Une contribution du multi-agent aux applications de travail coopératif

Jean-Christophe Routier — Philippe Mathieu

Equipe SMAC – LIFL – CNRS UPRESA 8022
Université des Sciences et Technologies de Lille
{routier,mathieu}@lifl.fr

RÉSUMÉ. Cet article présente une application de travail coopératif dont l'objectif est la tenue de conférences entre intervenants géographiquement distribués. Cette application fournit les capacités indispensables de visualisation et de navigation dans les diapositives. Une facette intéressante dans une conférence est que les rôles tenus par les différents intervenants peuvent évoluer dynamiquement : ici un simple auditeur peut devenir le conférencier et réciproquement. L'application développée doit donc prendre en compte cet aspect. Cette application a été réalisée à l'aide de la plate-forme multi-agents MAGIQUE. *Les difficultés liées à la distribution sont donc masquées par le mécanisme «naturel» de communication entre agents. De plus, une «analyse agent» du problème permet, par la mise en évidence des différents rôles et compétences, de faciliter l'approche du développeur et l'évolutivité de l'application.* MAGIQUE *offre une solution effective à l'évolution dynamique des rôles dans l'application en permettant l'évolution dynamique des capacités des agents. Le but essentiel de cet article n'est pas tant de montrer un nouveau collecticiel que de montrer comment celui-ci est facilement développable, maintenu et extensible grâce à l'approche multi-agent et à la plate-forme* MAGIQUE.

ABSTRACT. This article presents a co-operative work application dedicated to conferences with geographically remote people. This application provides the essential slide visualization and manipulation fonctionnalities. An intersting point in a conference is that the role plaid by the different members can dynamically evolved: here, a plain listener can become the speaker and conversely. The developped application must then take that into account. A prototype of this application has been developped using the multi-agent platform MAGIQUE. *Therefore, problems that arise from distribution are hidden by the «natural» communications between agents. Moreover, an agent oriented analysis of the problem lightens the different role and skills and thus eases the programmer task and the evolutivity of the application.* MAGIQUE *offers a real solution to the dynamic evolution of the roles in the application by providing the support to dynamic evolution of agent. The main goal of this article is not really to present a new groupware but rather to show how it can be easily developped, maintained and extended thanks to* MAGIQUE *and its multi-agent approach.*

MOTS-CLÉS : collecticiel, assistant, multi-agent, environnement de développement agent, MAGIQUE

KEYWORDS: groupware, assistant, multi-agent, agent programming framework, MAGIQUE

RSR-CP – 13/2001. Les télé-applications, pages 207 à 232

1. Introduction

Depuis quelques années, de nombreux chercheurs et industriels se sont penchés sur les applications de travail coopératif ([BAE 93, BAU 98, DER 95, ELL 91]). Il est en effet de plus en plus nécessaire d'offrir des outils permettant à différentes personnes éloignées physiquement et qui doivent travailler collectivement, de communiquer, de s'échanger des documents et de présenter leurs informations à l'ensemble d'une équipe. Actuellement, force est de constater que les quelques outils présents sur le marché n'offrent en général qu'une partie de ces possibilités. Ces outils sont difficiles à appréhender et surtout difficiles à étendre par les développeurs, tant les nouveaux concepts à ajouter sont parfois divers. Chaque application a en effet ses propres spécificités et, au delà de l'aspect traditionnel propre à chaque collecticiel, il est en général nécessaire d'ajouter de nouvelles fonctionnalités.

Parallèlement, les concepts du multi-agents se sont développés [FER 95]. Même s'il est encore difficile de trouver une définition réellement consensuelle au terme d'*agent* [FRA 96], les définir comme des *entités automones, réactives et communicantes* constitue une base généralement acceptée et reconnue. Les champs d'applications se sont également étoffés : agents assistants, éducation [REP 00, ROS 99], simulations [MAR 98], jeux [TRA 96], etc. A travers ces différentes applications, le multi-agent s'est également affirmé comme offrant une nouvelle approche de la programmation [SHO 93].

Ces dernières années, les travaux sur les modèles multi-agents [BEN 97, FER 98, MAR 98] se sont concrétisés à travers la réalisation de plate-formes de conception [GUT 00, ROU 00]. Ces plates-formes multi-agents offrent des outils particulièrement bien adaptés à la conception modulaire d'applications multicompétences telles que celles mentionnées précédemment. Les propriétés de réactivité et proactivité des agents permettent en effet un développement souple et évolutif de telles applications. De plus, l'interprétation en termes de rôles et de compétences des capacités des agents facilite le travail d'analyse et de décomposition.

Cet article présente une application coopérative basée sur la plate-forme multi-agents MAGIQUE développée par l'équipe SMAC[1] du LIFL[2] [BEN 97] (voir [TAR 99] pour un autre exemple d'approche du collecticiel par le multi-agent). L'objectif de cette application est d'offrir à l'utilisateur la possibilité d'échanger des informations, de type diapositive au format HTML (et ce qui va avec : applets, sons, images, hyperliens, etc.), à l'ensemble des collaborateurs. L'analyse fait apparaître deux types de rôles : le professeur ou *conférencier* et les élèves ou *auditeurs*. La différence sur ces rôles se détermine par la détention par le *conférencier* d'une unique télécommande. Elle permet de passer des diapositives aux auditeurs et elle peut éventuellement être demandée par les auditeurs. Il n'y a donc qu'un seul conférencier à un instant donné, mais le détenteur de ce rôle peut être amené à changer au cours de la conférence pour

1. Systèmes Multi-Agents et Coopération.
2. Laboratoire d'Informatique Fondamentale de Lille.

qu'un auditeur puisse lui aussi passer ses diapositives. A côté de cette fonctionnalité *sine qua non*, un ensemble d'outils de manipulation est fourni.

Le développement de cette application se base sur le système multi-agents MAGIQUE, qui est avant tout un système de conception d'applications agents physiquement distribuées sur réseau hétérogène. MAGIQUE utilise en totalité la notion de compétences qui existe dans ce SMA. En MAGIQUE, tout est compétence [ROU 00]. Etre auditeur est une compétence au même titre que posséder la télécommande. MAGIQUE offre des fonctionnalités importantes de gestion et d'échange de ces compétences et il est alors très aisé de passer la télécommande d'un utilisateur à un autre grâce à cet échange de compétences. Dès lors, les changements de rôles ne sont pas simplement des drapeaux disposés dans le code source, mais bien des changements dynamiques de rôles. Un agent pouvant avoir autant de compétences qu'il le souhaite, il est aisé de rajouter de nouvelles fonctionnalités au système pour qu'il couvre le champ souhaité par l'entreprise. Nous avons par exemple, après avoir réalisé une première version du prototype, ajouté une compétence »*assistant*« à nos *auditeurs* sans que cela n'ait aucun impact sur l'existant. L'assistant est une entité «intelligente», proactive et réactive, qui aide l'utilisateur dans sa gestion du travail coopératif en lui permettant de mieux contribuer à la conférence. L'assistant permet également une modélisation des autres intervenants aux conférences en déterminant leurs champs de compétences.

Le but essentiel de cet article n'est donc pas tant de montrer un nouveau collecticiel que de montrer comment celui-ci, grâce à l'approche multi-agents et à la plate-forme MAGIQUE et sa notion de compétences, est facilement développable, maintenu et extensible.

Dans une première partie, nous présenterons l'application et ses différentes fonctionnalités, et notamment l'assistant. Puis, après une très brève présentation de la plate-forme MAGIQUE, nous décrirons le principe de réalisation d'une telle application à l'aide de SMA.

2. L'application diapo-conférence

L'objectif de cette application de travail coopératif est de fournir un support permettant la tenue d'une conférence entre des intervenants physiquement distribués.

Chacun des intervenants dispose d'un ensemble de ressources documentaires («*diapositives*»). Dans une première version de cette application, ces ressources sont décrites par des documents HTML. Le choix de ce format de document se justifie facilement, dans la mesure où il offre un support unique pour des documents de formes très différentes : texte, images, animations, dynamicité *via* les applets, etc. Un second argument en sa faveur est la facilité à se procurer des interprètes et visualisateurs pour de tels documents. Cependant, le choix de ce format n'est pas une contrainte forte de notre application. Comme nous le montrerons par la suite, le développement basé sur le système multi-agent MAGIQUE et son principe de compétence, permet une évolu-

tion facile de l'application. Des documents au format XML pourraient ainsi être pris en compte par l'application sans qu'il y ait beaucoup à changer.

