bibliothèque de SWI-

16. Une macro-action système est une macro-action définie par l’administrateur, que toutes les ressources peuvent utiliser mais qu’aucune d’entre elles ne peut modifier ou détruire.

Prolog[17]. Le rôle de la méthode C++ qui implémente une opération IDL consiste à construire un but Prolog en fonction des paramètres, à le résoudre, puis à retourner les éventuels résultats.

Le codage déclaratif des états et des procédures est totalement portable et facilite la mise au point, mais la propriété la plus marquante concerne la possibilité de modifier le comportement des opérations déclarées en IDL sans interrompre le serveur (cf. exemple de la figure 7).

déclaration IDL de l’opération

```
typedef ListeString sequence<string>;
…
ListeString accointances(); // retourne la liste des agents
…
```

implémentation de l’opération via le mapping C++

```
ListeString *accointances(
  CORBA::Environment &env = CORBA::default_environment)
{
  /* mise en forme et appel du but Prolog “liste_agents(L)”
  puis retour de la liste des éléments de L */
}
```

première implémentation en Prolog : l’opération retourne une liste non triée

```
liste_agents(L) :-
   findall(X, agent(X), L). % L est la liste des X tels que ‘agent(X)’ est vrai
```

mise-à-jour dynamique de l’implémentation :
la liste sera désormais triée, sans que le serveur ait été interrompu

```
liste_agents(L) :-
   findall(X, agent(X), L1),
   sort(L1, L). % L est la liste des éléments de L1 triés dans l'ordre croissant
```

Figure 7. *Exemple d’implémentation Prolog d’une opération IDL via C++*

5.2.3. Les agents

Les agents sont des objets serveurs dont l’interface CORBA permet aux adaptateurs de contrôler la représentation de la ressource. L’état d’un agent est caractérisé par :

– ses compétences ;
– les macro-actions personnelles définies ;
– les requêtes simples (i.e. action élémentaire) ou composites (i.e. macro-action) émises ;
– les requêtes reçues.

17. SWI-Prolog v2.7.8, par Jan Wielemaker (Université d’Amsterdam — E-mail : jan@swi.psy.uva.nl — URL http://www.swi.psy.uva.nl/usr/jan/SWI-Prolog.html).

Quant à la problématique des accointances, primordiale pour toute coopération, tous les agents se connaissent et connaissent leurs compétences au sein d’un même serveur coopérateur.

De façon analogue à l’objet “coopérateur”, les objets “agent” sont programmés en C++ et en Prolog. Les opérations sont donc également modifiables dynamiquement, et l’état des agents est intégralement représenté par des clauses Prolog. Le noyau Prolog est partagé par les différents agents et le coopérateur, ce qui interdit tout accès concurrent (SWI-Prolog n’est pas un Prolog parallèle). Le processus serveur CORBA qui contient tous ces objets n’a donc pas lieu d’être *multi-thread*.

5.3. L’adaptateur d’utilisateur

5.3.1. Une applet Java

L’adaptateur correspondant à un utilisateur est une interface graphique programmée en Java, téléchargée au sein d’un visualisateur Web, depuis un serveur HTTP où se trouve également le serveur “coopérateur” (voir figure 8).

