Master 2 Recherche en informatique (IFSIC)

— Rapport de stage —

Comportements adaptatifs et motivés d'entités autonomes par systèmes de classeurs hiérarchiques

Ewen Creac'h

Encadrant :
Cédric Buche (buche@enib.fr)

Au laboratoire :
Laboratoire d'Informatique des Systèmes Complexes (LISyC, EA 3883)
Centre Européen de Réalité Virtuelle (CERV)
Equipe Atelier de Réalité Virtuelle (ARéVi)

11 juin 2007

Table des matières

— Introduction —

Aujourd'hui, de nombreux domaines scientifiques s'intéressent à la modélisation et à la simulation de *comportements complexes* et crédibles afin de peupler des environnements virtuels d'*entités autonomes*. Si l'on prend l'exemple d'une application de réalité virtuelle, le seul rendu réaliste des graphismes est insuffisant pour assurer la cohérence de l'environnement (Tisseau, 2001). Par conséquent, l'utilisateur du système de réalité virtuelle doit se trouver face à des entités avec des comportements correspondant à leur représentation (humain, chien, oiseau...) et qui répondent de manière rationnelle aux interactions de l'utilisateur. On peut aussi vouloir étudier les dynamiques d'un environnement peuplé d'entités autonomes (écosystème, foules...) en fonction d'évènements inattendus et non définis *a priori*.

De nombreuses solutions et architectures ont été développées et permettent de réaliser des comportements plus ou moins complexes. Cependant la notion d'*adaptation* aux changements de l'environnement en temps réel (apprentissage en ligne) n'est que très partiellement abordée. Différents types d'apprentissages en environnement dynamique peuvent être envisagés : imitation (Del Bimbo et Vicario, 1995), essais-erreurs (Sutton, 1991)...

Une idée serait qu'une entité dispose de *capacités de base* (telle que la marche, la saisie d'objet, l'observation...) et apprenne de manière autonome en évoluant dans l'*environnement* en fonction de ses *motivations* (la faim, la peur, l'agressivité...). Ceci permet, par analyse des conséquences sur l'environnement, de retenir les actions favorables en fonction des motivations, et d'inhiber celles qui sont inutiles (cf. figure 1). De plus, cette approche offre aux entités une *auto-adaptation* dans des environnements dynamiques (une action devenue inutile disparaît). Nous allons donc être amenés à mettre en place un système d'auto-apprentissage pour représenter ce phénomène.

Dans un premier chapitre, nous présentons les principales approches existantes pour réaliser des comportements. Ensuite, le besoin d'auto-adaptation étant primordial, nous décrivons le concept d'*apprentissage artificiel* ainsi que les types et paradigmes d'apprentissage existants, afin de rechercher ce qui correspond le mieux à nos besoins. Nous abordons ensuite les *systèmes de classeurs* qui offrent un mécanisme (à base de règles « condition => action ») de choix de décision en fonction de l'environnement et des états internes de l'entité qui permet de réaliser des comportements adaptatifs. Cependant, les limitations des systèmes de classeurs

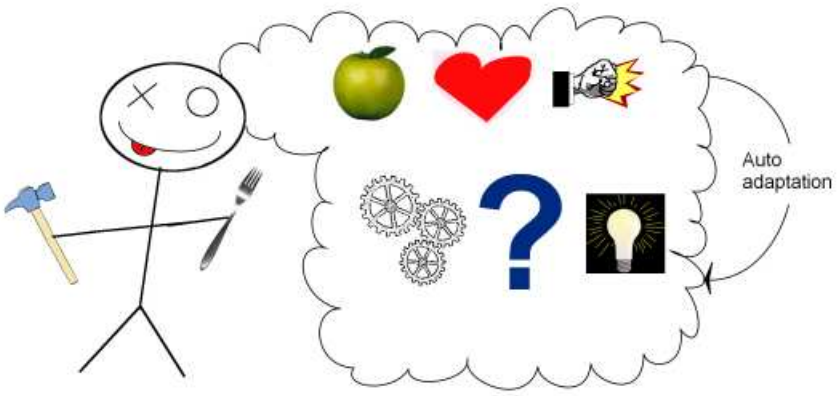

Fig. 1 – Une entité essaye de satisfaire des motivations (faim=« pomme », santé=« coeur », agressivité=« poing ») par un comportement complexe, auto-adaptatif, mais comment ?

classiques dans le cadre de comportements complexes nous amène à aborder le problème de la *modélisation de comportements complexes* par l'utilisation des systèmes de classeurs *hétérogènes* ou introduisant la notion de *hiérarchie*. Nous concluons cette première partie par une analyse concernant les hiérarchies de systèmes de classeurs, celle-ci nous a permis de déceler des problèmes fondamentaux concernant l'utilisation des hiérarchies de systèmes de classeurs, ce qui nous a amené à étudier une autre approche hiérarchique des systèmes de classeurs : les systèmes de classeurs hiérarchiques.

Le second chapitre présente l'architecture générale que nous avons proposée afin de satisfaire nos besoins (multi-motivations, réalisation de plans...). Nous décrivons son fonctionnement global pour ensuite rentrer dans le détail de chacun des éléments qui compose cette architecture (motivations, systèmes de classeurs hiérarchiques, mécanisme de décision, modes de fonctionnement...). Nous abordons ensuite la problématique de l'apprentissage au sein de notre architecture et plus particulièrement au sein des systèmes de classeurs hiérarchiques. De plus, nous avons réfléchi au problème de l'apprentissage au sein de notre architecture concernant l'évaluation des éléments de l'environnement mais aussi au niveau des systèmes de classeurs hiérarchiques et de la génération partielle de plans. Nous présentons donc nos résultats sur ce point.

Nous concluons en rappelant les résultats obtenus concernant le fonctionnement de notre architecture et sur l'apprentissage. Puis nous envisageons des perspectives de recherches futures que nous présentons.

Chapitre 1

— Problématique —

Ce premier chapitre s'intéresse à la problématique de la modélisation de comportements. Dans un premier temps, il présente les principales approches existantes dans ce domaine (cf. section 1.1). La question de l'apprentissage, indispensable à la mise en place d'une certaine autonomie de l'entité, est également abordée (cf. section 1.2). Puis une présentation des systèmes de classeurs permet de voir les avantages de cette approche pour modéliser des comportements (cf. section 1.3). Cependant, dans le cadre de comportements complexes les systèmes de classeurs classiques sont rapidement limités et l'utilisation de systèmes de classeurs particuliers peut apporter une piste de solution (cf. section 1.4). Nous concluons ce chapitre par une analyse des hiérarchies de systèmes de classeurs qui nous permet de prendre conscience de problèmes conceptuels les concernant (cf. section 1.5).

1.1 Architectures et modèles comportementaux

L'étude des comportements complexes est abordée par de nombreuses disciplines telle que la psychologie, le neurophysiologie, l'ergonomie, l'intelligence artificielle, la sociologie... La psychologie, dont le sujet principal est l'étude des comportements humains, propose de nombreux modèles et théories[1] dans le but d'en mieux comprendre les mécanismes tels que le langage, la mémoire, la perception, le contrôle moteur, les émotions et le raisonnement humain... Les informaticiens se sont inspirés de ces modèles et théories afin de tenter de réaliser des comportements.

1.1.1 Différents types d'agents, différents types de comportements

Dans le domaine de la modélisation de comportement, on peut considérer deux grands types : les comportements *réactifs* et les comportements *cognitifs*. Dans les deux cas, le cycle

[1] Donikian (2004) fait une présentation de ces théories et modèles psychologiques.

de fonctionnement d'une entité est celui-ci : *perception-décision-action*. Un agent réactif va uniquement prendre des décisions à partir de l'état du monde et de son état, sans mémoire du passé. Un agent cognitif va en plus tenir compte de ses buts et ambitions personnelles, afin de mettre en place un plan cherchant à les atteindre. Des approches *hybrides* tentent d'associer les deux points de vue afin d'offrir aux agents des comportements cognitifs, mais aussi des actions réactives de l'ordre du réflexe.

1.1.2 Méthodes existantes

Plusieurs techniques et modèles ont été proposés afin de réaliser des comportements plus ou moins réactifs et/ou cognitifs. Nous présentons ici les plus utilisés, qui sont issues de la recherche en réalité virtuelle, systèmes multi-agents, simulation comportementale et de l'approche animat[2] :

- ▷ Les *automates d'états finis* sont fréquements utilisés pour modéliser des comportements. Ils peuvent être mis en œuvre seuls ou combinés entre eux au sein d'une architecture hiérarchique et de piles d'automates afin de complexifier et/ou paralléliser les comportements (Brooks, 1986; Maes, 1991; Donikian et Rutten, 1995; Lamarche, 2004). Les automates permettent une *bonne interprétation* du fonctionnement grâce à un mécanisme de base simple (état, transition). Cependant, une modification du comportement est très difficilement réalisable et nécessite de profond changements dans le(s) automate(s). La réalisation de comportements adaptatifs à partir d'automates est donc très complexe.

- ▷ Les *réseaux de neurones artificiels* sont utilisés pour réaliser des comportements réactifs, voire réflexes (Van de Panne et Fiume, 1993). Les canaux sensoriels sont reliés aux effecteurs *via* le réseau de neurones. Malgré la *bonne capacité d'apprentissage* que présentent les réseaux de neurones, le *manque d'abstraction et d'interprétabilité* du fonctionnement du système pose problème pour la description ou l'analyse de comportements par un expert.

- ▷ Des *systèmes à base de règles* (réactives (Reynolds, 1987), floues (Coquelle et al., 2004), logiques...) permettent de réaliser des comportements. Les principaux avantages des systèmes à base de règles sont leur *modularité* et leur *interprétabilité*. Une règle peut être créée, détruite ou transformée sans interférence avec les autres. De plus toutes ces règles sont facilement compréhensibles par un expert.

- ▷ Des architectures cognitives, principalement basées sur des *logiques formelles* et des modèles psychologiques, essayent d'intégrer la notion de raisonnement par le biais de mécanismes d'*inférences* (chaînage avant : des faits vers les buts et/ou chaînage arrière : des buts vers les conditions nécessaires). Pour cela, ces architectures reposent en général sur une mémoire à long terme (procédurale) qui permet de stocker les règles permettant d'inférer, et une mémoire à court terme (de travail) qui contient l'état de l'entité et les faits avérés à l'instant présent. De nombreuses architectures sont basées sur ce modèle avec de nombreuses variantes : SOAR (Laird et al., 1987), ACT-R

[2] cf. conférence internationale biannuelle *« From animals to animats »* .

(Anderson et Lebiere, 1998), BCOOL (Lamarche, 2004), C4 (Isla et al., 2001)... De plus, d'autres architectures de ce type sont issues du monde des systèmes multi-agents : les BDI (Beliefs Desires Intentions) (Bratman, 1987) sont basés sur les croyances de l'entité sur l'état du monde, ses désirs (buts), ses intentions (plans à exécuter) ; PECS (Schmidt, 2000) et BRAHMS (Clancey et al., 1996) offrent une vision plus sociale en intégrant la notion de groupes. L'avantage de cette approche est la réalisation de comportements cognitifs complexes pour la résolution de problèmes, cependant l'apprentissage et l'adaptation aux changements de l'environnement ne sont que très peu abordés.

1.1.3 Bilan

Cette présentation, non exhaustive et très simplifiée, permet de situer différentes approches possibles pour aborder le problème de la modélisation de comportements plus ou moins complexes. De plus, ces approches, offrant des points de vue différents, sont *combinables* afin de garder les atouts de chacune. Il est donc possible d'allier comportements réactifs et cognitifs dans le cadre d'une entité hybride, capable d'effectuer un raisonnement et des tâches complexes, mais répondant de manière instinctive à des stimulus. Le problème de l'apprentissage en ligne de nouvelles solutions et de l'adaptation à des situations inconnues n'est que très partiellement abordé par les architectures actuelles. Cependant, cette notion d'*apprentissage en ligne*, bien qu'extrêmement complexe, est fondamentale dans le cadre d'*environnements dynamiques* où le comportement n'est pas entièrement descriptible *a priori*.

1.2 Apprentissage artificiel

« L'apprentissage artificiel englobe toute méthode permettant de construire un modèle de la réalité à partir de données, soit en améliorant un modèle partiel ou moins général, soit en créant complètement le modèle ». Cette définition donnée par Miclet et Cornuéjols (2002) donne un cadre très large à l'apprentissage artificiel et correspond à notre idée qui est de réaliser un modèle de comportement complexe pour une entité. Afin de mieux comprendre cette notion, nous sommes appelés à présenter les deux types d'apprentissages en informatique (symboliques ou numériques cf. 1.2.1), ainsi que les trois paradigmes (protocoles) qui permettent d'effectuer un apprentissage artificiel (cf. 1.2.2).

1.2.1 Types d'apprentissage

On peut distinguer deux types d'apprentissage artificiel, caractérisés par le type des données utilisées :

L'apprentissage symbolique

L'apprentissage symbolique, comme son nom l'indique, manipule des *symboles*. Il fonctionne grâce à la mise en place de *relations* entre ces symboles par le biais de *jugements*. L'idée est donc d'élaborer des méthodes permettant d'extraire des connaissances structurelles ou décisionnelles à partir d'instances peu structurées. L'avantage principal de l'apprentissage symbolique est sa *portée sémantique forte*. Un expert qui analyse le système apprenant va pouvoir comprendre la façon dont fonctionne celui-ci (par exemple un système à base de règles « si => alors » est compréhensible).

L'apprentissage numérique

L'apprentissage numérique ne manipule pas de symbole, il traite uniquement des *valeurs numériques quantitatives* qui vont être manipulées afin de réaliser l'apprentissage. Les méthodes d'apprentissage numérique se révèlent être *portables* et permettent un grande *adaptabilité*, car non dépendant de symboles. Par contre, le fonctionnement interne du système est opaque. Il est très difficile, voir impossible, de comprendre comment le système apprend (les réseaux de neurones artificiels en sont un bon exemple, il y a uniquement des interconnexions entre neurones).

1.2.2 Paradigmes d'apprentissage

En apprentissage, le fait qu'intervienne (ou non) un *enseignant* afin d'aider l'apprentissage, en fonction des informations dont il dispose, définit le protocole d'apprentissage.

Apprentissage non-supervisé

En apprentissage non-supervisé, le système ne reçoit *aucune information extérieure* concernant les résultats attendus. La connaissance ou l'information est donc « *incarnée* » dans la structure des informations en entrée, et le système ou l'algorithme doit *découvrir lui-même* les corrélations existantes entre les données qu'il a à traiter.

Apprentissage supervisé

Ici, un *expert* fournit la ou *les solutions attendues*, ou évalue l'erreur commise par le système. Durant l'apprentissage le système utilise les informations données par l'expert pour améliorer ses performances.

Apprentissage par renforcement

En apprentissage par renforcement, le système va essayer d'apprendre « seul » la façon d'améliorer ses résultats. La seule information dont il dispose est une *critique* sur son choix

précédent (cette critique peut venir de l'environnement, d'un critique extérieur ou être une auto-évaluation). En fonction des *récompenses* ou *punitions*, il doit modifier ou non son comportement afin d'améliorer ses performances.

1.2.3 Nos besoins pour l'apprentissage

Après cette présentation générale de l'apprentissage artificiel, il convient de situer nos besoins parmis les différentes types et paradigmes d'apprentissage (cf. tableau 1.1).

Premièrement, nous souhaitons que notre système soit *compréhensible* par un expert, afin de pouvoir *initialiser* la base de connaissances et la *valider* après apprentissage. Dans ce but l'utilisation d'un système à base de règles ou d'automates est judicieux. De plus, le souhait d'un système doté de capacités d'apprentissage (modularité et adaptabilité) nous pousse à choisir des règles « condition => action » afin d'associer à un état de l'environnement une/des action(s) à réaliser (Une règle peut être transformée/retirée/ajoutée sans remettre en cause le reste du système). Deuxièmement, notre besoin d'*évolution* et d'*adaptation en ligne* au niveau des règles (créations/transformations de règles) peut être amélioré par la mise en place d'algorithmes génétiques afin de faire évoluer les règles (apprentissage *numérique*). Nous serons donc appelés à associer apprentissage numérique et symbolique afin d'obtenir un *mécanisme adaptatif de comportement autonome à base de règles*.