L'application doit lui permettre de diffuser ces documents auprès des autres. Il ne peut cependant le faire qu'à la seule condition qu'il soit détenteur de la *télécommande* (unique) associée à l'application. Cette télécommande lui permet de parcourir et de diffuser ses ressources. Mais elle tient également lieu de «pointeur» que le conférencier peut utiliser pour attirer l'attention des autres utilisateurs sur un point précis du document diffusé, ce pointeur étant visible par tous les utilisateurs en temps réel.

On distingue donc deux *rôles* dans cette application : celui de *conférencier* pour celui qui détient la télécommande, et celui d'*auditeur* pour les autres participants. Comme dans une conférence réelle, le conférencier peut à tout moment céder la télécommande à un auditeur et les rôles respectifs tenus par les intervenants évoluent alors dynamiquement. Une attention particulière a été portée au respect de la conformité avec la tenue des conférences réelles.

Les figures 1 et 2 donnent un aperçu de l'application du point de vue du conférencier et du point de vue d'un auditeur quelconque. On peut distinguer différents outils sur les deux vues : la visionneuse sur la gauche des écrans (on peut remarquer que celle de l'auditeur a bien la même taille que celle du conférencier), le pointeur de souris que l'on peut apercevoir dans la visionneuse du conférencier est visualisée par un point dans la fenêtre de l'auditeur. D'autres outils comme la télécommande (active chez le conférencier et passive chez l'auditeur), les fenêtres d'équipe et de messagerie, ainsi que l'assistant en haut de chaque écran (il fait une annonce dans le fenêtre du conférencier) sont également visibles.

2.1. *Les fonctionnalités de base*

Dans l'application réalisée, les ressources sont désignées par des URL. L'avantage de ce format est évident. Il permet de désigner des documents correspondant à différents supports (texte, image, sons, vidéo, etc.). De plus, il permet de profiter des capacités de navigation liées aux hyper-liens ainsi que d'obtenir des documents dynamiques grâce aux applets par exemple. Chaque participant possède ses propres ressources et il peut les regrouper et les organiser afin de constituer un exposé structuré.

Afin de permettre la tenue de la conférence, les deux outils indispensables sont la *visionneuse* et la *télécommande*.

2.1.1. *La visionneuse*

Chaque participant dispose d'un support permettant la visualisation des ressources (il s'agit dans notre cas d'un navigateur d'URL).

Le conférencier a le contrôle des interactions de cet outil chez l'ensemble des participants. Les vues dans les différentes visionneuses sont commandées par le char-

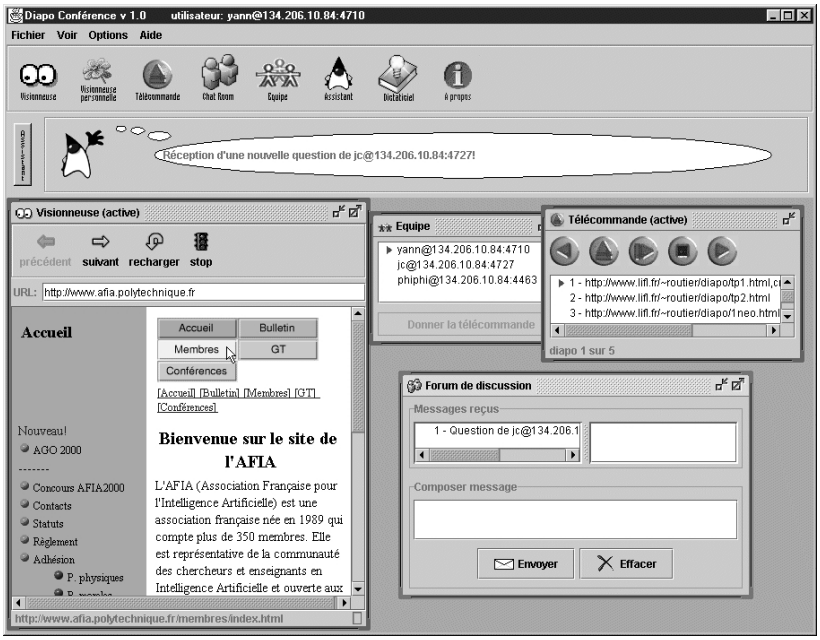

FIG. 1. *L'application du côté* conférencier

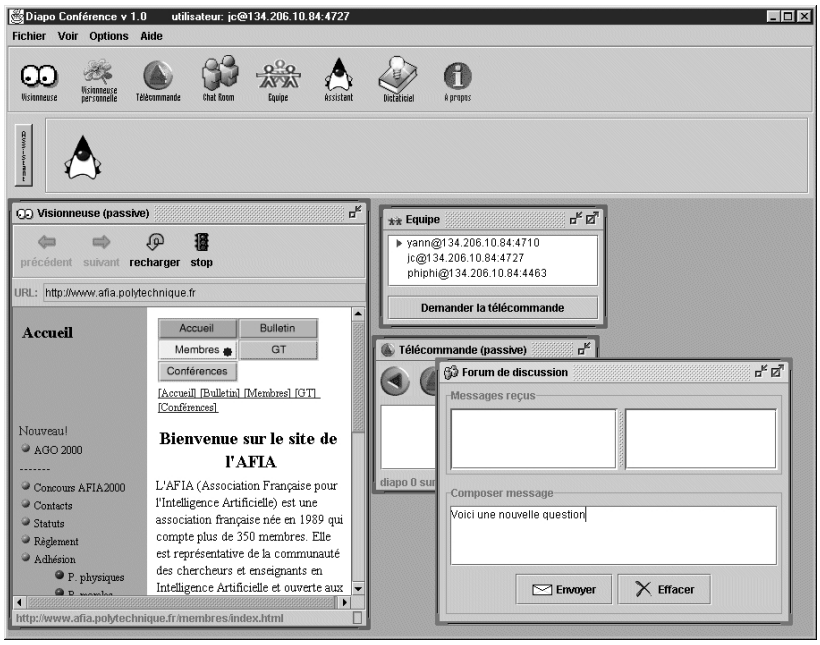

FIG. 2. *L'application du côté* auditeur

gement d'un document dans sa visionneuse par le conférencier, selon le principe du WYSIWIS [STE 87]. En fait, les différentes visionneuses des intervenants sont totalement synchronisées sur celle du conférencier. Ainsi, le pointeur de sa souris dans cette visionneuse est visualisé chez tous les intervenants, ce qui lui permet de mettre en évidence tel ou tel point de la diapositive (cf. figures 1 et 2). Un autre exemple : si le conférencier modifie la taille de la visionneuse sur son poste, la taille des visionneuses de chaque auditeur est également modifiée ; il en est de même lors du défilement du document à l'aide des ascenseurs.

2.1.2. *La télécommande*

La seconde fonctionnalité indispensable est bien sûr la télécommande. Elle est active chez le conférencier et inactive chez les auditeurs. A tout moment, un auditeur peut la demander et c'est au conférencier que revient la décision de céder ou non la télécommande. Son rôle premier est, lorsqu'elle est active, de permettre la diffusion à toute l'assemblée d'une diapositive ou d'enchaîner une séquence de diapositives automatiquement. En fait, le conférencier transmet aux différents intervenants l'adresse de l'URL du document qu'il souhaite diffuser, les visionneuses de chaque client ont donc la charge du téléchargement de la ressource et de son interprétation.

La télécommande fournit également les outils permettant de rassembler et d'organiser des diapositives en un exposé (cf. figure 3). C'est pourquoi la télécommande n'est pas simplement masquée pour le conférencier lorsqu'elle est inactive. Tout participant peut en effet construire, comme lors de la préparation d'un cours, un scénario pour l'enchainement de ses diapositives. Il peut alors réarranger à tout moment l'ordonnancement des diapositives et réorganiser l'exposé. La possibilité de prévisualiser localement les diapositives facilite cette tâche. Les différents scénario, construits peuvent évidemment être sauvegardés pour une réutilisation ultérieures.

2.2. *Les outils d'aide à la coopération*

En plus des fonctionnalités minimales présentées précédemment, l'application offre plusieurs outils qui favorisent la coopération et la prise de conscience par chacun des autres intervenants. Citons en trois principalement :

– la «visualisation» de l'assistance,

– un assistant,

– un système de messagerie.

Nous allons les présenter brièvement.

