CAHIER DU LAMSADE

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision
(Université Paris-Dauphine)
Unité de Recherche Associée au CNRS ESA 7024

CONCEPTION ORIENTÉE AGENT DU TRAVAIL COOPÉRATIF

CAHIER N° 157
octobre 1998

Samir AKNINE [1]
Suzanne PINSON [1]

reçu : juin 1998.

[1] *LAMSADE, Université Paris-Dauphine, Place du Maréchal De Lattre de Tassigny, 75775 Paris Cedex 16, France ({aknine,pinson}@lamsade.dauphine.fr).*

TABLE DES MATIÈRES

Pages

Agent Oriented Design of Cooperative Work

Abstract : More and more, the organisations requirements are competition, reduction of work process execution costs and development of new products and services. For these requirements, the wokflow technology answers by providing methodologies and software for the activity processes. The aim of our research is to provide a support for the parallel workflow applications with new activity representation models and software architectures for performing the desired functionalities. In this article, we first define a task model as well as formal relations between tasks. We then present a software agent architecture which addresses the needs of these systems by encapsulating several agents in the same structure. The three key aspects of our recursive agent model are: parallelisation of agent tasks, reuse of agent components and partial mobility of agent code.
Keywords : Multi-agent systems, agent model, recursivety, cooperative work, task model.

Conception Orientée Agent du Travail Coopératif

Résumé : Les exigences des organisations d'aujourd'hui portent sur la compétitivité, la réduction des coûts d'exécution des processus d'activités et la réalisation rapide de nouveaux services et de produits. A ces exigences, la technologie Workflow essaye de répondre en offrant des méthodologies et des outils logiciels qui supportent ces processus d'activité. L'objectif de nos recherches est de proposer un support pour les applications de Workflow parallèles au travers des modèles de représentation de l'activité et des architectures d'agents logiciels pour réaliser les fonctionnalités requises par ces systèmes. Dans ce but, nous définissons un modèle de tâche ainsi que différentes relations formelles entre tâches. Nous présentons ensuite un état de l'art sur les architectures d'agent puis le modèle que nous préconisons pour répondre aux exigences de ces systèmes. Le modèle encapsule plusieurs agents primitifs dans un même agent de base. Cette architecture d'agent récursive présente plusieurs avantages : la parallélisation des tâches de l'agent, la réutilisation des composants de l'agent et la mobilité partielle du code de l'agent.
Mots-clés : Systèmes multi-agent, modèle d'agent, récursivité, travail coopératif, modèle de tâches.

1 Introduction

Les recherches sur la conception de systèmes coopératifs poursuivent deux types d'objectifs : la conception de systèmes capables d'assister " intelligemment " un utilisateur dans la résolution d'un problème et des systèmes qui médiatisent (coordonnent) de façon " intelligente " l'activité d'un collectif de travail. Les travaux sur les systèmes qui médiatisent l'activité coopérative s'inscrivent dans le domaine du CSCW (Computer Supported Cooperative Work) dans lequel plusieurs travaux exploitent des concepts de l'Intelligence Artificielle Distribuée et contribuent en retour à enrichir cette discipline (Ishiguro et al, 1996 ; Aknine, 1998a ; Aknine, 1998b). Parmi ceux-ci, les recherches sur le workflow (Ellis, 1979 ; Coop'96 ; Divitini et al, 1996) figurent en bonne place.
Cet article s'inscrit dans un projet de recherche qui vise à tirer parti d'une approche multi-agent pour construire des systèmes de workflow intelligents capables de s'adapter de façon réactive aux évolutions de l'activité d'un collectif de travail tout en facilitant la parallélisation et l'ordonnancement de certains travaux. Ces systèmes de workflow constituent, pour le collectif de travail, des " mécanismes de coordination " dont nous souhaitons qu'ils apportent une aide active aux acteurs[1] en réagissant en fonction de l'état d'avancement de leur travail pour leur proposer une affectation des travaux optimale (Divitini et al, 1996). Pour apporter une réponse à ce type de problèmes, nous proposons dans cet article un modèle de tâche et nous définissons une architecture multi-agent et un modèle d'agent qui s'appuient sur les fonctionnalités du processus de travail coopératif.
Cet article est organisé comme suit : dans la section 2, nous présentons le paradigme de coordination réactive pour les systèmes de Workflow parallèles et nous l'illustrons sur l'exemple de rédaction coopérative de spécifications techniques dans le domaine des télécommunications. Dans la section 3, nous proposons un modèle de tâches et nous définissons trois types de relations entre tâches. Dans la section 4, nous argumentons les choix orientés agents que nous avons faits pour l'implémentation des systèmes d'aide au travail coopératif. Dans la section 5, nous présentons différents travaux sur les architectures multi-agents, nous détaillons ensuite le modèle d'agent proposé ainsi que son fonctionnement. Dans la section 7, nous discutons l'implémentation de notre maquette. Enfin, nous exposons nos conclusions dans la section 8.

2 Coordination réactive des processus d'activités coopératifs

Avant de décrire le modèle d'agent que nous proposons, nous présentons brièvement les caractéristiques de l'application coopérative de type workflow que nous avons étudiée pour valider ce modèle. Nous avons retenu comme exemple la rédaction de documents techniques dans le domaine des télécommunications. Le processus coopératif sollicite quatre opérations :

- la **rédaction** qui concerne tous les traitements nécessaires pour rédiger une spécification (dans notre exemple spécification, des composants téléphoniques) ;
- la **relecture** qui s'attache à la qualité rédactionnelle du document ;
- la **revue** qui consiste à vérifier qu'une spécification prend bien en compte l'ensemble des besoins du système de télécommunications ;
- la **vérification de cohérence** qui vérifie la conformité de la spécification par rapport à d'autres documents en cours de réalisation portant sur le même système.

L'application consiste à organiser les activités des acteurs et à les contrôler lors de la rédaction coopérative de documents techniques dans le domaine des télécommunications. Plusieurs acteurs participent au processus de rédaction coopérative de spécifications techniques. Une spécification technique est un document qui se décompose en trois sous-spécifications :

[1] Dans la suite de cet article, nous désignerons par acteur l'acteur humain du système coopératif.

- la **spécification fonctionnelle** (SF) ou **spécification de méthodologie** qui présente des architectures ou des composants d'architectures dans un objectif de définition conceptuelle, méthodologique ou de description des besoins. Ce document n'est pas normatif. Il n'est pas conçu de façon détaillée pour impliquer une contrainte forte de réalisation ou pour servir à la validation.
- la **spécification détaillée** (SD) ou **spécification de mise en œuvre** de composants du réseau téléphonique : ce document est construit à partir du précédent. Il est conçu pour une utilisation lors de la phase de réalisation.
- la **spécification de référence** (SR) qui rassemble dans un même document tous les systèmes ou éléments du réseau impliqués dans la définition de la spécification détaillée.

Pour décharger les acteurs de certaines tâches de coordination, nous avons conçu un système coopératif composé d'agents logiciels. A chaque acteur est associé un agent logiciel. L'acteur du système exécute les tâches du processus (rédaction, relecture, revue et vérification de cohérence) ; quant à l'agent, il assiste et représente l'acteur dans le système. Il vérifie les conditions d'activation des tâches qu'il annonce aux autres agents du système lorsque ces conditions sont satisfaites. Connaissant le profil de l'acteur qui correspond aux tâches que celui-ci sait faire, l'agent propose alors l'acteur qu'il représente pour réaliser les tâches annoncées (Smith et Davis, 1981). Une fois la tâche affectée à l'acteur compétent, un processus coopératif de résolution du problème est ainsi engagé entre l'agent et l'acteur auquel la tâche est allouée. Par exemple, lorsque l'acteur rédige sa spécification détaillée, l'agent logiciel peut analyser un ensemble de composants du réseau téléphonique selon plusieurs points de vue puis les proposer à l'acteur correspondant qui peut sélectionner le plus rentable pour l'entreprise. Les situations de conflits apparaissent lorsque les acteurs du système sélectionnent différents composants téléphoniques pour des spécifications dépendantes. Ces conflits seront résolus de façon réactive par les agents logiciels du système.

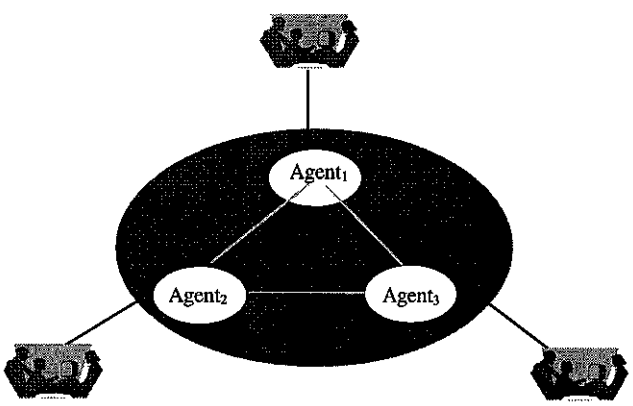
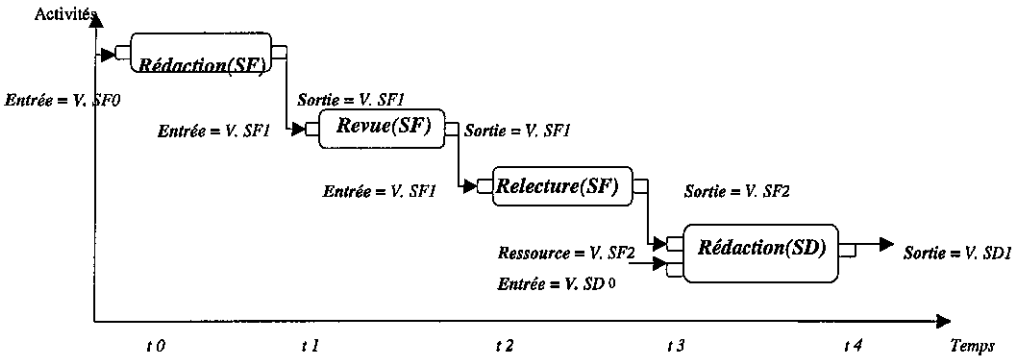


Figure 1. *Système coopératif acteurs – agents logiciels*

La figure 2 illustre le plan d'exécution du processus workflow suivant l'approche traditionnelle. L'activité de rédaction de la spécification fonctionnelle (SF) est activée à l'instant t_0. Conformément à la définition classique de la dépendance entre les activités, la revue, la relecture de la spécification fonctionnelle ainsi que l'activité de rédaction de la spécification détaillée sont des activités séquentialisables. Elles sont respectivement activées aux instants t_1, t_2 et t_3. On notera sur cette figure qu'un document non modifié par une activité apparaît en entrée

et en sortie d'une même activité avec la même version. Un document consultable lors du processus de rédaction est représenté comme une ressource de l'activité.