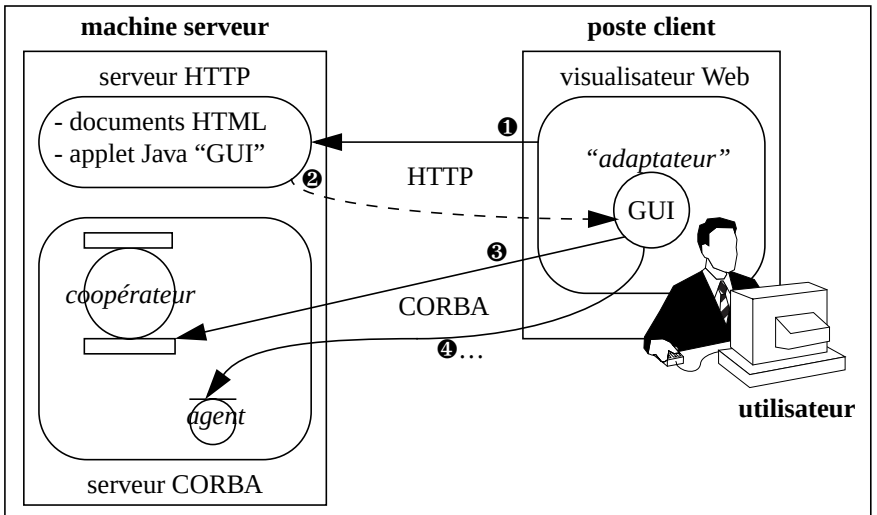


Figure 8. *Intégration de l’utilisateur au système*

Via cette applet, l’utilisateur peut déclarer la liste de ses compétences, ce qui revient à choisir un certain nombre d’actions élémentaires parmi celles disponibles. Il peut aussi consulter la liste des requêtes reçues, répondre à l’une d’entre elles ou en

rejeter une autre. Enfin, il peut émettre une requête simple ou composite, consulter la liste des requêtes émises et voir leur état.

Qu’il s’agisse de l’émission d’une requête ou de la déclaration des compétences, on notera que le principe des contraintes n’a pas encore été implémenté dans le système. En réalité, cela est d’ores et déjà prévu dans les déclarations IDL, mais la mise en œuvre effective demande un travail supplémentaire pour introduire la définition des caractéristiques des agents et des services, la spécification des contraintes par l’utilisateur, et leur prise en compte par les agents. Ceci a déjà été réalisé dans un système de réservation de salle multi-agents ([DIL 96]), avec notamment un moteur de résolution de contraintes écrit en Prolog qui serait assez directement réutilisable.

5.3.2. Emission d’une requête

L’utilisateur peut choisir une tâche quelconque parmi trois listes :
– actions élémentaires,
– macro-actions personnelles,
– macro-actions système.

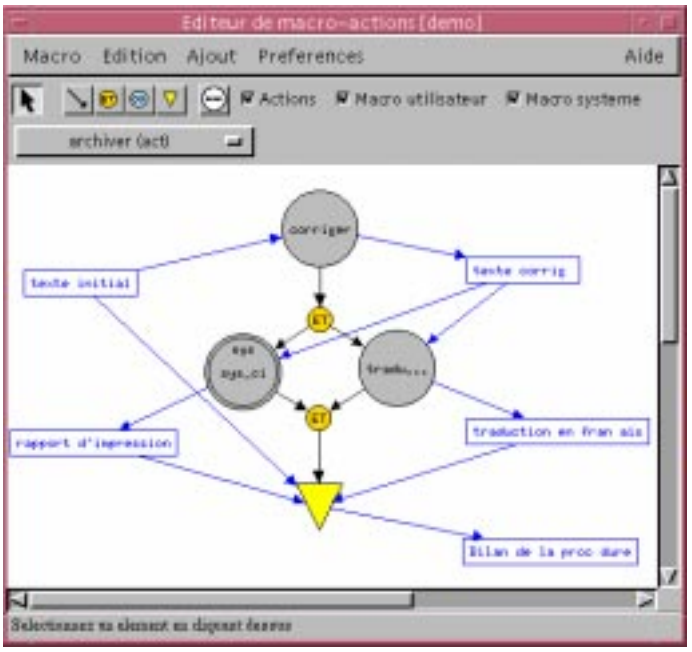
Des macro-actions peuvent être définies à loisir par l’utilisateur, en combinant des actions élémentaires, des macro-actions personnelles et des macro-actions système. Pour cela, l’utilisateur dispose d’un éditeur graphique de procédure (figure 9) reprenant le formalisme décrit en 3.2.2. Cet éditeur réalise très peu de contrôle de cohérence. L’essentiel de ces contrôles est réalisé par l’opération d’ajout de macro-action, qui retourne une exception si le graphe n’est pas correct. Cette exception contient un code d’erreur, un message, ainsi qu’une liste de nœuds incriminés, ce qui facilite la mise au point du schéma. Les macro-actions sont enregistrées sous forme de clauses Prolog, et les vérifications sont réalisées par une procédure Prolog, ce qui s’est véritablement révélé très commode.

