Une société d'agents temporels pour la supervision de systèmes industriels

Mohamad Khaled Allouche

THÈSE

présentée par

M. Allouche

pour obtenir le grade de Docteur
de l'Université Jean Monnet
et
de l'Ecole Nationale Supérieure des Mines de Saint-Etienne
(Arrêté ministériel du 30 mars 1992)

Spécialité : informatique

Une société d'agents temporels pour la supervision de systèmes industriels

Soutenue le 15 Octobre 1998

Président :	Mme. Suzanne Pinson
Rapporteurs :	Mme. Catherine Garbay
	M. Flavio Oquendo
Examinateurs :	M. Olivier Boissier
	M. Michel Dumas
	M. Philippe Ezequel
	Mme. Claudette Sayettat

Thèse préparée au sein du **L**aboratoire **S**ystèmes **I**ndustriels **C**oopératifs

THÈSE

présentée par

M. Allouche

pour obtenir le grade de Docteur
de l'Université Jean Monnet
et
de l'Ecole Nationale Supérieure des Mines de Saint-Etienne
(Arrêté ministériel du 30 mars 1992)

Spécialité : informatique

Une société d'agents temporels pour la supervision de systèmes industriels

Soutenue le 15 Octobre 1998

Président :	Mme. Suzanne Pinson
Rapporteurs :	Mme. Catherine Garbay
	M. Flavio Oquendo
Examinateurs :	M. Olivier Boissier
	M. Michel Dumas
	M. Philippe Ezequel
	Mme. Claudette Sayettat

Thèse préparée au sein du **L**aboratoire **S**ystèmes **I**ndustriels **C**oopératifs

Je tiens à remercier :

- Mme Suzanne Pinson qui m'a fait l'honneur de présider ce jury.
- Mme Claudette Sayettat de m'avoir accueilli dans son équipe. Je lui suis reconnaissant pour les conseils et la confiance qu'elle m'a accordée.
- Mme Catherine Garbay et Mr. Flavio Oquendo d'avoir accepté d'être rapporteurs de cette thèse.
- Mr. Michel Dumas et Mr. Philippe Ezequel d'avoir accepté de faire partie du jury.
- tous les membres de l'équipe "Systèmes Multi-Agents", pour leur convivialité et l'intérêt qu'ils ont porté à mon travail, ainsi que pour leur aide.
- l'ensemble des membres de SIMADE pour leur sympathie.

Je remercie Olivier Boissier à la fois pour son encadrement, ses conseils scientifiques et ses qualités humaines. Je remercie également sa famille de m'avoir soutenu pendant les moments difficiles.

David et Elke, merci de votre présence qui m'a beaucoup aidé.

Mes parents, je ne saurais jamais comment vous remercier ... je me contente de vous dédier ce manuscrit.

Table des matières

Liste des figures

Liste des exemples

Chapitre 1

Introduction

1.1 Problématique

La complexité croissante des *systèmes industriels* en termes d'expansion, d'hétérogénéité et en terme de décentralisation entraîne de plus en plus de contraintes dans leur fonctionnement. Ces contraintes sont souvent d'ordre temporel. Par exemple, dans un atelier de production composé d'un ensemble de machines, il existe des contraintes temporelles sur les tâches pouvant être exécutées sur les différentes machines. L'ordre et la durée des tâches sont souvent variables. Ils dépendent d'un ensemble de facteurs tels qu'une quantité suffisante de matière première ou alors la terminaison d'autres tâches.

De tels systèmes industriels sont dits *dynamiques*, *décentralisés* et *complexes* :

- un système est dit *dynamique*, s'il présente des changements d'états au cours de son fonctionnement. Ces changements peuvent être variables et imprévisibles ;
- un système est dit *décentralisé*, s'il n'existe pas de contrôleur central permettant de gérer l'ensemble de traitements du système ;
- la notion de *complexité* d'un système est liée à sa taille et à la forte interaction qui peut exister entre les différents éléments du système. Plus cette interaction est forte, plus les traitements deviennent difficiles à modéliser et à exécuter. Dans certains cas, la complexité d'un système devient un critère de décentralisation.

1.1.1 Prise en compte du temps dans les systèmes industriels

Afin de traiter des problèmes relatifs à de tels systèmes, les chercheurs ont été amenés à représenter explicitement le *facteur temporel* dans les outils de modélisation et de raisonnement sur ces systèmes.

Cette prise en compte du temps, a rendu ces outils capables d'exécuter et gérer des tâches soumises à des contraintes temporelles. Ces contraintes interviennent au sein d'une tâche (contraintes intérieures) et/ou entre les différentes tâches (contraintes extérieures). Les contraintes intérieures et extérieures d'une tâche ne sont pas toujours indépendantes. En effet, la durée d'exécution d'une tâche peut causer la violation d'une contrainte avec une autre tâche, si cette exécution est trop longue.

Cette dépendance entre les différentes contraintes rend leur gestion problématique quand il s'agit d'un système décentralisé et complexe. En effet, une centralisation des traitements étant absente dans un tel système, il devient difficile d'avoir une information complète sur l'ensemble des contraintes qui le régissent. La complexité du système se traduit par une complexité de ces contraintes et leur gestion devient de ce fait, encore plus difficile.

1.1.2 Le temps dans les systèmes multi-agents

Afin de répondre à un tel problème, des efforts ont été déployés pour adapter les outils de modélisation à la structure complexe et décentralisée de tels systèmes. Pour ce faire, de nouvelles techniques ont été adoptées telles que les *systèmes multi-agents*. Dans ces systèmes, il est possible de faire un ensemble de traitements par un ensemble d'*entités autonomes* appelées *agents*. Ces agents possèdent en plus des capacités de communication et de raisonnement sur autrui, leur permettant ainsi de *coopérer* afin de traiter un ensemble de sous-problèmes faisant partie d'un problème global.

Afin de répondre à la dynamique des systèmes, les formalismes utilisés hors du contexte multi-agent ont été réutilisés. Ainsi, le facteur temporel est pris en compte, au niveau de l'activité interne d'un agent uniquement. Une telle prise en compte demeure insuffisante. En effet, dans un système multi-agent, les agents font généralement partie d'un processus de *coopération*. Ainsi, il est impossible pour un agent de considérer son activité interne indépendamment de celles des autres agents. Cette *dépendance* entre les activités, doit également être prise en compte dans le raisonnement de cet agent. Pour ce faire, il doit admettre non seulement une *description du système industriel*, mais également une *description des autres agents*. Ainsi, son raisonnement porte sur son activité interne (*raisonnement individuel*) et sur celles des autres (*raisonnement social*).

Malheureusement, le facteur temporel est souvent négligé lors de la conception d'un système multi-agent, alors qu'il joue un rôle essentiel dans un contexte où l'activité des agents est en état d'évolution permanente.

Afin de traiter le temps dans un système multi-agent, nous avons choisi la *supervision de systèmes dynamiques* comme domaine d'application. La supervision d'un système dynamique suppose l'existence de plusieurs *modes de fonctionnement* possibles. Elle assure la *détection* d'un passage d'un mode de fonctionnement normal à un mode anormal, et l'*identification* des origines

d'une anomalie qui cause ce passage.

Dans un contexte de supervision, la modélisation d'un système dynamique admet souvent une représentation explicite du facteur temporel.

1.2 Objectif et contribution

L'objectif général de ce travail, est l'intégration des aspects temporels dans un système multi-agent. La supervision de systèmes dynamiques permettra d'illustrer ces aspects.