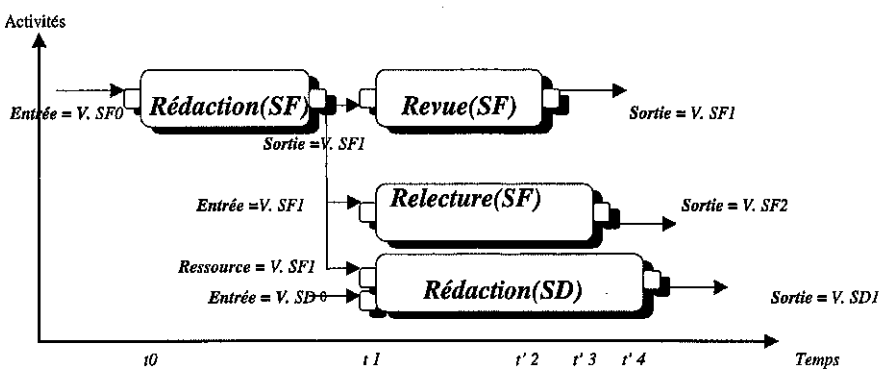


légende
SF Spécification fonctionnelle ; SD Spécification détaillée ; V. $\xi_{(j)}$ *: version 'j' du document* ξ.

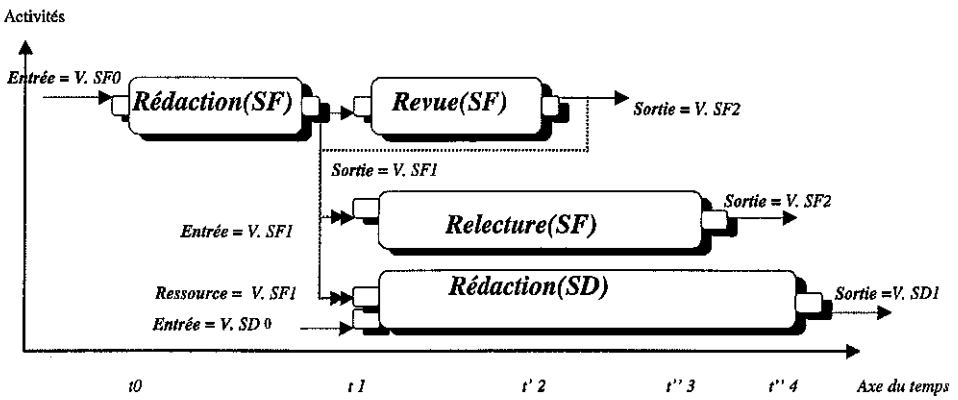
La version 1 de la spécification fonctionnelle, soit V. SF1, n'ayant pas été modifiée par l'activité de revue apparaît en entrée et en sortie de cette activité. Par contre, V. SF1 est modifiée par l'activité de relecture pour donner une deuxième version V. SF2. Cette version est en ressource de l'activité de rédaction de la spécification détaillée.

Figure 2 Déroulement du processus Workflow suivant l'approche traditionnelle.

Etant donné que les activités du processus peuvent parfois être indépendantes, elles sont donc soit totalement, soit partiellement parallélisables. Cette forme de parallélisation est intéressante dans la mesure où elle peut réduire la durée d'exécution du processus d'activités.
Une fois la rédaction de la spécification fonctionnelle terminée, les activités de revue et de relecture de la spécification fonctionnelle (SF) ainsi que l'activité de rédaction de la spécification détaillée (SD) sont immédiatement entamées. Le fait de disposer d'un modèle de représentation des indépendances partielles entre les activités et d'un mécanisme de contrôle réactif, i.e., pris en charge par des agents artificiels avertissant immédiatement l'acteur vérificateur des besoins, le relecteur de la spécification fonctionnelle et le rédacteur de la spécification détaillée chaque fois qu'une modification est apportée à la version de la spécification fonctionnelle stockée dans le système, permet aux agents de paralléliser l'exécution de certaines activités. Le processus de rédaction concurrent multi-acteurs que nous proposons est décrit par la figure 3.



(a) la parallélisation est favorable car la Revue n'a pas modifié le document.
(Même version *V. SF1* en entrée et en sortie de la tâche)



(b) la parallélisation est défavorable, le système réactive les activités de Rédaction et de Relecture.

légende
***SF** Spécification fonctionnelle ; **SD** Spécification détaillée ; **V.** $\xi_{(j)}$: version 'j' du document ξ.*

Figure 3 Illustration de la parallélisation des activités semi-indépendantes sur un extrait du processus coopératif.

La figure 3 définit un extrait du processus coopératif et illustre les cas favorable et défavorable pour l'exécution des tâches semi-indépendantes. Dans le premier cas (cf. fig. 3.a), la parallélisation est favorable en temps d'exécution car l'exécution de l'activité de revue de la spécification fonctionnelle enclenchée avec une version 1 de la spécification fonctionnelle (appelée *V. SF1*) n'a pas généré une nouvelle version de la spécification fonctionnelle. De ce fait, les relations d'indépendance partielle entre la revue, la relecture de la SF et la rédaction de la spécification détaillée se transforment ainsi en relations d'indépendance entre ces trois activités. Dans le deuxième cas (cf. fig. 3.b), une nouvelle version pour la spécification fonctionnelle (appelée *V. SF2*) générée par l'activité de revue allonge le temps d'exécution global du processus de rédaction parallélisé de *[t' 3, t'' 3]* pour l'activité de relecture de la SF et de *[t' 4, t'' 4]* pour l'activité de rédaction de la SD. En effet, les tâches de relecture de la SF et de rédaction de la SD sont reconduites avec la nouvelle version *V. SF2* de la spécification fonctionnelle. La relation d'indépendance partielle s'est transformée en une relation de dépendance entre les activités parallélisées.
Nous considérons la modification de la version d'un document sur lequel plusieurs activités sont exécutées en parallèle comme un cas d'incohérence qui crée des situations conflictuelles entre les agents logiciels du système qui contrôlent ce document. Dans la suite de cet article, nous verrons comment sont gérés les conflits entre les agents du système.

3 Définition d'un modèle de tâche et des relations entre tâches

Pour modéliser le processus de rédaction coopérative, nous avons besoin de définir un modèle de tâche ainsi que différents types de relations entre tâches.

3.1 Définition d'un modèle de tâche

Au niveau conceptuel, une tâche est une description des actions à réaliser pour atteindre un but. Il s'agit uniquement des actions de haut niveau à accomplir. La mise en œuvre de la tâche est

explicitée par les méthodes qui s'y rattachent (Ferber, 1995). En nous inspirant de (Glaser, 1997 ; Delouis et Krivine, 1992), nous considérons qu'une tâche est un script comportant une description déclarative, une description comportementale et une structure de contrôle (cf. fig. 4) que nous détaillons ci-après.

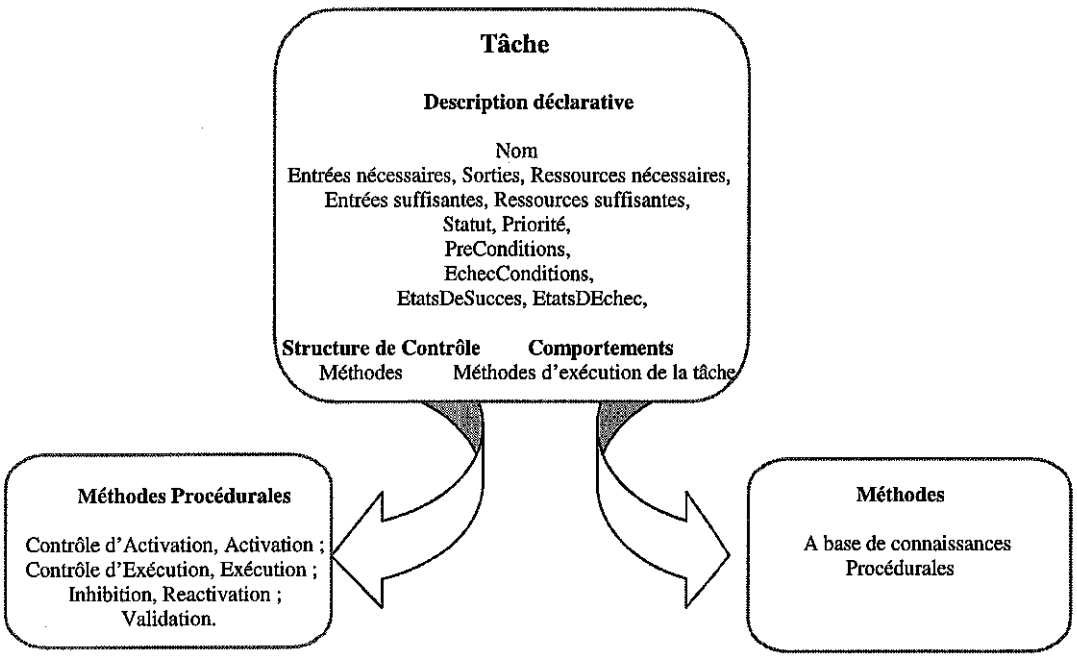


Figure 4 Description d'une tâche.

- **La description déclarative** est composée des éléments suivants :

➢ Les entrées suffisantes représentent pour notre application les documents permettant d'initier la réalisation d'une tâche. Par exemple, le document suffisant pour la réalisation de la tâche de relecture de la spécification fonctionnelle est V. SF1.

➢ Les entrées nécessaires sont les documents nécessaires pour valider les résultats d'exécution d'une tâche. Par exemple, le document en sortie de la revue est un document nécessaire pour valider les résultats de la tâche de relecture. Lorsque les entrées nécessaires sont les mêmes que les entrées suffisantes le processus d'activités parallélisables est favorable.

➢ Les ressources suffisantes représentent les documents qu'une tâche peut consulter au cours de son exécution.

➢ Les ressources nécessaires valident les résultats finaux de la tâche (voir définition4).

➢ Les préconditions de la tâche renferment les conditions suffisantes pour l'exécution de la tâche. Ces conditions portent sur la disponibilité des entrées et des ressources suffisantes pour la tâche.

➢ Les conditions d'échecs sont des contraintes sur les entrées nécessaires et les ressources nécessaires qui invalident l'exécution d'une tâche suivant la définition 4.

➢ Les états de Succès et d'Echec définissent les états des entrées et des ressources nécessaires et suffisantes pour lesquelles la tâche a été considérée comme valide ou invalide et qui correspondent dans notre cas aux versions des documents.

➢ La priorité est une valeur numérique attribuée à chaque tâche permettant de résoudre les conflits engendrés par la parallélisation des tâches semi-indépendantes. La priorité permet de séparer les tâches parallèles correctes des tâches incorrectes.

➢ Le Statut définit l'état de la tâche : active (i.e., les préconditions sont satisfaites), inactive (i.e., les préconditions sont insatisfaites), exécutée ou inhibée qui correspond à un état de passage temporaire lors de l'occurrence d'une incohérence dans le système.

• **La description comportementale** pointe vers les éléments opérationnels nécessaires à l'exécution de la tâche. Puisque, dans notre application de rédaction coopérative, les tâches sont réalisées par l'acteur en coopération avec son agent, la partie de la tâche réservée à l'agent logiciel se limite aux actions de réponse aux requêtes émises par l'acteur. Cette description renferme des règles d'inférence ou des procédures.

