Faire coopérer des agents hétérogènes par apprentissage de médiation

Romaric Charton, Anne Boyer, François Charpillet

Faire coopérer des agents hétérogènes par apprentissage de médiation

R. Charton[1,2] A. Boyer[1,2] F. Charpillet[1,3]
charton@loria.fr boyer@loria.fr charp@loria.fr

[1] LORIA UMR 7503 - Campus Scientifique - BP 239
54506 Vandœuvre-lès-Nancy – FRANCE

[2]Université Nancy 2 [3]INRIA Lorraine

Résumé :
Dans le cadre de nos travaux sur la collaboration entre agents hétérogènes, nous avons étudié l'interaction entre un agent humain et un service de recherche d'informations. Nous avons proposé d'introduire un agent médiateur dont le rôle est de formaliser la requête d'un utilisateur, selon son profil et selon l'environnement, puis de lui donner un nombre restreint de réponses pertinentes. Nous cherchons à déterminer la meilleure stratégie de médiation (donc une séquence d'interactions) avec un modèle stochastique (MDP) dont les états sont construits sur un référentiel d'attributs et sur la capacité de la source de répondre à la requête en cours de formalisation. Les actions permettent de poser une question à l'utilisateur ou de sonder la source. Les récompenses reflètent la satisfaction de l'utilisateur, la longueur du dialogue et la quantité de résultats. Nous avons implanté un prototype de médiateur utilisant un apprentissage par renforcement (Q-Learning) car il permet une adaptation en ligne, sans modèle a priori.

Mots-Clefs : Médiation, MDP, apprentissage par renforcement.

Abstract:
In the framework of our works on collaboration between heterogeneous agents, we studied the interaction between a human agent and an information search service. We proposed to introduce a mediator agent to formalize the request of a user, according to his/her profile and the environment, then to give a restricted number of relevant answers. We try to determine the best mediation strategy (as a sequence of interactions) with a stochastic model (MDP) whose states are built on a referential of attributes and the capacity of the source to answer the request that is under formalization. The actions allow to ask a question to the user or to probe the source. The rewards reflect the satisfaction of the user, the length of the dialogue and the quantity of results. We have implemented a prototype of mediator using reinforcement learning (Q-Learning) because it allows an on line adaptation, without requiring an a priori model.

Keywords: Mediation, MDP, reinforcement learning.

1 Introduction

L'exemple, illustré [FIG. 1], décrit un problème générique de collaboration entre des agents hétérogènes.

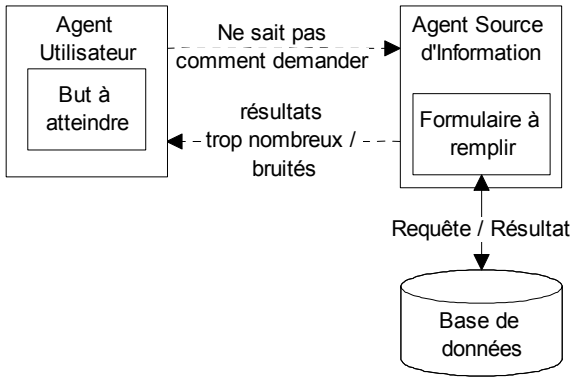

FIG. 1 - Un problème de coopération

Un utilisateur a un besoin d'information et doit interroger un service de recherche, cependant il n'est pas capable de formuler la bonne requête et/ou a des difficultés pour interpréter le résultat correctement. Le problème peut être spécifié de la façon suivante : "Etant donnés deux agents ***A*** et ***B*** et leurs langages de communication respectifs L_A (langue naturelle) et L_B (SQL par exemple), comment faire collaborer ***A*** et ***B*** ?". Comme il n'est pas possible de modifier ***A*** ou ***B***, une solution simple est d'introduire un troisième agent ***M*** que l'on appellera un médiateur. Ce médiateur peut dériver les flux de communications entre ***A*** et ***B*** avec différentes combinaisons, selon le degré d'intervention souhaité. ***M*** peut être un simple observateur du comportement exhibé par ***B*** sur un flux et donner des conseils à ***A***, et vice-versa. Il peut aussi cacher complètement un agent de l'autre. Dans ce cas ***M*** est un assistant pour ***A*** (inversement pour ***B***) car il indique comment répondre au comportement

de ***B*** (inversement pour ***A***) de façon adéquate. Dans notre approche, nous considérons que ***M*** va devoir gérer un dialogue entre ***A*** (l'utilisateur) et ***B*** (la source d'informations) en explicitant les requêtes exprimées par ***A*** dans le langage L_B et réciproquement pour les résultats. Dans cette nouvelle interaction, le médiateur tente de comprendre ce que veut l'utilisateur et cherche à atteindre ce but en utilisant la source d'information. La question à résoudre devient donc : "Comment conçoit-on un tel médiateur ?". Pour répondre, nous nous focalisons sur l'aspect décisionnel de l'agent et sur sa capacité à s'adapter aux agents ***A*** et ***B*** plutôt que sur la forme de l'interaction.