Notre idée consiste à étudier l'évolution de l'environnement et l'état des motivations de l'entité, en fonction des actions effectuées. Une action bénéfique est valorisée, ou dévaluée dans le cas contraire. Le paradigme qui se rapproche le plus de nos besoins est donc un *apprentissage par renforcement*[3], où les punitions et récompenses sont données *via* une analyse de l'évolution de l'état de l'entité et de l'environnement.

Les systèmes de classeurs offrent des mécanismes d'évolution et d'apprentissage par renforcement grâce à des règles de types « condition => action » (symbolique) et des algorithmes génétiques (numérique), ils répondent donc à nos besoins (cf. tableau 1.2).

1.3 Systèmes de classeurs

Les systèmes de classeurs[4] permettent de modéliser et de simuler (entre autres[5]) le *comportement d'une entité* (cf. figure 1.1) en fonction de l'*état de l'environnement* dans lequel elle évolue, et de ses *états internes*. Par exemple Girard et al. (2002) les manipule pour gérer le comportement de vaisseaux spatiaux, et Sanza (2001) gère les joueurs d'une simulation de football à l'aide de systèmes de classeurs (un par joueur) pour le choix de leurs actions.

[3] Un apprentissage supervisé n'est pas envisageable étant donné le caractère dynamique et ouvert de l'environnement, la volonté d'un apprentissage en ligne et l'influence des motivations de l'entité.

[4] Nous les présentons ici très brièvement, le lecteur intéressé trouvera un état de l'art plus détaillé dans (Buche et al., 2006a) et (Sigaud, 2007).

[5] De récentes études en classification et fouille de données (Bernardo et al., 2001) ont donné de bons résultats.

	Présence d'un enseignant	*Informations disponibles pour apprendre*
Non-supervisé	Non	Aucune
Supervisé	Oui (il montre ce qui est attendu)	Les résultats attendus, des exemples
Renforcement	Dépend de la source des récompenses	Récompenses/punitions
Nos besoins	L'environnement de manière indirecte	Evaluation de l'évolution de l'environnement et des motivations de l'entité

Tab. 1.1 – Caractéristiques des trois paradigmes d'apprentissage, ainsi que nos besoins.

Nos besoins	Apprentissage en ligne	Compréhensible par un expert	Apprentissage par évaluation de l'environnement
Systèmes de classeurs	Mécanisme de renforcement + algorithmes évolutionistes	Système à base de règles « condition => action »	Récompenses/punitions issues de l'environnement

Tab. 1.2 – Les systèmes de classeurs répondent à nos besoins.

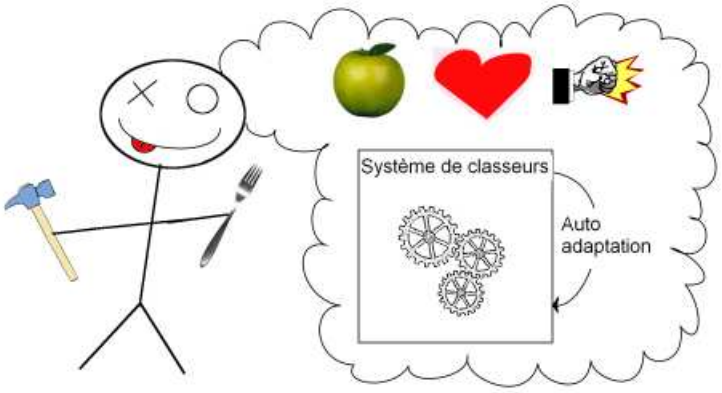

Fig. 1.1 – Un système de classeurs pour simuler un comportement complexe, auto-adaptatif cherchant à satisfaire des motivations.

1.3.1 Principes

Un système de classeurs gère un *ensemble de règles*, les *classeurs*, qui à une condition associent une action à réaliser. Chaque classeur est associé à une ou plusieurs *valeurs de qualité*, ce qui va permettre de faire des choix cohérents entre deux règles applicables au même instant. Cette valeur est réévaluée au cours du temps en fonction des récompenses ou des punitions engendrées par l'action du classeur associé. Cette valeur de qualité peut aussi faire office de fonction de *fitness* dans le cadre d'*algorithmes évolutionnistes* (génétiques) afin de créer de nouvelles règles issues des meilleurs individus parmis les classeurs déjà présents.

Le cycle de fonctionnement d'un système de classeurs (schématisé par la figure 1.2) est le suivant. L'environnement est perçu *via* une *interface d'entrée* et le système de classeurs en déduit les *règles applicables* qui vont associer à un état de l'environnement une action à réaliser. Parmi les règles applicables, les actions les plus intéressantes sont exécutées et le système de classeurs peut finalement percevoir une *rétribution* en fonction de l'évolution de l'environnement, qui va permettre de mettre à jour les règles et/ou leur valeur de qualité.

Le système de classeurs va donc, par *expérimentation*, apprendre l'ensemble des règles « condition => action » ainsi que leur qualité, afin de *maximiser les rétributions* de l'environnement.

Afin d'éviter que le système de classeurs ait à gérer un nombre trop élevé de règles, celles-ci sont *généralisantes*, c'est à dire qu'elles peuvent être appliquées pour plusieurs états de l'environnement. Il y a donc des mécanismes qui décident quand un ensemble de règles doit être généralisé en une seule, ou une règle générale, spécialisée en plusieurs, plus spécifiques. De plus, les algorithmes génétiques et de *covering* permettent de faire évoluer l'ensemble des règles.

Pour résumer, le système de classeurs est un mécanisme d'apprentissage dont les qualités des règles sont réévaluées dynamiquement par le biais d'un algorithme d'apprentissage par

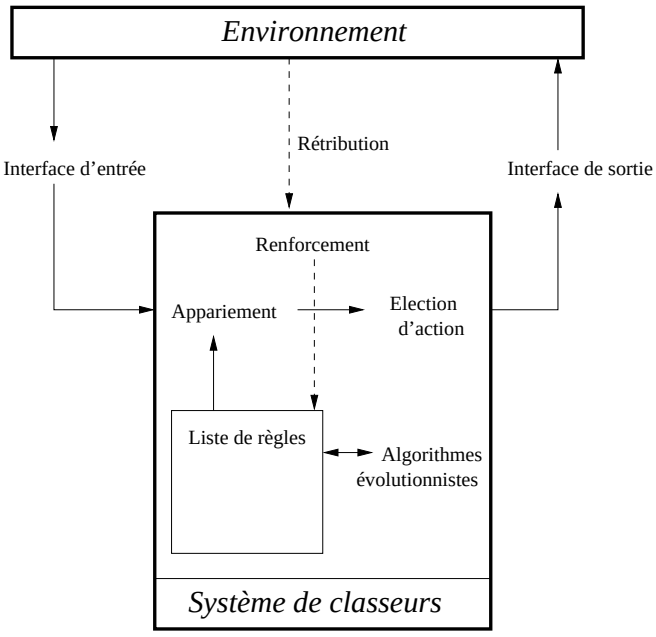

Fig. 1.2 – Fonctionnement général d'un système de classeurs d'après Buche et al. (2006a).

renforcement. De plus, l'apprentissage est optimisé en explorant de nouvelles solutions par génération de règles à l'aide d'*algorithmes évolutionnistes* et d'*heuristiques de recouvrement* (*covering*).

Généralisation de la base de règles

Dans le contexte habituel des systèmes de classeurs, les règles sont codées en binaire et nous manipulons donc avec des règles de type « condition => action » qui ressemblent à ceci : « 011001 => 011010 ». En général, un '0' dans la partie condition signifie qu'un capteur n'est pas activé, un '1' le contraire, de même pour les parties actions, un '1' active un moteur et un '0' le laisse inactif (cependant des codages plus fins sont possibles, 2 bits peuvent coder 2^2 états pour un capteur ou un moteur, 3 bits peuvent en coder 2^3 ...). Dans cet alphabet, il est possible d'utiliser un caractère « joker », le '#', pouvant représenter, soit un '0', soit un '1'. Ce procédé permet une généralisation des règles de la base, étant donné qu'un classeur contenant un ou plusieurs symboles '#' représente plusieurs classeurs classiques (le classeur « 00#1 => 1101 » correspond à la fois au classeur « 0001 => 1101 » et au classeur « 0011 => 1101 »).

Kovacs (1997) a proposé un classification, sachant que plus un classeur contient de symboles '#', plus il est généralisé, et moins il possède de symboles '#', plus il est spécialisé :

- ▷ Un classeur est dit surgénéralisé (*over-general*) s'il contient trop de symboles '#' pour obtenir de récompenses constantes et donc être efficace pour le problème posé. Un classeur surgénéralisé doit être spécialisé.

- ▷ Un classeur est dit généralisé au maximum (*maximally general*) s'il ne peut être ajouté aucun symbole '#' sans qu'il devienne surgénéralisé.

- ▷ Un classeur est dit généralisé sous-optimal (*suboptimally general*) s'il peut être généralisé tout en conservant sa même efficacité. Un classeur généralisé sous-optimal devrait être généralisé.

On constate que le problème de la généralisation des règles permet de réduire la taille de la base de règles à gérer. Cependant, si les règles sont généralisées à outrance, il y a un risque de faire du sur-apprentissage en étant beaucoup trop généralisé pour résoudre efficacement un problème.

Mécanisme de sélection

Il est courant que plusieurs classeurs soient activables en même temps (du fait de leur généralité). La sélection d'une action est alors *guidée par la qualité des règles*. Le plus souvent un mécanisme de *roue de la fortune* permet de sélectionner l'action à effectuer (plus une action a de la valeur, plus elle a de chance d'être choisie). Cela permet d'explorer de nouvelles pistes en trouvant un *compromis entre exploration et exploitation*. Ce compromis peut être géré par d'autres techniques afin de favoriser soit l'exploration ou l'exploitation (action prédominante par le nombre de règles la proposant, choix de la meilleur règle, exploration de pistes improbables, choix par tournoi...).

Mécanisme de rétribution

Le mécanisme de rétribution distribue les récompenses, ce qui améliore la valeur des règles ayant contribué à l'obtention de la récompense et diminue la valeur de celles qui en ont gêné l'obtention. Deux mécanismes de rétribution utilisés sont le « *Bucket Brigade* » (Holland, 1985) et le « Q-learning dérivé » que nous ne présenterons pas ici.

Mécanisme de génération

L'objectif du mécanisme de génération est de minimiser le nombre de règles tout en conservant celles menant à une rétribution. Une bonne règle est donc *généralisante* avec une *bonne valeur de qualité*. Les deux mécanismes de génération sont le *covering* et les *algorithmes génétiques*.

- ▷ Le covering permet de *créer des règles* lorsqu'aucun classeur ne correspond aux données perçues, ou que la qualité des classeurs compatibles est jugée insuffisante. Des règles sont donc créées de toutes pièces et une valeur de qualité leur est attribuée. Le covering est aussi une bonne solution pour *initialiser l'ensemble des règles* au début de l'apprentissage.

▷ Les algorithmes génétiques (Holland, 1975; Goldberg, 1989b) sont utilisés[6] pour effectuer des *croisements* entre les meilleurs classeurs (utilisation d'un mécanisme de roue de la fortune ou par tournoi) et obtenir ainsi des classeurs « fils » possédant des caractéristiques de chacun des parents[7]. Des *mutations* sur les classeurs sont aussi effectuées afin de rechercher des améliorations.

Les mécanismes de générations vont influer grandement sur l'*aptitude à s'adapter* à une évolution dynamique de l'environnement, par la création de nouvelles règles, la valorisation par croisement/mutation des meilleurs règles, et la destruction des règles devenues inutiles.

1.3.2 Bref historique

Les systèmes de classeurs ont été initialement proposés par Holland (1975) afin de proposer un *modèle d'apprentissage*. Deux approches ont été proposées et étudiées dans les débuts. L'approche de type *Pittsburg* (Smith, 1980a) propose un mécanisme adaptatif d'algorithmes génétiques qui s'applique à une population de systèmes de classeurs (un individu = un système de classeurs) afin de trouver celui qui répond le mieux au problème. L'autre approche, celle de *Michigan* travaille sur un système de classeurs unique et l'algorithme génétique concerne uniquement les règles de ce système de classeurs (un individu = un classeur) en collaboration avec un mécanisme d'apprentissage par renforcement.

Le premier système de classeurs, nommé « *CS1* » (Holland et Reitman, 1978), utilise des listes de messages internes ainsi qu'un mécanisme de rétribution. En raison de la complexité de fonctionnement de ce modèle (Wilson et Goldberg, 1989) et des difficultés à obtenir des résultats satisfaisants, les recherches sur les systèmes de classeurs se sont essoufflées durant les années 1980, jusqu'à la publication de l'algorithme d'apprentissage par renforcement de *Q-learning* par Watkins (1989). Les recherches sur les algorithmes d'apprentissage par renforcement ont permis la transformation du fonctionnement interne des systèmes de classeurs vers une architecture globale plus simple.

1.3.3 Trois grandes familles de systèmes de classeurs

Nous allons maintenant présenter très brièvement les trois principales familles de systèmes de classeurs qui sont caractérisées par la façon dont les classeurs sont évalués.

La famille des **ZCS** (Zeroth-level Classifier System) qui a été proposée initialement par Wilson (1994) est « *basée sur la force* », c'est-à-dire que c'est l'espérance de rétribution estimée qui sert à évaluer la qualité des classeurs. Un des principaux problèmes des ZCS est qu'il ne permet pas de valoriser des solutions éloignées du but (ayant peu de récompenses) mais qui mènent à plus ou moins long terme à une solution intéressante. Une idée pour résoudre ce genre de problèmes consiste à ne plus évaluer l'estimation de la récompense, mais la précision

[6] Les algorithmes agissent sur l'ensemble des règles ou uniquement sur celles sélectionnées à un instant, en fonction des versions (Wilson, 1994, 1995).

[7] Les algorithmes génétiques sont issus de la théorie de l'évolution de Darwin.

de cette estimation. Dans ce cas, même si un classeur n'offre que peu de rétribution avec une forte probabilité, il sera considéré comme intéressant. Cela a conduit à la famille des systèmes de classeurs « *basés sur la précision* », les **XCS** (Wilson, 1995).

L'idée des systèmes de classeurs « *basés sur l'anticipation* », les **ACS** (Anticipatory Classifier System) (Stolzmann, 1998; Butz et al., 2000), se démarquent clairement des deux précédents. Les ACS se basent sur l'*anticipation de l'état* de l'environnement après la réalisation d'une action. Les classeurs manipulés sont de la forme « [condition, action] => effet » et offrent donc un *modèle des transitions* entre états et des propriétés d'anticipation qui accélèrent l'apprentissage.

1.3.4 Limitations dans le cadre de comportements complexes

Malgré les bons résultats des systèmes de classeurs dans de nombreux domaines (notamment la classification de données), leur utilisation pour simuler des comportements complexes présente des limitations. En effet, un système de classeurs classique est rapidement limité quand il s'agit de réaliser des *tâches concurrentes*, *parallèles*, dans un contexte *multi-objectifs*. De plus une entité dotée d'un comportement complexe va avoir à traiter des données de types variés et d'*abstractions sémantiques différentes* (Barry, 1993).

1.4 Systèmes de classeurs pour des comportements complexes

Les limitations des systèmes de classeurs classiques pour gérer des comportements complexes peuvent être abordées par l'utilisation des systèmes de classeurs hétérogènes (cf. section 1.4.1), des hiérarchies de systèmes de classeurs (cf. section 1.4.2 - B) et des systèmes de classeurs hiérarchiques (cf. section 1.4.2 - C).

1.4.1 Systèmes de classeurs hétérogènes

Le principe des systèmes de classeurs hétérogènes est de gérer le traitement des données en fonction de leur portée sémantique et de les faire évoluer à l'aide d'algorithmes génétiques spécialisés pour chaque type de données[8] (cf. figure 1.3). De cette manière, une mutation, un croisement, n'amène plus à une valeur totalement incohérente, voir erronée, mais au contraire on peut maîtriser efficacement le mécanisme afin d'obtenir un *biais représentatif* améliorant les résultats. Par exemple, un flottant est représenté par son signe, son exposant et sa mantisse, une transformation « brute » entraîne un changement du tout au tout dans sa valeur, un traitement plus fin peut être préférable. Des implémentations de cette idée ont été proposées

[8] Au niveau de la machine, une chaîne de bits peut être un entier, un flottant, un intervalle... Un traitement particulier peut être nécessaire.

par Sanchez (2004) (HGCS : systèmes de classeurs hétérogènes généralisés) et Heguy (2003) (GCS : Generic Classifiers System).