Nous proposons un modèle de société d'agents où le facteur temporel est pris en compte à la fois au niveau du raisonnement individuel et au niveau du raisonnement social :

- le raisonnement individuel permet la gestion des contraintes temporelles relatives aux tâches à effectuer au sein d'un agent ;
- le raisonnement social est basé sur la notion de *relations de dépendance*, il tient compte des aspects dynamiques relatifs à ces relations, qui sont fonctions de l'évolution des activités des agents.

L'utilisation du raisonnement social favorise la *communication* entre agents. Nous proposons, un modèle de communication permettant l'expression des contraintes temporelles sous forme de délais.

Dans notre contexte, la supervision se fait par *reconnaissance par scénarios temporels*. Un scénario modélise un mode de fonctionnement particulier du système supervisé et sa reconnaissance, correspond à l'identification de ce mode.

Afin de distribuer la reconnaissance de scénarios, le modèle de scénario doit être adapté à un contexte où plusieurs agents peuvent reconnaître des scénarios en parallèle.

1.3 Organisation du document

Le présent manuscrit est organisé en trois parties. Chaque partie est composée de trois chapitres (figure 1.1).

Première partie : Systèmes multi-agents, raisonnement temporel et supervision de systèmes dynamiques

Dans cette partie, nous analysons les liens existant entre trois domaines : les systèmes multi-agents, le raisonnement temporel et la supervision de systèmes dynamiques.

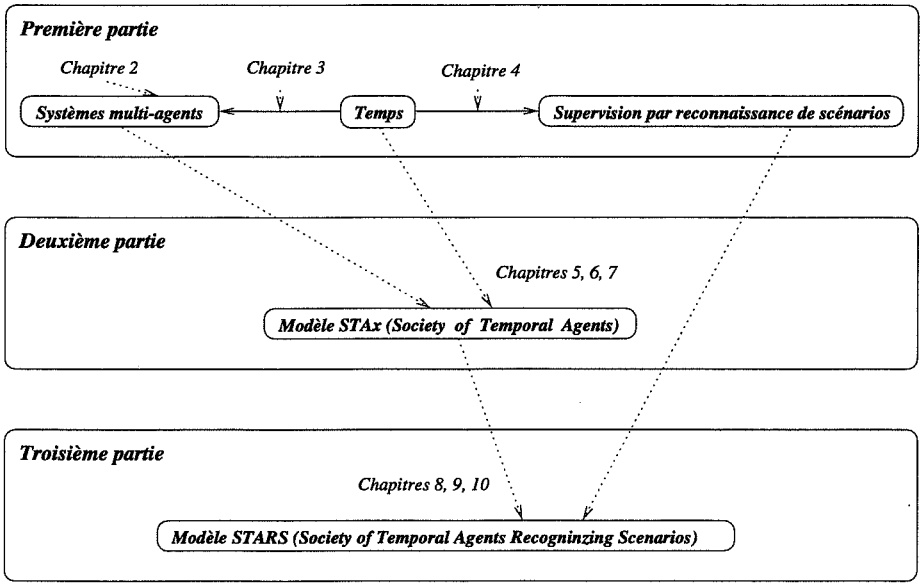


Figure 1.1 : Organisation du document.

Chapitre 2 : Systèmes multi-agents

Ce chapitre présente d'une manière générale les systèmes multi-agents à travers l'analyse d'un certain nombre d'aspects importants dans la cadre de notre travail tels que la coopération, les relations sociales et la communication.

Cette analyse conduit à la définition de deux niveaux au sein de l'agent : le *niveau individuel* et le *niveau social.*

Chapitre 3 : Raisonnement temporel dans les SMA

Ce chapitre présente le lien entre les systèmes multi-agents et le raisonnement temporel. D'abord, nous présentons des formalismes pour la représentation du temps dans un contexte général. Ensuite, nous présentons la prise en compte du temps dans un système multi-agent. Cette prise en compte est analysée à la fois au niveau individuel et au niveau social.

Chapitre 4 : Supervision des systèmes dynamiques

Dans ce chapitre, nous présentons le domaine de la supervision de systèmes dynamiques. Nous montrons à travers l'analyse d'un certain nombre d'approches, l'importance de la représentation explicite du facteur temporel au sein des modèles utilisés dans ces approches.

Son étude dans la supervision fait d'elle un domaine d'application bien adapté pour la prise en compte du temps dans un système multi-agent.

Deuxième partie : Société d'agents temporels

Dans cette partie, nous présentons le modèle élaboré *STAx* (*Society of Temporal Agents*) permettant la prise en compte du temps dans un système multi-agent.

Chapitre 5 : Modèle d'organisation temporelle

Nous proposons un modèle de société basé sur des relations de dépendance temporelles. Ces relations seront utilisées dans le raisonnement social des agents.

Chapitre 6 : Modèle d'interaction temporelle

L'utilisation des relations de dépendance dans le raisonnement, pousse les agents à communiquer. Le modèle de communication proposé permet l'expression de contraintes temporelles au niveau du langage et de sa sémantique. Afin de contrôler les communications entre agents, nous avons défini un ensemble de protocoles d'interaction.

Chapitre 7 : Modèle d'agent temporel

Dans ce chapitre, nous proposons un modèle d'agent adapté au modèle de société proposé dans le chapitre 5. Dans ce modèle, le facteur dynamique est pris en compte dans chaque fonction du niveau individuel et du niveau social de l'agent.

Troisième partie : Système *STARS*

Dans cette partie, nous présentons le *système STARS*, instanciation du modèle précédent à une société d'agents chargée de la supervision d'un système dynamique par reconnaissance de scénarios.

Chapitre 8 : Reconnaissance de scénarios temporels

Nous décrivons tout d'abord le modèle des scénarios et leurs modes de reconnaissance. Nous proposons ensuite, une extension du modèle pour pouvoir l'utiliser dans un contexte multi-agent. Nous montrons également comment l'application du modèle contribue à la résolution d'un certain nombre de problèmes liés à la supervision.

Chapitre 9 : Application à la supervision

Dans ce chapitre, nous présentons une instantiation du modèle de société décrit dans le chapitre 5, à un système de supervision par reconnaissance de scénarios. L'accent sera mis sur le raisonnement social des agents à travers

un exemple de supervision d'une flotte de bus dans le cadre du transport en commun.

Chapitre 10 : Implémentation du modèle

Ce chapitre présente une implémentation du modèle d'agent proposé dans le chapitre 7.

Première partie

Systèmes multi-agents, raisonnement temporel et supervision de systèmes dynamiques

Introduction

L'objectif de cette partie est d'analyse les liens entre trois domaines : les systèmes multi-agents, le temps et la supervision de systèmes dynamiques. Chacun sera traité dans un chapitre :

- le premier chapitre, permet de donner une vision globale des systèmes multi-agents, en mettant l'accent sur des aspects importants telles que la coopération, les relations sociales et la communication ;
- le deuxième chapitre, fait le lien entre les systèmes multi-agents et le raisonnement temporel. Ainsi, sont présentés des travaux sur la représentation du temps d'une manière générale et plus spécifiquement dans les systèmes multi-agents ;
- enfin, le dernier chapitre présente le domaine de la supervision à travers l'analyse d'un certain nombre d'approches. Cette analyse doit nous permettre de constater que dans un tel domaine d'application, la prise en compte du temps est indispensable.

Ces mises en relation de la supervision avec le temps, d'un coté et des systèmes multi-agents avec le temps d'un autre coté, font de la supervision un domaine d'application bien adapté si l'on souhaite traiter les aspects temporels dans un système multi-agent.