• **La structure de contrôle** permet aux agents logiciels d'être totalement indépendants du problème du domaine à résoudre. Elle comprend plusieurs méthodes : des méthodes de contrôle des préconditions pour l'exécution de la tâches ; des méthodes qui activent l'exécution de la tâche une fois les conditions d'exécution réunies ; des méthodes d'exécution de la partie comportementale de la tâche ; des méthodes pour contrôler cette exécution ; des méthodes d'inhibition de la tâche qui s'activent lors de l'occurrence d'un dysfonctionnement et des méthodes de reprise des états d'exécution de la tâche après inhibition temporaire par validation des résultats de la tâche ou par réactivation.

3.2 Relations entre tâches

Nous définissons formellement les différents types de relations entre tâches introduites dans la description du processus de rédaction détaillée dans la section 2.

Définition 1 : Relation de Dépendance entre tâches
Soient T_i et T_j deux tâches du domaine. Soit $\Re_D$ la relation de dépendance entre les tâches.
$T_i \Re_D T_j$ signifie que la tâche T_i est fonctionnellement dépendante de T_j lorsque les sorties de la tâche T_j sont nécessairement des entrées ou des ressources de la tâche T_i.

Définition 2 : Relation d'Indépendance entre tâches
Soient T_i et T_j deux tâches du domaine. Soit $\Re_I$ la relation d'indépendance entre les tâches.
$T_i \Re_I T_j$ signifie que la tâche T_i est fonctionnellement indépendante de la tâche T_j si aucune sortie de T_j n'est ni en entrée ni en ressource de la tâche Ti.

Définition 3 : Relation d'Indépendance partielle ou semi-Indépendance entre tâches
Soient T_i et T_j deux tâches du domaine. Soit $\Re_P$ la relation d'indépendance partielle entre les tâches.
$T_i \Re_P T_j$ signifie que la tâche T_i est partiellement indépendante de la tâche T_j lorsque les sorties de la tâche T_j "peuvent être" des entrées ou des ressources de la tâche T_j.
Le quantificateur "peut être" entre les sorties de T_j et les entrées et ressources de T_i montre l'existence de certaines situations dans lesquelles l'exécution de la tâche T_i est indépendante de l'exécution de la tâche T_j.

Les différentes opérations d'activation, de validation et d'invalidation des tâches du domaine sont effectuées par les méthodes de la partie contrôle d'une tâche conformément aux définitions suivantes :

Définition 4 : Exécution valide de tâche partiellement indépendantes
On considère que l'exécution d'une tâche T_i effectuée à partir des entrées et des ressources suffisantes est valide si aucune des tâches T_j exécutées en parallèle et de priorité supérieure ou égale à la priorité de la tâche T_i n'a généré une nouvelle sortie qui soit entrée ou ressource nécessaire pour la tâche T_i.

Formellement, cette définition se traduit par :
Soit T l'ensemble des tâches du domaine et T_i une tâche de T. Soient $E_{suf}(T_i)$, $E_{néc}(T_i)$ et $R_{suf}(T_i)$, $R_{néc}(T_i)$ les entrées et les ressources respectivement suffisantes et nécessaires pour l'exécution d'une tâche T_i et Sorties(T_i) les sorties de T_i. Soient $t_{init}(T_i)$ et $t_{final}(T_i)$ les instants de début et de fin d'exécution de T_i. Soient Valide un prédicat qui s'évalue à vrai si l'exécution de la tâche T_i est valide, Priorité (T_i) la priorité de la tâche T_i dans le système et état(x, t_i) un opérateur[2] qui renvoie l'état de l'entrée ou de la ressource x à l'instant t_i.

Valide (T_i) = vrai si $\forall T_j \in T$, telle que $T_i \, \Re_P \, T_j$ et Priorité (T_i) $\leq$ Priorité (T_j)

$$\forall x \in [E_{suf}(T_i) \cap \text{Sortie}(T_j)] \cup [R_{suf}(T_i) \cap \text{Sortie}(T_j)]$$
$$\text{état}(x, t_{init}(T_i)) = \text{état}(x, t_{final}(T_j))$$

Définition 5 : Tâche active
On considère qu'une tâche est active à un instant t_i, lorsque ses conditions d'activation sont satisfaites à cet instant ou bien lorsque ces mêmes conditions ont été satisfaites à un instant t_k précédant son activation et ses conditions d'invalidation ou d'échec n'ont pas été satisfaites sur l'intervalle $[t_k, t_i]$ [3].

La définition formelle d'une tâche active est la suivante :
Soit T l'ensemble des tâches du domaine et T_i une tâche de T. Soit Satisfait(p,t_i) un prédicat qui s'évalue à vrai si la condition p est vérifiée à l'instant t_i. T_i est considérée comme active à l'instant t_i si l'une des conditions suivantes est vérifiée :
. $\forall p_m \in$ PréCondition(T_i), Satisfait (p_m, t_i).
. $\forall p_m \in$ PréCondition(T_i), Satisfait (p_m, t_k) avec $k < i$ et
$\neg \exists c_n \in$ EchecCondition(T_i) tel que Satisfait(c_n, t_j) avec $k < j \leq i$.

Pour permettre aux agents de vérifier automatiquement que les tâches exécutées en parallèle dans le système ne sont pas en situations conflictuelles (cf. fig. 3.b), nous avons besoin de définir la cohérence d'exécution des tâches parallélisées.

Définition 6 : Cohérence d'exécution de tâches
On considère que l'exécution d'une tâche T_j est cohérente par rapport à l'exécution d'une tâche T_i si T_i ne partage aucune entrée de T_j (tâches indépendantes) ou si l'exécution de la tâche T_j ne modifie pas les entrées suffisantes ou les ressources suffisantes de la tâche T_i (cf. fig. 3.a dans laquelle T_i = relecture(SF) et T_j = revue(SF)).

Formellement, cette définition se traduit par :
Soit T l'ensemble des tâches du domaine, T_i et T_j deux tâches de T, état(x, t_i) un opérateur qui renvoie l'état de l'entrée ou de la ressource x à l'instant t_i. Soient $E_{suf}(T_i)$, $E_{néc}(T_i)$ et $R_{suf}(T_i)$, $R_{néc}(T_i)$ les entrées et les ressources respectivement suffisantes et nécessaires pour l'exécution de T_i. Soit Sorties(T_i) les sorties de la tâche T_i. Soient Cohérence (T_j , T_i) un prédicat qui s'évalue à vrai si l'exécution de la tâche T_j est cohérente par rapport à l'exécution de T_i, $t_{init}(T_i)$ et $t_{final}(T_i)$ les instants de début et de fin d'exécution de la tâche T_i.

Cohérence (T_j , T_i) = vrai si $\neg \exists x \in [\text{entrées}(T_i) \cap \text{entrées}(T_j)] \cup [\text{ressources}(T_i) \cap \text{entrées}(T_j)]$
tel que entrées (T_i) = $E_{suf}(T_i) \cup E_{néc}(T_i)$ et
ressources (T_i) = $R_{suf}(T_i) \cup R_{néc}(T_i)$
Ou bien
$\forall x \in [E_{suf}(T_i) \cap \text{Sortie}(T_j)] \cup [R_{suf}(T_i) \cap \text{Sortie}(T_j)]$
état (x, $t_{init}(T_i)$) = état (x, $t_{final}(T_j)$)

[2]Dans notre exemple fig. 2, on représente la version d'un document SF à l'instant t_1 par état(SF,t_1)=V.SF_1.
[3]Cette définition signifie que, soit une tâche peut être exécutée par un acteur, soit, il n'y a pas d'acteur disponible.

Dans le cadre de notre application, cette définition indique que la version d'un document qui correspond à l'état d'une entrée suffisante ou d'une ressource suffisante d'une tâche T_i ne doit pas être modifiée par une tâche T_j exécutée en parallèle avec T_i.
A partir de ce prédicat, nous définissons les tâches dont l'exécution est valide à l'instant t_i et l'ensemble des tâches autorisées à l'instant $t_{k\,(k>i)}$. Pour cette raison, nous introduisons la notion de processus cohérent.

Définition 7 : Un processus d'activités cohérent
Soit T_a l'ensemble des tâches actives à un instant t_i dans le système coopératif, t une tâche de T_a et ω un ensemble de tâches, $\omega \subset T_a$. ω est un processus cohérent, si pour chaque $t \in \omega$, on a :

1. $\forall t' \in \omega$ on a Cohérence (t , t') .
2. $\forall t'' \in T_a / \omega$ // T_a / ω est l'ensemble des tâches dont l'exécution est invalide et donc à réactiver aux prochains cycles.

   on a :

   a. si $\neg$ Cohérence (t , t'') alors t. priorité $\geq$ t''. priorité ou bien
      $\exists$ t* $\in$ ω tel que t* . priorité > t''. priorité.
   b. si Cohérence (t , t'') alors $\exists$ t* $\in$ ω tel que $\neg$ Cohérence (t* , t'') et
      t*.priorité > t''. priorité.

Pour affecter la priorité attribuée aux tâches du processus d'activités coopératives, l'agent se réfère à son modèle de dépendances et d'indépendances entre les tâches. La stratégie que nous avons retenue pour faire cette affectation consiste à attribuer les poids les plus forts aux tâches ayant une valeur maximale de semi-indépendance (plus une tâche a des relations de semi-dépendance avec d'autre tâches, plus elle peut générer des conflits).

Comme nous l'avons signalé précédemment, la forme de parallélisation des tâches semi-indépendantes provoque des situations de conflits. A ces conflits, les agents du système réagissent par la validation de l'exécution de certaines tâches et l'invalidation d'autres exécutions lorsque la priorité attribuée à chaque tâche permet aux agents d'initier ce processus décisionnel. Par exemple, une nouvelle version de la spécification fonctionnelle générée par la tâche de relecture n'affecte pas la tâche de revue de la SF exécutée en parallèle puisque les modifications apportées lors de la relecture d'un document portent uniquement sur sa qualité linguistique indépendamment de sa valeur sémantique et donc aucun conflit n'apparaît. Dans d'autres cas, i.e., lorsque les tâches possèdent les mêmes priorités, les agents font appel aux acteurs qui les assistent pour résoudre les conflits entre ces tâches (Pinson et Moraïtis, 1996).

4 Pourquoi une approche distribuée pour les systèmes d'aide au travail coopératif ?

Notre but principal est de fournir aux acteurs un support logiciel pour l'articulation du travail distribué, but qui est commun à plusieurs chercheurs en CSCW (Borghoff et al, 1997 ; Divitini et al, 1996 ; Ishiguro et al, 1996 ; Zacklad, 1995). L'orientation multi-agent de notre recherche a été motivée par la puissance potentielle que peut offrir une communauté d'agents dans un système coopératif lors de la résolution d'un problème complexe et distribué. Parmi les aspects qui nous ont poussé vers une distribution du contrôle dans les agents, nous retiendrons les suivants :

- La distribution intrinsèque d'un système coopératif. En effet, la centralisation de ses tâches est quasiment inexploitable. Il est plus facile de le considérer en tant que groupe d'agents logiciels en interaction permanente avec un groupe d'acteurs.
- La forte dynamicité lors de l'exécution des tâches d'un processus d'activités coopératif. Le système de Workflow doit s'adapter aux changements dynamiques, c'est-à-dire aux

changements intervenant dans une activité alors que celle-ci est en cours d'exécution. Gérer des changements dynamiques implique le besoin de vérifier que l'activité est consistante avant et après l'occurrence des changements. Ainsi, lorsque les changements interviennent, le système de workflow doit adapter les différentes activités du processus en fonction de ces changements sans modifier les activités qui ne sont pas altérées (Borghoff et al, 1997).