Quel que soit le type de la tâche sélectionnée, l’utilisateur doit fournir un certain nombre de paramètres spécifiques avant d’émettre la requête. Cette requête vient ensuite s’ajouter à la liste des requêtes émises.

5.3.3. Le suivi des requêtes émises

Dans la liste des requêtes émises, l’utilisateur peut :
– consulter l’état d’une requête,
– supprimer une requête,
– archiver l’état d’une requête.

Consulter l’état d’une requête simple revient à vérifier si elle est terminée, si elle a échoué, ou si elle est en cours. Lorsqu’il s’agit d’une requête composite, ces trois états existent au niveau global ainsi qu’au niveau des différentes actions ou macro-actions qui la composent. Pour cette raison, un visualisateur graphique de suivi permet de détailler l’état d’avancement selon une apparence analogue à l’éditeur de macro-action (figure 10).



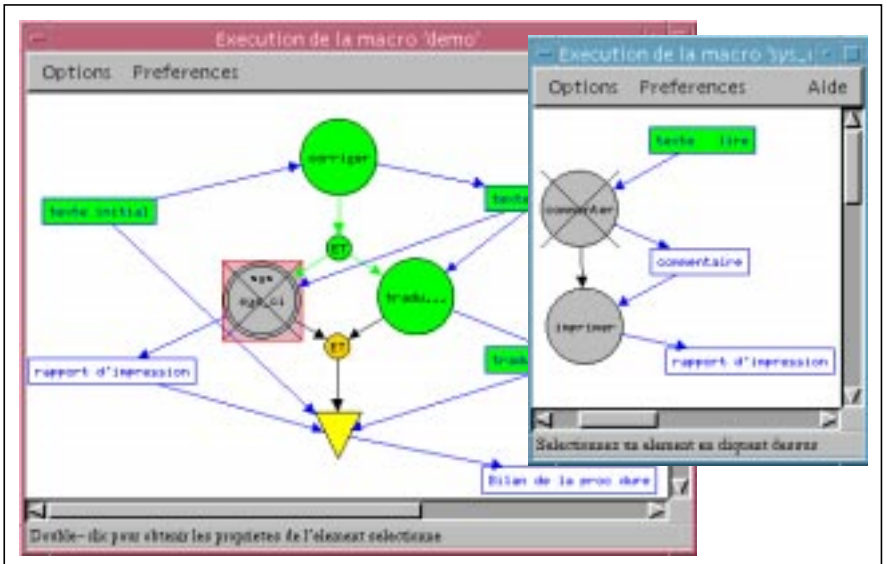
Dans cet exemple de macro-action, l’étape de départ consiste à produire un “texte corrigé” à partir d’un “texte initial” en mettant en œuvre l’action élémentaire “corriger”. Une fois cette action terminée, un parallélisme est introduit entre une action de traduction du “texte corrigé” et une macro-action système “sys_ci”. Celle-ci consomme le “texte corrigé” et produit un “rapport d’impression” (il est possible de visualiser le schéma de “sys_ci”). Après une synchronisation de “sys_ci” et de l’action de traduction, le système réalise une concaténation de la “traduction en français” et du “rapport d’impression” pour créer une unique information “Bilan de la procédure”.

Remarque : le “texte initial” étant consommé sans être produit au préalable, il s’agit d’un paramètre d’entrée de la macro-action.

Figure 9. *L’éditeur de macro-action — exemple de procédure*

Outre la mise en évidence des actions franchies, en cours, ou ayant échoué, le visualisateur offre la possibilité de voir les informations recueillies et de connaître le nom des agents contactés.