Chapitre 2

Systèmes multi-agents

2.1 Introduction

La complexité croissante des systèmes industriels et la délocalisation des traitements font de plus en plus appel à l'utilisation de nouvelles techniques où les traitements peuvent être décentralisés et pris en charge par des *entités* appelées *agents*. Outres leurs capacités individuelles à résoudre des problèmes, à exécuter des tâches, ces agents sont dotés de capacités de *communication* et de *raisonnement social*, c'est à dire qu'ils sont capables de raisonner les uns sur les autres. Ces capacités leur permettent donc de résoudre des problèmes collectifs et de parvenir à l'accomplissement de tâches communes. L'ensemble de ces agents, plus ces différentes capacités constituent un *système multi-agent* (*SMA*).

Bien qu'une définition précise des termes agent et SMA [BG88, PLE92, SDB92, Fer95, OJ96, HS98], ne soit notre objectif principal, nous tacherons à travers ce chapitre de donner notre vision des SMA, en analysant un certain nombre d'aspects importants: nous commençons tout d'abord, par la description de la *coopération* dans un SMA. Nous décrivons ensuite, différentes méthodes indispensables pour la mise en place de la coopération telles que la *coordination*, l'*engagement commun* et la *résolution de conflits*.

Par la suite, notre analyse est focalisée sur les *relations de dépendance et de pouvoir*, ainsi que sur la *communication*. Nous avons choisi de développer ces deux aspects car ils font partie des apports principaux de ce travail (deuxième partie). En effet, la coopération entre plusieurs agents nécessite la mise en correspondance de leurs activités. Celle-ci permet aux agents de déterminer leur relations de dépendance et de pouvoir. Ces relations constituent par la suite, une motivation pour la communication qui intervient dans la coordination de tâches et la résolution de conflits.

2.2 Coopération

Afin de résoudre un problème complexe par un ensemble d'agents autonomes et parfois hétérogènes [RJM91], ces derniers doivent agir de de façon à ce que la contribution de chacun fasse converger l'ensemble vers la solution ou une solution possible.

La coopération désigne l'ensemble des activités grâce auxquelles, un ensemble d'agents participe à la résolution d'un problème commun.

La coopération peut se présenter sous différentes formes [Fer94]. Elle peut être accidentelle, unilatéralement ou mutuellement intentionnelle : en effet, les différentes interactions entre les agents d'un groupe, peuvent être interprétées comme une forme de coopération quand elle sont perçues par un observateur extérieur. Ceci est indépendant de l'intention des membres du groupe à coopérer. Dans ce cas, la coopération est perçue au *niveau externe*. Par exemple, les formes de coopération observées dans la simulation des sociétés de fourmis [DCL95], sont des attitudes non intentionnelles. Quand la coopération est une attitude intentionnelle des agents, elle est perçue au *niveau interne* également.

Dans un contexte de coopération, l'ensemble d'agents a un but commun à atteindre. Deux agents ayant un but commun ne coopèrent pas forcément : si l'un des deux agents a *adopté* le but de l'autre [CMC91], dans ce cas, les deux buts sont *identiques*, sinon ces buts sont *parallèles* et les deux agents sont plutôt dans un contexte de *concurrence* [ZR91][KEL91] car ils peuvent poursuivre leurs buts séparément. Cette différenciation nous montre que les agents doivent *coordonner* leurs activités visant à atteindre leurs buts identiques, car ces derniers ne peuvent pas être poursuivis séparément.

2.2.1 Coordination

La *coordination* est un processus permettant aux agents de raisonner sur leur activité locale et sur les activités des autres de manière anticipée parfois, afin de s'assurer que tous les membres de la société agissent d'une manière "cohérente". Ce processus correspond à un certain nombre de tâches dont l'exécution n'est pas directement productive mais sert à ce que l'ensemble de toutes les actions puissent s'exécuter dans les meilleures conditions.

Situations d'interaction

La coordination consiste à déterminer et à gérer les dépendances entre différentes activités [LCd96]. En effet, lors de la résolution d'un problème global, chaque agent est chargé de l'exécution d'un certain nombre de tâches qui le concernent. L'exécution avec succès de ces tâches constitue une *contribution individuelle* à une solution du problème. Dans certains contextes,

cette contribution individuelle peut jouer un rôle négatif dans la résolution du problème. Ceci est dû au fait qu'un agent a en général une vue partielle de l'activité des autres.

Les agents doivent donc interagir, afin de coordonner leurs tâches. Mais la question est de savoir si les agents interagissent d'une façon optimale ? Pour cela, il faut définir les situations permettant de caractériser l'interaction. D'après [LCd95], il en existe trois : les *situations routinières*, les situations *familières* et enfin les *situations non-familières*. Dans ce dernier cas, les agents doivent prendre des décisions d'une manière dynamique. Les interactions deviennent de ce fait, dynamiques, incertaines et on parle de *coordination dynamique*, tandis que dans des situations routinières ou familières, les interactions peuvent être fixées une fois pour toute et la *coordination* est alors *statique*.

Techniques de coordination

Il existe différentes formes de la coordination :

- elle peut correspondre à une couche de contrôle local au sein de l'agent [DLC87], intégrant à la fois des connaissances sur le domaine d'application ou le problème à résoudre et des connaissances sur le réseau de coordination nécessaire pour une coopération cohérente : éviter la duplication des tâches, répartir équitablement les tâches et éviter de transmettre des informations ayant pour effet de diminuer la performance global du système. La réalisation de ce type de coordination passe par la mise en place d'une *structure organisationnelle* spécifiant les responsabilités à long terme ainsi que les formes d'interaction des différentes entités. Cette information guide les décisions locales portant sur le contrôle : un agent doit connaître les conséquences de ses décisions, mesurer leur impact sur lui même ainsi que sur les autres. Il doit également savoir les répercussions des décisions prises par les autres sur lui même ;

- dans [Jen92b] l'auteur tente de définir un modèle de *collaboration* dans la résolution de problèmes comme une contribution vers le développement d'un niveau "*connaissances coopératives*". Ce niveau a pour objectif d'offrir des modèles explicites des phénomènes sociaux (coopération, conflits, compétitions ...), il diffère du modèle individuel par le fait qu'il décrit des *actions* de groupes d'agents et non pas des individus uniquement. A ce niveau, il est important de rendre compte qu'avant d'entamer des actions communes, les agents doivent être conscients qu'ils ont un objectif commun et qu'ils doivent l'atteindre en collaborant. Ainsi, est définie une responsabilité commune spécifiant une sorte de code de conduite pour les agents pendant la résolution du problème. Cette séparation entre les connaissances relatives au domaine

d'application et les connaissances coopératives, permet la réutilisation de ces dernières pour différents domaines d'application.

2.2.2 Engagement commun

Quelle que soit la forme de la coordination, il ne suffit pas de définir des règles permettant aux agents de coopérer. Chaque agent faisant partie de ce processus de coopération doit en plus s'engager à respecter ces règles. On parle ainsi d'*engagement commun* ou d'*intention commune* à coopérer [Sea90, CL91, Jen92a, Jen93b, Jen95].

Un engagement est défini comme étant un lien entre trois entités : un état du monde et deux agents. Le premier agent s'engage auprès du deuxième à réaliser cet état du monde. Cet état du monde peut correspondre à l'exécution d'une action ou à la satisfaction d'un but ou l'exécution d'un plan ...