2.2.1. *La fenêtre des intervenants*

Le premier outil a pour objectif de souligner auprès de l'utilisateur qu'il «n'est pas seul au monde». La liste des participants est ainsi visualisable dans une fenêtre dans

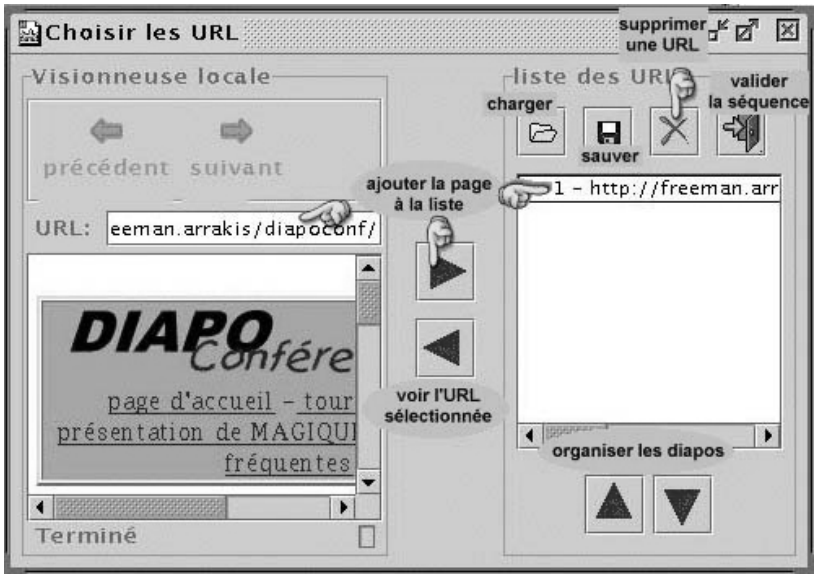

FIG. 3. *Création de séquences de diapositives*

laquelle le conférencier courant est clairement identifié. Cette nécessité de matérialiser une «conscience collective» est un problème bien connu des collecticiels [BRI 97].

La demande de télécommande peut se faire depuis cet outil. Les différentes requêtes effectuées par les différents auditeurs sont visualisables, ainsi que leur «ancienneté», par tous. Les noms des demandeurs sont en effet surlignés, et cet effet disparaît progressivement avec le temps.

L'intérêt est double : d'une part le conférencier peut directement visualiser depuis combien de temps une demande a été faite et peut par exemple considérer qu'une demande est «ancienne» et n'est probablement plus justifiée ; d'autre part, du point de vue de l'auditeur, celui-ci peut quant à lui hésiter à émettre une demande lorsqu'il s'aperçoit qu'il y en a déjà de nombreuses autres avant la sienne (cela correspond au nombre de doigts levés dans une assistance réelle).

2.2.2. *L'assistant*

Un second outil est présent pour permettre à chacun de mieux apporter sa contribution à la coopération. Il s'agit d'un *assistant* qui, comme son nom l'indique, doit faciliter l'exploitation de l'outil pour l'utilisateur.

On peut citer trois rôles principaux pour cet assistant :

– mise en évidence d'événements,

– aide à la participation à la coopération,

– modélisation des autres intervenants en vue de futures collaborations.

2.2.2.1. Mise en évidence d'événements

La nature distribuée de l'application implique la nécessité de valoriser un certain nombre d'événements qui seraient plus immédiatement perceptibles dans une conférence réelle. On peut citer par exemple les arrivées/départs des intervenants, le changement de conférencier, les messages envoyés par les autres (cf. figure 4), etc. La contextualisation des réactions de l'interface graphique de l'assistant en fonction des événements qu'il signale aide l'utilisateur en l'orientant vers les outils concernés. Cette fonction contribue donc à faciliter l'utilisation de l'application.

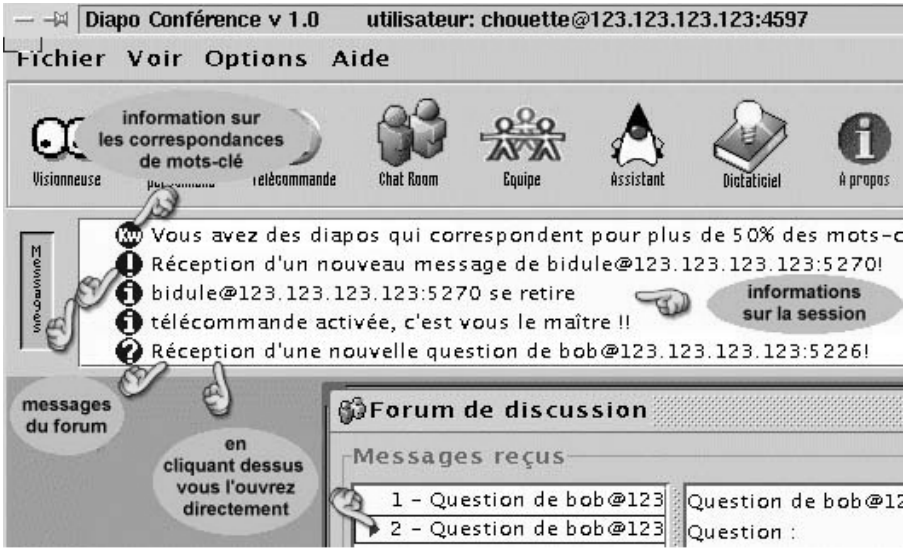

FIG. 4. *Evénements dans l'assistant*

2.2.2.2. Aide à la coopération

Parmi les «réactions» mentionnées, l'une d'entre elles fournit une aide à la construction de la coopération. En effet, lors de la tenue d'une diapo-conférence, l'assistant va aider l'utilisateur auditeur à apporter sa contribution à l'exposé. Pour chaque nouvelle diapositive transmise par le conférencier, l'assistant prévient l'auditeur quand il existe dans sa base de ressources une diapositive dont la thématique est corrélée à la diapositive courante. L'auditeur peut alors consulter le ou les documents sélectionnés par l'assistant dans une visionneuse locale, ce qui lui permet de juger par lui-même de la pertinence ou de la plus-value dans le discours courant de celui-ci. Il peut ensuite demander la télécommande pour devenir conférencier et ainsi «prendre la main». Mais il peut également choisir de demander à son assistant de transmettre la diapositive à

l'assistant du conférencier. Ce dernier prévient alors son utilisateur qu'une diapositive lui est proposée par un auditeur et celui-ci peut la diffuser s'il la juge pertinente.

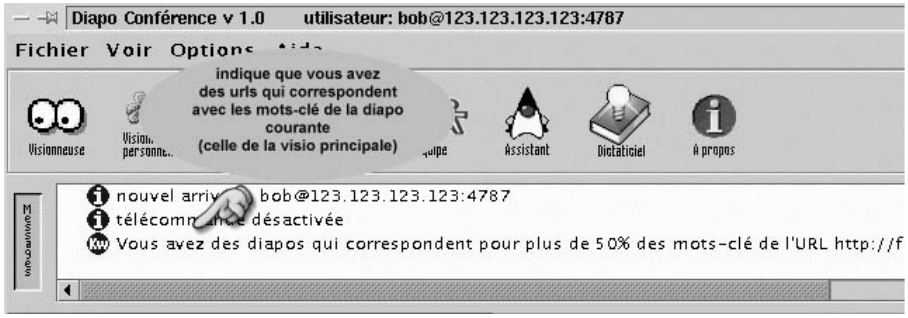

FIG. 5. *L'assistant m'avertit que j'ai des ressources corrélées*

Dans la version actuelle, l'aide à la décision fournie par l'assistant se fonde sur un système de mots-clés qui doivent avoir été précisés au niveau de chaque ressource (cf. figure 5). L'assistant consulte la base de ressources de son utilisateur et il se manifeste dès qu'un nombre suffisant de mots-clés correspond. Le critère de corrélation retenu est paramétrable par l'utilisateur.

2.2.2.3. Modélisation des intervenants

Mais les mots-clés peuvent avoir une autre utilité et être exploités différemment par l'assistant. Il est en effet possible de relever pour chaque diapositive diffusée, les mots-clés concernés et construire ainsi progressivement une image des domaines de compétences du conférencier, mais aussi une image des sujets d'intérêt des auditeurs. Ainsi, au fil des différentes sessions de conférence, une modélisation des différents intervenants et de leurs champs d'expertise peut être réalisée. Cette connaissance est ensuite utilisée pour faciliter la construction de conférences en invitant les experts sur les thèmes visés ou en prévenant les auditeurs qu'un sujet les intéressant sera abordé. Cette information pourra également être exploitée en dehors du cadre des conférences lorsque l'utilisateur cherche une information sur un domaine précis, il peut demander à son assistant s'il a connaissance d'une personne compétente sur le sujet.

2.2.3. *Le système de messagerie*

Enfin, un système de messagerie permet à chaque auditeur de poser une question au conférencier (et rien qu'à lui), alors que ce dernier peut diffuser un message (annonce ou réponse à une question) vers tous les auditeurs. Il est donc possible au conférencier de s'adresser à tous les auditeurs et de leur transférer les questions qui lui sont posées s'il trouve cela pertinent.