Les travaux sur les systèmes de dialogue homme-machine fournissent une bonne source d'inspiration pour faire coopérer des agents hétérogènes. Leur principal objectif est de permettre une interaction de haut niveau avec les utilisateurs (en langage naturel) afin de réaliser diverses tâches avec un niveau d'interaction plus bas. Cela nous a amené à baser le médiateur autour d'un gestionnaire de dialogue [1]. Actuellement, une voie prometteuse pour gérer les systèmes de dialogues est basée sur les modèles stochastiques [2] comme les Processus de Décision Markoviens (MDP) et les Processus de Décision Markoviens Partiellement Observables (POMDP). Elle est particulièrement intéressante pour construire un médiateur car ces modèles sont bien adaptés pour prendre des décisions en présence d'incertitude sur les perceptions et sur le résultat des actions. De plus, [3] montre que la conception des systèmes de dialogue est équivalente à un problème de MDP et que les solutions des MDP comme l'apprentissage par renforcement peuvent être appliquées pour apprendre des stratégies de dialogue. Ainsi, on dépasse les automates finis classiques qui n'évoluent plus après leur conception.

Notre approche est illustrée sur un problème classique de réservation de vols aériens. Dans cette application, une agence de voyage propose à ses clients des vols entre différents aéroports internationaux. Les clients interagissent avec le système au travers de bornes multimédia vocales. Le système d'information, appelé agent de service, accède à une base de données où les vols sont décrits par leurs aéroports de départ et d'arrivée, la classe de voyage, la compagnie aérienne et la tranche horaire. L'objectif du médiateur est d'aider les clients à trouver le vol qui correspond à leurs besoins et de le réserver. Le médiateur interagit avec les clients et l'agent de service : il construit une requête en posant des questions aux clients sur leur vol et demande les résultats à l'agent de service. Quand des réponses sont disponibles, elles peuvent être transmises au client qui va sélectionner ou rejeter les résultats. Un dialogue typique, où un usager (U) qui veut voyager de Paris à Moscou interagit avec le médiateur (M), est donné [FIG. 2].

M :	Quelle est votre ville de départ ?
U :	Paris
M :	Quelle est votre ville de destination?
U :	Moscou
M :	Il y a trop de réponses à votre demande.
M :	En quelle classe souhaitez-vous voyager?
U :	Je ne sais pas
M :	Voulez-vous voyager en classe économique ?
U :	oui
M :	Il y a deux réponses à votre demande...

FIG. 2 - Exemple de dialogue

2 Spécification du médiateur

Dans cette partie, nous proposons une structure d'agent médiateur et la généralisation des stratégies de médiation inter-agent. La [FIG. 3] présente un diagramme décrivant les modules qui composent le médiateur avec leurs dépendances :

- Le gestionnaire de dialogue est le module central qui envoie et reçoit les messages vers et depuis les autres agents. Il supervise les cycles de discussion et transmet les mises à jour nécessaires au gestionnaire de tâche.

- Le gestionnaire de tâche est responsable de la représentation de l'état de l'interaction et de sa mise à jour. Il est paramétré selon l'application et modélise la tâche à accomplir.

- Le module de décision, basé sur l'algorithme du Q-Learning [4], gère la sélection de la prochaine action à accomplir selon l'état courant abstrait, transmis par le gestionnaire de tâche (voir § 2.2.2). Afin d'améliorer sa

politique, et donc la stratégie de médiation, ce module reçoit également un signal de récompense du gestionnaire de dialogue.

• Le module de profil permet de gérer des connaissances probabilistes sur les différents agents avec lesquels le médiateur interagit.

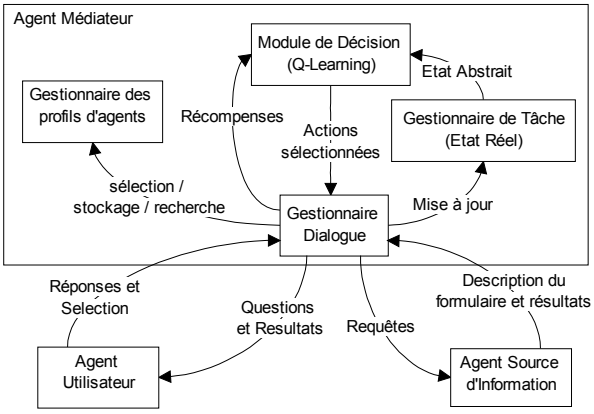

FIG. 3 - Composition de l'agent médiateur

2.1 Modélisation de la tâche

Le service se déroule en sessions pendant lesquelles un utilisateur contacte le médiateur afin que celui-ci lui trouve une information spécifique, appelée un but utilisateur. Le médiateur a pour tâche de découvrir ce but, en communiquant avec cet utilisateur. Pour cela, il doit déterminer comment formuler sa requête auprès de la source d'informations pour trouver les données correspondant à ce but. Le médiateur construit un modèle de la tâche en utilisant une description de la source d'informations. Nous supposons que la spécification de l'application fournit au médiateur un formulaire de requête type pour la source d'informations ou que le médiateur est capable de l'obtenir en interrogeant cette même source. Ce formulaire est ensuite décrit comme un espace multidimensionnel, avec approche similaire à [5] : les différents champs sont utilisés pour fournir un référentiel à cet espace de recherche sous la forme d'un ensemble d'attributs. Chaque attribut est associé à un domaine de valeurs possibles et à un ensemble de questions typiques. Ainsi, valuer les attributs revient à localiser un point dans l'espace généré par le référentiel, ce qui permet de cibler le but de l'utilisateur. Les données de la source d'informations sont supposées être distribuées sur l'ensemble des combinaisons d'attributs possibles. Le médiateur va devoir dialoguer avec l'utilisateur en lui posant une suite de questions et chaque réponse va lui permettre de construire progressivement une requête en déterminant les valeurs des attributs. Tout se passe comme si le médiateur se déplaçait dans l'espace de recherche jusqu'à ce qu'il ait réussi à localiser le but. Comme à chaque position de l'espace de recherche correspond potentiellement un ensemble de données, le médiateur peut sonder la source d'informations pour déterminer s'il y a ou non des données à cet endroit. Cependant, il ne s'agit pas de maximiser la quantité des données retrouvée, mais de ne retourner à l'utilisateur qu'une quantité limitée de données pertinentes.