- L'existence de plusieurs formes de coopérations et d'interactions : une forme de coopération horizontale (entre agents) et verticale (entre un agent et son acteur). L'étude de la coopération est un domaine de prédilection des recherches en Intelligence Artificielle Distribuée, principalement en systèmes multi-agents (Müller et al, 1996 ; O'Hare et Jennings, 1996 ; Perram et Muller, 1994 ; Wooldridge et Jennings, 1995).

5 Modèle d'agent logiciel

Les recherches effectuées sur les systèmes multi-agents concernent différents domaines. Dans cet article, nous nous focalisons sur les architectures d'agent et principalement celles liées à notre problème, c'est-à-dire les architecture collaboratives et hybrides. En nous appuyant sur les travaux bibliographiques effectués par plusieurs chercheurs [Wooldridge, M. et Jennings, N., R. 95] [Nwana, H. S. et Ndumu, D. T. 97], nous présentons les architectures les plus connues. Nous présentons, ensuite, l'architecture que nous préconisons pour les systèmes de workflow parallèle qui allient plusieurs caractéristiques des architectures citées précédemment.

5.1 Les architectures standards

5.1.1 Les Architectures collaboratives

Ces agents dont l'architecture est collaborative ont pour objectif l'accomplissement de certaines des tâches de leurs utilisateurs. Ils peuvent apprendre les comportements de ces derniers et réagir en fonction des connaissances acquises. Pour coordonner leurs activités, ces agents utilisent des protocoles de négociation. Ils peuvent être bénévoles, rationnels, sincères ou combiner plusieurs de ces caractéristiques. Le but visé lors de l'apparition de ce modèle est : (1) la coopération entre plusieurs agents hétérogènes en permettant ainsi à chaque agent de profiter des connaissances des autres [Huhns, M.,N. & al 94] ; (2) la résolution d'un problème complexe dont l'expertise ne peut être possédée par un unique agent central ; (3) fournir des solutions distribuées pour des systèmes qui sont fondamentalement distribués (ex. les réseaux de capteurs distribués DVMT [Durfee, E., H. & al 87] et le trafic aérien OASIS [Rao, A., S. & al 95] sont des systèmes de ce genre) [Nwana, H., S. & al 97].
Un exemple d'architecture collaborative est celle proposée par Sycara [Sycara, K. 95]. PLEIADES est une architecture à deux niveaux d'abstraction. Le premier niveau contient des agents de tâches-spécifiques et le second niveau contient des agents informations-spécifiques (cf. figure 6). Cette architecture a été utilisée pour développer un système de consultation (Visitor Hosting System) dans lequel les agents de tâches-spécifiques réalisent certaines des tâches de leurs utilisateurs (par exemple organiser des rencontres avec d'autres agents). Ces agents collaborent entre eux via le premier niveau pour résoudre les conflits ou intégrer de nouvelles connaissances. Ils rassemblent les informations nécessaires à leurs traitements grâce aux agents informations-spécifiques qui collaborent entre eux via le second niveau de l'architecture. Les sources d'information sont les différentes bases de données dans la sphère informationnelle mises à la disposition des agents informations-spécifiques. Ces agents possèdent les informations sur les formalismes de représentation des données dans ces bases et les informations leur permettant l'accès et l'extraction des connaissances de ces bases. Les agents tâches-spécifiques sont également dotés de capacités d'apprentissage. Un agent dans ce modèle est formé d'un module de planification et d'un moniteur d'exécution. Les agents

instancient des plans de tâches et coordonnent leur plans avec ceux des autres agents dans le système.

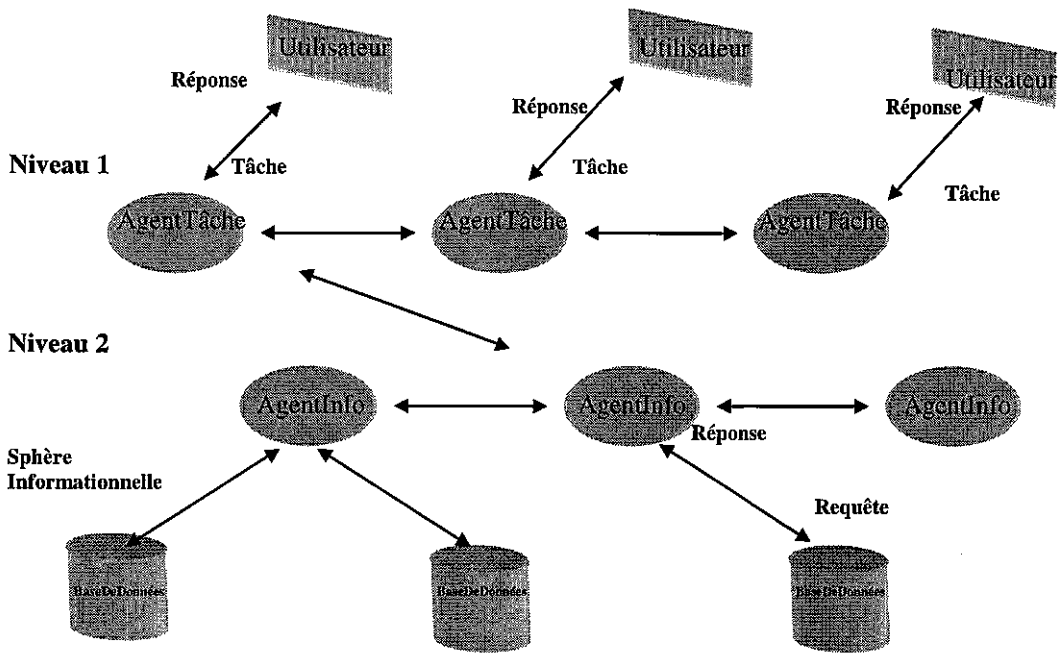


Figure 6 Architecture d'agents du système PLEIADES

D'autres chercheurs se sont également penchés sur ce genre d'architectures. Plusieurs prototypes de systèmes à base d'agents collaboratifs ont été développés, MII [Titmuss, R. & al 96] et ADEPT[O'Brien, P. & al 96]. MII est un système décentralisé de commerce électronique, ADEPT est une application du type Business Process Re-engineering.
Plus récents encore, un autre type d'agents dit d'interface ont été spécialement conçus pour être collaboratifs avec les utilisateurs. Les agents de ce type offrent un support aux travaux de ces derniers en recourant le plus souvent à des techniques d'apprentissage. Kozierok et Maes [Kozierok, R. & al 93] ont proposé un agent interface qui assiste son utilisateur dans l'organisation de ses meetings. A travers le temps, il apprend les préférences de son utilisateur. Dent et al [Dent, L. & al 92] ont également présenté un agent de ce type
Les recherches sur le projet IMAGINE [Haugeneder, H. & al 94] s'inscrivent également dans ce genre d'architectures. A la fois collaboratif et coopératif, le modèle d'agent proposé est constitué de trois parties : une partie fonctionnelle pour la résolution de problèmes (Agent Body), une partie coopération (Agent Head) et une partie communications externes avec les agents de l'environnement (Communicator). Une structure de coopération dans ce modèle est une instance d'une stratégie de coopération. Le modèle se fonde sur les trois structures suivante : le Contract Net Protocol, la structure maître-esclaves et la structure de coopération à base de Blackboard introduite dans le système Hearsay [Erman, L., D. & al 80].
Une autre catégorie d'agents collaboratifs sont les agents d'Internet spécialisés dans la recherche d'informations sur le Web à partir de différentes sources d'informations. Etzioni et Weld [Etzioni, O. & al 94] proposent un état de l'art sur les agents d'Internet appelés Softbot (Software Robot, par analogie aux robots matériels). Les utilisateurs de ces systèmes émettent des requêtes complexes aux agents qui utilisent des stratégies pour déterminer les méthodes de recherche à adopter. Liebermann a décrit Letizia, un agent de recherche d'informations qui assiste l'utilisateur du système[Liebermann, H. 95].

5.1.2 Les architectures hybrides

Les architectures hybrides sont conçues pour allier des capacités réactives à des capacités cognitives des agents, leur permettant ainsi d'adapter leurs comportements à l'évolution de l'environnement.
Ferguson [Ferguson I., A. 92] a proposé une architecture hybride (TouringMachines) dotée de plusieurs sous-systèmes de perceptions et d'actions organisés en trois niveaux (planification, réaction et modélisation) soumis au contrôle d'un système de médiation entre les différents niveaux. Chaque niveau s'exécute en parallèle avec les autres niveaux (cf. figure 7).

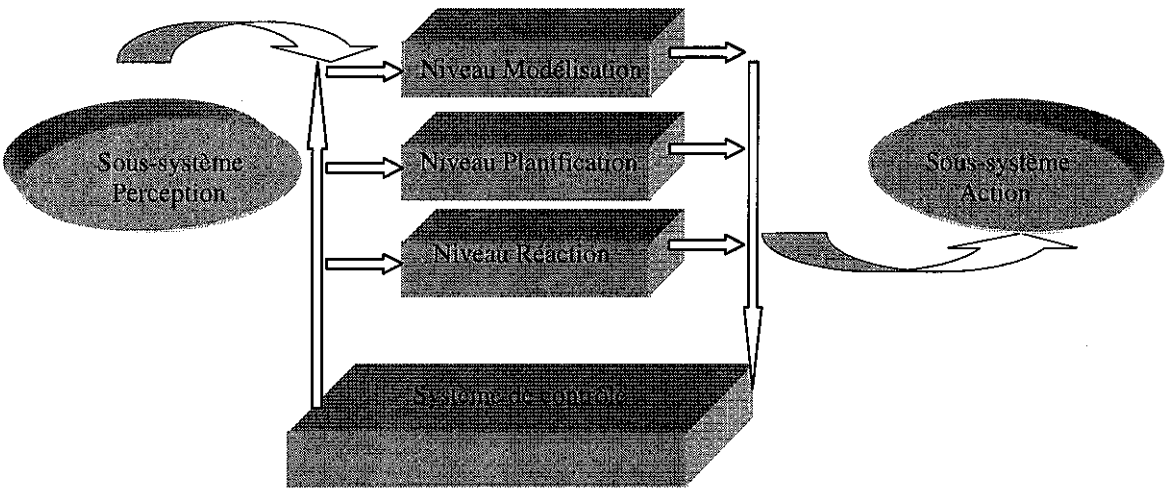


Figure 7 Modèle d'agent de Ferguson [Ferguson I., A. 92]

Le niveau planification construit des plans et sélectionne les actions exécutables par l'agent lui permettant d'atteindre son but. Le niveau réaction répond aux changements rapides dans l'environnement. Le niveau modélisation renferme des descriptions symboliques de toutes les entités dans l'environnement de l'agent. Ce modèle est utilisée par l'agent pour la résolution de conflits entre agents. Les trois niveaux communiquent entre eux par envoi de messages.
L'architecture COSY [Burmeister, B. & al 92] est une architecture BDI hybride formée de plusieurs composants (cf. figure 8). Le plus important est le composant cognitif qui gère les croyances de l'agent et ses intentions pour choisir les actions appropriées à réaliser. Le composant cognitif est formé d'une base de connaissances qui renferme les croyances de l'agent et trois composantes procédurales : un exécuteur de scripts, un manager de protocoles et un composant (raisonnement, décision et réaction). Les connaissances sur l'applications sont codées dans des plans de deux types : des scripts décrivant des actions permettant d'atteindre un certain but et des protocoles de coopération entre agents [Burmeister, B. & al 93]. Les scripts sont manipulés par le module d'exécution de scripts et les protocoles sont gérés par le manager de protocoles. Haddadi a montré la différence entre son architecture [Haddadi, A. 96] et celle de Cohen et Levesque [Cohen, P., R. & al 90] et Rao et Georgeff [Rao, A., S. & al 91] sur la modélisation stratégique des buts à long-terme, des préférences, des rôles et responsabilités, et des intentions tactiques directement rattachées aux actions de l'agent.