Formes d'engagement

Les deux agents impliqués dans un engagement, ne sont pas forcément différents. En effet, dans [Cas95] une distinction est faite entre trois types d'engagement :

- un *engagement individuel* correspond à la notion d'engagement définie par Cohen et Levesque [CL90a], elle relie un agent à une action. Quand un agent doit exécuter une action, il s'engage vis à vis de lui même à l'exécuter. Le *but* correspondant à cette action est persistent ;
- un *engagement social* est l'engagement d'un agent envers un autre au sein d'un groupe ;
- un *engagement collectif* est l'engagement interne d'un ensemble d'agent envers un certain but. Ainsi, le groupe d'agents peut être vu comme un agent ayant ses propres buts à atteindre.

Révision d'un engagement

Un engagement (individuel, social ou collectif) ne suffit pas pour coordonner l'ensemble des tâches devant être exécutées par un groupe d'agents pour satisfaire un but global. En effet, un agent doit être capable de réviser ses engagements internes pour différentes raisons : un changement externe dans l'*environnement* [1] ou l'arrivée de nouvelles informations remettant en cause un certain nombre de décisions prises auparavant. La révision des engagements peut dépendre d'un ensemble de *conventions* [Jen93a, Jen96].

[1] L'environnement d'un agent est formé de l'ensemble des ressources accessibles par cet agent. L'ensemble des autres agents fait partie également de cet environnement.

Elles décrivent les circonstances suivant lesquelles les agents sont amenés à réviser leurs engagements. Elles indiquent également les actions appropriées afin de retenir, rectifier ou abandonner un engagement. Un agent peut avoir plusieurs conventions à sa disposition mais à un engagement est appliquée une et une seule convention. Malgré le rôle important que jouent ces conventions, elles restent asociales dans la mesure où elles permettent de gérer les engagements d'un agent mais ne spécifient pas comment cet agent doit se comporter vis à vis d'un changement ou d'une altération des engagements des autres. Pour gérer les engagements sociaux les agents doivent avoir à leur disposition un ensemble de *conventions sociales* où la révision des engagements d'un agent doit tenir compte de celle des autres.

La révision des engagements est importante non seulement parce qu'elle constitue un moyen permettant aux agents de répondre à des changements imprévus dans l'environnement, mais également parce qu'elle est nécessaire à la résolution de conflits qui peuvent être le résultat de tels changements.

2.2.3 Conflits et Négociation

La participation d'un groupe d'agent à la résolution d'un problème commun implique l'existence de relations liant les différents membres du groupe [Die91]. Ces relations se manifestent par la mise en commun d'un certain nombre d'éléments nécessaires à la coopération. Par exemple, un ensemble de robots chargés du transport de boîtes dans un entrepôt doit avoir des *ressources* communes telles que les boîtes transportées ainsi que les couloirs que doivent emprunter ces différents robots. Le partage de tâches et de ressources impliquent que les connaissances des agents admettent une partie commune. La coordination des différentes tâches passant par la définition de responsabilités communes et d'intentions communes de coopérer ne fait que poser une contrainte supplémentaire aux agents dans leur autonomie[2]. Ces contraintes sont généralement acceptées par l'agent suite à son engagement à coopérer. Ayant une vue partielle de son *environnement* et des activités des autres membres du groupe, son mécanisme de raisonnement peut l'entraîner dans un *conflit* avec un ou plusieurs membres du groupe.

Par exemple, les robots transporteurs peuvent vouloir accéder à une ressource commune alors qu'une contrainte précise que cette ressource ne peut être accédée par deux entités distinctes en même temps (exclusion mutuelle), ce type de conflit est appelé *conflit de ressource*. Dans ce cas, il existe une "*relation négative*" [MT93] reliant les plans des agents ayant un conflit de ressource. Un autre conflit est celui des buts (*conflits de buts*): quand deux agents admettent deux buts incompatibles, c'est à dire que la satisfaction du premier implique que le deuxième ne pourra pas être satisfait par la suite.

[2]La notion d'autonomie est liée à la notion de dépendance et sera définie en détail dans la section 2.3.4

Dans l'un ou dans l'autre cas, les agents faisant l'objet d'un conflit doivent trouver une solution ou un compromis [GH89]. La *négociation* est une méthode souvent utilisée. Elle donne aux agents un moyen de résoudre leurs buts conflictuels, de corriger les incohérences dans leurs connaissances sur les autres et enfin un moyen de coordonner leurs actions suivant une stratégie commune :

- dans [MT93] le mécanisme proposé vise à gérer les recouvrements dans les plans des agents afin d'une part de ne pas exécuter plusieurs fois la même tâche par deux agents et d'autre part, de répartir équitablement les tâches quand il s'agit de déléguer une tâche faisant partie de deux plans qui se recouvrent.

- l'*argumentation* est souvent utilisée comme une méthode de négociation [PJ96], elle consiste à proposer une solution par un des partenaires. L'autre partenaire évalue la proposition en pesant le pour et le contre de cette dernière. Si cette proposition ne peut être acceptée, il construit un argument contre cette proposition et propose une nouvelle alternative. Le processus continue jusqu'à ce qu'une proposition ou une contre-proposition soit acceptable par les deux partenaires ou que la négociation échoue sans qu'il y ait une entente possible.

2.2.4 L'individuel et le social dans la coopération

Dans les sections précédentes, nous avons souligné à plusieurs reprises les termes *individuel*, *social*. Ces termes permettent de distinguer deux niveaux au sein d'un agent :

- *individuel.* Il concerne les aspects de l'agent qui ne font pas intervenir les autres agents. Par exemple, un but local se place au niveau individuel ;

- *social.* Il regroupe les activités et les connaissances qui font intervenir les autres agents, ainsi que le raisonnement qui utilise ces connaissances. Un engagement envers un autre agent, se place au niveau social ;

D'un point de vue externe, on retrouve ces deux niveaux, on peut y ajouter un niveau *collectif.* Il regroupe les connaissances qui concernent tous les agents et aucun en particulier. Les termes collectif et social sont souvent confondus en SMA. En effet, un agent admet des connaissances sociales qui lui permettent d'interagir avec les autres.

Lors d'un processus de coopération, un agent doit pouvoir passer d'un niveau à un autre si besoin est : un but local est situé au niveau individuel, si l'agent est incapable d'atteindre ce but, il doit interagir avec les autres pour leur demander de l'aide et il passe ainsi au niveau social.

2.3 Relations de dépendance et de pouvoir

Les dépendances entre agents constituent une motivation pour la coopération au sein du groupe et plus généralement, un guide pour les interactions mises en place. Dans cette section, nous présentons un modèle des interactions sociales [Sic95] en montrant le rapport qu'elles ont avec les *relations de dépendances* entre les agents interagissants.

2.3.1 Modèle d'interaction sociale

La définition d'un modèle d'interaction social détermine la manière selon laquelle des connaissances relatives à l'organisation vont être utilisée par les agents [Sic95]. Par rapport à l'organisation, les modèles d'interaction sont en général considérés selon deux points de vue :

- les *modèles descendants*. Dans ces modèles, pour atteindre un but global les interactions sont conditionnées par l'organisation qui est conçue à l'origine de façon statique. Les modèles fondés sur les *structures organisationelles* sont des modèles descendants. Cette organisation peut être représentée explicitement [HSBS98] ou seulement par une structure au niveau de l'implémentation. De plus, les organisations rigides ne permettent pas une évolution des interactions ;

- les *modèles ascendants*. Dans ces modèles, les interactions ne sont pas conditionnées et les agents n'ont pas de buts globaux à atteindre mais tout simplement chaque agent tente d'atteindre ses propres buts. Les modèles ascendants peuvent être décomposés en deux parties :

  - les modèles fondés sur l'*utilité* considèrent qu'un agent est rationnel. Il cherchera toujours à maximiser son utilité espérée dans la société à laquelle il appartient. Les agents de la société doivent alors se coordonner pour assurer un comportement global cohérent ;

  - les modèles fondés sur la *complémentarité* supposent que les agents possèdent des capacités complémentaires et qu'un agent ne pouvant pas atteindre son but peut toujours demander de l'aide à d'autres agents. Dans le chapitre 5, nous définissons un modèle d'interaction fondé sur la complémentarité. Ainsi, pour réussir l'exécution d'une tâche donnée un agent peut avoir besoin d'être aidé. Son unique moyen de savoir que cette possibilité d'aide existe, est l'utilisation de ses relations de dépendance avec les autres agents.