2.1.1 Définition du Référentiel

Le référentiel permet de positionner dans l'espace multidimensionnel les différents objets manipulés, comme les requêtes du médiateur, les informations de la source ou encore le but de l'utilisateur. Il s'agit d'un système d'axes $\boldsymbol{Attrib = \{a_1, ..., a_m\}}$ où chaque $\boldsymbol{a_i}$ est un attribut du problème. Dans l'exemple de réservation, un vol est caractérisé par une ville de départ, une ville d'arrivée, une classe de voyage, une compagnie et une plage horaire, ce qui donne le référentiel suivant : ***Attrib = {Départ, Arrivée, Classe, Compagnie, Horaire}***

2.1.2 Attributs

Un attribut défini par un tuple $\boldsymbol{<N, D, Q>}$ où $\boldsymbol{N}$ est le nom de l'attribut, où $\boldsymbol{D}$ est le domaine de l'attribut sous la forme d'un ensemble fini de valeurs possibles $\boldsymbol{\{v_1 ; \ldots ; v_k\}}$ et où $\boldsymbol{Q}$ contient des questions qui seront transmises à l'utilisateur. Pour faciliter l'accès aux différents composants d'un attribut, on définit les fonctions ***nom***, ***domaine*** et ***requêtes*** qui retournent respectivement $\boldsymbol{N}$, $\boldsymbol{D}$ et $\boldsymbol{Q}$. Un exemple d'attribut est donné en [Tab. 1]. Afin de constituer l'ensemble des questions à poser, on pourra par exemple utiliser le label du champ correspondant et l'on distinguera plusieurs types de questions :

• des questions de valuation, qui demandent quelle est la valeur de l'attribut (sans suggérer de valeur),

• les questions de proposition, dans lesquelles une valeur est proposée pour l'attribut,

• les questions de confirmation, qui demandent la confirmation d'une valeur d'attribut.

Certaines questions peuvent être paramétrées dynamiquement par une valeur du domaine de l'attribut placé entre les crochets. On pourra, par exemple, utiliser la valeur la plus probable selon le modèle de l'utilisateur ou alors, la valeur qui a été affectée à cet attribut dans une session précédente.

TAB. 1 - Exemple d'attribut pour l'application

Nom	Départ	
Domaine	{Paris, Londres, Rome, Luxembourg, Berlin, Madrid, Genève, Bruxelles}	
Questions	Valuation	Quelle est votre ville de départ ?
	Proposition	Souhaitez vous partir de [ville] ?
	Confirmation	Etes vous sûr de vouloir partir de [ville] ?

2.1.3 Représentation des buts

Soit $Goal = \{g_1, \ldots, g_n\}$, l'ensemble des buts possibles. Ces buts sont positionnés dans le référentiel à l'aide d'une fonction de position : chaque but $g_i \in Goal$, peut être décrit par un vecteur V de coordonnées $[v_{i,1}, \ldots, v_{i,m}]$ où $v_{i,j} \in Domaine(a_j)$, $\forall j \in [1;m]$ correspond à la valeur prise pour l'attribut a_i. On a alors :

$$position(g_i) = [v_{i,1}, \ldots, v_{i,m}]$$

Nous utiliserons la notation suivante :

$$g_i = \{ nom(a_1) = v_{i,1} ; \ldots ; nom(a_m) = v_{i,m} \}$$

Ce qui donne, par exemple :

$$g_i = \{ départ = "Paris"; arrivée = "Tokyo"; classe = "affaire"; compagnie = "Air France"; horaire = "matin" \}$$

Les buts se retrouvent indexés par des points dans le référentiel construit sur les attributs. Un utilisateur peut avoir une idée approximative de la position du but qu'il cherche à atteindre. Aussi, la tâche du médiateur est de dialoguer avec l'utilisateur pour déterminer la meilleure position ou le meilleur sous-espace possible contenant le point but. En cas d'échec, il doit indiquer que le but ne peut pas être identifié ou atteint.

2.1.4 Modélisation des états de la tâche

L'état de la tâche utilise une indexation partielle pour représenter la connaissance que le médiateur a du but utilisateur. Elle reflète aussi la connaissance sur la capacité de réponse de la source pour cette requête. Pour chaque dimension j du référentiel $\{a_1; \ldots; a_m\}$, nous utilisons un couple (ea_j, val_j). La variable d'état ea_j prend ses valeurs dans l'ensemble $Ea = \{Ouvert, Affecté, Fermé\}$ et la seconde variable $val_j \in Domaine(a_i)$ stocke la valeur correspondante, si elle existe (quand ea_j est affecté). Dans [Tab. 2], on retrouve les différents cas pouvant être rencontrés. L'état complet de la tâche, va mémoriser :

• pour chaque attribut, les couples (ea_i, val_j) contenant l'état d'affectation, ainsi que la valeur qui est affectée (ou nul s'il n'y en a pas).