Là encore, l’utilisation de Prolog pour réaliser le moteur d’exécution des macro-actions s’est avérée particulièrement adaptée.



Dans cet exemple d’exécution de la macro-action décrite figure 9, les actions “corriger”, “traduire” et le connecteur de parallélisme ont été franchis. Cela signifie que des requêtes élémentaires ont été émises et acceptées par des ressources du système. Aussi, les informations “texte initial”, “texte corrigé” et “traduction en français” ont pu être collectées. En revanche, le parcours du schéma est bloqué sur le connecteur de synchronisation en raison de l’échec de la macro-action “sys_ci”. En examinant l’état d’avancement de cette macro-action système, on se rend compte que c’est l’action “commenter” qui n’a pas pu être réalisée, soit par absence de ressource ayant la compétence ad hoc, soit parce que les ressources compétentes ont toutes rejeté la requête.

Figure 10. *Le visualisateur de suivi d’exécution*

5.3.4. L’archivage de l’état des requêtes

L’état d’une requête peut être conservé sous forme d’un document électronique. Il existe en effet une action particulière d’archivage (figure 14) qui engendre un fichier HTML et le place dans une arborescence propre à l’utilisateur, sur un serveur HTTP.

5.3.5. Traitement des requêtes reçues

Les requêtes reçues par un agent consistent forcément à effectuer une tâche élémentaire. L’utilisateur consulte la liste des requêtes reçues, auxquelles il peut répondre, en consultant les paramètres transmis et en renseignant les arguments produits par l’action (i.e. les *résultats* de l’action). L’utilisateur peut également rejeter une requête. Dans ce cas, l’agent émetteur de la requête cherchera un autre agent ayant la compétence ad hoc. La requête est en échec si tous les agents contactés la

rejettent, ou si aucun agent ne possède la compétence requise. Si cette requête émane d’une requête composite, celle-ci est également en échec.

5.4. Autres adaptateurs

5.4.1. Propriétés communes aux adaptateurs

Le modèle multi-agents associé à notre palette d’outils permet d’intégrer très rapidement des ressources pré-existantes : la constitution des agents est uniforme, et les adaptateurs, seuls composants spécifiques, représentent très peu de développement.

Le cas de l’interface utilisateur nous a permis de détailler comment se déroule la coopération et l’exécution d’une action ou d’une macro-action. Il s’agit d’un cas particulier d’adaptateur, dont nous allons présenter quelques autres exemples. Les adaptateurs possèdent des propriétés communes :

– il s’agit de clients au sens CORBA,
– ils jouent un rôle mixte client/serveur selon la vision multi-agents,
– ils peuvent être installés sur n’importe quelle machine reliée au réseau.

Le comportement typique d’un adaptateur consiste à consulter périodiquement sa boîte aux lettres, i.e. la liste des requêtes reçues. Il traite alors les requêtes dans l’ordre d’arrivée, soit en réalisant l’action correspondante, soit en rejetant.

5.4.2. Imprimante

L’adaptateur d’imprimante établit le lien entre un agent ayant la compétence “imprimer” (figure 11) et un serveur d’impression.

Il consulte périodiquement la liste des requêtes reçues. Le traitement d’une requête consiste à effectuer l’impression et à retourner un rapport d’exécution spécifiant, par exemple, la pièce où le document est imprimé.

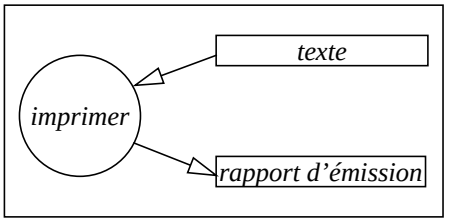


Figure 11. *Action “imprimer”*

5.4.3. Traducteur

Le traducteur est un exemple d’adaptateur intégrant la ressource. En effet, il consulte périodiquement sa boîte aux lettres et traite lui-même les requêtes de traduction. Il prend un texte en paramètre, traduit en français des noms d’animaux figurant dans ce texte, et retourne le texte résultant (figure 12).