2.3.2 Relations de dépendance

Une relation de dépendance r existe entre un agent a et un agent b, si pour atteindre son but, l'agent a doit avoir recours aux services offerts par l'agent b, sans lesquels ce but ne sera jamais atteint. La relation r contient les informations permettant à l'agent a de savoir qu'il dépend de l'agent b et comment cet agent est susceptible de l'aider.

Les relations de dépendance sont importantes pour la coopération : un agent dépendant d'un autre agent pourra lui demander de l'aide en cas de besoin. L'autre agent est en mesure d'aider cet agent sans même une demande explicite, à condition d'être au courant de la relation de dépendance existant entre eux. L'évolution des interactions entre agents rend les relations de dépendance dynamiques. Un agent doit être en mesure de déterminer continuellement ses dépendances avec les agents faisant partie de son réseau de connaissances.

D'une manière générale, les dépendances entre agents ne constituent qu'une sous-classe des dépendances qui peuvent motiver les interactions entre agents. Ces dépendances sont définies entre trois entités à savoir l'*agent*, la *tâche* et les *ressources* (Agent-Agent, Agent-Tâche, Agent-Ressource, Tâche-Agent, Tâche-Tâche, Tâche-Ressource, Ressource-Agent, Ressource-Tâche et Ressource-Ressource). Ces dépendances constituent la base de la coordination [LCd96].

Certaines de ces dépendances peuvent être déduites à partir des autres (figure 2.1). Dans les sections suivantes, nous nous intéressons plus particulièrement aux deux relations *Tâche-Tâche* et *Agent-Agent* car elles vont donner lieu au modèle décrit dans le chapitre 5.

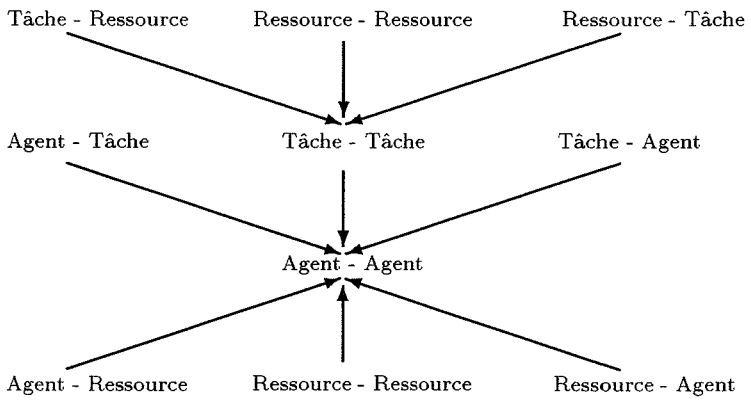


Figure 2.1 : La formation de relations de dépendance à partir d'autres relations.

2.3.3 Dépendances entre tâches

La relation *Tâche-Tâche* est une combinaison des relations d'une tâche avec ses sous-tâches ainsi que les relations avec les autres tâches. Elle peut être qualifiée de deux manières :

- négative : la réalisation d'une tâche empêche la réalisation de l'autre. Par exemple, la réparation d'une installation électrique dans un atelier peut entraîner une coupure générale d'électricité, ce qui empêche les autres machines d'effectuer leurs tâches. Ici, c'est typiquement la relation *Tâche-Ressource* entre les tâches et la ressource commune qui est utilisée ;

- positive : possibilité d'aide entre les deux tâches. La prise en compte d'un tel type de dépendance permet d'augmenter considérablement les performances du système. Par exemple, si un robot *Rob* est chargé du transport d'un objet dans un chariot d'un point A à un point B. Arrivant au point B, ce même chariot pourra servir pour le transport d'un autre objet par le robot *Robby* du point B au point C. Ainsi, nous pouvons déduire un relation de dépendance entre les deux tâches dédiées aux deux robots. Dans cet exemple, le gain produit est une économie de temps et d'énergie. La relation de dépendance entre les deux tâches des deux agents sont formées à partir des relations *Ressource-Tâche* et *Tâche-Ressource*. En effet, la tâche exécutée par *Rob* a rendu le chariot (ressource) disponible, sachant que la tâche que doit exécuter *Robby* dépend de ce chariot.

2.3.4 Dépendances entre agents

Pour entreprendre des interactions, les agents ont d'abord besoin de déterminer les relations de dépendance existant entre eux. Dans [Sic95, CMC92], une formalisation de deux types de relations de dépendance est proposée. La première est une relation de dépendance de ressources (R-DEP), et est définie de manière similaire à la relation *Agent-Ressource* citée ci-dessus. La deuxième est une relation de dépendance sociale (S-DEP) définie par le fait qu'un agent a besoin d'atteindre un but qui dépend d'une action qui ne peut être exécutée que par un autre agent. Ce type de dépendance correspond à la relation *Agent-Agent* et peut impliquer deux ou plusieurs agents.

Caractéristiques d'une dépendance sociale

Une dépendance sociale peut être composée de manière *conjonctive* (*et-dépendance*) comme dans la situation où plusieurs actions nécessaires à atteindre un but peuvent être entreprises par des agents différents (*dépendance*

multi-partite) ou dans la situation où un agent dépend d'un autre pour atteindre plusieurs de ses buts. Les relations de dépendance peuvent être décomposées de manière *disjonctive* (*ou-dépendance*) comme dans la situation où une action nécessaire pour atteindre un but peut être entreprise par un nombre d'agents différents (*alternative de partenaires*). Une autre situation est celle où plusieurs actions nécessaires pour atteindre un but peuvent être entreprises par un nombre d'agents différents (*alternative d'actions*).

Les dépendances peuvent être *unilatérales*, c'est le cas d'un agent dépendant d'un autre agent par rapport à un but, ou *bilatérales* et dans ce cas, la dépendance peut être soit *mutuelle*, soit *réciproque* :

- dans le premier cas, ayant le même but, deux agents sont capables d'entreprendre chacun une partie différente d'action permettant d'atteindre le but. Les deux agents ont un rôle complémentaire vis-à-vis du but ;

- dans le deuxième cas, un agent dépend d'un autre agent pour atteindre un certain but et ce dernier dépend du premier pour atteindre un autre but.

A partir des dépendances mutuelles et réciproques sont définies :

- la *Dépendance Mutuelle Localement Reconnue*: un agent déduit une dépendance mutuelle envers un autre en utilisant ses propres plans mais pas les plans de l'autre ;

- la *Dépendance Mutuelle Mutuellement Reconnue*: un agent déduit une dépendance mutuelle envers un autre en utilisant indifféremment ses propres plans ou ceux de l'autre ;

- la *Dépendance Réciproque Localement Reconnue*: un agent déduit une dépendance réciproque envers un autre en utilisant ses propres plans mais pas les plans de l'autre ;

- la *Dépendance Réciproque Mutuellement Reconnue*: un agent déduit une dépendance réciproque envers un autre en utilisant indifféremment ses propres plans ou ceux de l'autre.