• l'ensemble ***Rep*** des résultats de la dernière réponse de la source d'informations, s'il y en avait, afin de pouvoir les transmettre à l'utilisateur.

TAB. 2 - Description de l'état des attributs

Valeur	***Etat de l'attribut***
Ouvert	L'information sur cet attribut n'a pas encore été demandée à l'utilisateur et la dimension correspondante n'est pas contrainte. La notation "***?***" indiquera que la valeur val_j est encore indéterminée.
Affecté	Cet état, marqué par un "***A***" indique que l'attribut a été instancié au cours de l'interaction avec une valeur val_j.
Fermé	Une question a déjà été posée mais le médiateur n'a pas reçu de réponse utilisable. Par exemple, l'utilisateur a dit "*je ne sais pas*". Cet état sera noté par un "***F***" qui signifie que la valeur val_j restera inconnue, mais aussi qu'une question a déjà été posée.

L'état ***s*** de la tâche prendra la forme suivante :

$$s = \{ (ea_1 , val_1) ; \ldots ; (ea_m , val_m) ; Rep\}$$

La taille de l'espace d'état est très grande car il faut non seulement considérer toutes les combinaisons de valeurs d'attributs possibles, mais aussi les résultats qui dépendent de la source d'informations. L'ensemble ***Rep*** sera limité aux nr_{max} premières réponses, où nr_{max} est le nombre maximum de réponses permises. Comme nous l'expliquons plus loin, seule une abstraction de l'état de la tâche sera utilisée par le module de décision.

2.2 Module de décision à base de MDP

Formuler le comportement de l'agent médiateur en terme de Processus de Décision Markovien requiert de définir un espace d'état ainsi que les actions possibles. Comme elles dépendent de la réponse de l'utilisateur, les transitions du modèle ne peuvent pas être données, mais sont apprises indirectement par renforcement. La solution est obtenue sous la forme d'une politique qui indique pour chaque état, quelle action sélectionner. Dans les systèmes de dialogue, les politiques MDP peuvent être interprétées comme des stratégies de dialogue [3] et elles dictent quelles sont les meilleures questions à poser à l'utilisateur avec un coût minimal. Dans notre approche, résoudre le problème revient à atteindre un état où le but de l'utilisateur est suffisamment reconnu et à recevoir de l'utilisateur une sélection valide dans une liste de suggestions. Ainsi, l'état du processus de décision sera une abstraction de l'état de la tâche et les actions sont définies à partir des questions attachées à un attribut donné, sous la forme de requêtes transmises à l'utilisateur. Des actions, dirigées vers un autre agent du système sont également possibles, comme par exemple une requête transmise à la source d'informations. Afin de quantifier l'efficacité et la qualité du comportement de l'assistant et de comparer différentes politiques, il est également nécessaire de définir des récompenses données au médiateur. Les récompenses peuvent être des valeurs positives ou négatives selon la satisfaction finale de l'utilisateur, la longueur de l'interaction, le coût d'utilisation des ressources, etc. Pour mesurer la satisfaction de l'utilisateur, divers indices peuvent être exploités. Le plus simple est d'utiliser une notation explicite à la fin de l'interaction, mais cela peut être aussi le résultat d'un calcul sur l'état final (prenant par exemple en compte le nombre d'objets retournés par une requête). Nous avons choisi de donner une récompense positive quand l'utilisateur sélectionne une réponse et une récompense négative quand il ne le fait pas ou que le dialogue est trop long.

2.2.1 Etude de la taille du problème

Le problème considéré est la combinaison d'une partie ***U***, dépendante de l'utilisateur et de sa requête et d'une partie ***I***, dépendante de la source d'informations. L'espace d'états ***S*** peut se décomposer en $S = S_U \times S_I$. De la même façon, la fonction de récompense $r = f(r_U, r_I)$ est composée d'une partie venant du dialogue avec l'utilisateur et d'une autre partie venant de l'interaction avec la source d'informations. Il en est de même pour l'ensemble des actions possibles $A = A_U \times A_I$. L'espace S_U correspond à l'ensemble des requêtes partielles possibles. Chaque attribut de la requête peut prendre son état d'affectation dans l'ensemble ***{'?' ; 'F' ; 'A'}***. Quand un attribut est affecté, on lui associe de plus une valeur et considérant que l'on a en moyenne ***i*** valeurs possibles pour un attribut, cela donne ***2+i*** états possibles. Soit ***m*** le nombre d'attributs, la taille de l'espace S_U est donc $Card(S_U) = (2+i)^m$. L'espace des résultats S_I représente l'ensemble des réponses possibles de la source d'informations à une requête. Il s'agit donc de l'ensemble des parties de l'ensemble de toutes les entrées de la source d'informations, soit $\sum_{i=0}^{n} C_n^i = 2^n$. La taille de l'espace d'état est donc :

$$Card(S) = Card(S_U) \times Card(S_I) = (2+i)^m 2^n$$

où ***i*** est le nombre moyen de valeurs par attribut, où ***m*** est le nombre d'attributs et où ***n*** est le nombre d'éléments de la base.