3. MAGIQUE

Nous avons présenté succinctement dans la section précédente les fonctionnalités offertes par l'application de diapo-conférence. Le problème du choix d'un outil adapté au développement d'une telle application distribuée se pose. Nous soutenons que l'utilisation de plates-formes de développement multi-agent, et plus particulièrement la plate-forme MAGIQUE, facilite la réalisation de telles applications.

L'avantage d'un environnement de développement multi-agents tel que MAGIQUE est qu'il fournit toutes les fonctionnalités de distribution des agents ainsi que les mécanismes de communication entre agents. Cela permet au développeur de se débarrasser de ces problèmes pour se concentrer sur les aspects purement applicatifs. Il s'agit dans ce cas de déléguer aux agents la gestion des différentes parties de l'application ; on peut ainsi utiliser un ou plusieurs agents pour chaque client.

Nous allons présenter *rapidement* la plate-forme multi-agents MAGIQUE et nous montrerons ensuite comment elle a été utilisée pour développer l'application diapo-conférence.

3.1. *Présentation de* MAGIQUE

MAGIQUE signifie «Multi-AGent hiérarchIQUE»[3]. MAGIQUE est à la base un modèle d'organisation d'agents qui propose une organisation hiérarchique de systèmes multi-agents. Cette structure permet de proposer un mécanisme de délégation de compétences entre agents, facilitant ainsi le développement.

Concrètement, MAGIQUE existe sous la forme d'une API JAVA permettant le développement de systèmes multi-agents (SMA) hiérarchiques. Dans ce cas, MAGIQUE correspond à un *framework* générique pour le développement de SMA. Le concepteur de SMA peut donc s'appuyer sur les fonctionnalités offertes par MAGIQUE telles que la communication entre agents, la distribution de l'application, l'évolution dynamique, etc., charge à lui de développer les *compétences* applicatives dont il a besoin.

MAGIQUE peut être utilisée pour la réalisation de diverses applications multi-agents[4]. Citons quelques exemples non limitatifs :

– système de vente aux enchères avec des agents acheteurs à stratégies différentes et un agent commissaire-priseur,

– parcours d'un territoire par des agents explorateurs,

– jeux multijoueurs avec un agent arbitre-ordonnanceur et des agents joueurs,

– etc.

3. cf. `http://www.lifl.fr/SMAC` rubrique MAGIQUE.
4. voir pour différents petits exemples `http://www.lifl.fr/MAGIQUE` rubrique TOY PROBLEMS.

3.2. *Les agents, les compétences, la hiérarchie*

Dans MAGIQUE, un agent est une entité possédant un certain nombre de compétences. Ces compétences permettent à un agent de tenir un *rôle* dans une application multi-agents. Les compétences d'un agent peuvent évoluer dynamiquement (par échanges entre agents) au cours de l'existence de celui-ci, ce qui implique que les rôles qu'il peut jouer (et donc son statut) peuvent évoluer au sein du SMA. Un agent est construit dynamiquement à partir d'un agent élémentaire «vide», par enrichissement/acquisition de ses compétences.

Du point de vue de l'implémentation, une compétence peut être perçue comme un composant logiciel regroupant un ensemble cohérent de fonctionnalités. Les compétences peuvent donc être développées indépendamment de tout agent et donc réutilisées dans différents contextes. Une fois ces compétences créées, la construction d'un agent MAGIQUE se fait très simplement par un simple mécanisme d'enrichissement de ces compétences par l'agent à construire (il s'agit de l'«éducation de l'agent»). L'ajout de nouvelles compétences (et donc fonctionnalités) dans une application SMA est donc facile.

L'ensemble des agents sont regroupés au sein d'un SMA organisé selon une structure hiérarchique. Dans une hiérarchie, les agents feuille sont appelés «*spécialistes*» et les autres «*superviseurs*», ces derniers doivent être capables (*i.e.* «avoir la compétence pour») de gérer leur *équipe* d'agents (la sous-hiérarchie) dont ils sont la racine. Une hiérarchie représente en fait le support par défaut du réseau des communications entre les agents. Un lien hiérarchique représente donc l'existence d'un canal de communication bidirectionnel entre ces agents, et lorsque deux agents d'une même structure hiérarchique communiquent, le chemin emprunté par les messages qu'ils s'échangent colle, par défaut, à la hiérarchie.

L'organisation des agents peut être amenée à évoluer dynamiquement si une réorganisation de la structure du SMA se justifie en cours d'utilisation. Dans MAGIQUE, cette dynamique opère à plusieurs niveaux :

– *individuel* : un agent peut acquérir ou oublier des compétences. Les avantages de cet aspect seront discutés plus loin. Dans MAGIQUE, cela se traduit par un échange effectif de compétences entre agents, éventuellement distants, à l'initiative des agents eux-mêmes ;

– *relationnel* : Des liens d'accointances peuvent être créés lorsque des relations favorisées apparaissent entre deux agents de la hiérarchie. Cela a pour effet de supprimer les communications récurrentes le long de l'arborescence, puisque dès lors, la communication entre les agents est directe. La décision de la création de tels liens est une prérogative des agents eux-mêmes.

Couplés avec le mécanisme de délégation, cette création dynamique de relations de communication contribue à rendre *non déterministe* le fonctionnement d'une application SMA : deux exécutions successives ne produiront pas nécessairement la même structure de communication et les compétences ne seront pas réalisées nécessairement par les mêmes agents.

– *organisationnel/architectural* : des agents peuvent être créés ou détruits afin d'adapter le SMA à certaines contraintes. Deux exemples parmi d'autres :

- un agent peut se voir submerger par un afflux de requêtes pour l'une de ses compétences ; il peut alors décider de créer une équipe d'agents qui auront cette compétence et vers laquelle il pourra rediriger tout ou partie des requêtes qui lui parviennent,

- un agent doit pouvoir quitter temporairement le SMA et retrouver sa place ultérieurement, les messages qui lui sont destinés doivent alors avoir été mis en attente. Cela peut être essentiel pour des agents situés sur des ordinateurs portables (ou des téléphones ou tout appareil nomade).

3.3. *L'invocation de compétences*

Cette organisation hiérarchique offre un mécanisme simple de délégation entre les agents présents dans le SMA. Un agent a une tâche à accomplir qui requiert l'exploitation d'un certain nombre de compétences qu'il peut ou non posséder en propre. Dans les deux cas, le mode d'«invocation» de ces compétences par l'agent va se réaliser de manière identique dans MAGIQUE. La délégation éventuelle de la réalisation à un autre agent est transparente et prise en compte par l'organisation hiérarchique. Cela a pour conséquence qu'un agent cherchant à déléguer la réalisation d'une compétence n'a pas nécessairement à connaître explicitement l'agent qui réalisera effectivement cette compétence, l'organisation se charge de le trouver pour lui. C'est une grande différence entre l'approche «objets distribués» et l'approche multi-agent, les invocations ne sont pas nommées et on n'a pas nécessairement à savoir «qui fait quoi», c'est le SMA qui s'en charge. Le principe en est le suivant :

– l'agent «possède» la compétence, il l'«invoque» directement,

– l'agent ne «possède» pas la compétence, plusieurs cas de figure possibles :

- il a une accointance particulière pour cette compétence, il lui demande alors de la réaliser pour lui,

- sinon, il est superviseur et un membre de sa hiérarchie possède cette compétence, il transmet (récursivement *via* la hiérarchie) la délégation de réalisation de cette compétence à qui de droit,

- sinon, il demande à son superviseur de trouver quelqu'un de compétent pour lui et celui-ci réapplique ce même mécanisme récursivement.

Du point de vue de la programmation, l'invocation de compétence est très facile à réaliser. Là où en programmation objet, vous écririez :

```
object.ability(arg...);
```

pour faire un appel à une méthode `ability`, vous devez maintenant écrire :

```
perform("ability",arg...);
```

Cela a pour effet que la compétence nommée «`ability`» sera «invoquée» (sans avoir à savoir par qui[5]).

La primitive `perform` est dédiée à l'invocation de compétences pour lesquelles on n'attend pas de réponse. Il existe principalement trois autres primitives : `ask` quand une réponse asynchrone est désirée, `askNow` pour une réponse immédiate et `concurrentAsk` pour une invocation concurrente.

L'avantage de cette délégation du point de vue du programmeur, en comparaison des appels nominatifs, est qu'il n'a pas besoin de connaître explicitement les agents qui seront présents dans l'application. Les références se situent en effet au niveau des compétences et il suffit pour le concepteur de savoir que la compétence est présente dans le SMA, sans nécessairement connaître l'agent compétent. Le code produit gagne alors en réutilisabilité. Et l'application SMA gagne, quant à elle, en robustesse, car l'agent réalisateur d'une compétence n'étant pas nécessairement prédéfini, il peut éventuellement varier au fil du temps, si un agent devient indisponible ou surchargé par exemple.