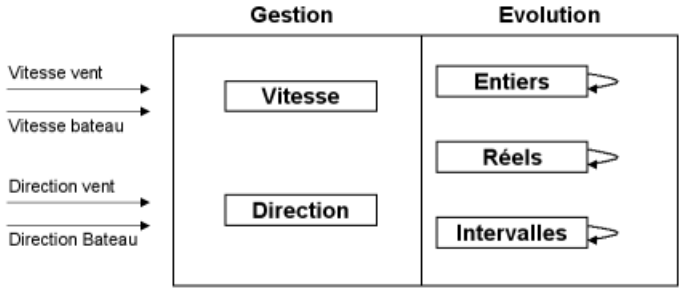

Fig. 1.3 – Traitement des données en fonction de la sémantique et du type (cas d'un voilier simpliste). Ici un module gère les vitesses du système, l'autre gère les directions. Des modules sont spécialisés pour faire évoluer les différents types de données.

1.4.2 Systèmes de classeurs et hiérarchie

Il existe deux manières d'aborder la hiérarchie dans le cadre des systèmes de classeurs, les hiérarchies de systèmes de classeurs (cf. section 1.4.2 - B) sont composés de systèmes de classeurs « classiques » qui vont s'activer les uns les autres et activer des actions. Pour les systèmes de classeurs hiérarchiques (cf. section 1.4.2 - C), la hiérarchie réside dans la structure des règles qui les composent. Cependant, avant d'aborder ces deux modèles, il convient de présenter les deux principaux mécanismes de parcours d'une architecture décisionnelle, les architectures WTA (*« Winner Take All »*) et FFH (*« Free Flow Hierarchy »*) (cf. section 1.4.2 - A).

1.4.2 - A Architectures WTA et FFH

Il existe deux grands principes de mécanisme de diffusion pour les architectures de décision, « Winner Take All » (« gagnant prend tout ») et « Free Flow Hierarchy » (« Hiérarchie à libre flux »).

- Dans une architecture de type WTA (*« Winner Take All »*) (cf. figure 1.4, schéma 1), proposée initialement par Tinbergen (1951), le premier niveau diffuse les activations (stimuli internes ou externes) aux différents éléments du niveau inférieur. Seul l'élément recevant l'activation la plus forte pourra diffuser son activation dans les éléments du niveau inférieur. Il y a donc une compétition forte entre les différents éléments d'un même niveau, qui se traduit par la sélection d'un seul vainqueur par niveau et donc une seule action finale.

- Rosenblatt et Payton (1989) ont proposé le principe des architectures en FFH (*« Free Flow Hierarchy »*) (cf. figure 1.4, schéma 2) qui permettent de réaliser des compromis entre plusieurs objectifs. Contrairement à une architecture de type WTA, aucun choix

n'est fait avant le dernier niveau de la hiérarchie. En outrepassant ainsi le principe de subsomption de Brooks (1986), cette architecture offre la possibilité d'explorer différentes pistes de recherche pour enfin effectuer le ou les choix les plus cohérents (en terme d'efficacité, de compromis, ...) au dernier moment, une fois toutes les informations en mains.

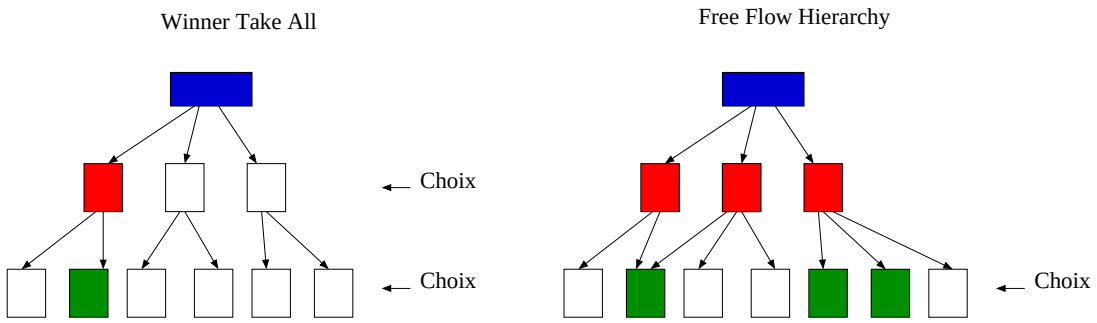

Fig. 1.4 – Les architectures en *« Winner Take All »* font un choix à chaque étape ce qui exclu des compromis, tandis que les architectures en *« Free Flow Hierarchy »* font un choix au dernier moment.

Les deux architectures sont donc fondamentalement opposées, quant une architecture en WTA fait un choix définitif à chaque étape du raisonnement, une architecture en FFH envisage de nombreuses options et choisit à la fin. Comme le souligne Steegmans et al. (2004) les architectures en FFH offrent de nombreux avantages, notamment le fait qu'elles n'ont pas besoin de toutes les informations en haut de la hiérarchie, ce qui permet un traitement des informations disponibles uniquement quand elles sont nécessaires. Puisque aucun choix définitif n'est fait, la réception d'une information au niveau d'un noeud inférieur n'est pas gênant et une information n'est traitée qu'aux noeuds susceptibles de l'utiliser. De plus, le fait de choisir au dernier moment permet de faire son choix en tout état de cause et de faire des compromis entre plusieurs motivations.

Les architectures décisionnelles, quelles que soient leur type, peuvent fonctionner soit en WTA, soit en FFH (ou éventuellement combiner les deux). Il existe des hiérarchies de systèmes de classeurs en WTA ou FFH, de même un système de classeurs hiérarchique peut être utilisé au sein d'une architecture de type WTA ou FFH.

1.4.2 - B Hiérarchies de Systèmes de Classeurs (HSC)

Les hiérarchies de systèmes de classeurs mettent en place une *architecture hiérarchique* permettant de décomposer un problème en sous-problèmes plus simples et plus précis (Barry, 1996). De cette façon, un *sous-problème* va nécessiter uniquement certaines données, sans tenir compte d'autres et les systèmes de classeurs qui traiteront de ces sous-problèmes vont être spécialisés donc beaucoup plus efficaces. De plus, dans le cas de l'application à la modélisation de comportements, les systèmes de classeurs hiérarchiques permettent de mettre en place des *personnalités* propres aux entités, en pondérant de manière significative les systèmes de classeurs composant la hiérarchie, notamment concernant les *motivations*.

Hiérarchies de Systèmes de Classeurs en WTA (HSC WTA)

Dans un HSC WTA, un système de classeurs arbitre va élire le prochain système de classeurs à activer parmi ceux du niveau inférieur (chaque système de classeurs du niveau inférieur correspond à une motivation). Le système de classeurs élu choisit lui même un système de classeurs de niveau inférieur ou l'action appropriée s'il n'y a pas de niveau inférieur. Ce mécanisme réalise à chaque étape un choix définitif entre plusieurs systèmes de classeurs au niveau inférieur, autrement dit à chaque instant il fait un choix entre les motivations qu'il a à satisfaire. De ce fait ce type d'approche exclut donc la possibilité de chercher un compromis entre plusieurs motivations mais augmente la vitesse de décision par un choix instantané. Un exemple de ce type d'architecture est ALECSYS (A LEarning Classifier SYStem) de Dorigo (1995), qui permet de contrôler un robot souris qui est tiraillé entre deux objectifs, suivre la lumière et échapper à un prédateur en se cachant dans un terrier. Cette première implémentation ne repose que sur un seul système de classeurs arbitre et deux systèmes de classeurs de niveaux inférieurs (« suivre la lumière » et « fuir ») qui ne peuvent s'éxécuter en même temps. On se retrouve donc dans une situation où l'entité ne peut pas effectuer de compromis entre les motivations

Hiérarchies de Systèmes de Classeurs en FFH (HSC FFH)

Les HSC FFH premettent de chercher des compromis entre différentes motivations en explorant plusieurs solutions possibles, pour finalement choisir, une fois toutes les possibilités évaluées, celle qui est la plus à même de satisfaire le plus de motivations. Une implémentation de cette idée est proposée par Robert (2002b), c'est une architecture hiérarchique sur quatre niveaux qui permet de choisir les actions qui vont offrir la meilleure alternative pour satisfaire toutes les motivations de l'entité (qui peuvent être contradictoires). Les niveaux supérieurs gèrent les motivations et dans les niveaux inférieurs, les actions élues doivent se partager des ressources. Des priorités sur les actions élues vont règler les conflits éventuels (si une même ressource est requise par plusieurs actions) alors que les actions non concurentes sont réalisables en même temps (cf. figure 1.5). De plus, en pondérant de manière significative les motivations de l'entité, il est possible d'intégrer la notion de personnalité (entité farouche ou intéressée par la collecte de nourriture...)

1.4.2 - C Systèmes de Classeurs Hiérarchiques (SCH)

Les Systèmes de Classeurs Hiérarchiques (SCH) ont été proposés initialement par Donnart et Meyer (1994), leur particularité réside dans l'utilisation d'un contexte interne (liste de messages) qui permet de stocker des informations sur l'état de l'environnement ou sur un « raisonnement » en cours (cf. figure 1.6). Bien que l'ajout de listes de messages pour les systèmes de classeurs soit très peu utilisé du fait des difficultés de mise en place et de gestion d'un tel système (Wilson et Goldberg, 1989), leur utilisation offre un intérêt pour créer des comportements plus cognitifs avec notamment la gestion de plans. Par exemple à un instant donné l'entité peut être dans un état de « recherche de nourriture », puis une fois qu'une proie est repérée elle passe en état de « chasse de la proie » pour enfin passer en mode « consommation de la nourriture » une fois la proie chassée.

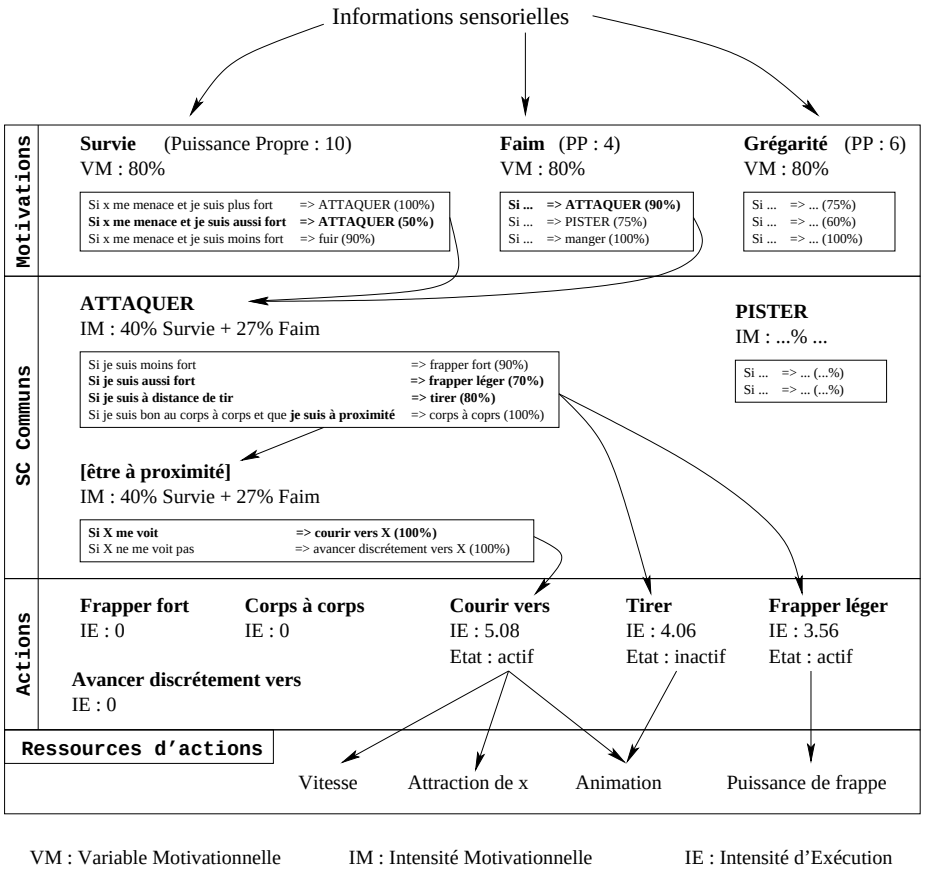

Fig. 1.5 – Architecture sur quatre étages de MHICS (Robert, 2002b). Les systèmes de classeurs s'activent en fonction des motivations et de l'environnement pour choisir les actions qui satisfont au mieux l'ensemble des motivations. Ici, au niveau des actions, « Courir vers » et « Tirer » sont concurentes mais « Courir vers » est prioritaire et peut se réaliser sans conflit avec « Frapper léger ». VM : Valeur des Motivations, IM : Intensité d'activation provenant des différentes Motivations, IE : intensité d'éxécution pour les actions.

Un SCH est un système de classeurs modifié pour obtenir une organisation hiérarchique des règles le composant, ce qui permet aussi bien la mise en place de plans pro-actifs, que des comportements réactifs. Ceci est rendu possible par l'utilisation de deux types de règles. Les règles externes envoient le choix des actions directement aux muscles ou moteurs et les règles internes modifient l'état interne (liste de message) du système.

Voici un exemple de règles internes et externes :

- ▷ règle interne : « Je suis en quête de l'objet A et j'ai l'objet A => Je passe en mode utilisation de l'objet A »
- ▷ règle externe : « Je suis en quête de l'objet A et l'objet A est dans la pièce à côté => Je vais dans la pièce à côté »

Cette liste de messages internes permet donc de connaître dans quel contexte se situe le système, le contexte pouvant définir un état d'esprit (affamé, terrifié, ...) ou encore l'état de

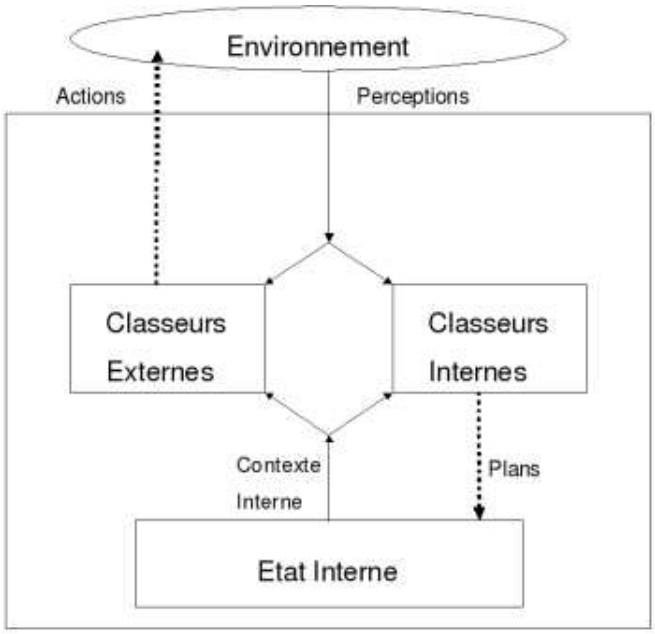

Fig. 1.6 – Architecture générale des Systèmes de Classeurs Hiérarchiques. Les classeurs externes et internes sont activés par les perceptions et le contexte interne de l'entité. Les classeurs internes font évoluer l'état interne de l'entité par ajout et suppression de messages. Les classeurs externes effectuent des actions sur l'environnement.

réalisation d'un plan (je suis à l'étape trois du plan B). On va donc, en ajoutant ou supprimant des éléments de cette liste de message, effectuer les différentes étapes d'un plan jusqu'à sa fin. Par exemple, le contexte interne *« find food »* va permettre de réaliser les actions jusqu'à l'obtention de nourriture, une fois la nourriture obtenue on pourra passer à un autre contexte interne comme *« eat food »* , *« cook food »* ou encore *« store food »* .