Sur l'application de voyage aérien, avec ***8*** villes de départ, ***6*** villes d'arrivée, ***3*** classes de voyage, ***8*** compagnies aériennes et ***4*** créneaux horaires, il y a potentiellement ***4608*** vols possibles. Même si la base ne contient que

10% de ces vols, soit ***461***, l'ensemble des réponses possibles à une requête atteint tout de même 2^{461}. Par ailleurs, en considérant en moyenne ***6*** valeurs par attribut, le nombre de requêtes partielles possible est $(2+6)^5 = 8^5 = 32768$. La taille de l'espace d'état complet est donc de $8^5 \times 2^{461} = 2^{476}$, soit $1{,}9.10^{143}$ états.

2.2.2 Abstraction des états

Le processus de décision doit travailler sur une représentation réduite de l'espace d'état. En effet, les états de la tâche ne peuvent pas être donnés tels quels au processus de décision, à cause de leur surnombre. La solution la plus simple pour réduire la complexité est de faire abstraction de la valeur de l'attribut. Ainsi, l'état ***(ea, val)*** de chaque attribut sera simplifié en ne gardant que l'état d'affectation ***ea***, ce qui donne, par exemple, ***A*** au lieu de ***(A ; valeur***). Les résultats de la requête passée sont également abstraits en appliquant deux seuils au nombre ***nr*** de réponses : un seuil minimal, fixé à une réponse et un seuil maximal nr_{max}. Ainsi, seule une mesure de qualité des résultats ***qr*** sera utilisée, comme indiqué [Tab. 3]. Les états abstraits transmis au module de décision seront notés avec la forme suivante :

$$< ea_1 \; ; \ldots ; ea_m \mid qr >$$

Comme chaque $\mathbf{ea}_i$ prend une valeur dans ***('?', 'A' , 'F')*** et que ***qr*** prend une valeur dans ***('?', '0', '+', '*')***, on obtient une taille de 4×3^m, soit ***108*** états pour ***3*** attributs et ***972*** pour ***5*** attributs.

TAB. 3 - Qualité de la réponse

Nombre de réponses	*Qualité réponse*
Requête non exécutée	$qr = '?'$
Aucune : $nr = 0$	$qr = ' 0 '$
Suffisant : $nr \in [1; nr_{max}]$	$qr = ' + '$
Trop : $nr \in] nr_{max} ; +\infty [$	$qr = ' * '$

2.2.3 Actions

La liste des actions possibles du médiateur est donnée [Tab. 4]. Il y a trois actions par attribut (valuer, proposer et confirmer), mais aussi deux autres actions (une pour la requête vers la source d'informations et une autre pour demander à l'utilisateur d'effectuer une sélection). Dans le cadre des MDP, il est tout à fait possible de modifier l'ensemble des actions réalisables selon l'état, aussi la colonne "conditions" indique à quel moment une action peut être choisie.

TAB. 4 - Actions possibles

Code	*Conditions*	*Description de l'action*
value	$ea_s = '?'$	Demander à l'utilisateur la valeur de l'attribut a_s.
propose	$ea_s \neq 'A'$	Proposer à l'utilisateur une valeur pour a_s.
confirm	$ea_s = 'A'$	Demander à l'utilisateur de confirmer la valeur de a_s.
rqte	$qr = '?'$	Envoyer une requête vers la source d'informations.
select	$qr = '+'$ ou $'*'$ si la tâche est pleinement contrainte	Demander à l'utilisateur la sélection d'une entrée dans la liste des résultats.

2.2.4 Récompenses

Le gestionnaire de dialogue donne les récompenses au module de décision, donc au MDP. Ces récompenses proviennent de chacune des interactions. Elles peuvent venir de l'utilisateur, quand il sélectionne ou qu'il refuse un résultat, s'il abandonne la session en cours ou quand l'interaction est trop longue (t_{max} interactions). Une récompense peut aussi être obtenue lorsque le médiateur interagit avec la source de données et qu'il y a trop de réponses.

La fonction de récompense r_t obtenue au temps t dans l'état s_t est initialisée à 0 puis elle est modifiée au fur et à mesure des entrées perçues. Les valeurs données [Tab. 5] peuvent venir s'accumuler quand arrive une réponse utilisateur ou une réponse de la source d'informations. Le problème est que seul l'utilisateur connaît g_U et c'est donc à lui que revient la vérification de la satisfaction globale de la tâche. Nous avons ajouté d'autres récompenses, mais elles ne sont pas obligatoires. En effet, une requête qui échoue, à cause d'un surnombre de réponses, d'une absence de réponse ou d'un refus de sélection

pénalise la séquence d'actions en terme de "temps" dépensé alors qu'il aurait été préférable de sélectionner une autre action.