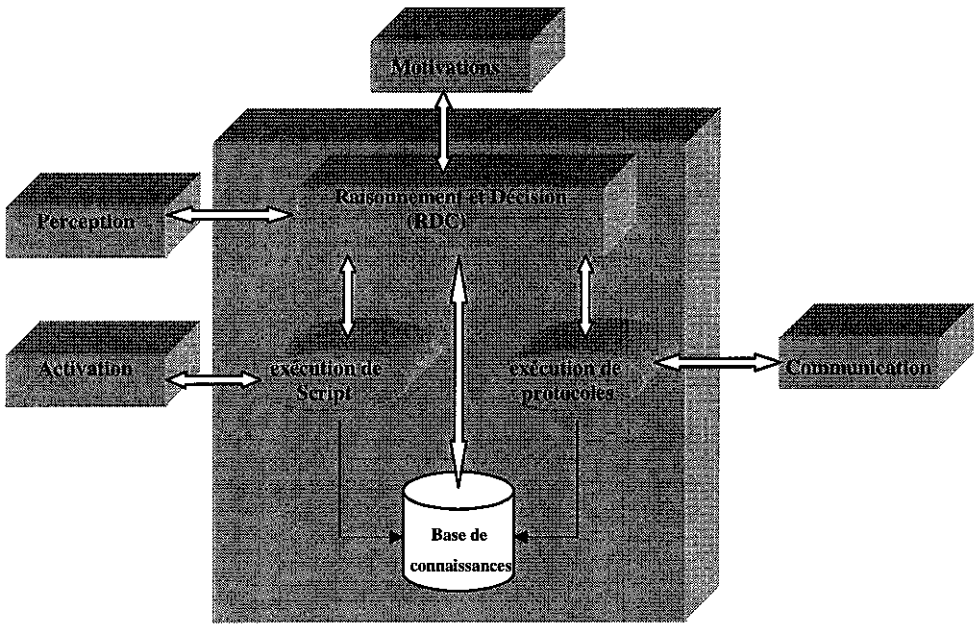


Figure 8 Modèle COSY [Haddadi, A. 96]

Dans le même axe des travaux de Ferguson, Müller et Pischel [Müller, J., P. & al 94] ont développé l'architecture InteRRap. Comme le souligne Müller, ce modèle à plusieurs niveaux offre plusieurs avantages : (1) il garantit la modularité de l'agent ; (2) il permet le traitement parallèle des connaissances dans les différents niveaux et donc une amélioration des capacités de traitement de l'agent ; (3) il augmente la réactivité de l'agent.
Dans ce modèle, chaque niveau de l'agent est subdivisé en deux autres sous-niveaux : un premier sous-niveau de connaissances manipulé par un second sous-niveau de contrôle qui contient différents composants (cf. Figure 9).

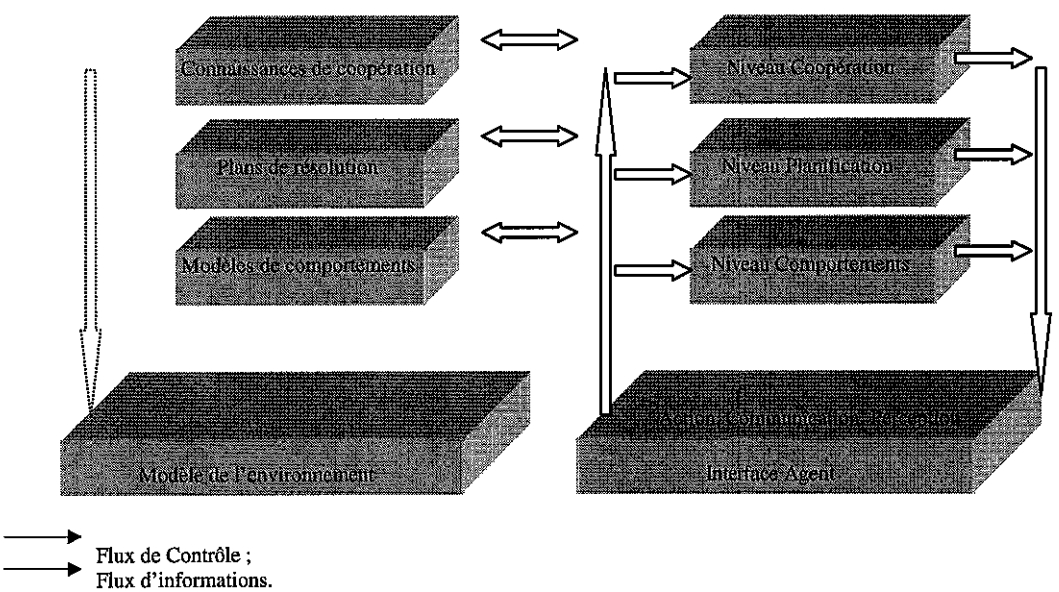


Figure 9 Modèle d'agent InteRRap

Le dernier niveau de l'architecture contient une interface permettant à l'agent de communiquer avec son environnement. Au-dessus de ce niveau, le niveau comportements gère et contrôle les comportements réactifs de l'agent. Le niveau planification génère et évalue les plans d'actions exécutables par l'agent. Le niveau coopération possède la même structure que le niveau planification mais dont le but est la construction de plans communs à plusieurs agents afin de coordonner leurs activités dans l'environnement partagé. L'architecture est dotée de deux mécanismes de contrôle hiérarchiques. Les résultats du niveau coopération sont des plans partiels raffinés par le niveau planification faisant appel aux comportements du niveau comportements. Ce modèle a été appliqué dans plusieurs domaines [Fischer K. & al 96].

5.1.3 Les agents mobiles

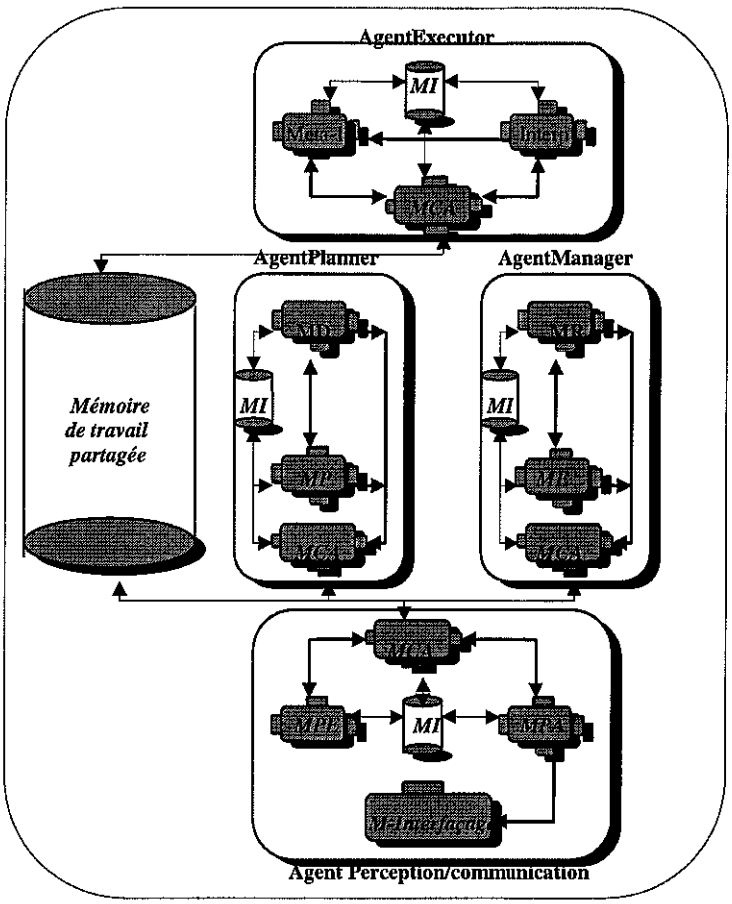
Les agents mobiles sont des agents capables de se déplacer de site en site à travers un réseau afin de collecter ou de traiter de l'information sur des sites distants. Ils sont classés dans la catégorie d'agents car ils respectent les principes d'autonomie et de coopération des agents. Les avantages de ce modèle apparaissent en particulier au niveau des applications sur Internet. Les agents mobiles fonctionnent grâce aux langages de programmation d'applications distribuées tels que : Java, Telescript et même C/C++. Appleby et Steward [Appleby, S. & al 94] ont présenté un prototype d'agents mobiles programmés en C/C++ pour contrôler les réseaux de télécommunications.
Les premières applications de ce modèle commercialisées sont Magic PDA et PIC (Personal Intelligent Communicator). Plu [Plu, M. 95] mentionne que FranceTelecom a également implémenté certains de ses services avec Telescript.