3.4. *L'acquisition dynamique de compétence*

Nous l'avons dit, la construction et évolution d'agent dans MAGIQUE se base sur l'acquisition (et l'oubli) dynamique de compétences. Les avantages sont multiples :

le développement est facilité. Construire un agent, c'est lui enseigner des compétences. La programmation d'un agent est donc «réduite» à celle de compétences, et une fois qu'une compétence a été développée, elle peut être réutilisée dans différents contextes. Les compétences peuvent être vues comme des *composants logiciels*, avec tous les avantages liés à cette notion : modularité, réutilisabilité, etc. ;

efficacité. Un agent peut décider de déléguer l'accomplissement de telle ou telle tâche à un autre agent (si celui-ci l'accepte). Mais cela a un coût (dû à la communication par exemple) et dépend du bon vouloir de l'autre. C'est pourquoi dans certains cas, s'il a à accomplir souvent cette tâche par exemple, un agent peut «préférer» apprendre une compétence et supprimer ainsi la nécessité de la déléguer. D'un autre point de vue, si l'agent «se sent» submergé par des requêtes des autres pour exploiter une de ses compétences, il peut choisir de l'enseigner à d'autres agents (qu'il aurait d'ailleurs pu créer et éduquer dans ce but spécifique) afin d'alléger sa charge ;

robustesse. Si pour quelque raison un agent doit disparaître du SMA et qu'il est le détenteur d'une compétence critique, il peut l'enseigner à un autre agent du SMA et ainsi garantir la pérennité et la cohérence de l'ensemble ;

5. Bien évidemment, MAGIQUE offre également la possibilité de préciser un destinataire si besoin est.

autonomie et évolutivité. Au cours de sa «vie», une compétence donnée d'un agent peut évoluer et être améliorée et de nouvelles compétences peuvent être ajoutées. Ainsi, l'agent accroît ses capacités et son autonomie ;

évolution dynamique. Quand une compétence d'un agent doit être changée (*a priori* pour l'améliorer), il n'est plus nécessaire d'accomplir le cycle classique (et pénible) : «l'arrêter, modifier le source, compiler et redémarrer». La nouvelle compétence peut être dynamiquement enseignée à l'agent (qui doit oublier l'ancienne). Cela peut être particulièrement important pour un agent dont la durée de vie est longue et qui est dédié à un rôle qui ne peut tolérer la moindre interruption ;

optimisation de mémoire. Si vous considérez un agent disposant d'une faible capacité mémoire (situé sur un téléphone portable ou un organizer), vous pouvez choisir de charger votre agent avec uniquement les compétences requises à un moment donné (et le décharger des autres).

Avec cette possibilité d'évolution dynamique d'un agent, il n'est plus possible d'utiliser le terme de «classe» d'agents. Même si pour différents agents vous partez d'une base commune, dans la mesure où ils peuvent (et vont probablement) recevoir une éducation différente due à leurs «expériences» individuelles, ils vont bientôt diverger et il sera de ce fait impossible de les considérer comme appartenant à une même «classe». Cette notion n'a définitivement plus de sens dans ce contexte. Cela constitue une forte différence entre une telle programmation orientée agents et la programmation orientée objets.

D'un point de vue programmation, les compétences qui permettent de gérer cette acquisition dynamique de nouvelles compétences sont connues par les agents MAGIQUE.

Ainsi, pour qu'un agent `agent` acquiert dynamiquement une nouvelle compétence de nom `"skill"`, il suffit que la ligne suivante soit interprétée :

```
agent.perform("addASkill",new Object[]{"skill",...args...});
```

Pour que cette même compétence lui soit cette fois enseignée par un agent de nom `"teacher@..."`, il faut cette fois interpréter :

```
agent.perform("learnSkill",
              new Object[]{"skill","teacher@...",...args...});
```

Cet enseignement sera effectif, y compris si `"teacher@..."` se trouve sur une autre plate-forme/machine que `agent` et sans qu'aucune hypothèse ne soit nécessaire sur la présence du bytecode de `"skill"` sur la machine hôte de `agent`. Il suffit que l'agent `"teacher@..."` «possède» ce bytecode, celui-ci sera transmis d'une plate-forme à l'autre si nécessaire.

3.5. *Un petit exemple complet*

Dans la prochaine section, nous montrerons que MAGIQUE permet un développement facile de l'application de téléconférence présentée précédemment. Il n'est cependant évidemment pas possible de donner le code source qui serait nécessaire, et ce serait d'ailleurs pénible pour le lecteur. Néanmoins, ce même lecteur souhaite certainement mieux comprendre comment cette mise en œuvre se réalise en MAGIQUE. C'est pourquoi nous allons à travers un petit exemple présenter plus en détail la programmation avec MAGIQUE. Evidemment, l'exemple que nous allons étudier sera volontairement très rudimentaire (on pourrait même peut être contester son caractère vraiment agent). L'intérêt de sa simplicité est de permettre de montrer l'ensemble du code et de présenter ainsi les principaux concepts de programmation de SMA avec MAGIQUE. Nous allons montrer ici que, pour toute personne connaissant le langage JAVA, le surcoût de l'utilisation de MAGIQUE est quasiment nul tout en permettant une distribution facile de l'application sur un réseau[6].

3.5.1. *Le problème*

Il s'agit d'un calcul de vérification de la conjecture dite de Collatz ou de Syracuse ou encore le problème $3x + 1$ [LAG 85]. Le problème étudié est le suivant :

Considérons la fonction f qui a tout entier n associe :

$$f(n) = \begin{cases} \frac{n}{2} & \textit{si } n \textit{ est pair} \\ 3n+1 & \textit{si } n \textit{ est impair} \end{cases}$$

La conjecture affirme qu'à partir de tout n, on peut toujours atteindre 1 après suffisamment d'itérations de la fonction f :

$$\forall n \in \mathbb{N}, \exists k \in \mathbb{N}, \underbrace{f(...(f}_{k \textit{ fois}}(n)..) = 1$$

Il s'agit donc de disposer d'un SMA qui pour un entier n donné produit la suite des valeurs des itérations de la fonction jusqu'à 1.

3.5.2. *Les compétences*

Le compétences nécessaires à la résolution de ce problème peuvent être facilement identifiées :

`ParitySkill :` le test de la parité,

`MultSkill :` multiplication de deux entiers,

`AddSkill :` addition de deux entiers,

`DividerSkill :` division de deux entiers,

6. Le code source complet de cet exemple peut être récupéré (et donc testé) sur le site : `http://www.lifl.fr/MAGIQUE`.

`CollatzSkill :` calcul de la conjecture.

Il faut donc écrire le code de chacune de ces compétences. Pour cela, il suffit de créer une classe qui hérite directement ou indirectement de l'interface `Skill` du paquetage `fr.lifl.magique.skill`. Chacune des «méthodes» publiques de cette classe pourra ête utilisée par tout agent qui possèdera cette compétence.

Pour les quatre premières compétences, rien de particulier à ajouter par rapport à un code JAVA classique. La figure 6 montre le code complet de la compétence `Parityskill`, le code des trois autres est immédiat.

```
public class ParitySkill implements fr.lifl.magique.Skill {

    public Boolean isEven(Integer x) {
        return new Boolean( (x.intValue()%2) == 0 );
    }

} // ParitySkill
```

FIG. 6. *Le code source complet de la compétence* `ParitySkill`

La compétence `CollatzSkill` nécessite un peu plus d'explications. En effet, pour accomplir sa tâche, elle doit faire appel aux autres compétences. C'est ici qu'interviennent les primitives MAGIQUE d'invocation de compétences mentionnées précédemment. Dans ce cas particulier, puisqu'il s'agit de calcul nécessitant que la réponse à une requête soient connues avant de poursuivre, c'est la méthode `askNow` qui doit être utilisée. Ainsi, pour invoquer la méthode `isEven` de la compétence `ParitySkill` sur l'`Integer x`, il faut écrire :

```
Boolean value = (Boolean) askNow("isEven",x);
```

On obtient pour la compétence `CollatzSkill` le code complet donné à la figure 7.

On peut noter aussi l'invocation de la compétence `display`, grâce à la primitive `perform`, car ici il n'y a pas de réponse attendue. Il s'agit de déléguer à la compétence qui convient (par défaut la compétence `DisplaySkill` présente dans tout agent MAGIQUE gère cette invocation) la réalisation de l'affichage demandé.