TAB. 5 - Récompenses obtenues

Récompense (Valeur)	***Condition d'attribution***
$R_{selection}$ (+5)	Le module de décision a choisi un ***select*** et l'utilisateur a sélectionné dans l'ensemble des réponses $Rep(s_i)$ un élément ***r*** qui satisfait son but.
R_{absrep} (1)	Il n'y a pas de réponses alors que la tâche est totalement contrainte. Le dialogue se termine car le système a rempli son rôle, mais c'est la source qui n'a pas donné de résultat.
$R_{surnombre}$ (-1)	La source d'informations a retourné trop de réponses ($qr = '*'$).
$R_{troplong}$ (-4)	Il s'est déroulé plus d'interactions que la limite fixée.
R_{refus} (-5)	Le module de décision a choisi un ***select*** et l'utilisateur a refusé l'ensemble des réponses $Rep(s_i)$.

2.2.5 Apprentissage par renforcement

Le principe de l'apprentissage par renforcement est de laisser le système apprendre le meilleur comportement, de sorte à optimiser la récompense espérée à plus ou moins long terme (selon un coefficient d'atténuation γ). Une introduction à l'apprentissage par renforcement peut être trouvée dans [6]. L'avantage cette approche est de ne pas nécessiter une connaissance complète du modèle sous-jacent (comme les tables de transition). Nous utilisons un algorithme classique : le Q-Learning de Watkins [4] qui est qui tente d'apprendre une table contenant la récompense attendue pour réaliser une action donnée dans un état donné. Ce choix est motivé par le fait que le Q-Learning peut être utilisé en ligne, ce qui permet au gestionnaire de dialogue de continuer son apprentissage durant les différentes sessions.

2.3 Gestion du dialogue

Le rôle du gestionnaire de dialogue est de gérer les tours de parole et d'envoyer les mises à jour au gestionnaire de tâche en fonction des entrées.

2.3.1 Etat de départ

L'état de départ va dépendre de la connaissance que le médiateur a de l'utilisateur. Si l'on ne connaît rien de lui a priori, il va partir d'une position où le but est ouvert. Il s'agira typiquement de l'état $<?, \ldots, ? \mid ?>$. Par contre, s'il y a déjà eu des sessions passées, il peut être intéressant de considérer les états finaux de celles-ci, afin de trouver un point plus astucieux à partir duquel il est possible de redémarrer directement.

Examinons de plus près ce problème : un nouvel utilisateur se présente au temps $t = 0$ et commence une première session. Le médiateur tente de se positionner en fonction des besoins de cet utilisateur dans l'espace de requête en $U_0 = < u_{0,0}, \ldots, u_{n,0} >$. Le système va poser une série de questions afin d'atteindre ce point. A la fin du service, l'utilisateur n'interagit plus avec le médiateur et celui-ci peut mémoriser ce dernier emplacement avant de passer à une toute autre session. Lorsque le même utilisateur revient plus tard dans une nouvelle session, ses besoins sont maintenant au point $U_1 = < u_{0,1}, \ldots, u_{n,1} >$ et le médiateur doit alors se rendre maintenant à ce point U_1, et ainsi de suite...

Sur l'exemple de voyage aérien, un usager peut demander de Paris à Tokyo en classe économique, puis revenir ensuite dans une autre session et partir de Tokyo vers Istanbul en classe économique. On peut se dire que cet usager voyage souvent en classe économique et la lui proposer directement quand il revient... on peut aussi se dire qu'il repart de là où il est arrivé la dernière fois et lui suggérer un vol au départ d'Istanbul.

Nous disposons donc là d'une piste pour modéliser le comportement de l'utilisateur : une loi qui relierait la requête U_t à la ou les précédentes $U_0, \ldots, U_{t-1}$. Dans l'absolu, connaître cette loi permettrait de s'adapter en

se préparant à ses requêtes futures et ainsi d'optimiser la recherche du but. La loi d'évolution des requêtes dépend de l'utilisateur, mais elle peut également dépendre de l'application. Une idée est donc de chercher à apprendre cette loi, en s'appuyant éventuellement sur des connaissances additionnelles.

2.3.2 Evolution de la tâche

TAB. 6 - Observations possibles

Code	***Action contexte***	***Perception***
val	value	Valeur d'attribut valide donnée par l'utilisateur
oui / non	propose, confirm	Réponse de l'utilisateur à une demande simple
table	rqte	Résultats donnés par la source d'informations
numéro / refus	select	L'utilisateur a sélectionné / refusé une proposition
stop	value, propose, confirm, select	Arrêt par l'utilisateur
nsp	value, propose, confirm, select	L'utilisateur indique qu'il ne sait pas répondre
aberrant	value, propose, confirm, select	Perception aberrante ou inattendue

Le gestionnaire de dialogue met à jour la tâche qui évolue en fonction des actions du médiateur et de leurs effets sur l'environnement. Deux types d'évolutions sont possibles. D'une part, le système peut s'adresser à l'utilisateur et recevoir la réponse (en posant une question sur une valeur d'attribut ou en fournissant une sélection de résultats) et d'autre part, il peut envoyer une requête vers la source d'informations et obtenir les résultats. Diverses sources d'incertitude peuvent apparaître à ce point. Un premier aspect vient des considérations des ressources et des médias, comme la qualité de la transmission : si une reconnaissance de parole est utilisée, les mots peuvent être perçus avec plus ou moins de certitude et la réponse peut être altérée, selon les performances du moteur utilisé. Une autre source importante d'incertitude est l'utilisateur, car il peut ne pas comprendre une question et donner une réponse inadaptée. Dans [Tab. 6], nous donnons un ensemble de perceptions possibles du médiateur. On laisse à l'utilisateur la possibilité d'indiquer qu'il ne connaît pas de réponse à la question posée (on notera cette perception par '***nsp***'). Les cas concrets dans lesquels les états des attributs évoluent sont détaillés [Tab. 7], où les entrées aberrantes sont ignorées.