5.2 Modèle d'agent logiciel proposé

L'étude des différentes architectures proposées dans la littérature ainsi que la description des processus d'activités coopératives décrites dans la section 3, nous ont permis d'identifier les besoins suivants. Les agents doivent (1) reconnaître les événements se produisant dans leur environnement ; (2) répondre à ces événements et réagir en temps opportun ; (3) sélectionner dynamiquement les tâches à réaliser, ce qui permet une construction dynamique des plans d'activités des acteurs qu'ils assistent (cf. section 3) ; (4) s'assurer que toutes les conditions relevant d'une tâche sont satisfaites avant, durant et après son exécution ; (5) coordonner leurs tâches avec celles des acteurs qu'ils assistent lors de l'exécution du processus de rédaction coopérative. Ces aspects ont motivé l'architecture récursive d'agent que nous allons détailler dans la section suivante.
Le modèle d'agent proposé peut être considéré comme un nouveau modèle d'agent générique pour les systèmes coopératifs. Il est composé de différents agents primitifs (cf. fig. 11) : les agents de décision (un AgentManager et un AgentPlanner), l' agent d'exécution (AgentExecutor), l' agent de perception/communication et la mémoire de travail.
Le premier agent de décision (AgentManager) intègre les comportements qui gèrent l'exécution des tâches du domaine. Le second agent de décision (AgentPlanner) permet de planifier dynamiquement l'enchaînement de ces tâches, ce qui implique une génération dynamique des plans d'activités pour chaque acteur. Quant à l'AgentExecutor, il participe avec son acteur à l'exécution des tâches sélectionnées par les deux agents de contrôle. Chacun des agents de l'agent logiciel communique par l'intermédiaire de la mémoire de travail partagée.

Cette structure récursive d'agent présente plusieurs avantages. (1) Elle assure la parallélisation des tâches de l'agent, notamment l'exécution des tâches actives et la planification des futures tâches à activer, et donc la réactivité dans la prise des décisions provoquée par l'évolution rapide de l'environnement, avantages soulignés par différents auteurs (Demazeau, 1990 ; Stinckwich, 1994 ; Occello et Demazeau, 1997 ; Marcenac, 1997) ; (2) cette architecture prend

en compte les exigences de mobilité des nouvelles applications sur Internet dont l'intérêt a été largement discuté dans (Parc et Leuker, 1997). Elle permet de traiter de façon efficace le travail coopératif dans le cas d'utilisation de l'application sur Internet. Comme le souligne B. Crabtree dans (Nwana et al, 1997) : "imagine having to download many images just to pick up one. Is it not more natural to get your agent to 'go' to that location, do a local search and only transfer the chosen compressed image back across the network?". Le caractère multi-agent de l'agent permet la migration partielle de son code, i.e., l'AgentExecutor est capable de migrer pour réaliser la tâche sollicitée dans le site concerné alors que les composants non directement impliqués dans la réalisation de cette tâche peuvent rester dans le site initial. Ils peuvent échanger des messages avec l'AgentExecutor via leurs modules de communication. La migration partielle du code est une alternative particulièrement intéressante car elle permet de réduire les risques de submersion du réseau par du code non exploité.



Légende

Flux de contrôle ;
Transfert de la connaissance ;
***MI** Mémoire Interne ; **MR** Module de Reconnaissance ; **MD** Module de décision ;*
***ME** Module d'Exécution ; **MP** Module de Planification ; **Meta-I** Meta Interpréteur ; **Interp** Interpréteur ;*
***MCA** Module de Communication Agent ; **MPE** Module de Perception Environnementale ;*
***MPA** Module de Perception Acteur ; **M-Interfaçage** Module d'interfaçage.*

Figure 11 Modèle générique d'un agent logiciel

De plus, en nous référant aux arguments fournis dans (Rothermel et al, 1997), le transfert des données via le réseau sans surveillance constante peut parfois remettre en cause leur authenticité et leur sécurité. Dans le cadre de notre application, le déplacement des agents nous assure la confidentialité et l'authenticité des données. En effet, certaines données nécessaires aux traitements de l'agent ne sont plus transférées

via le réseau ; l'agent les traite directement au niveau de leurs sites de stockage. A partir de ces sites, il renvoie uniquement ses propres résultats.
(3) La séparation entre le contrôle de l'agent (totalement indépendant du domaine d'application) et les comportements réalisant les tâches du domaine permet à l'agent de raisonner sur lui-même, le dotant ainsi d'un comportement réflexif. Ceci permet une réutilisation plus facile du système d'aide au travail coopératif pour d'autres applications, uniquement par redéfinition des tâches du domaine que doit exécuter l'agent ;
Dans la suite de cette section, nous présenterons en détails chacun des agents primitifs de notre modèle d'agent.

- **Agents de décision**

Les deux agents de décision sur lesquels repose cette architecture sont l'AgentManager et l'AgentPlanner. Nous avons choisi de séparer les deux niveaux, planification et gestion des tâches, afin de permettre une réaction instantanée des agents aux événements se produisant dans leur environnement. Ce choix est nécessaire vu la forme de parallélisation des tâches semi-indépendantes que nous avons retenue (cf. section 3). Ainsi, la détection et le traitement des conflits entre agents sont immédiats.
Ces agents activent, désactivent, valident et invalident (cf. section 3) les tâches exécutées en parallèle dans le système coopératif. Ils peuvent autoriser l'inhibition de certaines tâches si les événements perçus par l'AgentPerception/Communication nécessitent une réaction immédiate afin de confirmer la validité des résultats des tâches parallélisées.

L'AgentManager et AgentPlanner sont décrits dans la figure 11. Ils ont un fonctionnement complètement imbriqué. À chaque cycle de contrôle de l'agent logiciel, ses connaissances sont remises à jour grâce à l'AgentPerception/Communication. Les nouvelles connaissances sont insérées dans la mémoire de travail commune aux quatre agents primitifs. Elles sont ensuite analysées et orientées par l'AgentManager vers l'AgentPlanner responsable de la prise des décisions.
Le module d'exécution ME de l'AgentManager est subdivisé en trois sous-modules : le module d'initiation des processus de négociation (MIN), le module de transmission des données vers l'AgentPlanner (MTP) et le module de transmission des données vers l'AgentExecutor (MTE).
Le module de planification MP de l'AgentPlanner est subdivisé en trois sous-modules : le module de sélection des tâches exécutables par l'acteur et l'agent logiciel (MS), le module de validation (MV) et le module d'activation des tâches négociées (MAT).

À chaque cycle de contrôle, l'AgentManager vérifie les conditions d'activation des tâches qui correspondent aux "PréConditions" et les conditions d'exécution des tâches activées au cours des cycles précédents. En reconnaissant de nouvelles tâches actives dans le système, le module MIN initialise un processus de négociation avec le reste des agents du système par un appel d'offres auquel tous les agents compétents répondent. L'offre émise par chaque agent évalue l'aptitude de l'acteur qu'il contrôle et sa disponibilité pour la réalisation de la tâche.
Pour tout événement perçu par l'AgentPerception/Communication (par exemple un changement de l'état d'un document), le module de reconnaissance des situations (MR) de l'AgentManager définit des types d'actions à réaliser (ex. inhibition des tâches) en réponse à l'événement.
Le module MTP de l'AgentManager transmet tous les dysfonctionnements rencontrés au niveau de l'exécution des tâches parallélisées à l'AgentPlanner. Un dysfonctionnement correspond à un conflit entre deux ou plusieurs agents. Le module de validation MV de l'AgentPlanner réagit à ces dysfonctionnements par application de l'algorithme de validation des tâches actives. Cet algorithme, décrit ci-dessous, calcule l'ensemble des tâches parallélisées dont l'exécution est cohérente dans le système (voir définition 6, section 3). Toute décision prise au niveau du

module de validation des tâches MV de l'AgentPlanner est transmise par le module MTE à l'AgentExecutor et à l'AgentPerception/Communication.
A la réception de nouveaux appels d'offres émis par d'autres agents logiciels pour la réalisation de nouvelles tâches ou bien de nouvelles propositions à des appels émis par l'AgentManager, l'AgentPlanner évalue chaque offre grâce à son module de décision MD et chaque proposition grâce à son module de sélection MS. Quant au module d'activation (MAT), il active les tâches allouées par l'agent logiciel.

Algorithme de validation de l'exécution des tâches (EVA)

L'algorithme reçoit en entrée l'ensemble T des tâches T_i actives à l'instant t_i.
Méthode ValidationTâches (T)
Début T' = Trier (T); // Trier est une méthode de tri des tâches T par ordre de priorité décroissante.
Pile = ()
Tant que (T' ≠ ())
Début
Dépiler (S, T');
Si ∀ x_j ∈ Pile, Cohérence (S , x_j) // cf. définition 6, (section3)
Empiler (S, Pile);
Fin
Retour Pile;
Fin

Le théorème suivant montre que l'algorithme retrouve le processus d'activités cohérent dans le système à tout instant t_i et se termine après un nombre fini d'étapes. La démonstration est proposée dans (Aknine, 1997).

Théorème 8
Soit T un ensemble fini de tâches exécutées à un instant t_i, les propriétés suivantes de l'algorithme EVA sont vérifiées :

a. l'algorithme EVA se termine.
b. le résultat T' de l'algorithme EVA (T) est un processus cohérent de T. // cf. définition 7, (section3)

• L'agent de Perception/Communication

L'agent de perception/communication est composé de deux modules : un module de perception environnementale (MPE) qui communique avec les agents du système et un module de perception acteur (MPA) qui analyse les traces d'exécution des tâches par l'acteur du système coopératif. Les résultats de l'analyse des traces sont signalés à tous les agents dont les tâches courantes dépendent partiellement ou totalement de la tâche de l'acteur associé à l'agent. Ces résultats sont des événements spécifiés sous la forme d'un 5-uples :

(<Instant>, <Tâche>, <Evénement>, <Acteur>, <Agent>)

Par exemple, dans le cadre de notre application, ces événements correspondent aux changements d'états des entrées suffisantes ou des ressources suffisantes des tâches parallèles.

(<t_i>, <Revue (SF)>, < V.SF3 >, <Acteurj>, <Agentj>)

Le module de perception environnementale de l'AgentPerception/Communication perçoit ces événements au niveau de chaque agent récepteur. Il transmet ces événements à l'AgentManager. Ce dernier inhibe les tâches en cours d'exécution afin de permettre à l'AgentPlanner de résoudre les situations de conflits éventuelles. En réponse à ces événements, l'AgentPlanner prend des

décisions qu'il transmet à l'AgentPerception/Communication par des commandes exprimées sous la forme d'un 5-uple:

(<Instant>, <Tâche>, <Action>, <Acteur>, <Agent>)

Par exemple, la commande d'inhibition de tâche émise par l'agent du système coopératif vers l'acteur est la suivante.

(<t_j>, <Relecture (SF)>, <Inhibition>, <Acteurj>, <Agentj>)

D'autres commandes émises par l'acteur portant sur le début et la fin des tâches, l'interruption et la réactivation des tâches parviennent également au module de perception acteur (MPA) de l'AgentPerception/Communication.