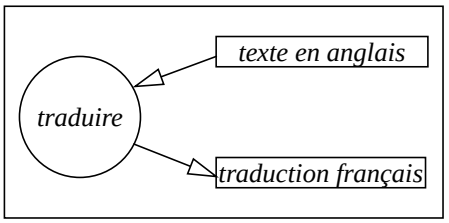


Figure 12. *Action “traduire”*

5.4.4. E-mail

L’adaptateur de messagerie électronique est un exemple qui montre comment notre approche peut s’intégrer à l’intranet. Cet adaptateur dialogue suivant deux protocoles :

– avec son agent CORBA,

– avec un serveur SMTP.

Les paramètres et le résultat de l’action rendue par cet adaptateur sont indiqués par la figure 13.

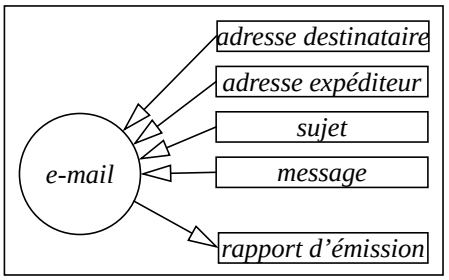


Figure 13. *Action “e-mail”*

5.4.5. Archiveur

Autre illustration du caractère intranet, l’archiveur permet de générer un fichier HTML à partir d’un document, et de le placer au sein d’un serveur HTTP en fonction d’un plan de classement. Comme l’indique la figure 14, ce plan est structuré suivant une arborescence à 4 niveaux : dossiers, tiroirs, armoires et une racine implicite.

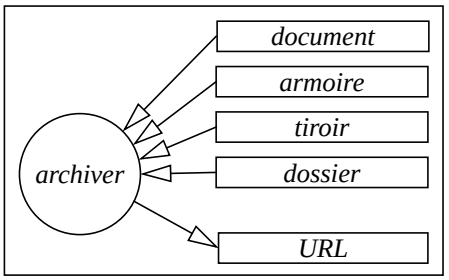


Figure 14. *Action “archiver”*

Cette action et cet adaptateur ont un rôle très important dans le système, puisqu’ils réalisent l’archivage de l’état des requêtes (cf. 5.3.4.). L’adaptateur rend également un service de désarchivage, qui permet de retirer un document en fournissant son URL en paramètre.

5.5. Bilan de l’expérience

La combinaison des technologies choisies (Prolog, Java, Web), autour de l’infrastructure de communication offerte par CORBA, apparaît comme un bon support pour réaliser le modèle multi-agents proposé.

Le caractère multiplate-forme offert par le téléchargement des applets Java (cf. interface utilisateur) fonctionne bien en général[18]. Cette association de Java avec le Web simplifie ainsi le développement et le déploiement des clients CORBA. De plus, elle offre un accès banalisé au système d’information, adapté aux utilisateurs nomades.

L’intégration d’un langage de type Prolog dans un serveur CORBA se fait sans difficulté et n’est pas pénalisante au vu des accès réseau. Elle nous a apporté un outil

18. Quelques problèmes ont été rencontrés avec certains visualisateurs Web, sur certains environnements.

adapté à la résolution de problèmes (cf. représentation des compétences, gestion de la coopération). De plus, son caractère interprété a considérablement facilité la mise au point du serveur, et offre la possibilité de modifier son comportement en cours de fonctionnement.

La rapidité avec laquelle ce travail a été effectué prouve que ces différentes technologies sont suffisamment mûres pour que leur assemblage proposent des solutions pertinentes aux problèmes posés.

6. Conclusion

L’implémentation de notre modèle sur l’architecture proposée offre une coopération à la fois

– structurée (niveau macroscopique permettant de définir un enchaînement de tâches impliquant diverses ressources),
– et souple (prise en compte du contexte d’exécution dans le choix des ressources).