TAB. 7 - Mise à jour de l'état réel des attributs

$(ea, val)_t$	Act_t	$Percept_t$	$(ea, val)_{t+1}$
?	propose [v]	nsp	F
?	propose [v]	non	?
?	propose [v]	oui	A; v
?	value	nsp	F
?	value	v	A; v
A, v	confirm	nsp	F
A, v	confirm	non	?
A, v	confirm	oui	A; v
F	propose [v]	non/nsp	F
F	propose [v]	oui	A; v
F	value	nsp	F
F	value	v	A; v

En ce qui concerne l'action de requête ***rqte***, elle peut être sélectionnée à partir de n'importe quel état et la table des résultats est mise à jour en fonction des réponses. Le résultat modifie l'état courant au niveau de la qualification de la dernière requête. Ainsi, on passera par exemple d'un état ***<?, ?, ? | ?>*** à l'état ***<?, ?, ? | *>*** si on fait une requête alors que l'on ne connaît rien et que la base de données retourne logiquement une grande quantité de résultats. A l'inverse, une requête fortement contrainte effectuée depuis un état ***<A, A, A | ?>*** peut ne pas donner de résultats. Dans ce dernier cas l'état devient alors ***<A, A, A | 0>***. Il faut noter par ailleurs que toute transition modifiant l'état des attributs fait repasser la qualité de requête dans l'état inconnu (***?***). A chaque fois que l'état comporte une qualité de requête positive (+), il est alors possible de demander la sélection dans la liste à l'utilisateur grâce à l'action ***select***. Dans ce cas, deux cas de figure peuvent se présenter.

Dans le premier cas, on reçoit un numéro sélectionné, alors la tâche est accomplie et le dialogue s'arrête en stockant l'état courant pour l'utilisateur. Dans le second cas, on reçoit un refus de sélection et la zone de réponses est vidée, ce qui a pour effet de faire repasser la qualité de requête à ***0***. Un dernier type de transition existe : lorsque l'utilisateur a demandé l'arrêt du service. Si cela se produit, le service s'arrête après avoir mémorisé l'état courant pour cet utilisateur.

2.4 Gestion des profils utilisateurs

De sorte à prendre en compte les préférences des utilisateurs, le médiateur contient également un module qui gère les profils utilisateurs. Chaque utilisateur est représenté avec un ensemble de données et un profil par défaut peut être utilisé pour les nouveaux utilisateurs.

Les profils utilisateurs ont été conçus pour stocker les préférences comme des distributions de probabilités $p(v \mid u, a)$ sur les valeurs v de chaque attribut a et chaque utilisateur u. La distribution de probabilité est initialisée avec une loi uniforme et mise à jour quand l'utilisateur sélectionne une valeur. Ce module de préférence utilisateur permet de donner une valeur dans le cas où l'on formulerait une question de proposition. Par exemple, un utilisateur qui a voyagé deux fois en classe affaire, une fois en classe normale et sept fois en classe économique va plus probablement voyager en classe économique la prochaine fois. La sélection d'une question de proposition sur l'attribut de classe donnera donc : "*Voulez-vous voyager en classe économique ?*".

3 Description du prototype

Un prototype basé sur la spécification du médiateur a été implanté en Java et testé sur l'application de réservation de vol aérien. Le prototype est construit autour d'un corps générique d'agent appelé SmallMu qui gère les ressources avec une pompe action/évènement. Il utilise des ressources génériques de reconnaissance et de synthèse de parole basées sur le système vocal *IBM ViaVoice* au travers de *Java Speech API*. Le "cerveau" ressource du médiateur est composé du gestionnaire de dialogue lié au gestionnaire de tâche et au module de modélisation des utilisateurs. Il contient également le module de décision qui utilise l'algorithme du Q-Learning. Le prototype est relativement générique car il peut être paramétré en fonction du problème donné, à condition d'avoir le formulaire de tâche, les grammaires et les prompts correspondants. La structure de la tâche est une frame décrivant les différents champs du formulaire de requête et ses diverses caractéristiques. Une fois la frame analysée, le gestionnaire de tâche construit le référentiel d'attributs, en utilisant une bibliothèque spécialisée dans la cartographie d'objets et initialise l'état de la tâche à ***<?, ... ,? | ?>***. Le module de décision sélectionne l'action à accomplir. Si une question doit être posée, alors les commandes sont envoyées à la pompe d'action et transmises aux moteurs vocaux : le moteur de synthèse reçoit les prompts à générer et respectivement, le moteur de reconnaissance reçoit les règles de grammaire à activer. Quand une action mettant en œuvre la source d'information est sélectionnée, la requête est formée et envoyée à la ressource de gestion de données qui la transmet à la source de données. Quand une réponse est obtenue ou quand des résultats de la source sont disponibles, l'entrée est retournée dans la pompe à événements vers le gestionnaire de tâche qui met à jour l'état.