Deux autres types de dépendance faisant intervenir trois partenaires sont les *dépendances externes* et les *dépendances transitives*. La première traduit le fait, qu'un agent devient dépendant d'un autre agent car il a adopté le but d'un tiers. La deuxième correspond à la situation où un agent a devient dépendant d'un autre agent b car ce dernier a adopté le but d'un tiers (agent c). Il est clair qu'avant l'adoption du but, une dépendance existait entre l'agent a et l'agent c.

Dépendance sociale et autonomie

Une dépendance sociale donne aux agents la possibilité de déterminer leur degré d'*autonomie*, et par conséquent la possibilité de savoir quelles sont les tâches qui doivent faire partie du processus de coordination. Un agent peut être autonome par conception, c'est à dire qu'il possède une existence propre, indépendamment de l'existence des autres agents. Il peut être autonome par rapport à l'environnement, par rapport à ses propres buts ou par rapport à ses motivations (il est capable de décider s'il va coopérer avec les autres ou non) [Cas90].

La notion d'autonomie est liée à la notion d'*interférence sociale*. Elle existe entre deux agents a et b si un but atteint par a admet un effet sur un ou plusieurs buts de b. Cet effet peut être positif : le fait que a atteigne ses buts aide b à atteindre ses buts à son tour. Il peut être négatif lorsqu'il y a concurrence sur une même ressource ou quand l'atteinte d'un but par a cause un conflit entre un but de a et un but de b (il n'existe aucun état du monde où ces deux buts peuvent être atteints en même temps). L'existence d'une interférence sociale au sein d'un groupe d'agents permet différents types de *situation sociale* :

- l'*exploitation* est une situation dans laquelle un agent ayant le même but qu'un autre agent n'a plus qu'à attendre que cet agent l'atteigne ;
- l'*influence* est une situation dans laquelle un agent ne pouvant pas atteindre son but et croyant qu'un autre agent peut l'atteindre, va influencer cet agent pour faire en sorte que ce but soit atteint. Le degré d'influence dépend du pouvoir d'un agent d'offrir certains services en échange ;
- l'*agression* d'un agent a par un autre agent b est le résultat d'une interférence négative traduite par un conflit entre les buts de deux agents. Ainsi, l'agent b peut empêcher a d'atteindre son but ;
- l'*adoption* est plutôt utilisée dans un contexte de coopération. Un agent a ayant un but g à atteindre, un agent b adopte le but de a, s'il admet comme but "que a atteigne g".

Ces différents types d'action permettent de différencier deux classes d'*interaction sociale*. La première est une interaction qualifiée de *positive* où les agents s'entraident afin d'atteindre leurs buts mutuels. Ce type d'interaction prend place dans un contexte de coopération et d'échange social. La deuxième est une interaction qualifiée de *négative* où les agents s'empêchent mutuellement d'atteindre leurs buts en conflit ou en concurrence. Ce type d'interaction peut correspondre aussi à une situation de *compétition* où les deux agents ont les mêmes buts et sont capables d'entreprendre les mêmes actions dans le monde.

2.3.5 Exemples d'utilisation des relations de dépendance

Les relations de dépendance constituent un moyen permettant à un agent d'avoir un *raisonnement social.* Ce dernier couvre la classe de tous les raisonnements utilisant des informations sur les autres pour tirer certaines conclusions [SCCD94]. En effet, en utilisant ses relations de dépendance avec les autres, un agent est capable de mesurer son degré d'autonomie par rapport à un ensemble de buts et par rapport aux autres agents. Lorsqu'une dépendance est détectée par l'ensemble des agents concernés, ces derniers peuvent suivre des stratégies et exécuter des plans en fonction de ces dépendances.

Un raisonnement social nécessite une représentation des autres au sein de l'agent. On parle ainsi de *description externe* dans [DM91]. La *description interne* correspond à la description de l'agent qu'il a de lui même. Il n'existe pas de modèles standards d'une description externe, elle peut être composée des *buts* d'un agent (atteints ou non), des *actions* qu'il peut entreprendre, des *ressources* qu'il contrôle et enfin des *plans* qu'il est capable d'exécuter en utilisant des actions et des ressources afin d'atteindre un certain but. Cette description peut également contenir les relations de dépendance d'un agent avec les autres.

Formation de coalitions

La stratégie devant être suivie par un agent pour atteindre un but donné (individuel ou commun), consiste généralement à exécuter un ou plusieurs *plans* d'*actions.* Les relations de dépendance ne suffisent pas pour le choix d'une stratégie. Un agent doit également prendre en compte la *possibilité d'exécuter un plan* et la *possibilité de réaliser un but.* Pour ce faire, dans [Sic96], l'auteur définit les notions de *plan exécutable* et de *but réalisable.* Il suppose d'abord que les agents n'ont pas de planification dynamique, mais ils utilisent plutôt des plans préétablis. Quand ils choisissent un but à suivre, il évaluent d'abord son *importance* à l'aide d'une fonction dédiée. Le choix d'un but nécessite le choix d'un plan à exécuter et si un agent n'est pas capable d'exécuter un plan tout seul, il peut proposer à d'autres agents (*susceptibles* de l'aider) de former une coalition et exécuter le plan en question. Le choix des partenaires est basé sur les relations de dépendance existant entre cet agent et les autres.

Résolution des incohérences

Pour exploiter bénéfiquement le raisonnement social, un agent doit avoir une base de croyances *cohérente* avec la réalité et l'ensemble des croyances des autres. En pratique, la description externe que possède un agent j concernant un agent i doit contenir des informations correctes et cohérentes avec les informations contenu dans la description interne de i. Or, le mécanisme de mise à jour de ces deux descriptions n'est le même pour les deux agents.

En effet, afin de mettre à jour la description externe de l'agent i, l'agent j se sert de son raisonnement social. Ce raisonnement utilise les informations perçus de l'environnement, ainsi que celles reçues des autres à travers les interactions mises en place. L'agent i quant à lui, utilise son raisonnement interne pour mettre à jour sa description interne. Pour cette raison, un agent a besoin d'une mesure de crédibilité sur les informations reçues afin de préserver la cohérence dans la description des autres.

Pour contribuer à la résolution de ce problème, dans [SD96], les auteurs ont identifié trois sources d'information : le raisonnement, la perception et la communication. Ensuite, ils ont défini une relation d'ordre sur ces sources en fonction de leur crédibilité. La règle est la suivante : Un agent i donnera une crédibilité maximale à une information provenant d'un agent j et décrivant l'agent j lui même. Quand cette information n'est pas fournie par l'agent j, il cherchera à croire toute information inférée par son mécanisme de raisonnement interne. S'il lui est impossible d'inférer des informations, il cherchera à croire en toute information délivrée par son mécanisme de perception. En dernier recours, il ne lui reste qu'à croire en toute information concernant l'agent j communiquée par d'autres agents que l'agent j lui même.

Une incohérence au niveau des descriptions externes entraîne des incohérence au niveau des relations de dépendance déduites par les agents [SD95] : les relations de dépendance d'un agent avec les autres, sont déduites à partir d'autres relations telles que les relations *Tâche-Tâche*, *Tâche-Agent* et *Agent-Tâche* (cf. section 2.3.2). Ces dernières font généralement partie de la description interne de cet agent. Ainsi, la description externe correspondante chez un autre agent, doit contenir les mêmes relations. Par exemple, considérons la situation où deux agents i et j ont le même but g et pour atteindre ce but, ils ont besoin d'exécuter le plan p composé de deux actions $a1$ et $a2$. Supposons que l'agent i (respectivement j) est capable d'exécuter seulement l'action $a1$ (respectivement $a2$) et supposons également que la description externe que possède l'agent i concernant l'agent j est correcte alors que celle que possède l'agent j concernant l'agent i est incorrecte, à savoir qu'elle ne contient pas d'informations précisant que i est capable d'exécuter l'action $a1$. Dans cet exemple, l'agent i va déduire une *dépendance mutuelle mutuellement reconnue* entre lui même et l'agent j. De son coté, l'agent j va déduire une indépendance.