Donnart et Meyer (1994) utilisent ce principe pour réaliser le mécanisme de décision d'une entité virtuelle dont le but est de rejoindre un objectif (problème mono-objectif) dans un environnement contenant des obstacles. Cette application fait une utilisation des systèmes de classeurs hiérarchiques dans un contexte faiblement symbolique où l'entité connaît ses coordonnées, celles de l'objectif et peut percevoir un obstacle quand elle le rencontre. L'entité va apprendre un chemin vers l'objectif en décomposant le chemin initial en ligne droite, en sous-chemins qui mènent à l'objectif en évitant les obstacles. On a donc le plan initial (ligne droite vers l'objectif) qui est décomposé en sous-plans (plusieurs lignes droites formant un slalom vers l'objectif).

De Sevin (2006) s'est inspiré des SCH pour des entités devant satisfaire plusieurs motivations dans un environnement informé. Le contexte de cette application est donc fortement symbolique avec la manipulation de symboles abstraits tels que « loin », « pizza », « eau »...). En fonction de l'état d'activation des motivations, l'entité va décider de réaliser le plan pour satisfaire la motivation la plus urgente. Cependant même si cette architecture permet de réaliser des actions/plans afin de satisfaire des motivations, aucun apprentissage n'est réalisé afin de permettre à l'entité de s'adapter à des changements dans l'environnement. De plus, dans cette architecture il n'est pas possible de faire des choix qui pourraient être des

compromis entre plusieurs motivations. Cette architecture ne permet pas non plus de réaliser plusieurs actions en parallèle, et ne s'inquiète donc pas du partage éventuel de ressources entre des actions qui pourraient être réalisables en même temps.

1.5 Analyse préalable

Notre première idée concernant l'objectif du stage était de créer un modèle générique (et son implémentation) permettant de réaliser des architectures hiérarchiques de systèmes de classeurs de type FFH afin de modéliser des comportements cognitifs multi-motivation et nécessitant des ressources en nombre limité à partager (Par exemple : avec deux mains, il n'est pas possible de saisir trois objets qui nécessitent une main chacun pour être porté). Cette architecture hiérarchique devait permettre la mise en place d'apprentissage par renforcement et éventuellement par algorithme génétique.

Nous avons donc commencé à étudier cette possibilité, avec la conception d'un modèle générique et d'une application « test » afin de valider la cohérence et le fonctionnement de notre modèle générique. L'analyse concernant la conception du modèle générique était bien avancée, par contre lors de la définition de l'application qui nous aurait permis de tester ce modèle nous avons soulevé certains problèmes au sujet du concept même des hiérarchies de systèmes de classeurs :

1. Il s'avère très difficile de trouver une architecture cohérente pour modéliser un comportement souhaité, de plus cette difficulté augmente énormément avec la complexité du comportement à modéliser. En effet, il est très difficile de savoir si on doit faire un seul système de classeurs pour un comportement (l'attaque par exemple) ou s'il faut en faire un pour chaque sous-comportement (un par type d'attaque). De plus au niveau des liaisons entre systèmes de classeurs (un système de classeurs active/inhibe un autre système de classeurs), il est très complexe de trouver la façon dont sont corrélés les comportements (est-ce que l'attaque doit être corrélée avec la fuite ou la recherche de nourriture, et dans quelle mesure). Il est de plus très difficile de savoir si les choix sont crédibles et cohérents. Ce problème de difficulté à modéliser le comportement est crucial du fait qu'un des objectifs principaux des hiérarchies de systèmes de classeurs est d'en simplifier la modélisation.

2. Le fait de faire une décomposition d'un comportement en sous-comportements ne semble pas toujours efficace et cohérent par rapport à un système de classeurs classique, que ce soit au niveau de la qualité du comportement final ou d'un gain lors de l'éxécution. En effet, contrairement aux HSC en WTA où le fait de « trancher » en cours de recherche permet de ne pas inspecter une partie du sous-arbre de la hiérarchie, en FFH on va propager l'information dans toute la hiérarchie et donc presque tout les SC seront interrogés. De ce point de vue, au niveau de l'exécution, on inspectera presque autant de règles que la hiérarchie en contient, ce qui se rapproche de l'utilisation d'un système de classeurs classique. La question se pose donc de savoir si l'effort de modélisation complexe, se justifie pour le peu de gain obtenu.

3. La mise en place d'un algorithme génétique ou de *covering* au sein de chaque système de classeurs de l'architecture peut entraîner des incohérences concernant la base de règles de ce système de classeurs par rapport à son objectif. En effet, dans une hiérarchie de systèmes de classeurs, chaque système de classeurs la composant possède un but, tel que « attaquer » , « fuir »... Si l'on laisse un algorithme génétique ou de *covering* transformer la base de règles, le système de classeurs peut au final perdre complètement son utilité principale pour faire quelque chose de totalement différent, ce qui devrait se traduire par une uniformisation de la base de règles des systèmes de classeurs et donc se rapprocher de l'utilisation d'un système de classeurs unique. Pour remédier au problème généré par l'utilisation d'algorithmes génétiques, nous avions imaginé la possibilité d'utiliser ce qui pourrait être perçu comme un « pool génétique » qui limiterait les symboles utilisables par les algorithmes génétiques de chaque système de classeurs. De ce fait, chaque système de classeurs serait borné sur un domaine prédéfini (par exemple, le système de classeurs « attaquer » pourrait créer des règles concernant l'attaque mais pas des règles concernant la collecte de nourriture).

Aux vu de ces problèmes conceptuels importants, nous nous sommes questionnés sur l'intérêt de poursuivre la réalisation d'un modèle pour une architecture hiérarchique de système de classeurs, qui se révélerait inutile car non intéressante pour modéliser des comportements. Finalement la prise de contact avec Gabriel Robert, nous a confortées dans l'idée de ne pas continuer l'étude des hiérarchies de systèmes de classeurs en « *Free Flow Hierarchy* » , étant donné que lui-même a abandonné l'idée de les utiliser dans la version finale de son architecture MHICS Robert (2005) pour les mêmes raisons.

Nous avons donc finalement décidé de porter notre attention sur l'étude des systèmes de classeurs hiérarchiques qui permettent la réalisation de plans pro-actifs, mais aussi de comportements réactifs. Il nous est paru intéressant d'essayer de faire un rapprochement entre les travaux de Robert (architecture multi-motivationnelle avec systèmes de classeurs, partage de ressources, apprentissage mais pas de gestion de plan) et De Sevigne (architecture multi-motivationnelle avec systèmes de classeurs hiérarchiques, pas de partage de ressources, pas d'apprentissage mais avec gestion de plans) en réalisant une architecture composée de systèmes de classeurs hiérarchiques qui offre un mécanisme de décision pour une entité poussée par plusieurs motivations devant partager plusieurs ressources. L'accent a été porté sur la possibilité de trouver des compromis entre plusieurs plans et motivations, la gestion de ressources qui sont à partager (plusieurs actions exécutables en mêmes temps devant se partager des ressources), et la mise en place d'apprentissage permettant à l'entité de s'adapter aux changements de l'environnement.

Chapitre 2

— Proposition —

Nous nous intéressons ici à la problématique de la réalisation d'une architecture permettant de gérer un comportement poussé par plusieurs motivations, éventuellement contradictoires, et devant se partager des ressources nécessaires à la réalisation des actions. Notre proposition consiste en une architecture se rapprochant de celle proposée par Robert (2005) mais où l'utilisation des systèmes de classeurs « classiques » est remplacée par des systèmes de classeurs hiérarchiques afin de pouvoir gérer des plans (De Sevin, 2006). De plus, nous avons porté une attention toute particulière à la possibilité de chercher des compromis entre plusieurs motivations et à l'étude de pistes pour effectuer un apprentissage artificiel afin de générer des parties de plans et de mettre à jour les connaissances de l'entité concernant la qualité et le coût des actions et éléments de l'environnement. Nous allons donc, tout d'abord, présenter l'architecture et ses composants dans le détail, ainsi que son fonctionnement global (cf. section 2.1). Puis, nous présenterons les recherches que nous avons faites concernant l'apprentissage (cf. section 2.2).

2.1 Architecture

Afin de pouvoir modéliser des comportements cognitifs multi-motivations, nous avons réalisé une architecture hiérarchique qui permet de gérer le cycle « perception, décision, action » des entités virtuelles.

Nous partons du principe qu'une entité :

- observe son environnement,
- prend des décisions,
- afin de réaliser une action,
- pour satisfaire une/des motivations,
- apprend en analysant les conséquences des choix précédents.

Notre proposition porte sur la prise de décision et sur l'apprentissage. Avant d'aborder ces deux thématiques, il convient de préciser le mécanisme de perception utilisé et la notion d'action.

Mécanisme de perception

Afin de permettre à l'entité de trouver les éléments dont elle a besoin dans l'environnement, elle est dotée d'un mécanisme de perception qui va repérer les éléments visibles et retenir leur position.Des approches complexes de modèles 3D pour les perceptions visuelles et sonores existent (cône de perception par exemple), cependant, notre intérêt portant uniquement sur la prise de décision nous avons utilisé une approche simpliste où l'entité perçoit tout les éléments présents dans la pièce dans laquelle elle se trouve. Nous connaissons ainsi l'endroit précis où se situe l'élément perçu et ses propriétés (nom de l'objet, type de l'objet[1]) Une fois l'entité dans une autre pièce, elle va pouvoir se souvenir qu'elle a aperçu certains éléments qui sont actuellement nécessaire à la réalisation d'un plan. Cependant des éléments perçus auparavant peuvent avoir été pris par quelqu'un d'autre, il pourait donc être intéressant d'utiliser des probabilités afin de supposer la présence ou non d'un élément perçu précédemment en tenant compte du temps écoulé depuis sa perception et du passage de personnes dans les alentours de cet élément.

L'idée de recherche active peut aussi être intéressante à creuser dans le contexte où l'entité recherche un type d'élément bien particulier pour réaliser un plan, mais qu'elle ne connaît pas l'emplacement d'un tel élément dans son environnement. Dans ce cas, une perspective intéressante serait de mettre en place des zones de recherche pour tel ou tel type d'élément (si l'entité recherche de la nourriture, une fouille de la cuisine semble plus judicieuse que la salle de bain...) dans lesquelles l'entité pourrait fouiller avec une plus ou moins forte espérance de trouver un élément satisfaisant.

Le mécanisme de perception présenté, nous détaillons maintenant la notion d'action pour notre architecture.

Actions

Au sein de notre architecture, il faut différencier deux types d'actions :

- ▷ Les actions intermédiaires ont un coût (CE : coût estimé, CR : coût réel), elles sont les étapes de plans jusqu'à une action finale. Ces coûts peuvent dépendre de nombreux critères (CE_i : coût estimé pour le critère $_i$, CR_i : coût réel pour le critère $_i$) qui peuvent avoir plus ou moins d'importance. Par exemple, une action peut avoir un coût faible en temps d'exécution mais fort en consommation d'énergie, en fonction de la situation ou du problème à résoudre on pourra donner plus ou moins d'importance à tel ou tel critère.
- ▷ Les actions finales vont satisfaire une ou plusieurs motivations. Elles sont réalisées en fin de plans et engendre une rétribution (RE : rétribution finale estimée, RR : rétribution finale réelle) pour les motivations concernées.

Quel que soit le type d'action, deux cas nous intéressent :

[1] Comme nous le verrons plus tard l'entité va se faire une connaissance propre par rapport au type de chaque objet.

- ▷ L'action est réalisée sans utiliser d'élément de l'environnement. Le coût pour chaque critère (cas d'une action intermédiaire) ou la rétribution engendrée (cas d'une action finale) dépend donc uniquement de cette action.
- ▷ L'action est réalisée en utilisant un élément de l'environnement. Le coût pour chaque critère (cas d'une action intermédiaire) ou la rétribution engendrée (cas d'une action finale) dépend donc complètement de l'élément utilisé. Dans ce cas, le coût est donc définit au niveau de l'élément pour cette action et la rétribution est définie au niveau de l'élément pour cette action dans le cadre d'une motivation.

Il faut de plus, bien noter la différence entre les valeurs (coûts et rétributions) réelles et estimées. En effet, notre entité possède une connaissance qui va lui permettre d'estimer les coûts et rétribution espérée. Cette connaissance, définie *a priori* ou apprise au fur et à mesure, est utilisée pour estimer les plans possibles à un instant donné. Les coûts ou rétributions réels ne sont connus qu'au moment de la réalisation effective du plan et peuvent donc être utilisés afin de réévaluer les valeurs estimées par l'entité comme nous le verrons dans la section 2.2.1.

2.1.1 Principes de base de l'architecture de décision

Maintenant que ces aspects de la perception et des actions ont été abordés, nous présentons notre architecture décisionnelle et les mécanismes d'apprentissage qui ont été intégrés afin de permettre d'une part, une mise à jour des connaissances qualitatives sur les éléments de l'environnement, d'autre part la découverte partielle de plans. Notre architecture décisionnelle permet de :

- ▷ gérer l'évolution des motivations et leur activation (cf. section 2.1.2) ;
- ▷ évaluer les plans satisfaisants les motivations (cf. section 2.1.3 et 2.1.4) et d'effectuer un choix concernant les prochaines actions à réaliser (cf. section 2.1.5) ;
- ▷ effectuer un choix parmi les plans évalués (cf. section 2.1.5) ;
- ▷ gérer le partage de ressources entre plusieurs actions concurrentes (cf. section 2.1.5) ;

Nous allons maintenant présenter l'architecture générale, puis détailler chaque élément de cette architecture.

Notre architecture (cf. figure 2.1) est composée d'un premier niveau (niveau 1) gérant l'évolution des motivations (Motivations$_i$) et leur activation. Ces motivations activent les Systèmes de Classeurs Hiérarchiques (SCH$_i$) du niveau 2. Ces derniers sont spécialisés pour réaliser un ou plusieurs plans spécifiques pour satisfaire une motivation. Les SCH déroulent donc des plans en fonction des éléments présents dans l'environnement. Ces plans sont valués en fonction du coût des actions intermédiaires réalisées tout au long du plan et de la rétribution finale espérée. Chaque SCH propose en fin d'évaluation la première action de chaque plan qu'il a évalué. Chaque système de classeurs hiérarchique va donc fournir au mécanisme de décision (niveau 3) la liste contenant la première action de chaque plan évalué. Puis un pré-traitement (premier pré-traitement) est effectué sur les actions proposées par les SCH d'une même motivation afin de les réévaluer en fonction de leur qualité et du nombre de fois où elles ont été proposées. On obtient donc une nouvelle liste d'actions (LA Motiv$_i$) contenant

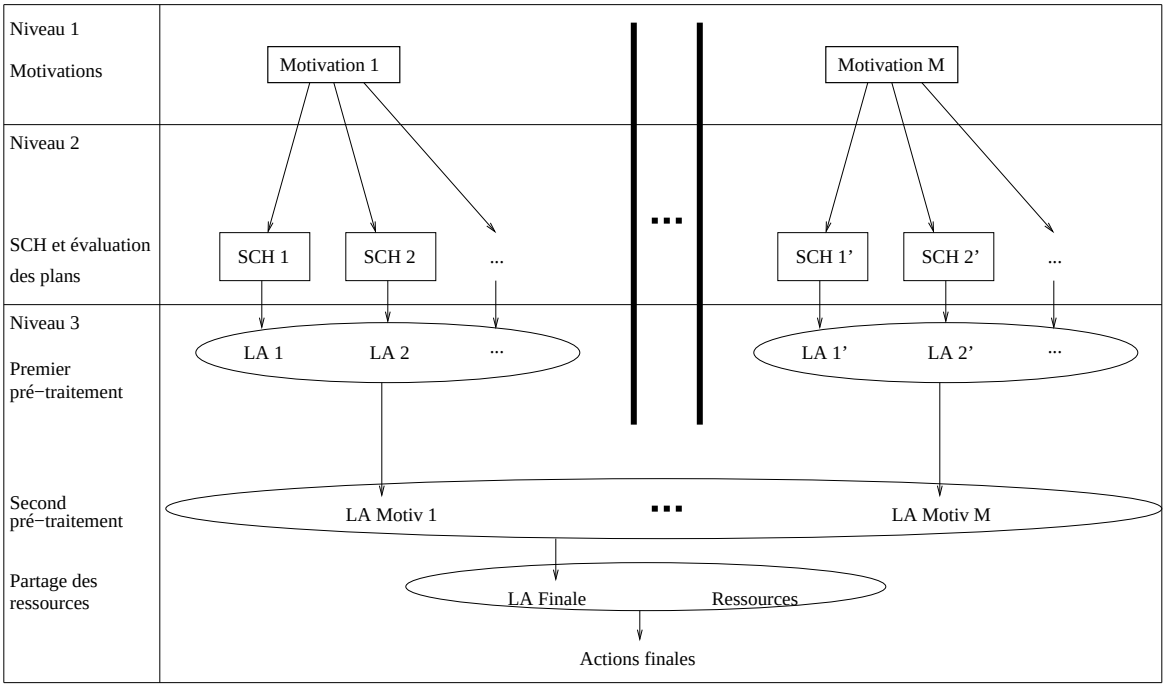

Fig. 2.1 – Architecture globale de notre modèle.

toutes les actions proposées pour la motivation$_i$. Le mécanisme de décision réévalue une seconde fois la valeur des actions proposées (second pré-traitement) afin de permettre de trouver d'éventuels compromis entre toutes les motivations. Nous avons donc une liste finale (LA Finale) contenant toutes les actions proposées et leurs valeurs. Les actions valuées sont finalement ordonnées afin de pouvoir se partager les ressources par ordre de priorité. Le pseudo-code suivant (cf. algorithme 1) permet de visualiser de manière simplifiée le déroulement d'un cycle de décision.