- **L'agent d'exécution (AgentExecutor)**

Cet agent répond aux requêtes émises par l'acteur à travers le module d'interfaçage de l'AgentPerception/Communication. Une réponse de l'agent est le résultat de l'activation d'une ou plusieurs méthodes contenues dans la spécification comportementale de la tâche en cours d'exécution par l'acteur. Certaines correspondent à des méthodes Java et d'autres à des bases de règles. Par exemple, pour chaque composant du réseau téléphonique retenu par l'acteur dans la spécification détaillée, l'agent logiciel peut calculer le gain de l'entreprise pour un tel choix par rapport à cette spécification et le gain global par rapport à d'autres spécifications. Il peut par exemple proposer d'autres composants plus rentables pour l'entreprise grâce aux connaissances qu'il détient dans sa base de connaissances sur le domaine. Un méta-interpréteur est nécessaire pour permettre à l'agent d'observer son comportement (Kornman, 1995 ; Pitrat, 1990 ; Guessoum et Durand, 1996). A la réception d'un message de la part de l'AgentManager, le méta-interpréteur définit les opérations qu'il doit exécuter et les changements à apporter sur les connaissances de l'interpréteur. Par exemple, lorsque l'AgentManager décide d'inhiber l'exécution d'une activité courante, il transmet deux types de messages, l'un pour l'acteur via l'agent de perception/communication et l'autre pour l'AgentExecutor. Le méta-interpréteur chargé d'assurer le contrôle des opérations de l'AgentExecutor récupère ces messages et bloque temporairement les opérations en cours d'exécution par l'AgentExecutor. En fonction de la décision finale prise par l'AgentPlanner, le méta-interpréteur peut réactiver de nouveau l'interpréteur avec les nouvelles connaissances.

- **La mémoire de travail**

Elle représente le médiateur informationnel central entre les quatre agents qui composent l'agent logiciel. Elle contient les connaissances déclaratives et algorithmiques sous la forme de script des tâches à réaliser (cf. section 4), les états courants des tâches en cours d'exécution par l'AgentExecutor et celles en cours de négociation par l'AgentManager et l'AgentPlanner avec le reste des agents de l'environnement. Ces états sont sauvegardés sous la forme suivante :

(<Instant>, < Tâche >, <Evénement>, <Acteur>, <Agent>)

Par exemple, l'événement d'activation de la tâche de rédaction de la spécification fonctionnelle à l'instant t_i par l'agent $Agent_j$ pour être exécutée par l'acteur $Acteur_j$ est enregistré sous la forme :

(<t_i>, <Rédaction (SF)>, <Activation>, <Acteurj>, <Agentj>)

Dans notre application, les tâches sont prédefinies lors de la conception du système. Les agents primitifs de l'agent logiciel modifient la valeur des attributs de la tâche au cours de leur raisonnement.

6 Interactions agent / acteur

Comme il a été indiqué dans la section (2), chaque acteur est représenté dans le système coopératif par un agent. Les interactions agent / acteur sont du type Client / Serveur (cf. figure 1). Cette relation spécifie que l'agent du système fournit des services à l'acteur qui se veut client. L'agent sert un seul client à la fois et régule les activités de plusieurs acteurs en même temps lorsque ces derniers interviennent sur des tâches semi-indépendantes (cf. définition 3). Les agents et les acteurs interagissent par un mécanisme d'échange de messages. Un message répond à un service demandé ou offert. A l'intérieur du système multi-agent, les agents communiquent entre eux par des actes de langages et se coordonnent suivant le protocole de réseaux de contrats (Smith et Davis, 1981). L'intégrité des activités des acteurs est obtenue grâce à la centralisation des opérations de résolution de conflits et de maintenance de la cohérence du système (cf. section 3.2) au niveau de chaque agent qui alloue des tâches parallélisables et partiellement indépendantes.
L'agent communique avec son acteur à travers son interface (MI) et avec les autres agents grâce à son module de perception environnemental (MPE) (cf. section 5.2). Le processus de communication entre agents est décrit dans la section 5.2.

7 L'interface

Il est important pour une interface acteur de s'adapter aux besoins des utilisateurs et permettre une interactivité avec l'acteur durant l'exécution du processus d'activités. En accord avec cet objectif, nous avons décidé d'implémenter l'interface acteur en utilisant les Applets Java. Puisque l'intérêt de l'utilisateur se limite à l'exécution de sa tâche et à la cohérence de ses résultats par rapport aux résultats de l'exécution des tâches concurrentes par les autres acteurs du système coopératif, nous avons décidé de rendre la représentation et la validation des résultats des tâches totalement transparentes à l'acteur. La figure 12 montre l'interface associée à un acteur dans notre maquette.

L'interface est composée des parties suivantes :

- Une fenêtre pour les notifications agent qui permet à l'agent l'affichage des événements internes se produisant dans le système. Les événements portent sur les changements effectués sur les documents en entrées ou en ressources de la tâche exécutée par l'acteur.
- Une fenêtre pour les requêtes acteurs qui lui permet de signaler les événements externes tels que le début, la fin et l'interruption de l'exécution d'une tâche ;
- Une fenêtre pour l'affichage de la base de tâches internes qui présente la liste des tâches planifiées pour l'acteur et la tâche en cours de réalisation ;
- Une fenêtre pour l'affichage des tâches concurrentes à la tâche courante de l'acteur et qui peuvent altérer son exécution. Enfin, la partie inférieure droite est réservée à l'exécution de la tâche courante, i.e., la rédaction d'un document dans le cas de notre exemple.

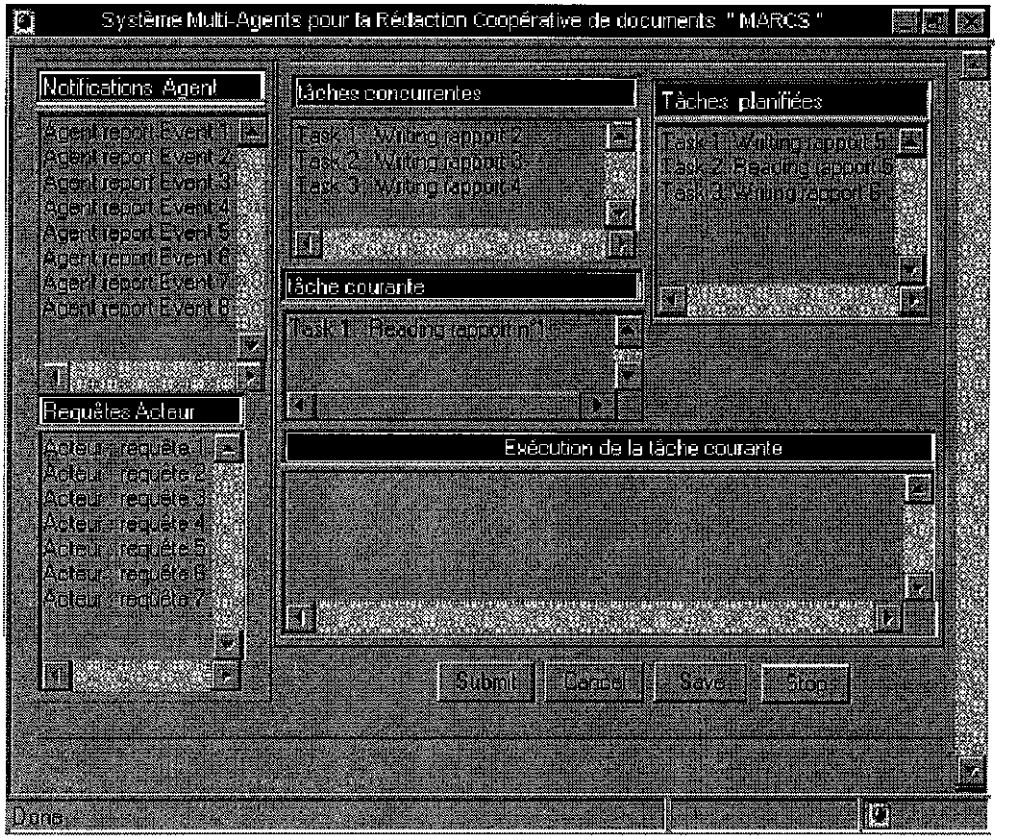


Figure 12 Interface graphique acteur pour l'exécution des tâches du processus d'activités.

Après que l'acteur ait initialisé son système à la tâche courante (cf. figure 12), l'agent spécifie à travers l'interface ''événements'' (cf. figure 13), les événements se produisant au niveau du système coopératif et qui altèrent la tâche de l'acteur. L'agent notifie l'acteur immédiatement après l'occurrence d'un événement et avant que celui-ci n'atteigne la fin de sa tâche, afin de pouvoir réagir opportunément aux événements, et si besoin est, modifier les résultats de l'activité courante de l'acteur.

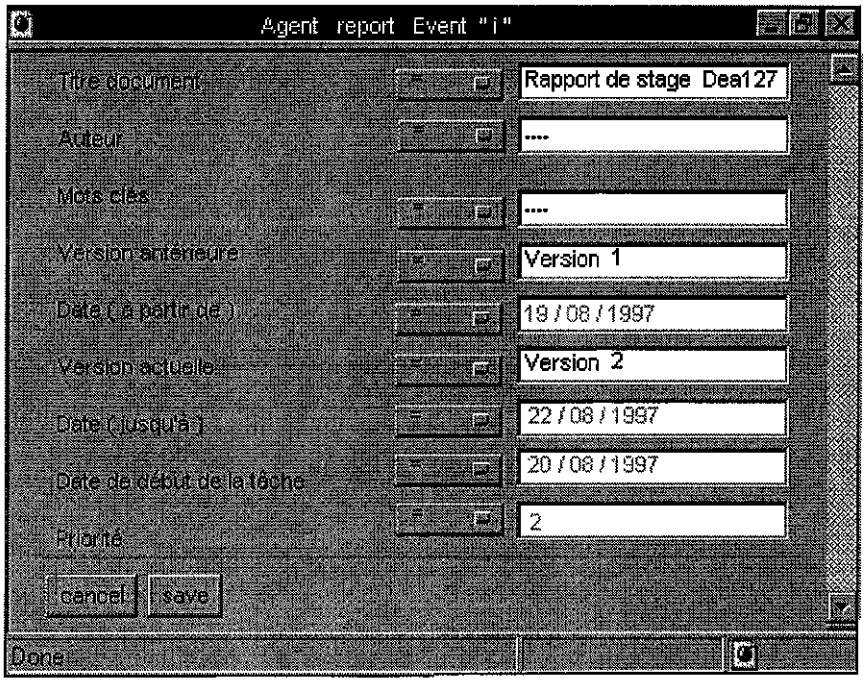


Figure 13 Interface pour l'affichage des événements survenants dans le système coopératif.

La maquette du système a été implémentée en VisualJ++. Le système est spécialement conçu pour tenir compte des exigences des nouvelles applications en mobilité en proposant une solution intéressante pour une migration partielle du code de l'agent.