Voici donc pour les compétences impliquées. On voit que MAGIQUE n'apporte que peu de particularités en comparaison de classe et méthodes JAVA. La seule vraie différence est que pour certains appels, là où en objet on écrit :

```
objet.method(args...);
```

avec MAGIQUE, on a :

```
perform("method",args...);  // ou ask/askNow
```

```
import fr.lifl.magique.*;
import fr.lifl.magique.skill.*;
public class CollatzSkill extends MagiqueDefaultSkill {
   private Integer x;
   public CollatzSkill(Agent myAgent, Integer x) {
     super(myAgent);
     this.x = x;
   }
   private Boolean testParity(Integer x) {
     Boolean result = (Boolean) askNow("isEven",x);
     return result;
   }
   private Integer xByTwo(Integer x) {
     Integer r=(Integer) askNow("quotient",x,new Integer(2));
     return r;
   }
   private Integer threeTimesPlus1(Integer x) {
     Integer y = (Integer) askNow("mult",x,new Integer(3));
     Integer z =(Integer) askNow("add",y,new Integer(1));
     return z;
   }
   public void conjecture(Integer x) {
     this.x = x;
     conjecture;
   }
   public void conjecture() {
     while (x.intValue() != 1) {
       perform("display",new Object[]{"x  = "+x.intValue()});
       if (testParity(x).booleanValue()) {
           x = xByTwo(x);
       }
       else {
           x = threeTimesPlus1(x);
       }
     }
     perform("display",new Object[]{"x = 1 ** finished"});
   }
} // CollatzActionSkill
```

FIG. 7. *Le code source complet de la compétence* `CollatzSkill`. *Les méthodes* `conjecture` *permettent de tester la conjecture*

Une différence notable est que le destinataire n'a pas à être nommé (à la différence de la programation objet). Un des intérêts évident est que la compétence pourra facilement être réutilisée dans différents contextes (cela est facilement imaginable pour les quatre premières compétences) et avec différents agents. On peut d'ailleurs remarquer

que nous avons étudié les compétences sans même encore avoir discuté des agents qui seront impliqués dans le SMA. C'est ce que nous allons faire maintenant.

3.5.3. *La hiérarchie et les agents*

Il s'agit de répartir les compétences parmi les agents du SMA puis d'organiser ces agents dans une hiérarchie. Nous allons imposer arbitrairement la structure du SMA et la répartition des compétences. Nous travaillerons avec sept agents : un pour chacune des compétences précédemment citées, un agent qui supervisera les quatre agents possédant les compétences arithmétiques de base et un superviseur global du SMA. D'autres choix auraient pu être faits.

On dispose donc d'un SMA hiérarchique correspondant à la structure présentée à la figure 8. Rappelons que l'existence de cette structure hiérarchique permet la gestion automatique de la gestion des invocations des compétences. Ainsi, lorsque la compétence `CollatzSkill` requiert la compétence `ParitySkill`, l'agent `collatz` n'a pas besoin de connaître explicitement l'agent `parity`, MAGIQUE gère l'acheminement de la requête. En fait, la seule chose qui importe est de savoir que la compétence `ParitySkill` est présente dans le SMA.

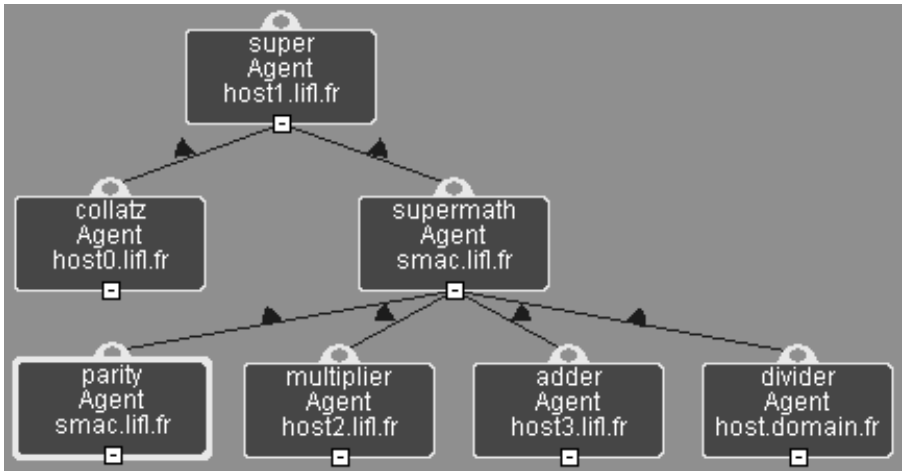

FIG. 8. *La structure hiérarchique choisie*

Venons-en à la création de nos agents. MAGIQUE se base sur la notion de plate-forme, qui est en fait l'équivalent de la JVM pour MAGIQUE. Cette plate-forme prend en charge les problèmes liés à la distribution qui est ainsi masquée pour le développeur. *A priori*, on ne trouve qu'une plate-forme par machine, même si cela n'est pas une obligation. Un agent est donc créé par une plate-forme et il faut ensuite faire son «éducation» en lui enseignant les compétences que l'on souhaite lui donner.

La figure 9 présente la création de l'agent possédant la seule compétence `ParitySkill`. Cet agent doit se rattacher dans la hiérarchie à son superviseur. C'est

ce qui est fait par l'appel à la méthode `connectToBoss`. Ici, on suppose que ce superviseur se trouve dans la même plate-forme (ce superviseur n'a besoin d'aucune compétence particulière, il n'est là que pour chapeauter les agents «mathématiques») et qu'il doit lui-même être connecté au superviseur global appelé `"super"` et placé dans une autre plate-forme sur la machine `host1.lifl.fr`. Cette simple commande de connexion va suffir pour gérer les communications distantes entre les agents.

```
import fr.lifl.magique.*;
public class ParityImp {
  public static void main(String[] args) {
              // création et lancement de la plate-forme
    Platform p = new Platform();
              // création du superviseur "mathématique"
    Agent supermath = p.createAgent("supermath");
              // connexion à son superviseur
    supermath.connectToBoss("super@host1.lifl.fr:444");
              // création de l'agent pour la parité
    Agent parity = p.createAgent("parity");
             // enseignement de la compétence de parité
    parity.addSkill("ParitySkill");
             // connexion à son superviseur
    parity.connectToBoss("supermath");
  }
}
```

FIG. 9. *Le code source complet de création du superviseur mathématique et de l'agent possédant la compétence* `ParitySkill`

Pour les autres agents, il en va de même. Il faut juste savoir quels agents évoluent sur la même machine et donc dans la même plate-forme. Cependant, pour que le calcul se fasse, il faut invoquer la compétence `conjecture` ; cela peut être fait par exemple après la création de l'agent `collatz` (cf. figure 10). Mais avec le mécanisme de délégation, cette invocation aurait pu être faite par n'importe quel autre agent.

Si l'on dispose en plus des programmes, `ParityImp`, `CollatzImp`, des autres `AdderImp`, `MultiplierImp`, `DividerImp`, `SuperImp` écrits sur le même principe (en supposant donc que tous ces agents se trouvent sur des plates-formes (donc *a priori* machines) différentes), on peut maintenant, en exécutant ces programmes sur leurs machines respectives, «exécuter» ce SMA.

3.5.4. *Environnement de développement*

Un environnement graphique permettant la construction de la structure hiérarchique, la création des agents et l'enseignement de leurs compétences existe (cf. figure 11). Vous pourrez le récupérer pour le tester sur le site de MAGIQUE. Cet environnement permet également la distribution automatique des agents sur le réseau et fournit une console d'administration distante de ces agents.

```
import fr.lifl.magique.*;
public class CollatzImp {
  public static void main(String[] args) {
    Platform p = new Platform();
    Agent colAgent = p.createAgent("collatz");
    colAgent.connectToBoss("super@host1.lifl.fr:444");
    colAgent.addSkill("CollatzSkill",colAgent,17);
             // calcul de la conjecture avec x=17
    colAgent.perform("conjecture");
  }
}
```

FIG. 10. *Le code source complet de création du superviseur mathématique et de l'agent possédant la compétence* `ParitySkill`