Algorithm 1 Pseudo-code d'un cycle de décision

```
1: for each m ∈ [Motivations] do
2:    if m.is_active() then
3:       LA_list = m.evaluate_plans()
4:       LA_motiv[m] = first_treatement(LA_list)
5:    end if
6: end for
7: LA_final = second_treatement(LA_motiv)
8: Final_actions = sharing(LA_final,Ressources)
9: return Final_actions
```

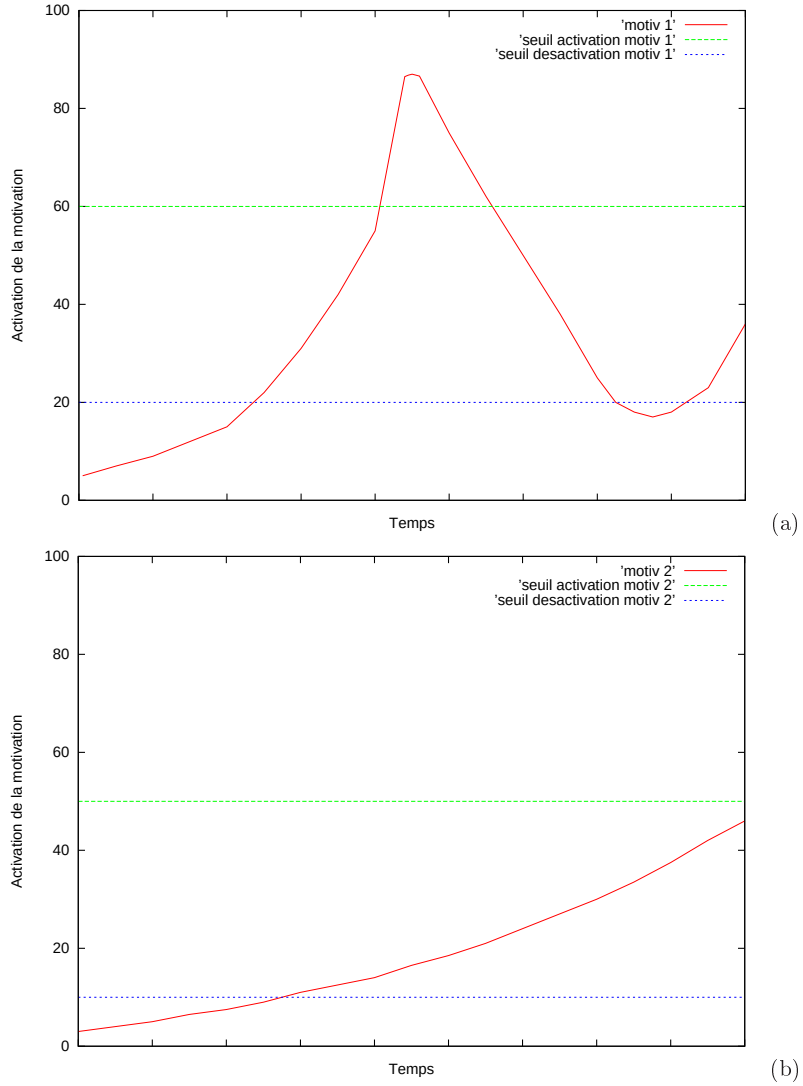

Fig. 2.2 – On peut voir sur ces deux graphiques l'évolution de deux motivations suivant chacune une loi différente. Pour la première motivation (a), la loi se rapproche d'une exponentielle et le seuil d'activation est de 60, à partir du moment où le niveau d'activité de la motivation atteint ce seuil, les SCH de cette motivations sont activés. Puis une fois que la motivation est en court de satisfaction son niveau d'activité diminue jusqu'au seuil de désactivation (ici 20), une fois ce seuil atteint, les SCH sont désactivés jusqu'à ce que la motivation atteigne à nouveau le seuil d'activation. La motivation représentée par le second graphique (b) suit une loi beaucoup plus douce et les seuils d'activation et de désactivation sont différents.

2.1.2 Gestion des motivations (Niveau 1)

Chaque module de motivation gère une des motivations de l'entité. Une motivation est caractérisée par son niveau d'activation qui se situe dans un intervalle entre 0 et 100 (cf. figure 2.2). La valeur de cette motivation va augmenter au cours du temps en suivant une loi représentative de la motivation (linéaire, exponentielle...) de manière plus ou moins

indépendante. En effet, une motivation peut être stimulée ou inhibée par une autre motivation ou des actions qui sont en cours ou ont eu lieu (c'est à dire que l'activation d'une motivation ou de certaines actions peut augmenter ou diminuer la pente d'évolution de la motivation). Par exemple un sentiment de danger va inhiber toutes les autres motivations pour se focaliser sur la défense ou la fuite, alors que le fait d'avoir mangé va avoir tendance à assoupir. Lorsque ce niveau d'activation atteint le seuil d'activation, les SCH associés à cette motivation sont activés et cherchent donc à la satisfaire en recherchant le meilleur plan à cet instant. Les priorités entre les motivations sont abordées par l'utilisation d'un coefficient qui définit l'importance de chaque motivation, plus le coefficient est fort, plus la motivation a de l'importance et inversement. Au moment de l'activation des SCH du niveau inférieur, la valeur de ce coefficient est utilisée pour pondérer la valeur des plans évalués. Plus ce coefficient sera fort, plus les plans associés auront d'importance et donc de chance d'être choisi. Si un plan associé à cette motivation est élu, il sera exécuté jusqu'à atteindre une action bénéfique pour la motivation. A partir du moment où cette action bénéfique est activée, le niveau d'activation de la motivation va redescendre progressivement jusqu'à un seuil de désactivation, ce qui va désactiver les SCH correspondant à la motivation. A chaque instant, il peut y avoir plusieurs motivations à satisfaire, le mécanisme de décision doit donc essayer de trouver le meilleur compromis entre toutes les motivations.

2.1.3 Systèmes de classeurs hiérarchiques (Niveau 2)

Un ou plusieurs SCH peuvent être associés à chaque motivation et chaque SCH peut proposer des plans utilisant des éléments de l'environnement afin de satisfaire cette motivation. Au moment où un SCH est activé par une motivation, il sélectionne, dans les connaissances de l'entité, les éléments qui sont supposés lui être nécessaire. Ensuite il va, pour chacun de ces éléments, tester les plans susceptibles de satisfaire la motivation. Chacun des plans testés va être évalué en fonction de différents critères, qui peuvent évoluer en fonction du type de problème à traiter (cf. section 2.1.4).

Cette approche est intéressante du fait que la connaissance n'est pas contenue dans l'environnement comme c'est la cas dans un environnement informé classique. Chaque entité se fait une représentation de l'utilité de tel ou tel élément et du coût des actions qu'il peut faire, même si la façon dont ils sont utilisés au sein des plans est défini *a priori* par le concepteur. Il y a donc ici une part de connaissance propre à chaque entité, une représentation interne de l'utilité d'éléments de l'environnement.

Certains plans, plus complexes, peuvent nécessiter plusieurs éléments différents au cours du temps. Dans ce cas, le SCH concerné peut consulter les éléments potentiellement intéressants pour la suite du plan, quand cela est nécessaire. Ainsi, à la fin d'une étape d'un plan, le système de classeurs hiérarchique va pouvoir inspecter uniquement les éléments nécessaires à la réalisation de la prochaine étape du plan.

Notons qu'il est possible pour un SCH d'activer un autre SCH de la même motivation (voir d'une motivation différente). De ce fait, en plus de pouvoir passer d'une étape d'un plan à l'autre, on peut ainsi créer un lien entre différents plans et comportements, on obtient donc une structure entre les SCH en plus d'avoir une structure au sein des SCH. Par exemple, nous avons la possibilité d'activer un comportement à la fin de la réalisation d'un plan afin

de lier des plans entre eux et donc créer des procédures qui peuvent être perçues comme des plans de plans. Ceci permet de simplifier grandement la conception du comportement *a priori* et l'analyse *a posteriori* en regardant le fonctionnement et la structure des SCH gérant une motivation. Dans une simulation mettant en scène des procédures plus complexes, comme avec des pompiers, on pourrait utiliser ce principe pour gérer les différentes étapes d'une procédure. Le départ de la procédure activerait des SCH gérant le début de la procédure (étude du plan de secours et des procédures d'évacuation, recherche et encadrement de blessés...). Une fois le début de la procédure fini, les systèmes de classeurs hiérarchiques correspondant activeraient les systèmes de classeurs hiérarchiques gérant la suite de la procédure (déployer les lances à incendie, éteindre l'incendie...). Ces derniers, une fois leur tâche finie, activeraient les systèmes de classeurs hiérarchiques finissant la procédure (fouille du site, enquête...). Cette approche permet de combiner différentes granularités (tâches au sein d'un plan, plans au sein d'une procédure...).

Nous présentons maintenant la façon dont nous évaluons les plans en fonction de différents critères qui peuvent être le coût en temps, en énergie, en difficulté...

2.1.4 Evaluation des plans (Niveau 2)

Comme précisé précédemment, l'évaluation d'un plan peut varier en fonction du critère choisi. Cette évaluation se fait à partir de la rétribution finale espérée par le plan pour la motivation (qui peut varier de la rétribution finale réelle), à laquelle on retire le coût estimé (qui peut aussi varier du coût réel) de l'ensemble des actions intermédiaires. C'est sur le coût intermédiaire que le critère de jugement peut varier. En effet, en fonction du problème traité on peut vouloir juger différemment le coût d'une action en fonction de sa durée d'exécution, de son coût en énergie, de sa difficulté... Pour un problème où l'on cherche la solution la plus rapide, on choisira une évaluation des actions en fonction de leur durée d'exécution. Pour un problème où l'on cherche une solution optimale en terme d'efficacité, on choisira d'ignorer le coût des actions intermédiaires pour se focaliser sur la rétribution finale. Dans un contexte de consommation d'énergie, on tiendra uniquement compte des coûts en énergie des actions intermédiaires. Ici aussi, comme au niveau des motivations, il va donc être possible de mettre en place un compromis concernant la façon d'évaluer les actions intermédiaires, le coût des actions sera donc défini en fonction de plusieurs critères.

Notons que les coûts de certains critères peuvent, de plus, dépendre des éléments de l'environnement. Si une action intermédiaire utilise un élément, celui-ci peut influencer le coût de l'action (plus l'objet manipulé est lourd, plus l'action de le transporter est coûteuse en temps et en énergie). Nous avons donc deux cas :

1. L'action ne nécessite aucun élément, le coût pour chaque critère est donc directement lié à l'action.

   ▷ CE est le coût estimé de l'action.
   ▷ α_i est le coefficient donné pour le critère i, il représente l'importance du critère i dans le total des coûts.
   ▷ CE_i est le coût estimé de l'action pour ce critère i.

$$CE = \sum_i (CE_i \times \alpha_i)$$

2. L'action nécessite un élément, le coût pour chaque critère est donc défini au niveau de l'élément cible pour cette action.

- ▷ CE est le coût estimé de l'action.
- ▷ α_i est le coefficient donné pour le critère i, il représente l'importance du critère i dans le total des coûts.
- ▷ CEe_i est le coût estimé de l'action avec l'élément e pour ce critère i.

$$CE = \sum_i (CEe_i \times \alpha_i)$$

Le mécanisme pour calculer le coût réel des actions intermédiaires suit le même principe

Le second type d'action sont les actions finales qui sont bénéfiques pour les motivations. Il y a là aussi deux cas :

1. L'action ne nécessite aucun élément et sa valeur est donc liée à l'action elle même. La rétribution finale pour une motivation (RE) est égale à la valeur de l'action pour cette motivation.

2. L'action nécessite un élément et la valeur de cette action est donc liée à la valeur de cet élément pour cette action dans le cadre de cette motivation. Un élément qui peut être utilisé dans plusieurs actions va donc avoir une valeur pour chacune de ces actions. Un verre d'eau a une valeur élevée pour satisfaire la soif s'il est bu, et une valeur nulle s'il est jeté. De plus ce même verre d'eau peut être utilisé pour satisfaire une toute autre motivation telle que éteindre un feu, en le jetant sur les flammes. La rétribution finale pour une motivation (RE) est égale à la valeur de l'action avec l'élément pour cette motivation.

Notons la différence entre valeur estimée et valeur réelle des plans. La valeur estimée est calculée *a priori* à partir des connaissances de l'entité sur son environnement et de ses capacités. La valeur réelle est calculée lors de la réalisation du plan et permet de mettre à jour les connaissances qu'à l'entité sur son environnement comme nous le verrons dans la section 2.2.

Une fois obtenu le coût des actions et la rétribution finale, on peut calculer la valeur estimée d'un plan. Nous avons donc :

- ▷ VA est la valeur du plan, valeur qui est reportée sur la première action du plan.
- ▷ CE_j est le coût de l'action j.
- ▷ RE est la rétribution finale espérée pour le plan.

$$VA = RE - \sum_j CE_j$$

Si l'on prend un exemple simple, où l'entité ne s'intéresse qu'à un seul critère pour évaluer le coût des actions (le temps d'éxecution par exemple), on peut estimer l'évaluation d'un plan qui consiste à atteindre un objectif qui va nous rapporter 60 (l'unité est ici non définie) et les actions qui vont nous permettre d'atteindre cet objectif nous coûtent :

- ▷ aller dans la pièce ($15sec$) ;
- ▷ se déplacer vers le meuble ($3sec$) ;
- ▷ ouvrir et fouiller le meuble ($5sec$) ;
- ▷ prendre l'objectif ($2sec$).

La valeur du plan est donc :

$$VA = 60 - (15 + 3 + 5 + 2) = 35$$

Possédant les valeurs estimées pour les plans de chaque motivation, le mécanisme de décision va essayer de trouver un compromis entre plusieurs motivations en cherchant parmi les premières actions de ces plans, lesquelles offrent le meilleur compromis entre les motivations.

2.1.5 Mécanisme de décision (Niveau 3)

Le mécanisme de décision a pour objectif de faire un choix entre toutes les actions proposées par les SCH (la première action de chaque plan évalué), dans le but de trouver le meilleur compromis une fois toutes les données du problème en main (principe des architectures en FFH). Afin de distinguer les actions qui offrent un compromis, le mécanisme de décision va réévaluer la valeur des actions proposées par les SCH en deux étapes.