2.3.6 Dépendances et pouvoir social

Les dépendances entre agents sont des relations objectives, c'est à dire qu'elles existent même si les agents n'en tiennent pas compte. Ces relations quand elles sont prises en compte, peuvent être mise en correspondance avec la notion de *pouvoir*. Cette notion est souvent négligée quand il s'agit

d'étudier des situations sociales telles que l'autonomie et l'adoption de but. Dans [Cas90], l'auteur met l'accent sur ce facteur et décrit deux types de pouvoir :

- un agent a le *pouvoir d'un but*, s'il est capable d'atteindre ce but. Cette notion est importante dans le cadre d'un raisonnement social, en effet, il est important qu'un agent i sache qu'un agent j a le pouvoir d'un but g, surtout si ce but intervient dans un des plans de i (g peut appartenir à un des plans de i ou la réalisation de g peut provoquer la réalisation d'un autre but g' appartenant à un plan de i). Ce type de pouvoir correspond à la relation *Tâche-Agent* décrite dans la section 2.3.2, à condition de mettre en relation une tâche et un but ;

- à partir de la notion de pouvoir définie ci-dessus, l'auteur définit la notion de *pouvoir social* (pouvoir entre agents) :

  - un agent i *a le pouvoir sur* un agent j par rapport à un but, si l'agent i peut aider j à atteindre le but g ou au contraire, s'il peut l'empêcher de l'atteindre. La relation *Agent-Agent* correspond à ce type de pouvoir dans un contexte coopératif ;
  - la notion de *possession* est liée à la notion de ressource, elle est également liée aux relations *Agent-Ressource* et *Ressource-Agent*. Un agent possédant X (il a le pouvoir d'accéder à X, peut utiliser et empêcher un autre agent d'utiliser X. Donc, il a du pouvoir sur cet agent ;
  - le *pouvoir d'influence* par rapport à un but se traduit par le fait qu'un agent a le pouvoir de réduire ou d'augmenter la probabilité qu'un autre agent suive le but en question. Ce type de pouvoir social est intéressant dans le cas où un agent i est dépendant d'un agent j par rapport au but $g1$ et j a le pouvoir de $g1$ et sachant que j est dépendant d'un but $g2$ dont i a le pouvoir, i peut *influencer* j à réaliser $g1$ en lui promettant de réaliser $g2$ comme *récompense*. Si la réalisation du but $g2$ peut empêcher la réalisation d'un des buts de j, alors i peut *menacer* j de réaliser ce but, à moins qu'en échange j ne promette de réaliser $g1$. Ce type de pouvoir social est également applicable dans une situation où i projette de *mettre en place une coopération* avec j. En cas d'acceptation j aura une attitude coopérative envers i (influencé par la proposition de i).

2.3.7 L'individuel et le social dans les dépendances

Les relations de dépendance entre agents (*Agent-Agent*) sont des relations sociales qui guident les interactions dans un processus de coopération.

Les relations *Tâche-Ressource*, *Ressource-Tâche*, *Tâche-Tâche* et *Ressource-Ressource* sont des relations placées au niveau application, car elles ne font intervenir aucun agent. Les éléments appartenant à ce niveau concernent uniquement le domaine d'application et le problème à résoudre. Les relations *Agent-Tâche*, *Tâche-Agent*, *Agent-Ressource* et *Ressource-Agent* sont plutôt des relations individuelles car elles ne font pas intervenir les autres agents. Or, la relation *Agent-Agent* est souvent construite à partir de ces relations individuelles (cf. section 2.3.2).

Ce passage des relations individuelles aux relations sociales est très important : les relations individuelles rendent compte qu'un agent est incapable d'exécuter une tâche. A partir des relations individuelles, l'agent peut déduire les relations sociales qui vont lui permettre d'interagir avec les autres pour leur demander de l'aide.

Le pouvoir social est associé aux relations de dépendance, mais il peut être considéré comme une extension de ces dernières. En effet, les relations de dépendance sont souvent utilisées dans un contexte coopératif où il s'agit d'aider un agent ou d'être aidé par un ensemble d'agents. Le pouvoir social peut être utilisé en plus dans une situation de concurrence. Quelque soit le contexte utilisé, les relations de dépendance et de pouvoir permettent aux agents de mieux gérer leurs interactions.

2.4 Communication

La coopération au sein d'un groupe d'agents nécessite que les agents se partagent un ensemble d'éléments: un environnement, des tâches, des informations de contrôle et de synchronisation. En effet, aucun agent n'est capable de connaître parfaitement son environnement et encore moins les intentions et les plans des autres agents. Ce partage est à l'origine des relations de dépendance. Pour ces agents, la communication représente à la fois un moyen leur permettant de compléter le manque d'informations et un moyen d'utiliser leurs relations dépendance. Le manque d'informations qui dans la plupart des cas, introduit des incohérences au niveau des croyances de l'agent, peut même dans certains cas, rendre la coordination impossible. En effet, si les agents ne possèdent pas d'informations suffisantes sur les actions des autres, ils sont incapables de coordonner leurs actions dans un contexte de coopération et sont incapables de calculer leurs relations de dépendance.

2.4.1 Formes de communication

Werner a identifié cinq façons de communiquer entre agents [Wer89] :

- *pas de communication* : c'est la communication sous sa forme la plus simple ! En effet, dans certains contextes, les agents sont capables d'inférer les plans des autres sans qu'il y ait besoin d'une communication

directe, à travers l'observation de l'environnement par exemple. Ces agents peuvent également avoir à leur disposition un ensemble de scénarios préétablis, précisant à chacun des agents comment atteindre son but sans l'aide des autres ;

- *envoi de signaux* : pour assurer la synchronisation, les agents peuvent se contenter de s'envoyer des signaux. Un signal est un message codé connu par les interlocuteurs et peut prendre n'importe quel forme ;

- *envoi de plans* : afin d'assurer la cohérence de leurs croyances et la coordination de leurs tâches, les agents s'envoient mutuellement leurs plans ;

- *envoi de message* : la plupart des systèmes multi-agents utilisent ce mode de communication. Pour communiquer leurs intentions et leurs besoins les agents s'envoient des messages en exécutant des méthodes dédiées à cet effet. Un message respecte un format connu par tous les agents. En fait, ce format décrit le *langage d'interaction* utilisé. A la réception d'un message, son contenu doit être interprété correctement, afin que l'agent sache comment il doit réagir. La théorie des actes de langage est souvent utilisée pour permettre une compréhension mutuelle des communications entre agents.