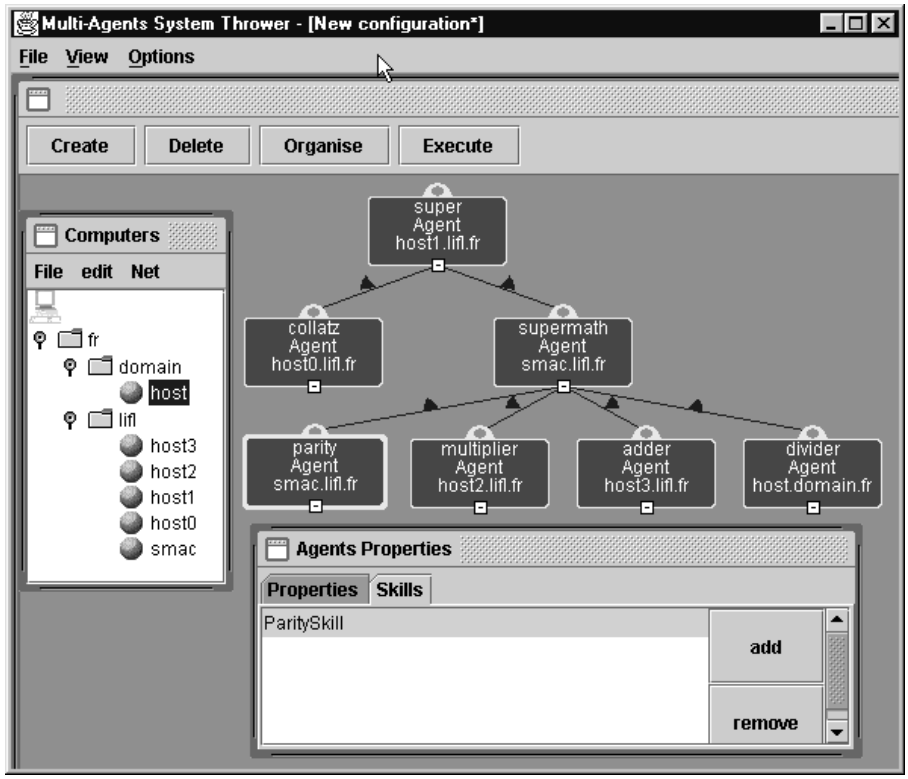

FIG. 11. *Le SMA de collatz créé avec l'environnement de conception graphique*

3.5.5. *Conclusion*

Avec cet exemple complet, nous espérons avoir convaincu le lecteur que pour peu que l'on connaisse le langage JAVA, programmer avec MAGIQUE ne présente pas de

difficultés. En outre, les problèmes de communication et de distribution sont entièrement masqués par MAGIQUE. Cet exemple devrait permettre de mieux comprendre la description de la réalisation d'application de téléconférence présentée maintenant.

4. Diapo-conférence : la réalisation avec MAGIQUE

Nous allons maintenant étudier comment l'application de télé-conférence présentée précédemment peut être développée facilement avec MAGIQUE. En fait, comme nous l'avons vu précédemment, le surcoût de développement lié à MAGIQUE et aux aspects multi-agents est presque inexistant et n'est quasiment dû qu'à l'agent coordinateur que nous présenterons par la suite. Les fonctionnalités applicatives comme la visionneuse, la gestion des équipes ou autres doivent de toute façon être mises en œuvre.

Examinons plus en détail cette réalisation. Il convient de déterminer les agents intervenants et les compétences de chacun, puis de définir l'organisation du système et les interactions possibles.

4.1. *Analyse*

Il s'agit dans un premier temps d'identifier clairement les rôles nécessaires à l'application et les compétences qui y sont attachées.

4.1.1. *Les rôles*

Nous avons déjà clairement identifié deux de ces rôles : ceux de *conférencier* et d'*auditeur*. Il existe cependant un troisième rôle qui n'a pas encore été explicitement nommé. Il est en effet nécessaire, pour que la conférence puisse avoir lieu, que les intervenants puissent la quitter ou la rejoindre et qu'une entité fasse le lien entre les différents participants. Nous appelerons *coordinateur* ce rôle mais il peut être vu comme l'*environnement*, habituel en SMA, puisqu'il constitue le support de l'interaction et concrétise par son existence la cohabitation de l'ensemble des participants à la conférence.

4.1.2. *Les compétences*

Pour le rôle *coordinateur*, les compétences attachées sont assez limitées, il suffit de fournir aux agents un point d'entrée à la conférence et de maintenir la cohérence en cas de déconnexion (notamment si c'est le conférencier qui quitte la session, il faut passer le relais – la télécommande – à un des auditeurs).

Pour les rôles de *conférencier* et d'*auditeur*, les compétences sont assez similaires et correspondent aux fonctionnalités décrites dans la première section. Les voici de manière informelle :

– *visionneuse* : pouvoir récupérer, interpréter et afficher les ressources, gérer le pointeur et les manipulations

– *la fenêtre des intervenants* : gérer les demandes de télécommande ainsi que ses transferts

– *la messagerie* : permettre les échanges entre les participants en respectant le statut *conférencier* et *auditeur*, cette compétence doit donc être contextualisée au rôle.

– *assistant* : fournir les fonctionnalités d'aide à la coopération et à la modélisation des intervenants mentionnées précédemment.

Et il reste pour ces deux rôles à traiter la compétence de gestion de la *télécommande*. Cette compétence doit permettre :

– au rôle *conférencier* de diriger la conférence (passage de diapositives, gestion du pointeur et de la zone de visualisation),

– aux deux rôles de gérer leurs ressources et d'organiser des séquences de diapositives (cette fonctionnalité étant un outil fournit en plus.

Le cas de cette compétence est particulièrement intéressant, car c'est justement elle qui crée la différence entre les deux rôles de *conférencier* et d'*auditeur*. L'évolution dynamique des rôles en cours de session (de *conférencier* à *auditeur* et réciproquement) implique une évolution dynamique de la compétence connue par les agents. Il est en effet nécessaire d'accorder au nouveau conférencier les fonctions pour mener la conférence et de les retirer à l'ancien conférencier.

4.2. *Les choix d'organisation du SMA*

Comme cela a déjà été mentionné, nous avons utilisé la plate-forme multi-agents MAGIQUE pour développer un prototype pour cette application. Chaque intervenant est défini par un agent qui le représente, un de ces agents aura le rôle de *conférencier* et les autres seront donc *auditeurs* (cf. figure 12). Nous avons décidé de créer en plus un agent particulier, pour qu'il joue uniquement le rôle de *coordinateur* (même si ce rôle aurait pu être joué par un des autres agents, l'initiateur de la conférence par exemple, ou pourrait même évoluer dynamiquement).

Nous avons expliqué que les SMA dans MAGIQUE étaient organisés hiérarchiquement. Dans ce cas, nous avons choisi de placer l'agent *coordinateur* à la racine du SMA et de placer tous les autres agents au niveau inférieur. Remarquons qu'une autre possibilité était de placer l'agent jouant le rôle de *conférencier* à la racine, ce qui impliquait une réorganisation de la hiérarchie lors du changement des rôles. Ce qui est réalisable avec MAGIQUE.

L'évolution dynamique des rôles *conférencier*$\longleftrightarrow$*auditeur* est facilement prise en compte par MAGIQUE. Cette plate-forme a en effet l'avantage de permetttre l'évolution dynamique des compétences des agents par échanges entre eux. Dans MAGIQUE, les agents apprennent et oublient *effectivement* des compétences. Les échanges de

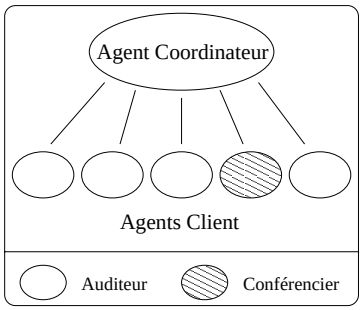

FIG. 12. *L'organisation hiérarchique du SMA*

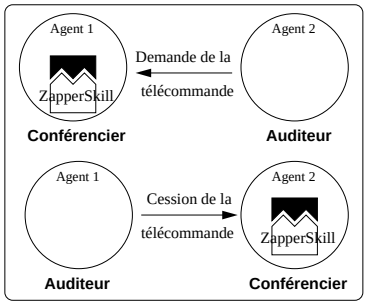

FIG. 13. *Evolution dynamique de rôles par échange de compétences*

compétences sont réels, indépendamment de toute hypothèse sur le code de ces compétences et de la distribution sur le réseau des agents. L'agent *conférencier* pourra donc, comme cela se passe lors du passage de micro dans une conférence réelle, donner la compétence de gestion de la télécommande au nouveau *conférencier* et de ce fait la *perdre* (cf. figure 13). Et il ne s'agit pas ici de la simple modification de la valeur d'un attribut quelconque, mais d'une mutation concrète de l'agent et du rôle qu'il tient et donc en quelque sorte de ses droits dans l'application.

4.3. *La programmation*

Pour réaliser cette application avec MAGIQUE, il «suffit» de développer les différentes compétences mentionnées précédemment.