4 Résultats expérimentaux

De façon à entraîner le module de décision du médiateur, un simulateur a été construit avec un comportement de réponse naïf aux questions. Ce simulateur sélectionne un but et répond aux questions du médiateur avec des perturbations aléatoires. Les graphes [FIG. 4] et [FIG. 5] ont été obtenus pour un dialogue à 3 attributs, une durée maximale de 20 interactions. Le premier montre l'évolution du pourcentage de dialogues réussis (terminés par une sélection ou par une absence de résultat) selon le nombre d'itérations. Le second montre l'évolution de la longueur moyenne d'un dialogue selon le nombre d'itérations.

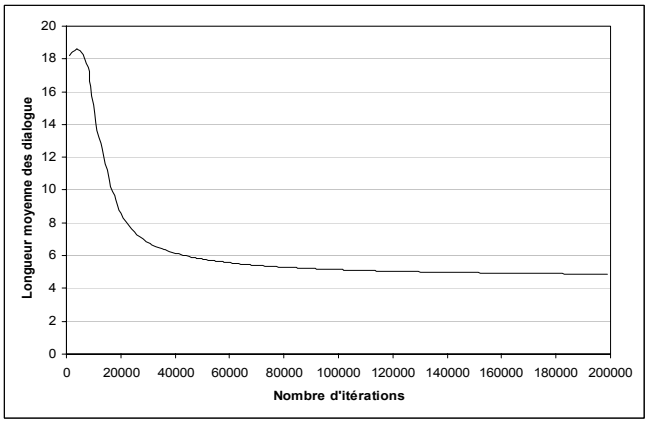

FIG. 4 - Evolution de la longueur du dialogue

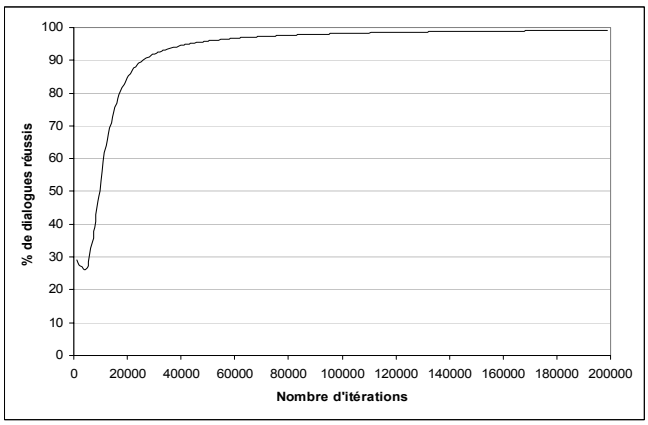

FIG. 5 - Evolution du nombre de succès

On constate qu'un nombre important d'interactions est nécessaire pour obtenir une politique acceptable : Choisir un attribut et le faire valuer. On voit que les dialogues arrivent en moyenne à 5 interactions (soit 25% de la limite haute), ce à quoi on pouvait s'attendre. En effet, il faut au moins une question pour valuer un attribut (donc trois dans ce cas précis), une action pour la requête vers la source d'informations et une pour demander la sélection. Pour dépasser la limite basse, une idée est d'utiliser les dépendances entre les attributs. Par ailleurs, le nombre d'itérations nécessaires rend l'apprentissage très difficile en dehors d'une simulation. Pour produire un comportement plus intéressant, le médiateur doit être entraîné et de meilleurs résultats sont attendus d'un simulateur basé sur des traces d'interactions réelles puis l'approche sera validée sur des applications, comme la gestion automatisée d'un portefeuille d'actions en collaboration avec la société Dialoca.

5 Conclusion

Introduire un agent médiateur permet de pallier l'hétérogénéité de la coopération entre agents. De plus, cette solution est applicable, même si les autres agents ne sont pas contrôlables puisque le médiateur apprend à interagir avec eux et non l'inverse. Considérer l'interaction multi-agent comme si c'était un dialogue permet d'établir un parallèle avec les systèmes de dialogue et de transposer les approches à base de MDP existantes et les solutions qui sont utilisées dans ce domaine. Nos expérimentations ont montré que les méthodes d'apprentissage par renforcement pouvaient être appliquées pour apprendre une stratégie de médiation. Les travaux futurs seront orientés pour s'attaquer au problème de représentation de la tâche et viseront à réduire la quantité de données nécessaire à l'apprentissage.

Références

[1] Charton R., Boyer A. et Charpillet F, Providing users with adapted services: Dynamic building of dialogues to make heterogeneous agents cooperate. ISSPIT 2002, Marrakech, 2002.

[2] Young S., Probabilistic Methods in Spoken Dialog Systems. Royal Society, Londres, 1999.

[3] Levin E, Pieraccini R. et Eckert W. Using Markov Decision Process for Learning Dialogue Strategies. ICASSP'98, Seattle, 1998.

[4] Watkins C., Learning from Delayed Rewards. PhD Thesis, King's College, Cambridge, 1989.

[5] Goddeau D., Meng H., Polifroni J., Seneff S. et Busayapongchaiy S., A Form-Based Dialogue Manager For Spoken Language Applications. ICSLP'96, Philadelphie, 1996.

[6] Sutton R. S. et Barto A. G. Reinforcement Learning: An Introduction. MIT Press Cambridge MA, 1998.