1. Pour chaque motivation, le mécanisme de décision pondère la valeur des actions proposées par les SCH de cette motivation en fonction du nombre de fois où elles sont proposées. Si une même action est proposée plusieurs fois par une même motivation, sa nouvelle valeur est égale à la moyenne des valeurs précédentes (une autre approche serait de garder la plus grande valeur afin de rechercher une solution optimale). Après ce premier traitement, chaque motivation propose donc des actions toutes différentes. Nous avons donc :

   - ▷ VA' est la valeur de l'action après le premier traitement.
   - ▷ VA_i est la valeur de la i^{e} occurence de cette action.
   - ▷ i est le nombre de fois que cette action a été proposée.

   $$VA' = \frac{\sum_i VA_i}{i}$$

2. Si une même action est proposée par plusieurs motivations différentes, cette fois ci les valeurs vont être ajoutées en tenant compte d'un coefficient propre à chaque motivation, afin de pouvoir pondérer l'importance de chaque motivation (plus une motivation est

importante, plus son coefficient sera grand). Ce second traitement permet de valuer à la hausse les actions qui offrent un compromis entre plusieurs motivations. Nous avons donc :

- ▷ VA'' est la valeur de l'action après le second traitement.
- ▷ VA'_j est la valeur de l'action après le premier traitement pour la j^{e} occurence de cette action.
- ▷ α_j est le coefficient de la motivation j proposant cette action.

$$VA'' = \sum_j \alpha_j \times VA'_j$$

Finalement, le mécanisme de décision possède une liste valuée d'actions et va la parcourir dans l'ordre décroissant (de la meilleur à la plus faible) afin de trouver toutes les actions réalisables en même temps. Pour chaque action, il va regarder s'il possède suffisamment de ressources pour l'exécuter (par exemple, prendre un objet nécessite une main de libre), si oui, il la sélectionne et retire de la liste des ressources libres celles qui sont utilisées par cette action. De cette manière, plusieurs actions vont être réalisables en même temps, si elles n'utilisent pas les mêmes ressources (« tenir un objet » qui nécessite une ou deux mains et « aller vers » qui nécessite l'animation des jambes vont être exécutables en même temps).

2.1.6 Mode de fonctionnement de l'architecture

Nous proposons deux modes de fonctionnement de l'architecture. Le premier est dynamique, les plans peuvent être interrompus afin de satisfaire au mieux les motivations. Le second fonctionnement est statique et ne permet pas l'interruption d'un plan une fois celui-ci commencé. Le choix d'un mode de fonctionnement a une forte influence au niveau des résultats et de la possibilité ou non de mettre en place un apprentissage.

Fonctionnement dynamique

Pour ce principe de fonctionnement (cf. figure 2.3), le mécanisme de choix est exécuté à intervalle régulier afin de permettre une remise en cause de ce qui est effectué à tout instant. De ce fait, si un plan A1 est en cours de réalisation, dans le but de satisfaire un motivation A, et que ce plan A1 n'est plus prioritaire (poussée soudaine d'une autre motivation, apparition d'un nouvel élément dans l'environnement), alors ce plan A1 sera interrompu au profit du plan B1 qui aura pris la priorité. Ce qui est intéressant est la possibilité pour le plan A1 d'éventuellement reprendre où il en était à la fin du plan B1. En effet, le système de classeurs hiérarchique gérant A1 va garder en mémoire l'état interne dans lequel il se trouve afin de pouvoir reprendre son activité par la suite, si cela est nécessaire

Comme on peut le constater le principal intérêt de ce fonctionnement dynamique est de pouvoir interrompre des plans. Cependant, ce mode de fonctionnement interdit l'utilisation de

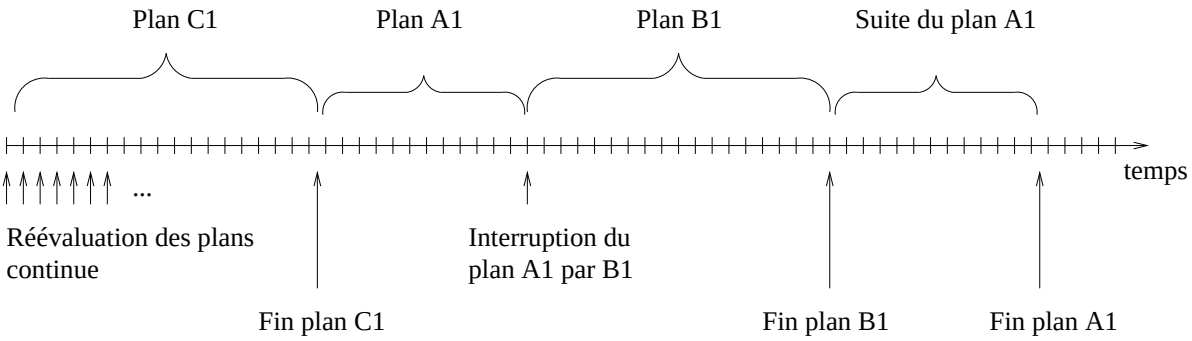

Fig. 2.3 – Echelle de temps d'une éxécution dynamique.

mécanisme de choix aléatoire des règles, qui est pourtant une des caractéristiques principales des systèmes de classeurs classiques. Un mécanisme de choix aléatoire des règles entraînerait l'apparition d'oscillations dans le choix des plans pour au final rendre quasi impossible la finition d'un plan. Si l'on a cinq règles qui font partir vers cinq plans différents, le choix aléatoire (mécanisme de roue de la fortune par exemple) va par exemple choisir à l'instant 1, la règle 3, puis à l'instant 2 choisir la règle 5, ce qui rendra difficile l'atteinte d'un objectif. Nous sommes donc obligés d'utiliser un mécanisme déterministe qui va empêcher les oscillations au niveau des choix de plans.

Fonctionnement statique

Le second mode de fonctionnement possible est statique (cf. figure 2.4). Une fois qu'un plan est choisi, il va être réalisé jusqu'au bout sans continuer à chercher des solutions. On se retrouve donc dans des situations où l'entité réalise le plan choisi et ne se rend pas compte qu'au bout de quelques temps elle pourrait arrêter le plan en cours pour le remplacer par un autre plus efficace (découverte d'un nouvel élément plus efficace dans l'environnement, apparition d'un obstacle imprévu, ...). Dans un autre cas, une motivation A pourrait avoir déclenché un plan A1, mais une autre motivation B, devenue soudainement très urgente (situation de danger, arrivée d'un élément très perturbateur pour cette motivation, ...), ne pourrait pas prendre la main et interrompre le plan A1. Malgré les problèmes, au niveau de la dynamicité, du fonctionnement statique, l'utilisation d'un mécanisme de choix aléatoire des règles redevient possible. En effet, le problème d'oscillation entre plusieurs plans n'existe plus étant donné qu'une fois un plan commencé, il est forcement suivi. On peut donc choisir quelles règles doivent être choisies de manière aléatoire ou pseudo-aléatoire (utilisation du mécanisme de roue de la fortune, plus une règle est valable, plus elle a de chance d'être sélectionnée), pour au final obtenir un plan qui sera exécuté jusqu'à sa fin.

Dynamique *vs* statique

Les modes de fonctionnement dynamiques et statiques sont profondement différents. Ces différences sont résumées dans le tableau 2.1. Le mode de fonctionnement dynamique permet donc l'interruption entre plans, mais par contre effectue ces choix de manière déterministe, en choisissant le « meilleur » choix à un instant donné, ou du moins celui qui semble le

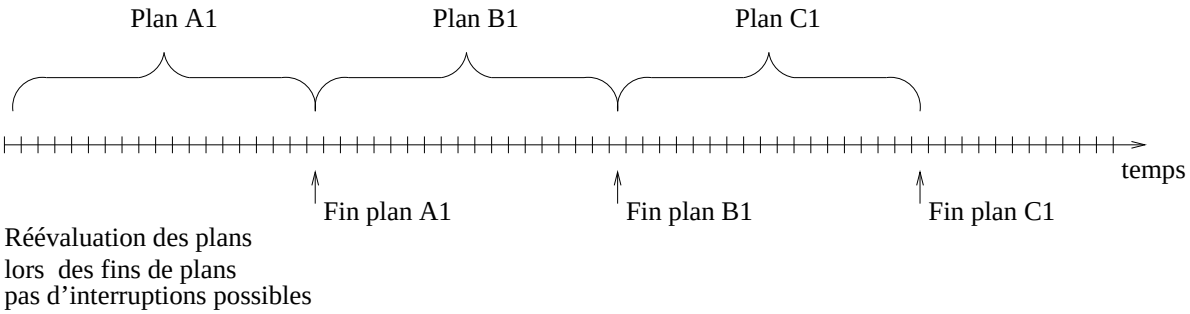

Fig. 2.4 – Echelle de temps d'une éxécution statique.

« meilleur » au vu des critères qui ont été sélectionnés. Le mode statique ne permet pas d'interruption de plans et offre donc la possibilité de choisir les plans de manière pseudo-aléatoire par mécanisme de roue de la fortune. Le choix est donc à faire en fonction du comportement désiré, un comportement dynamique mais déterministe ou un comportement statique mais non déterministe. De plus, notons que le mode fonctionnement dynamique ne permet pas ou peu d'exploration de nouvelles solutions, alors que le mode statique permet par un mécanisme de roue de la fortune d'explorer des nouvelles solutions. Il peut donc être judicieux d'alterner des cycles d'exploration (en mode statique) et des cycles d'exploitation (en mode dynamique).

	Dynamique	*Statique*
Exploration/ Exploitation	Exploitation (optimum)	Exploration possible (roue de la fortune)
Principe de fonctionnement	Déterministe	Non-déterminisme possible
Rapidité des réactions	Les plans peuvent être interrompu	Pas d'interruption de plan possible
Déclenchement du mécanisme de recherche de plans	On recherche le « meilleur » plan à intervalle régulier	On recherche le prochain plan à la fin de l'exécution du plan courant

Tab. 2.1 – Différences entre les modes de fonctionnement dynamique et statique.

2.1.7 Exemples d'utilisation

Afin de présenter les intérêts de notre architecture dans le cadre d'une utilisation dynamique, nous présentons maintenant deux exemples simples.

Exemple 1, le compromis

Le premier exemple (cf. figure 2.5) illustre le problème de la gestion des compromis. Il y a ici deux motivations à satisfaire (Boire et Manger) et pour les satisfaire, il y a deux éléments

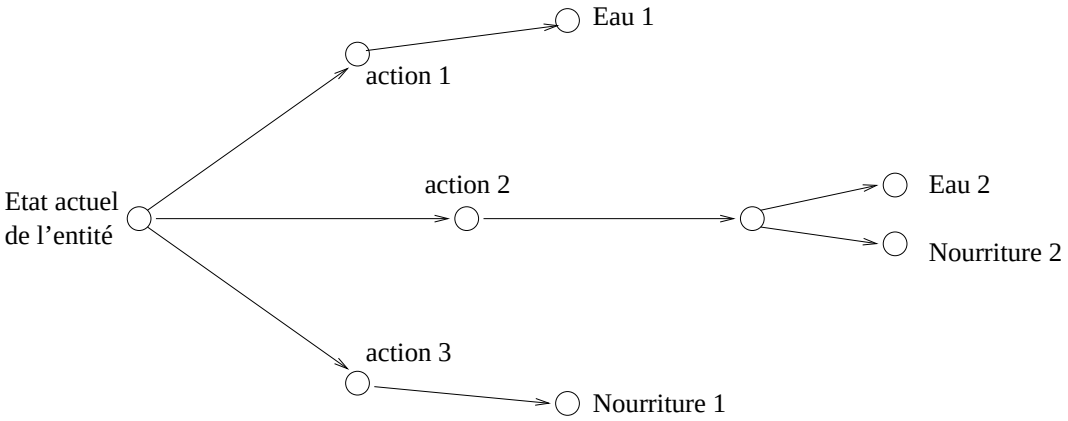

Fig. 2.5 – Illustration d'une gestion de compromis.

pour Boire (Eau 1 et Eau 2) et deux éléments pour Manger (Nourriture 1 et Nourriture 2). Suite à l'évaluation des plans nous obtenons ces valeurs (plan vers Eau 1 : 30, plan vers Eau 2 : 20, plan vers Nourriture 1 : 35, plan vers Nourriture 2 : 25). Comme expliqué précédemment, les valeurs des plans sont reportées sur la première action de chaque plan et ces valeurs sont retraitées deux fois. Dans cet exemple le premier pré-traitement n'a pas de conséquence car chaque motivation propose des première actions toutes différentes). Le second pré-traitement, par contre, va réévaluer l'action 2 de cette manière :

- ▷ VA'' est la valeur de l'action 2 après le second traitement.
- ▷ VA'_{Boire} est la valeur de l'action 2 après le premier traitement pour la motivation Boire.
- ▷ α_{Boire} est le coefficient représentant l'importance de la motivation Boire.
- ▷ VA'_{Faim} est la valeur de l'action 2 après le premier traitement pour la motivation Manger.
- ▷ α_{Faim} est le coefficient représentant l'importance de la motivation Manger.

$$VA'' = (VA'_{Boire} \times \alpha_{Boire}) + (VA'_{Faim} \times \alpha_{Faim})$$

Si l'on considère que les motivations A et B ont un coefficient de 1, on a :

$$VA'' = (20 \times 1) + (25 \times 1) = 45$$

Nous avons donc les valeurs pour les prochaines actions réalisables (Action 1 : 30, Action 2 : 45, Action 3 : 35). On voit ici que le mécanisme de décision va favoriser un compromis qui est d'aller un peu plus loin afin de pouvoir satisfaire deux motivations, plutôt que d'aller s'isoler pour satisfaire une motivation, puis repartir dans l'autre sens afin de satisfaire une seconde motivation.

Exemple 2, l'interruption de plan

Le second exemple nous illustre l'interruption de plan pour un autre plus prioritaire. Si l'on se place dans le contexte d'une entité réalisant un plan A, assez long, cette entité va réévaluer à intervalle régulier ce qu'elle doit faire. Imaginons que l'entité perçoive une alarme incendie, ce qui va activer une motivation de fuite (cette motivation possède un coefficient d'activation élevé ce qui en fait une motivation fortement prioritaire). Ici encore, lors du second pré-traitement le plan B de fuite va être très surévalué du fait du coefficient de motivation très élevé. Même si le plan A possède toujours une forte activation, le plan B de fuite va être réalisé du fait de l'intensité de son activation supérieure à celle du plan A. Le plan A va donc être interrompu, au profit du plan B, pour être éventuellement repris plus tard.

Ces deux exemples ont permis d'illustrer deux aspects intéressants de l'architecture, la gestion de compromis entre motivations et l'interruption de plan afin de s'adapter à l'évolution des motivations et de l'environnement. Dans la section suivante nous abordons la mise en place d'apprentissage artificiel afin de permettre d'accroître cette adaptation aux changements de l'environnement.

2.2 Apprentissage artificiel

En plus des solutions offertes par le modèle, nous avons souhaité étudier la possibilité d'intégrer une part d'apprentissage, qui est indispensable pour l'autonomisation des entités afin de s'adapter aux changements de l'environnement.

Nous avons notamment abordé cet apprentissage dans le cadre de l'utilisation des systèmes de classeurs hiérarchiques dans le contexte de données symboliques (symbolique : utilisation de symboles abstraits, numérique : utilisation de données numériques), qui n'a pour l'instant jamais été étudié. Cependant, il est apparu que le mode de fonctionnement de l'architecture (cf. section 2.1.6) influence énormément les possibilités de réaliser un apprentissage. Comme présenté précédemment (cf. section 1.2), nous nous sommes intéressés aux apprentissages concernant la façon dont l'entité évalue les plans (cf. section 2.2.1) mais aussi au niveau de l'amélioration de la structure des SCH (cf. section 2.2.2). Nous avons également abordé la question de la génération partielle des plans (cf. section 2.2.3).

2.2.1 Apprentissage au niveau de l'évaluation des plans

Dans cette partie, nous montrons comment nous réévaluons le coût des actions intermédiaires ou la valeur des actions finales en utilisant des mécanismes d'apprentissage par renforcement.

Réévaluation des actions intermédiaires

Les différentes actions qui composent un plan, jusqu'à l'action finale, ont un coût pour chaque critère (coût en temps, coût en difficulté, ...). Ces coûts peuvent varier au cours du temps et il semble donc intéressant de pouvoir mettre à jour l'estimation de ces coûts en fonction de leur évolution. Ainsi, nous proposons ici un mécanisme qui permet de mettre à jour les coûts estimés des actions intermédiaires par rapport à leurs coûts réels. Nous effectuons cette mise à jour à la fin d'une action intermédiaire selon un coefficient d'apprentissage α qui va permettre de définir la rapidité de l'apprentissage.

Nous avons donc :

- ▷ CE est la valeur estimée du coût de l'action (avec ou sans utilisation d'un élément).
- ▷ CR est la valeur réelle du coût de l'action, calculée par l'entité lors de sa réalisation.
- ▷ α est est le coefficient d'apprentissage.

$$CE = CE + \alpha \times (CR - CE)$$

Le coefficient d'apprentissage est à régler afin de trouver un juste milieu entre un apprentissage trop rapide et pas forcement pertinent (α proche de 1) et un apprentissage trop lent et donc inutile (α proche de 0). Avec ce mécanisme on va pouvoir, par exemple, retenir qu'un endroit n'est plus accessible alors qu'il l'était jusqu'à présent, ou qu'un mécanisme quelconque ne fonctionne plus.