Dans cette implémentation, les agents sont co-localisés et liés à un unique “coopérateur”. Sans modifier le modèle multi-agents et l’architecture définie, l’étape suivante consisterait à introduire un niveau de coopération entre plusieurs coopérateurs répartis à travers le réseau.

Remerciements

Merci au projet PREVISIA pour nous avoir offert l’occasion de concrétiser nos idées. Merci à Philippe Maurice pour son ferme soutien au sein de PREVISIA. Merci à Jan Wielemaker de l’Université d’Amsterdam pour la qualité de SWI-Prolog et de sa documentation. Merci à Christophe “RFC-Tof” Trompette, pour sa contribution initiale à notre approche multi-agents et ses interventions de dépannage. Merci à Bruno Hocq et Thierry Larigot pour la qualité de leur travail sur l’éditeur graphique de macro-action. Merci à Jean-Marc Deshayes et René de Smet pour la transmission de leur expérience et leur aide au sein du module CIDRIA.

Références bibliographiques

[BEY 95] Claire Beyssade, Patrice Enjalbert, Claire Lefèvre. *Cooperating logical agents: model, applications.* IJCAI-95 Workshop on Agent theories, architectures, and languages. Montréal, 1995, pp. 1-14.

[BOU 89] François Bourdon, Anne Béguin, Jean-Marc Deshayes, Didier Tourrade, Pierre Touzeau. *CIDRE: intelligent circulation of distributed folders.* 5th International Workshop on Telematics, Denver, 1989.

[COM 91] *The COMANDOS Distributed Application Platform.* Research report, ESPRIT project 2071, 1991 Springer-Verlag, vol. 1, pp. 123-139.

[DIL 95] Bruno Dillenseger, François Bourdon. *Towards a multi-agent model for the office information system : a Prolog-based approach.* Practical Applications of Prolog 1995, pp. 191-200.

[DIL 96] Bruno Dillenseger. *Une approche multi-agents des systèmes de bureautique communicante.* Thèse de doctorat de l'Université de Caen, novembre 1996.

[ELL 79] Clarence A. Ellis. *Information Control Nets: A mathematical model of office information flow.* Conference on Simulation, Measurement and Modeling of Computer Systems, 1979, pp. 225-239.

[GAS 91] Les Gasser. *Social conceptions of knowledge and action: DAI foundations and open systems semantics.* Artificial Intelligence 47, pp. 107-137, Elsevier 1991.

[HAB 89] Sabine Habert. *Gestion d'objets et migration dans les systèmes répartis.* Thèse de doctorat de l'Université Paris-VI, décembre 1989.

[HEW 91] Carl Hewitt. *Open information systems semantics for DAI.* Artificial Intelligence 47, pp. 79-106, Elsevier 1991.

[HUH 87] M. Huhns. *Distributed Artificial Intelligence.* London, Pitman, 1987.

[ISA 90] *Esprit project ISA : Enterprise and Information Modelling*. octobre 1990.

[KUW 95] Kuwabara, K., Ishida, T., and Osato, N.. *AgenTalk: Coordination Protocol Description for Multiagent Systems*, Proc. First International Conference on Multi-Agent Systems (ICMAS '95) p. 455.

[LEA 89] Rodger Lea, Vadim Abrossimov, Jean-Marc Deshayes. *The CIDRE distributed system based on Chorus.* Proc. of TOOLS'89, Paris, pp. 521-530.

[MAU 90] Philippe Maurice, Anne Béguin. *Présentation du projet SEPT-CNET "AMBIANCE"*. Document interne, janvier 90.

[ODP] *Open Distributed Processing.* Normes ISO 10746-1,2,3,4 (ou ITU-T X.901, X.902, X.903, X.904)

[OMG 95] *The common Object Request Broker: Architecture and specification (revision 2.0)*. Object Management Group, juillet 1995.

[SMI 80] Reid 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, décembre 1980, pp. 1104-1113.

[THI 93] R.A. Thiétart. *Ordre et chaos dans les Organisations*. Journée d'étude AFCET "Les systèmes d'information, autonomie et chaos", Paris, 1993.