8 Conclusion

Dans cet article, nous avons proposé un nouveau modèle d'agent pour la construction de systèmes de workflow parallèles et intelligents capables de s'adapter de façon réactive aux évolutions de l'activité d'un collectif de travail tout en facilitant la parallélisation et l'ordonnancement de certains travaux. La forme de parallélisation des activités d'un workflow proposée exploite l'indépendance partielle entre les activités d'un processus workflow. Nous l'avons illustrée en détail sur l'exemple de la rédaction coopérative de spécifications techniques dans le domaine des télécommunications.
Sachant que, dans un système de workflow parallèle, un agent intelligent doit coordonner ses activités avec les activités des autres agents et de son acteur et qu'il doit identifier et réaliser des tâches logiquement correctes en réponse à des événements externes, nous avons proposé un modèle d'agent autonome capable de raisonner à partir de sa représentation (de l'environnement, des autres agents, de l'acteur avec lequel il coopère et de ses propres connaissances) pour répondre aux exigences de la parallélisation des tâches semi-indépendantes proposée. L'architecture présentée est caractérisée par la récursivité et axée sur les trois principaux aspects des nouvelles applications : la mobilité, la réutilisabilité et la flexibilité. Les avantages d'une telle architecture sont les suivants :

(1) la séparation entre le contrôle de l'agent (totalement indépendant du domaine d'application) et les comportements réalisant les tâches du domaine permet à l'agent de raisonner sur lui-même, le dotant ainsi d'un comportement réflexif. Ceci permet une réutilisation plus facile du système d'aide au travail coopératif pour d'autres applications, uniquement par redéfinition des tâches du domaine que doit exécuter l'agent ;
(2) l'architecture multi-agent de l'agent de base permet à l'agent d'accomplir plusieurs tâches en parallèle, notamment l'exécution des tâches actives et la planification des futures tâches à activer ;
(3) cette architecture permet en outre de prendre en compte les exigences de mobilité des nouvelles applications sur Internet dont l'intérêt a été largement discuté dans (Parc et Leuker, 1997).
Il serait intéressant d'utiliser une telle architecture pour d'autres applications dans le sens où la description et l'implémentation des agents logiciels sont totalement indépendantes du domaine d'application (la rédaction coopérative de documents dans notre cas) et où le contrôle des agents est complètement indépendant des tâches du domaine qu'ils réalisent. Nous avons l'intention de poursuivre nos recherches dans ce sens et de valider le modèle sur différentes applications.

Bibliographie

Aknine, S. Langage de Tâches et Coopération, Rapport de DEA, Université Paris Dauphine, septembre 1997.

Aknine, S. ''Using Multi-Agent Systems to Support Workflow Telecommunications Applications'', IEEE ISIC/CIRA/ISAS '98, A Joint Conference on the Science and Technology of Intelligent Systems, Maryland U.S.A., September 1998a.

Aknine, S. ''A Reflexive Agent Architecture applied to Parallel WorkFlow Systems'', CE'98, 5th ISPE International Conference on Concurrent Engineering Research and Applications, Tokyo, Japan, July 15-17 1998b.

Appleby, S. & Steward, S., ''Mobile Software Agents for Control in Telecommunications Networks'', BT Technology Journal, Vol. 12, n°2, 1994.

Borghoff, M., Bottoni, P., Mussio, P. et Pareschi, R. ''Reflective Agents for Adaptive Workflows'', 2nd International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM'97), 1997.

Burmeister, B. Haddadi, A. & Sundermeyer, K., ''Generic Configurable Cooperation Protocols for Multi-agent Systems'', MAAMAW'93, Université de Neuchâtel, 1993.

Burmeister, B. & Sundermeyer, K., ''Cooperation Problem-Solving Guided by Intentions and Perception'', Werner, E. & Demazeau, Y. (eds.), Decentralized AI 3, Elsevier Science Publishers B. V., 1992.

Delouis, I. et Krivine, J. P. ''Implementing a Conceptual Model: Towards an Architecture that Allows Better Cooperation between the User and the System'', Twelfth International Conference on Artificial Intelligence, Expert Systems, Natural Language, Avignon, 1992.

Demazeau, Y. ''From Interactions to Collective Behaviors in Agent-Based System'', European Conference on Cognitive Science, St. Malo, avril 1995.

Demazeau, Y., Müller, J. P. (eds). Decentralized Artificial Intelligence, North Holland, 1990.

Dent, L., Boticario, J., McDermott, J., Mitchell, T. & Zabowski, D. A., "A Personal Learning Apprentice", 10th National Conference on Artificial Intelligence, 1992.

Divitini, M., Simon, C. et Schmidt, K. "ABACO : Coordination Mechanisms in a Multi-agent Perspectives", Second International Conference on the Design of Co-operative Systems, 1996.

Durfee, E. H., Lesser, V. R. & Corkill, D., "Coherent Cooperation among Communicating Problem Solvers", IEEE Transaction on Computers, C-36, n°11, 1987.

Ellis, C. A. "Information Control Nets: A Mathematical Model of Office Information Flow", Conference on Simulation, Measurement and Modeling of Computer Systems, 1979.

Erman, L. D., Hayes-Roth, F. & Lesser, V. R., "The Hearsay II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty", ACM Computing Surveys, 12 (2), 1980.

Etzioni, O. & Weld, D. "A Softbot-Based Interface to the Internet", Communications of the ACM, 37, n°7, 1994.

Ferber, J. Les systèmes Multi-Agent. Vers une Intelligence Collective, InterEditions, 1995.

Ferguson I. A., "Towards an Architecture for Adaptative, Rational, Mobile Agents", Werner, E. & Demazeau, Y. (eds.), Decentralized AI 3, Elsevier Science Publishers B. V., 1992.

Fischer, K., Müller, J. P. & Pischel. M., "Scheduling an Application Domain for DAI", Applied Artificial Intelligence, An International Journal, Vol. 10: 1- 33, 1996.

Glaser, N. "The CoMaMas Methodology and Environment for Multi-agent Systems Development", in Zhang, C. and Lukose, D. (eds.), Lecture Notes on Computer Science, Multi-agent Systems Methodologies and Applications, Springer Verlag, 1997.

Guessoum, Z. & Durand, R. "Des Agents Intelligents pour Modéliser l'évolution des Entreprises", Muller, J. P. et Quinqueton, J. (eds.), IA distribuée et Systèmes Multi-Agent, JIFIADSMA'96, Hermès, 1996.

Haddadi, A., Communication and Cooperation in Agent Systems: A Pragmatic Theory, Vol 1056 of Lecture Notes in Artificial Intelligence, Springer-Verlag, Heidelberg, 1996.

Haugeneder, H., Steiner, D. & McCabe, F. G., "IMAGINE: A Framework for Building Multi-agent Systems", Deen, S., M. (eds.), International Working Conference on Cooperating Knowledge Based Systems (CKBS'94), University of Keele, UK, 1994.

Huhns, M. N. & Singh, M. P., "Distributed Artificial Intelligence for Information Systems", CKBS'94, Tutorial, University of Keele, UK, 1994.

Ishiguro, Y., Tarumi, H., Asakura, T., Kida, K., Kusmi, D. et Yoshifi, K. "An agent architecture for personal and group work support", Second International Conference on Multi-Agent Systems, ICMAS 96, 1996.

Jennings, N. R., ''Specification and Implementation of a Belief Desire Joint Intention Architecture for Collaborative Problem Solving'', Journal of Intelligent and Cooperative Information Systems, 2 (3), 1993.

Kornman, S. ''A Meta-Level Architecture for Self Monitoring'', Workshop on Reflection and Meta-Level Architecture and their Applications in Artificial Intelligence, IJCAI 95, Montreal, 1995.

Kozierok, R. & Maes, P., ''A learning Interface Agent for Scheduling Meetings'', ACM-SIGCHI, International Workshop on Intelligent User Interfaces, Florida, 1993.

Liebermann, H., ''Letizia: An Agent that Assists Web Browsing'', International Joint Conference on Artificial Intelligence, AAAI Press, 1995.

Marcenac, P. ''Modélisation de Systèmes Complexes par Agents'', Technique et Science Informatiques, Vol.16, n°8, Octobre, 1997.

Müller, J. P. & Puschel, M., ''An architecture for dynamically Interacting Agent'', International Journal of Intelligent and Cooperative Information Systems, IJICIS, 3(1), 1994.

Müller, J., P., Wooldridge, M., J. et Jennings, N., R. (eds). Intelligent Agent III, Agent Theories, Architectures and Languages, European Conference on Artificial Intelligence, Budapest, Hungary, 1996.

Nwana, H., S. & Ndumu, D., T., ''An Introduction to Agent Technology'', Nwana, H., S. & Azarmi, N. (eds.), Software Agents and Soft Computing, Towards Enhancing Machine Intelligence, Springer, 1997.

O'Brien, P. & Wiegand, M., ''Agents of Change in Business Process Management'', BT Technol., J., n°4, 14, 1996.

Occello, M. et Demazeau, Y. ''Une Approche du Temps Réel dans la Conception d'Agents'', dans Quinqueton, J., Thomas, M., C. et Trousse, B. (eds), Journées Francophones d'Intelligence Artificielle et Systèmes Multi-agent, Hermès, France, 1997.

O'Hare, G.M.P. et Jennings, N. R. (eds), Foundations of Distributed Artificial Intelligence, John Wiley, 1996.

Parc, A. S. et Leuker, S. ''A Multi-agent Architecture Supporting Service Access'', in Rothermel, K. et Popescu-Zeletin, R., S. (eds), First International Workshop on Mobile Agents, MA'97, Berlin, Germany, 1997.

Perram, J. W. et Muller, J. P. (eds). Distributed Software Agents and Applications, 6th European Workshop on Modeling Autonomous Agents in a Multi-agent World, MAAMAW'94, 1994.

Pinson, S. et Moraïtis, P. ''An Intelligent Distributed System for Strategic Decision Making'', Group Decision and Negotiation, N° 6, Kluwer Academic Publishers, 1996.

Pitrat, J. Métaconnaissances, futur de l'intelligence artificielle, Hermès, 1990.

Plu, M., ''Software Agents in Telecommunications Environments'', Unicom Seminar on Intelligent Agents and their Business Applications, London, 1995.

Rao, A. S. & Georgeff, M. P., ''Modeling Agents Within a BDI Architecture'', Fikes, R. & Sandewall, E. (eds.), 2nd International Conference on Knowledge Representation and Reasoning (KR'91), Cambridge, Morgan Kaufmann, 1991.

Rao, A. S. & Georgeff, M. P., ''BDI Agents: From Theory to Practice'', ICMAS'95, San Francisco, USA, 1995.

Rothermel, K. et Popescu-Zeletin, R. S. (eds), First International Workshop on Mobile Agents, MA'97, Berlin, Germany, 1997.

Smith, R. G. et Davis, R. ''Frameworks for co-operation in Distributed Problem Solving'', IEEE Transaction on System Man and Cybernetics, 1981.

Stinckwich, S. Modèles Organisationnels et réflexifs des architectures à objets concurrents Implémentation en Smalltalk-80, Thèse de doctorat, Université de Savoie, 1994.

Sycara, K., ''Intelligent Agents and the Information Revolution'', UNICOM Seminar on Intelligent Agents and their Business Applications, London, 1995.

Titmuss, R., Crabtree, I., B. & Winter, C. S. (1996). ''Agents, Mobility and Multimedia Information'', BT Technol., J., n°4, 14.

Wooldridge, M. & Jennings, N. R. (1995). ''Agent Theories, Architectures and Languages: A Survey'', Wooldridge, M. & Jennings, N., R. (eds.), Intelligent Agents. Springer-Verlag.

Zacklad M. et Rousseaux F. « Modelling Co-operation in the Design of Knowledge Production Systems : the MadeIn'Coop Method », Computer Supported Cooperative Work: The Journal of Collaborative Computing, 5:133-154, 1996.