Réévaluation des actions finales et des éléments utilisés

La valeur d'une action finale, pour une motivation, peut dépendre de l'action seule (cas d'une action qui n'utilise pas d'élément) ou de la valeur de l'élément pour cette action (une verre d'eau, s'il est bu améliore la motivation « soif » efficacement, par contre s'il est jeté contre des flammes, il satisfait la motivation « éteindre un feu » de manière moyenne). On a donc pour chaque type d'élément une connaissance sur son efficacité, pour la réalisation des actions satisfaisant les motivations. Par exemple un extincteur sera très utile pour éteindre un feu mais ne permettra pas de boire, par contre une carafe d'eau est fonctionnement possiblepeu utile pour éteindre des flammes (suffisant pour des petites flammes) par contre elle permet de boire. Chaque type d'élément aura plus ou moins d'efficacité pour telle ou telle motivation.

D'une manière assez semblable aux actions intermédiaires, les valeurs des actions, ou des éléments pour ces actions, peuvent être réévaluées en fonction de l'efficacité réelle avec laquelle elles améliorent la ou les motivations qui les ont engendrées. On va donc, au moment où une action finale est active, réévaluer la qualité estimée à priori par rapport à la qualité réelle pour la ou les motivations qui l'ont élue, là encore à l'aide d'un coefficient d'apprentissage α'. Pour une action qui n'utilise aucun élément, la valeur est définie pour l'action dans le cadre d'une motivation. Et pour une action utilisant un élément, la valeur est définie au niveau de l'élément pour cette action dans le cader d'une motivation. Pour toutes les motivations concernées :

- ▷ RE est la valeur estimée de la rétribution de l'action (avec ou sans élément) pour la motivation
- ▷ RR est la valeur réelle la rétribution de l'action (avec ou sans élément) pour la motivation.
- ▷ α' est est le coefficient d'apprentissage.

$$RE = RE + \alpha' \times (RR - RE)$$

Par ce mécanisme, on peut retenir qu'une action ou un élément qui était bon pour une motivation peut avoir perdu sa qualité au cours du temps (usure d'un élément utilisé, ...) mais aussi qu'une action ou un élément qui était peu intéressant au départ peut avoir gagné en qualité.

2.2.2 Apprentissage au sein des systèmes de classeurs hiérarchiques

Nous abordons maintenant la question d'un apprentissage au niveau des SCH. Pour cela nous utilisons un renforcement au niveau des règles des SCH, mais aussi une valuation des SCH eux-mêmes.

Renforcement au niveau des règles

Un des principes fondamentaux des systèmes de classeurs est l'utilisation d'un mécanisme de choix entre les différentes règles éligibles à un instant donné (roue de la fortune, tournoi, ...). Cependant, ce mécanisme pseudo-aléatoire est très difficilement applicable à un système de classeurs hiérarchique en fonctionnement dynamique du fait des oscillations qu'il entraînerait. En effet, quand un système de classeurs classique fait un choix uniquement pour l'instant suivant, un système de classeurs hiérarchique fonctionne sur une suite d'actions. Si à chaque instant de réflexion l'entité « raisonne » de manière aléatoire, la poursuite d'un plan devient quasi-impossible du fait de la remise en cause permanente en mode dynamique.

En mode de fonctionnement statique, il redevient par contre possible de faire des choix pseudo-aléatoires concernant la sélection des règles, étant donné qu'une fois ce choix fait, le plan est effectué jusqu'au bout s'il ne recontre aucun problèmes au sein de l'environnement qui l'empêcheraient de se terminer.

Dans ce contexte, la solution la plus simple pour effectuer un renforcement sur les règles qui vont être les plus valables semble être d'utiliser un apprentissage latent qui retient les cheminements jusqu'à la fin d'un plan, afin de rétribuer les règles qui ont participé à l'obtention de la récompense finale. Cependant ce type d'apprentissage reste peu adapté aux systèmes de classeurs hiérarchiques qui définissent dès le départ une hiérarchie afin de créer des plans, alors qu'un apprentissage latent devrait permettre de mettre en place une cohérence implicite entre une suite ordonnée d'actions (qui pourraient être perçues comme un plan). De plus,

l'architecture est conçue dès le départ pour évaluer des plans (définis *a priori*) à partir des coûts intermédiaires et de la qualité de la rétribution finale. De ce point de vue, il devient difficile de pouvoir effectuer un choix entre les règles. Il conviendrait plutôt de tester les différentes règles et d'analyser pour chacune, les coûts et rétributions finales afin de faire un choix à partir de leur valeur estimée.

Le fait de devoir tester toutes les règles pose le problème de la complexité algorithmique qui va de même augmenter énormément à chaque étape d'un plan. Une solution est donc de limiter le nombre de règles applicables à chaque étape, par l'utilisation de plans peu complexes au niveau du nombre d'embranchements possibles. Cette complexité a aussi été réduite par l'utilisation d'une « attention active » qui permet à l'entité de ne s'intéresser qu'à un sous-ensemble potentiellement intéressant des éléments qu'elle connaît pour résoudre chaque plan (Si l'entité a soif, elle ne vas pas s'intéresser au pain pâté sur la table). Cette « attention active » est possible du fait que les règles utilisées indiquent les éléments ou types d'éléments intéressants à cet instant.

Nous ne nous sommes donc, pour l'instant, pas intéressés à l'idée d'utiliser des valeurs pour les règles, principalement du fait que ce choix ne correspondait pas avec l'évaluation des plans utilisés par l'architecture. Il est à noter que ce choix de ne pas utiliser de valeur pour les règles, éloigne l'utilisation que l'on fait des systèmes de classeurs hiérarchiques, de l'utilisation des systèmes de classeurs classiques (ne gérant pas explicitement de plans) avec un arbitrage de règles en fonction de leur valeur.

Cependant, il pourrait être intéressant d'approfondir cette approche afin de voir comment la valuation des règles peut intervenir dans l'évaluation des plans.

Renforcement au niveau des systèmes de classeurs hiérarchiques

La possibilité de donner une valeur à chaque système de classeurs hiérarchique, abordée précédemment, amène aussi la question de la mise en place d'un apprentissage de cette valeur.

Premièrement l'utilisation de cette valeur peut varier en fonction du mode fonctionnement. En dynamique, on pourrait éventuellement pondérer la valeur d'un plan en fonction de la valeur du système de classeurs hiérarchiques qui l'a proposé. En statique, on pourrait comme pour les règles, départager les systèmes de classeurs hiérarchiques afin de choisir lequel doit être élu, mais là encore, cela va à l'encontre du mode d'évaluation choisi qui fait une projection de la rétribution, par rapport aux coûts.

Ici encore, même si nous avons écarté pour l'instant l'utilisation d'une valuation des SCH, un apprentissage par renforcement devrait permettre de rétribuer un système de classeurs hiérarchique qui aura proposé un plan efficace et inversement.

2.2.3 Génération partielle de plans

Nous avons abordé la problématique de la génération partielle de plans par l'utilisation des algorithmes génétiques mais le contexte fortement symbolique nous a poussé à élargir nos

recherches à l'étude d'algorithmes de covering. Nous présentons maintenant ces deux aspects.

Algorithmes génétiques

Nous nous sommes intéressés à la possibilité d'utiliser des algorithmes génétiques, qui dans un contexte de manipulation de données symboliques posent certains problèmes.

Premièrement, le principe même des algorithmes génétiques est de faire des croisements et des mutations sur des individus. Cependant, même si tout en informatique n'est que '0' ou '1', donc potentiellement croisable et mutable, le résultat final peut perdre tout son sens et arriver à quelque chose d'aberrant (cf. section 1.4.1). Ceci est encore plus vrai dans le cadre de données symboliques qui ne supportent pas les mutations classiques. Si un symbole « outils » interprétable facilement par un algorithme logique quelconque devient « autuls », il ne voudra plus rien dire pour l'algorithme l'utilisant et ne pourra donc pas être traité.

Cela limite donc les possibilités d'utilisation des algoritmes génétiques sur des données symboliques à des échanges de symboles, ce qui nous a inspiré l'idée d'utiliser un « pool génétique » contenant tous les symboles potentiellement utilisables dans un contexte précis. Si une règle concerne le domaine A, on peut effectuer des échanges au sein du « pool génétique » Ac pour la partie condition de la règle et du « pool génétique » Ae pour la partie effet de la règle.

Une autre caractéristique des systèmes de classeurs hiérarchiques restreint la possibilité d'utiliser des algorithmes génétiques. En effet, la structure hiérarchique des règles d'un système de classeurs hiérarchique est assez sensible, il suffirait d'une mutation dans une seule des règles du plan pour rendre impossible sa réalisation. Si une étape est omise dans un plan en cours, la suite du plan ne peut plus se réaliser, cette structure hiérarchique est donc à conserver, ce que ne permet pas un algorithme génétique classique.

Au vu de ces problèmes nous avons rejeté l'idée d'utiliser des algorithmes génétiques classiques[2] mais nous avons cherché à élaborer un algorithme de *covering* (cf. section 1.3.1) afin de permettre de trouver certaines étapes de plans automatiquement.

Algorithmes de covering

Le principe de base des algorithmes de *covering* est de créer des règles éligibles par l'entité à un instant donné. Pour cela, la partie condition de la règle correspond à l'état actuel dans lequel se trouve l'entité et la partie action est générée.

La structure principale d'un système de classeurs hiérarchique est définie par les classeurs internes qui définissent les étapes d'un plan. L'évolution de l'état interne de l'entité, le place dans les conditions pour réaliser la suite du plan. Nous avons réfléchi au moyen de trouver le cheminement entre les étapes du plan (comment passer de tel classeur interne au classeur interne suivant). Pour cela, nous partons de l'état de départ de l'entité et des classeurs internes donnés *a priori*, dans l'ordre dans lequel ils doivent être élus, définissant la structure du plan.

[2] Robert (2005) à lui aussi repoussé l'idée à cause du caractère symbolique des règles.

Par exemple :

- ▷ Le classeur interne 1 est : « Je passe en mode recherche d'un objet de type A »
- ▷ Le classeur interne 2 est : « J'ai un objet de type A dans la main ET je suis en mode recherche d'un objet de type A => je passe en mode recherche d'un objet de type B »
- ▷ Le classeur interne 3 est : « J'ai un objet de type B dans la main ET je suis en mode recherche d'un objet de type B => je passe en mode utilisation des objet de type A et B »
- ▷ Le classeur interne 4 est : « J'ai les objets de type A et B dans la main ET je suis en mode utilisation des objet de type A et B => je passe en mode préparation des objets de type A et B »
- ▷ ...

Le premier classeur interne nous place donc dans un état de recherche afin d'atteindre le second classeur interne. Pour atteindre ce second classeur interne, nous orientons notre recherche vers un objet ou un type d'objet particulier (ici un objet de type A) qui correspond à nos besoins. Au lieu de chercher la transition entre deux classeurs internes de manière complètement aléatoire, nous nous focalisons donc sur un seul élément du type qui est souhaité et l'on teste toutes les actions possibles le concernant. De ces actions nous gardons la « meilleure » [3] afin de créer la règle externe correspondante et nous continuons ainsi jusqu'à être dans l'état désiré par le second classeur interne. Ceci est répété jusqu'à avoir atteint tous les classeurs internes et avoir obtenu une rétribution finale.

Dans le cadre de l'exemple précédent pour passer du classeur interne 1 au classeur interne 2, l'entité crée les règles lui permettant d'atteindre et de prendre un objet de type A, ce qui lui permet d'activer le classeur interne 2 pour continuer la réalisation du plan.

Cette approche a été testée sur des exemples relativement simples où le plan consiste à collecter des éléments puis à effectuer une action sur eux pour au final satisfaire une motivation. Nous avons ainsi pu trouver de manière dynamique les classeurs externes nécessaires à la réalisation du plan, en ne précisant au départ que les étapes du plan.

2.2.4 Bilan sur l'apprentissage

Notre étude concernant l'apprentissage nous a permis de mieux situer les problèmes liés à un apprentissage avec des données symboliques et surtout dans le contexte des systèmes de classeurs hiérarchiques qui sont conceptuellement différents des systèmes de classeurs classiques. En effet, un système de classeurs classique va choisir une seule action à réaliser alors qu'un système de classeurs hiérarchique fait un choix à plus long terme en déroulant un plan, ce qui en complique l'apprentissage.

[3] L'estimation de la qualité des actions au sein de l'algorithme de *covering* peut dépendre de critères différents, en général il s'agit de la distance nous séparant de l'objet visé.

Les apprentissages concernant l'évaluation que fait l'entité sur l'efficacité des actions (intermédiaires et finales) et des éléments présents dans l'environnement permet une adaptation, mais pas de réévaluations des plans, ceux-ci sont considérés valables *a priori* et nous recherchons la meilleure exécution et combinaison possible afin de satisfaire toutes les motivations.

Les problèmes de la valuation des règles (comme dans les systèmes de classeurs classiques) et des systèmes de classeurs hiérarchiques, ainsi que de l'utilisation qui est faite de ces valeurs restent très complexes et le cadre du stage n'aura permis que de les aborder rapidement au travers de quelques pistes. Cependant, certains points intéressants notamment au niveau de la dualité des fonctionnements statique et dynamique des systèmes de classeurs hiérarchiques, et des conséquences que cela entraine sur l'apprentissage auront été abordés et pourraient être l'objet d'une étude plus poussée.

Notre étude concernant les algorithmes génétiques, qui sont par définition peu compatibles avec des données symboliques, a permis de trouver quelques éléments qui semblent intéressants comme l'utilisation de « pools génétiques » afin de limiter les créations et mutations de règles dans un domaine restreint. Finalement, l'utilisation d'un algorithme de *covering* orienté vers les buts a été testée et a permis de trouver des cheminements entre les différentes étapes (définies *a priori*) d'un plan.

Nous retiendrons finalement que la mise en place d'apprentissage interne aux systèmes de classeurs hiérarchiques dans un contexte symbolique n'est pas aisée, de par la structure même des systèmes de classeurs hiérarchiques qui sont conçus pour suivre un plan. De ce point de vue il nous a semblé plus cohérent d'effectuer cet apprentissage sur les connaissances qu'a l'entité sur l'environnement et l'efficacité des actions. Nous avons de plus, réussi à mettre en place un apprentissage interne aux systèmes de classeurs hiérarchiques par le biais de l'algorithme de *covering* qui permet de simplifier le travail de modélisation de plans en ne définissant que les étapes principales de ceux-ci.

— Conclusion —

Nous nous sommes intéressés, tout au long de cette étude, au problème de la modélisation de comportements complexes pour des entités virtuelles multi-motivées dans des environnements dynamiques. En effet il se pose la question de savoir comment rendre crédibles les comportements des entités au sein de ces environnements. Nous partons du principe que dans le cadre d'un environnement ouvert et dynamique, cette crédibilité implique une certaine autonomie, afin qu'une entité découvrant une situation inconnue sache adapter son comportement pour y faire face. Cette adaptation va être possible grâce à la mise en place d'un mécanisme apprentissage artificiel, afin que l'entité acquière de la connaissance à partir d'autres entités présentes dans l'environnement (apprentissage supervisé, par observation et imitation) ou de son expérience personnelle (apprentissage par renforcement). Bien que dans un contexte faiblement symbolique (où l'on ne manipule pas de symboles abstraits) cette autonomie puisse être abordée « plus simplement » au détriment de la complexité du comportement modélisé (par exemple un réseau de neurones possède des capacités d'apprentissage mais ne permet pas de comportements de haut niveau), dans un contexte fortement symbolique, permettant des comportements cognitifs plus complexes, cette autonomie est difficile à aborder, du fait de l'utilisation des symboles qui sont peu adaptés à certaines formes d'apprentissage artificiel. Malgré les nombreux travaux dans le domaine, il n'y a pour l'instant, pas de réponse satisfaisante au problème de la modélisation de comportements complexes crédibles et autonomes dans le cadre de simulations en environnements ouverts et dynamiques. Notre approche propose de rechercher des compromis afin de satisfaire plusieurs motivations dans un environnement dynamique.