Chacune représente un composant. Une fois l'interface de la compétence définie (cf. figure 14), il convient de l'implémenter sous la forme d'objets JAVA, la seule différence à prendre en compte représente la nécessité d'effectuer des interactions entre les agents et les différentes autres compétences et nous avons déjà vu précédemment que cela se faisait facilement grâce au mécanisme de délégation à l'aide des primitives `perform`, `ask`, etc.

Ensuite, la construction des agents se fait très simplement, il suffit d'«enseigner» à l'agent de base les compétences requises une par une. Ce qui donne quelque chose comme ce qui est présenté à la figure 15.

La couche MAGIQUE gère ensuite les connexions des agents et le transport des messages dûs aux interactions entre agents. Ces communications n'apparaissent pas explicitement au niveau du code, puisqu'elles sont induites par les invocations de compétences et donc prises en charge par MAGIQUE.

```
public interface HTMLSkillInterface
{
  public JPanel getHTMLSkillPanel();
  public URL getHTMLSkillURL();
  public void setActiveHTMLSkill(Boolean b);

  /** Charger une diapo. */
  public void loadPage(URL url);

  public Boolean nextHTMLSkillPage();
  /** Options */
  public Boolean isHTMLSkillCursorVisible();
  public void setHTMLSkillCursorVisible(Boolean b);
  public Boolean isHTMLSkillAutoResize();
  public void setHTMLSkillAutoResize(Boolean b);
}
```

FIG. 14. *L'interface de la compétence* `HTMLSkill` *(sans les commentaires). Il suffit donc de l'implémenter pour l'adapter à l'agent indépendamment des autres compétences. Il est notamment possible d'adapter la compétence de chargement de page en fonction du support et ce pour chaque agent indépendamment des autres*

```
import fr.lifl.magique.*;
    ...
    // création d'un agent "vide"
    Agent myAgent = new Agent("myName");
    // l'agent apprend ses compétences
    myAgent.addSkill("MaZapperSkill");
    myAgent.addSkill("MaHTMLSkill");
    myAgent.addSkill("MonAssistantSkill");
    ... autres enrichissement de skill ...
    ...
```

FIG. 15. *Création d'un agent*

4.4. *Evolutivité et adaptativité*

Le concept de compétence facilite l'évolution de l'application et son adaptation à différents contextes.

Il est ainsi possible d'étendre facilement les fonctionnalités offertes à travers la création de nouvelles compétences qu'il suffit ensuite d'«enseigner» aux agents. On peut ainsi sans aucun problème envisager une construction incrémentale de l'application.

Par exemple, une compétence permettant le support de la voix pourrait ainsi être ajoutée pour faciliter l'échange d'informations, sans que cela ait d'impact sur le reste

de l'application. Il suffirait en effet de prévoir les interactions entre les agents représentant l'application chez les différents utilisateurs et de leur «enseigner» la compétence créée pour qu'il soit possible de l'utiliser.

L'adaptation à différents supports ou différents types de ressources (XML par exemple) peut également être facilement obtenue. Il suffit de changer l'interprète de document pour le visualisateur. De même, si l'un des auditeurs souhaite utiliser l'application sur un client léger (type Palm Pilot ou Psion), il peut adapter la compétence de visualisation en «enseignant» à son agent la compétence visionneuse adaptée à ce client à la place de celle prévue par défaut.

Enfin, pour donner un dernier exemple illustrant l'ouverture d'une telle application développée avec MAGIQUE, il est facile d'envisager d'adapter l'assistant pour chacun des utilisateurs. Il est par exemple évidemment envisageable de définir différentes stratégies pour déterminer la corrélation des ressources du propriétaire de l'agent avec les documents diffusés ; ou encore différents critères pour réaliser la modélisation des autres intervenants.

5. Conclusion

Nous avons présenté une application de conférence distribuée. Cette application fournit à la fois le support à la coopération et des outils pour la faciliter. Il s'agit d'outils essentiels comme la visualisation et la télécommande. Cette dernière est l'outil pivot, puisque sa possession détermine le rôle de chaque participant et son transfert provoque l'évolution dynamiquement de ces rôles. D'autres outils plus annexes, mais dans la pratique tout aussi indispensables, sont également fournis : la gestion des équipes, la possibilité d'organiser ses ressources, ou la messagerie. Enfin, l'asssistant permet de mieux exploiter ses propres ressources lorsqu'elles sont en relation avec l'exposé courant, mais il permet également une modélisation des coparticipants aux conférences, facilitant ainsi la tenue de future coopération.

Un prototype de cette application a été développé sous forme d'un système multi-agents à l'aide de la plate-forme MAGIQUE. L'utilisation d'une telle plate-forme permet de se décharger des problèmes liés à la distribution et aux communications à travers les interactions «normales» des agents. De plus la construction incrémentale des agents dans MAGIQUE, par acquisition dynamique de compétences, facilite le développement et l'évolutivité de l'application.

Ce prototype et une documentation peuvent être récupérés à l'adresse :

`http://www.lifl.fr/SMAC` rubrique MAGIQUE

6. Bibliographie

[BAE 93] BAECKER R., Ed., *Readings in groupware and computer supported cooperative work*, Morgan Kaufman, 1993.

[BAU 98] BAUDOUIN-LAFON M., Ed., *Computer-Supported Cooperative Work*, John Wiley Sons Ltd, 1998.

[BEN 97] BENSAID N., MATHIEU P., « A Hybrid and Hierarchical Multi-Agent Architecture Model », *Proceedings of PAAM'97*, 1997, p. 145–155.

[BRI 97] BRINCK T., MCDANIEL S., « CHI 97 Workshop on Awareness in Collaborative Systems », *Proc. of CHI'97 : Human Factors in Computing Systems*, (position papers available via `http ://www.usabilityfirst.com/groupware/awareness/`), 1997.

[DER 95] DERYCKE A., HOOGSTOEL F., « Le travail coopératif assisté par ordinateur : quels enjeux pour les concepteurs », *Actes du XIIIe Congrès INFORSID (INFormatique des ORganisations et Systèmes d'Information et de Décision)*, Grenoble, 1995.

[ELL 91] ELLIS C., GIBBS S., REIN G., « Groupware : some issues and experiences », *Communications of the ACM*, vol. 34, n° 1, 1991, p. 38–58.

[FER 95] FERBER J., *Les systèmes multi-agents. Vers une intelligence collective*, InterEditions, 1995.

[FER 98] FERBER J., GUTKNECHT O., « A meta-model for the analysis and design of organizations in multi-agent systems », *Proceedings of ICMAS'98*, IEEE Press, 1998.

[FRA 96] FRANKLIN S., GRASSER A., « Is it an Agent, or just a Program ? : A Taxonomy for Autonomous Agent », *Proceedings of the 3rd International Workshop on Agent Theories, Architectures, and Languages*, Springer Verlag, 1996.

[GUT 00] GUTKNECHT O., J.FERBER, MICHEL F., « MadKit : une plate-forme multi-agent générique », rapport, 2000, FIPA.

[LAG 85] LAGARIAS J., « The $3x + 1$ problem and its generalizations », *American Math Monthly*, , 1985, p. 3-23.

[MAR 98] MARCENAC P., GIROUX S., « GEAMAS », *Journal of Applied Intelligence*, , 1998.

[REP 00] REPENNING A., « AgentSheets® : an Interactive Simulation Environment with End-User Programmable Agents », *Interaction 2000. Tokyo. Japan.*, 2000.

[ROS 99] ROSCHELLE J., DIGIANO C., KOUTLIS M., REPENNING A., PHILLIPS J., JACKIW N., SUTHERS D., « Developing Educational Software Components », *IEEE Computer*, vol. 32, 1999, p. 50-58.

[ROU 00] ROUTIER J., MATHIEU P., SECQ Y., « Dynamic Skills Learning », rapport n° 2000-06, 2000, LIFL.

[SHO 93] SHOHAM Y., « Agent-Oriented Programming », *Artificial Intelligence*, vol. 60, 1993, p. 51–92.

[STE 87] STEFIK M., BOBROW D., FOSTER G., LANNING S., TATAR D., « WYSIWIS revised : Early Experiences with Multi-user Interfaces », *ACM Transactions on Office Information Systems*, vol. 5, n° 2, 1987, p. 147-167.

[TAR 99] TARPIN-BERNARD F., DAVID B., « AMF : un modèle d'architecture multi-agents multi-facettes », *Techniques et Sciences Informatiques, Hermès*, vol. 18, n° 5, 1999, p. 555-586.

[TRA 96] TRAVERS M., « Programming with Agents : New metaphors for thinking about computation », PhD thesis, MIT, 1996.