Bilan

Nous avons conçu une architecture hiérarchique qui permet de gérer plusieurs motivations qui doivent être satisfaites par la réalisation de plans visant des objectifs. Ces motivations activent des SCH qui proposent des plans dans le but de satisfaire ces motivations puis un mécanisme de décision va évaluer ces plans afin de rechercher les meilleurs compromis entre toutes les motivations actives. Nous avons aussi abordé les modes de fonctionnement possibles

pour notre architecture (statique et dynamique), ce qui nous à permis de bien cerner les conséquences de chacun, notamment au niveau de l'exploration de nouvelles solutions et des possibilités de mise en place d'un apprentissage. Le mode de fonctionnement dynamique ne permet pas d'exploration mais offre la possibilité d'interrompre des plans contrairement au mode statique qui permet de faire de l'exploration mais pas d'interruption de plans.

Nous nous sommes intéressés au moyen de réaliser un apprentissage afin de permettre à notre entité de s'adapter aux changements potentiels de l'environnement. Pour cela, nous avons mis en place un mécanisme d'apprentissage qui permet de remettre en cause les connaîssances sur l'environnement (coûts d'exécution des actions et de qualités des éléments et des actions pour les motivations). Nous avons aussi souligné que la mise en place d'apprentissage au sein des SCH est complexe, du moins dans un contexte fortement symbolique. Cependant, les pistes abordées concernant la valuation des règles et des SCH seraient à explorer plus en détail. Concernant l'apprentissage partiel de plan, nous avons aussi utilisé un algorithme de *covering* afin de rechercher de manière automatique, les règles permettant de réaliser un plan dont les étapes ont été définies *a priori*, ce qui simplifie la modélisation des comportements.

Ces résultats prometteurs ne sont cependant qu'un début et ils ouvrent la voie vers de nombreuses perspectives de recherches.

Perspectives

De nombreuses perspectives découlent de ces travaux, que ce soit au niveau de l'architecture ou de l'apprentissage. En effet, la réflexion que nous avons eue nous a permis d'élaborer cette architecture qui nous ouvre des perspectives plus larges notamment au niveau de la gestion des motivations, l'évaluation des plans, la réalisation de compromis et l'apprentissage au sein de ce type d'architecture. Cette architecture et son implémentation ne sont pour l'instant qu'une ébauche de solution et chacun des points qui y ont été abordé mériterait une étude approfondie.

Motivations

Les liens entre les motivations et les influences qu'elles ont les unes sur les autres. Il en est de même au niveau des SCH quant à la notion de lien entre eux concernant leurs activations et inhibitions mutuelles. La notion d'émotion et de caractère peut aussi jouer un rôle prépondérant sur la gestion des motivations et mériterait une étude approfondie.

Apprentissage

Nous avons pu constater la difficulté de mettre en place un apprentissage au niveau des SCH du fait de leur structure assez rigide, ils serait donc intéressant d'étudier la possibilité

d'aborder cette gestion des plans par un autre système que celui-ci ou encore d'effectuer un travail plus poussé sur le problème de la mise en place d'apprentissage au sein de SCH, et notamment au niveau de la valuation potentielle des règles et des SCH eux-mêmes.

La façon dont nous avons abordé la recherche de solutions entre les étapes d'un plan à partir d'un algorithme de *covering* n'est qu'une ébauche et les exemples de plans utilisés pour tester ce mécanisme étaient simples. Un effort pour explorer cette approche de recherche de plans serait bienvenue.

Gestion des compromis

Notre approche de la gestion des compromis, se faisant au niveau de la prochaine étape de chaque plan, limite la recherche de compromis à une vue relativement courte, il serait donc intéressant d'étudier la possibilité de rechercher ces compromis à plus long terme avec par exemple le nombre d'étapes en commun entre deux plans et les moments du plan où ont lieu ces croisements.

Evaluation multi-critères

Au niveau de l'évaluation des plans dans un contexte multi-critères (coûts en temps, coûts en energie...), des études plus poussée seraient possibles pour creuser cet aspect afin de pouvoir tenir compte de l'impact de l'état « mental » de l'entité sur l'utilisation de tel ou tel critère prioritaire (une entité pressée ne va s'intéresser qu'au coût en temps en essayant de le minimiser).

Partage de ressources

Le mécanisme de gestion des ressources qui permet à plusieurs actions de se les partager serait aussi améliorable afin de permettre l'utilisation de ressources en collaboration entre plusieurs entités, et une étude au niveau de la concurrence et collaboration par plusieurs entités serait intéressante.

Complexité de l'exemple

De nombreuses choses restent à faire dans ce domaine et notre architecture présente quelques pistes intéressantes qui nécessitent une recherche plus spécifique et approfondie. En effet l'application réalisée est simple (deux motivations, une seule entité dans l'environnement) mais un développement est envisagé pour tester notre architecture dans un contexte plus poussé de réalité virtuelle et de jeux vidéo. Nous pensons donc complexifier les comportements, augmenter le nombre de motivations et d'entités présentes dans l'environnement.

Anticipation

Le fait d'avoir de nombreuses entités dans l'environnement nous amène donc à refléchir sur la question de l'anticipation du comportement des autres entités, ce qui devrait améliorer le mécanisme de décision. Notre approche actuelle peut être vue comme un prémice à l'anticipation, dans la mesure où l'on essaye d'anticiper les coûts et gains potentiels d'un plan. Il serait intéressant d'aller plus loin dans cette voie en essayant de se faire un modèle du fonctionnement des entités qui nous entourent afin d'anticiper leur comportement et leurs réactions (c'est ce que nous faisons tout les jours dans la vie quotidienne). De même, l'anticipation des conséquences d'une action réalisée par une entité permettrait un gain au niveau de la prise de décision pour des entités en environnement virtuel. Nous pensons que cette approche de l'anticipation peut améliorer la prise de décision automatique dans tous les domaines de l'informatique.

— Références bibliographiques —

Anderson, J. et Lebiere, C. (1998). The atomic components of thought. *Lawrence Erlbaum Associates*.

Barry, A. (1993). The Emergence of High Level Structure in Classifier Systems - A Proposal. *Irish Journal of Psychology*, 14(3) :480–498.

Barry, A. (1996). Hierarchy Formulation Within Classifiers System – A Review. Dans Goodman, E. G., Uskov, V. L., et Punch, W. F., éditeurs, *Evolutionary Computation and Its Applications (EVCA'96)*, pages 195–211. The Presidium of the Russian Academy of Sciences.

Bernardo, E., Llorà, X., et Garrel, J. M. (2001). Xcs and gale : a comparative study of two learning classifier systems with six another learning algorithms on classification tasks. Dans *Proceedings of the fourth international workshop on Learning Classifier Systems*.

Bratman, M. (1987). Intentions, plans and practical reason. *Harvard university Press*.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1) :14–23.

Buche, C., Septseault, C., et De Loor, P. (2006a). Les systèmes de classeurs. une présentation générale. *Revue des Sciences et Technologies de l'Information, série Techniques et Sciences Informatiques (RSTI–TSI)*, 25(8-9) :963–990.

Butz, M. V., Goldberg, D. E., et Stolzmann, W. (2000). Introducing a genetic generalization pressure to the anticipatory classifier system - part 1 : Theoretical approach. Dans Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., et Beyer, H.-G., éditeurs, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 34–41, Las Vegas, Nevada, USA. Morgan Kaufmann.

Clancey, W., Sachs, P., Sierhuis, M., et van Hoof, R. (1996). Brahms : Simulating practice for work systems design.

Coquelle, L., Buche, C., et Chevaillier, P. (2004). Un langage à base de logique floue pour la simulation de comportements individuels d'animaux. Dans *Rencontres Francophones sur la Logique Floue et ses Applications (LFA'04)*, pages 379–386. Cépadues–Éditions.

De Sevin, E. (2006). *An Action Selection Architecture for Autonomous Virtual Humans in Persistent Worlds*. Thèse de doctorat, École Polytechnique Fédérale De Lausanne.

Del Bimbo, A. et Vicario, E. (1995). Specification by example of virtual agents behavior. *IEEE transactions on visualization and computer graphics*, 1(4).

Donikian, S. (2004). *Modélisation, contrôle et animation d'agents virtuels autonomes évoluant dans des environnements informés et structurés*. Habilitation à diriger les recherches, IRISA.

Donikian, S. et Rutten, E. (1995). Reactivity, concurrency, data-flow and hierarchical preemption for behavioural animation. *Programming paradigms in graphics*.

Donnart, J.-Y. et Meyer, J.-A. (1994). A hierarchical classifier system implementing a motivationally autonomous animat. Dans *From Animals to Animats 3. Proceedings of the Third International Conference on Simulation of Adaptive Behavior (SAB94)*, pages 144–153. A Bradford Book. MIT Press.

Dorigo, M. (1995). Alecsys and the AutonoMouse : Learning to Control a Real Robot by Distributed Classifier Systems. *Machine Learning*, 19 :209–240.

Girard, B., Robert, G., et Guillot, A. (2002). Jeu vidéo et intelligence artificielle située. *In Cognito*, 22.

Goldberg, D. E. (1989b). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

Heguy, O. (2003). *Architecture comportementale pour l'émergence d'activités coopératives en environnement virtuel*. Thèse de doctorat, Université Paul Sabatier.

Holland, J. et Reitman, J. (1978). Cognitive systems based on adaptive algorithms. Dans Waterman, D. A. et Hayes-Roth, F., éditeurs, *Pattern-directed inference systems*. New York : Academic Press. Reprinted in : Evolutionary Computation. The Fossil Record. David B. Fogel (Ed.) IEEE Press, 1998.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems : An Introduction Analysis with Applications to Biology, Control, and Artificial Intelligence*. university of Michigan Press.

Holland, J. H. (1985). Properties of the bucket brigade algorithm. Dans *International Conference on Genetic Algorithms and Their Applications*, pages 1–7.

Isla, D., Burke, R., Downie, M., et Blumberg, B. (2001). A layered brain architecture for synthetic characters. Dans *Proceedings of the seventeenth international conference on artificial intelligence (IJCAI-01)*, pages 1051–1058. Morgan Kaufmann Publishers, Inc.

Kovacs, T. (1997). XCS Classifier System Reliably Evolves Accurate, Complete, and Minimal Representations for Boolean Functions. Rapport technique CSRP-97-19, School of Computer Science, University of Birmingham, Birmingham, U.K.

Laird, J. E., Newell, A., et Rosenbloom, P. S. (1987). Soar : an architecture for general intelligence. *Artificial Intelligence*, 33(1) :1–64.

Lamarche, F. (2004). *Humanoïdes virtuels, réaction et cognition : une architecture pour leur autonomie.* Thèse de doctorat, Université de Rennes I.

Maes, P. (1991). The agent network architecture (ana). *SIGART Bulletin*, 2(4) :115–120.

Miclet, L. et Cornuéjols, A. (2002). *Apprentissage artificiel, Concepts et algorithmes.* Eyrolles.

Reynolds, C. W. (1987). Flocks, herds and schools : a distributed behavioral model. Dans C.), S. M., éditeur, *Computer Graphics (SIGGRAPH'87 Proceedings)*, pages 25–34.

Robert, G. (2002b). Contribution des méthodologies animat et multi-agent à l'élaboration des jeux en ligne, persistants et massivement multi-utilisateurs.

Robert, G. (2005). *MHiCS, une architecture de sélection de l'action Motivationnelle et Hiérarchique à Systèmes de Classeurs pour Personnages Non Joueurs adaptatifs.* Thèse de doctorat, LIP6/AnimatLab, Université Pierre et Marie Curie.

Rosenblatt, J. K. et Payton, D. W. (1989). A fine-grained alternative to the subsumption architecture for mobile robot control. Dans *Proc of the IEEE Int. Conf. on Neural Networks*, volume 2, pages 317–324, Washington, DC. IEEE Press.

Sanchez, S. (2004). *Mécanismes évolutionnistes pour la simulation comportementale d'acteurs virtuels.* Thèse de doctorat, Université des Sciences Sociales Toulouse I.

Sanza, C. (2001). *Evolution d'Entités Virtuelles Coopératives par Système de Classifieurs.* Thèse de doctorat, Université Paul Sabatier.

Schmidt, B. (2000). The modelling of human behaviour. *The society for computer simulation international.*

Sigaud, O. (2007). Les systèmes de classeurs : un état de l'art. *Revue d'Intelligence Artificielle*, 21 :75–106.

Smith, S. (1980a). *A Learning System Based on Genetic Algorithms.* Thèse de doctorat, University of Pittsburgh.

Steegmans, E., Weyns, D., Holvoet, T., et Berbers, Y. (2004). Designing roles for situated agents. pages 17–32.

Stolzmann, W. (1998). Anticipatory classifier systems. Dans *Third Annual Genetic Programming Conference*, pages 658–664. Morgan Kaufmann.

Sutton, R. (1991). Reinforcement learning architectures for animats. Dans Meyer, J.-A. et Wilson, S. W., éditeurs, *From Animals to Animats : First International Conference on Simulation of Adaptive Behavior (SAB91)*, pages 288–296.

Tinbergen, N. (1951). *L'étude de l'instinct.* Payot.

Tisseau, J. (2001). *Réalité Virtuelle - autonomie in virtuo.* Habilitation à diriger des recherches, Université de Rennes I.

Van de Panne, M. et Fiume, E. (1993). Sensor-actuator networks. Dans T.), K. J., éditeur, *Computer Graphics (SIGGRAPH'93 Proceedings)*, pages 335–342.

Watkins, C. J. (1989). *Learning from delayed rewards.* Thèse de doctorat, Cambridge university.

Wilson, S. W. (1994). ZCS : A zeroth level classifier system. *Evolutionary Computation*, 2(1) :1–18.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2) :149–175.

Wilson, S. W. et Goldberg, D. E. (1989). A critical review of classifier systems. pages 244–255.

Nom Prénom : Ewen Creac'h

Nom de l'encadreur : Cédric Buche

Centre Européen de Réalité Virtuelle
25 rue Claude Chappe
BP 38 F-29280 PLOUZANÉ, France

Modélisation de comportements adaptatifs par systèmes de classeurs hiérarchiques

Contexte :

La problématique de réaliser des entités dotées de comportements autonomes est au coeur des recherches actuelles dans le domaine de la simulation comportementale en réalité virtuelle. Dans cette optique, une approche est de vouloir doter les entités de capacités d'adaptations aux changements de l'environnement. Un système de classeurs est un mécanisme de décision à base de règles qui offre la possibilité de réaliser un apprentissage par renforcement (l'entité ou le système apprenant va renforcer les règles considérées comme bonnes et dévaloriser les mauvaises). De plus un algorithme évolutionniste permet d'améliorer l'apprentissage en effectuant des croisements et des mutations entre les meilleures règles. Un système de classeurs est donc un mécanisme de décision adaptatif qui offre de bon résultats dans de nombreux domaines. Cependant dans un contexte de modélisation de comportements multi-motivations, avec une nécessité de réaliser des tâches planifiées, les systèmes de classeurs classiques sont rapidement limités. Les systèmes de classeurs hiérarchiques permettent, de par leur structure interne de réaliser des tâches complexes qui peuvent nécessiter un ordonnancement. Cependant les notions d'apprentissage (renforcement et algorithmes génétiques) ne sont que très peu abordées dans le cadre des systèmes de classeurs hiérarchiques, nous allons donc travailler sur ce point précis afin d'essayer de cerner les possibilités dans ce domaine.

Travail réalisé :

L'application réalisée sera donc un mécanisme de décision pour une entité dotée de plusieurs motivations. Chacune des motivations va proposer ses choix concernant les prochaines actions à réaliser, en fonction de son niveau d'activation, de son importance et de son ou ses systèmes de classeurs hiérarchiques. Au niveau inférieur un mécanisme choisira le meilleur compromis entre tous les choix proposés afin de satisfaire au mieux toutes les motivations. Nous travaillerons sur le moyen de réaliser un apprentissage par renforcement avec ce mécanisme de décision ainsi que la découverte des plans satisfaisant les motivations par algorithmes génétiques et de *covering*.

Domaines d'application :

Réalité Virtuelle, mécanismes de décisions, modélisation d'entités multi-motivées, IA, animation comportementale, sciences cognitives

Environnements :

C++, UML