2.4.2 Actes de langage

La théorie des actes de langage se propose d'étudier et de comprendre les communications humaines. Selon cette théorie, l'acte de *communiquer* est considéré comme l'exécution d'une *action* en ayant une certaine *intention.*

Austin est le premier à avoir introduit cette théorie [Aus62]. Il souligne que pour communiquer, les humains utilisent une classe d'expressions appelées *performatifs.* Comme les actions, ces performatifs peuvent échouer, ainsi il a défini pour un performatif un ensemble de *conditions de satisfaction.* Il affirme également qu'un humain prononçant une phrase exécute trois types d'actions :

- un *acte locutoire.* Correspond à l'acte de formulation d'une phrase dans le langage utilisé ;

- un *acte illocutoire.* Correspond dans la plupart des cas à l'énonciation d'un verbe. Une *force illocutoire* est associée à un ou plusieurs verbes afin de préciser l'*intention* du locuteur ;

- l'*acte perlocutoire.* Représente l'*effet* qu'un acte produit sur l'allocutaire ou sur le locuteur.

Force illocutoire

Comme deux actes illocutoires peuvent admettre la même force illocutoire, Searle a proposé une typologie pour mieux classer les actes illocutoires [Sea69][Sea79].

Vanderveken a utilisé cette typologie pour différencier différents types de but illocutoire [Van88]. Selon lui, chaque force illocutoire peut être divisée en six composantes :

- le *but illocutoire* ;
- le *mode d'accomplissement* exprimant comment le but illocutoire doit être accompli ;
- les *conditions sur le contenu propositionnel* ;
- les *conditions préparatoires* précisant ce qui est supposé être vrai par le locuteur avant l'énonciation ;
- les *conditions de sincérité* ;
- et enfin le *degré de puissance* qui peut caractériser les conditions de sincérité, par exemple les verbes "*supplier*", "*demander*" et "*commander*" correspondent à des degrés de puissance dans un ordre croissant.

But illocutoire

Le but illocutoire est la composante principale de la force illocutoire, il exprime qu'à travers un acte illocutoire, le locuteur cherche à établir une correspondance entre *les mots* et *le monde* suivant une certaine *direction d'ajustement.* Cinq groupes de but illocutoire ont été identifié par Searle et Vanderveken[Van88, Sea79] :

- *Assertif*, il exprime la valeur de vérité d'une expression, comme par exemple, le cas du verbe *affirmer*, la direction d'ajustement est celle des mots au monde ;
- *Directif*, il représente une tentative du locuteur de faire faire une action par l'allocutaire, comme le verbe *commander.* Ce but a la direction d'ajustement du monde aux mots ;
- *Promissif*, représente un engagement du locuteur pour entreprendre une action, le cas typique est celui du verbe *promettre*, ce but permet d'ajuster le monde aux mots ;
- *Expressif*, exprime un état psychologique du locuteur comme la *gratitude* exprimée par le verbe *remercier*, la direction d'ajustement est vide ;

- *Déclaratif*, consiste à accomplir une action par le seul fait de l'énonciation comme par exemple, *Je déclare la séance ouverte* ou *Ils déclarent la guerre*, ce but a une direction d'ajustement double.

Ces différents buts illocutoires constituent un moyen pour exprimer plus clairement l'intention du locuteur. C'est dans ce sens, que Cohen et Levesque dans leur théorie des actes de langage [CL90b], affirment qu'il n'est pas nécessaire de reconnaître la force illocutoire d'un acte, mais plutôt de reconnaître l'intention du locuteur. Ils précisent également que les actes de langage sont une sorte d'*événements complexes* et correspondent à des *actions complexes* composées de plusieurs actions plus simples.

Les actes de langage permettent aux agents de communiquer leurs intentions d'une manière claire et d'entamer des conversations avec les autres. Pour ce faire, les agents doivent utiliser un langage de communication permettant l'utilisation des actes de langage. Cependant, si les agents n'ont pas à leur disposition des moyens pour contrôler leurs conversations, celles-ci peuvent dégénérer. L'utilisation d'un ensemble de protocoles d'interaction permet de répondre à ce problème. Elle indique à agent comment réagir face à une situation donnée ou suite à la réception d'un message.

2.4.3 Langages et protocoles d'interaction

La communication explicite entre agents nécessite l'utilisation d'un langage de communication connu par tous les agents, surtout si ces agents sont hétérogènes. En effet, les agents doivent être capables de construire des messages exprimant leur intention et l'information qu'ils souhaitent communiquer à autrui. Souvent les langages d'interaction utilisent la théorie des actes de langage, par exemple, dans *KQML* [FFMM94] (*Knowledge Query and Manipulation Laguage*) une expression se compose d'un message au format *KIF* [Gin91] (*Knowledge Interchange Format*) et un verbe performatif pour exprimer l'intention du locuteur.

Parmi ces performatifs, on peut citer par exemple, "*achieve*" pour exprimer que le locuteur veut que l'allocutaire accomplisse une certaine action, "*reply*" pour exprimer une réponse à une demande et "*ask-one*" pour demander une réponse à une question. Malgré les avantages que présente un tel langage, il est critiqué à cause de son incomplétude, car en effet, les performatifs utilisés dans KQML correspondent à deux buts illocutoires : le but assertif (*assert*) et le but directif (*command*). De plus, ce langage permet des contradictions entre le contenu propositionnel et les conditions de sincérité d'un acte. Par exemple, un agent peut également exprimer une requête à l'aide du performatif utilisé mais le contenu propositionnel précise que l'agent ne souhaite pas la satisfaction de la requête [BC96].

L'utilisation des actes de langage dans un langage d'interaction permet aux interlocuteurs d'exprimer leurs intentions dans les messages échangés.

L'effet de l'acte est local, il ne permet pas de déterminer la suite des interactions qui peuvent se mettre en place. Pour répondre à ce problème, une *théorie du dialogisme* a été proposée [TB88] mais son application aux systèmes multi-agents reste très limitée car ces derniers ne présentent pas encore à l'heure actuelle des capacités de communication assez évoluées. Les enchaînements conversationnels sont alors traités sous forme de "*protocoles d'interaction*" [CD90]. Un protocole d'interaction est un moyen de contrôler une conversation entre plusieurs agents. Il décrit les séquences valides de messages et ainsi, un agent recevant un message aura plusieurs choix de réponse en fonction du protocole suivi. Par exemple, à l'aide d'un protocole, un agent pourra gérer une requête provenant d'un autre agent concernant l'exécution d'une tâche donnée. Le protocole précise qu'à la réception d'une requête, l'agent admet deux choix : soit il exécute la tâche, soit il envoie une réponse demandant qu'on lui accorde un délai supplémentaire. Le protocole précise également les conditions pour chacun des choix. Par exemple, si l'agent est libre, il peut exécuter la tâche. Sinon il doit demander un délai supplémentaire. Des situations plus complexes peuvent également être exprimées par un protocole, telles que les négociations en présence de conflits [CW92].

Plusieurs formalismes sont employés pour représenter les protocoles. Les plus usuels sont les automates à états finis [WF86][PPS93] ou les réseaux de pétri [EG92].

2.4.4 L'individuel et le social dans la communication

La communication est une activité de l'agent, située au niveau social car elle fait intervenir au moins un autre agent. Comme nous l'avons vu avec les actes de langage, elle permet à un agent d'exprimer ses intentions d'une manière claire et concise. Les protocoles d'interaction constituent un moyen pour contrôler cette communication

Dans la section 2.3.7, nous avons montré que les relations de dépendance entre les trois entités agent, tâche et ressource et les liens qui existent entre elles, permettent à l'activité de l'agent de passer du niveau individuel au niveau social. Ce passage est dans l'objectif de communiquer avec un autre agent afin de lui demander de l'aide ou lui fournir une information. La communication permet au locuteur d'exprimer ses intentions et à l'allocutaire de mettre à jour des connaissances appartenant au niveau individuel et/ou au niveau social. Ainsi, la communication assure une liaison directe entre les niveaux sociaux des deux agents et une relation indirecte entre leurs niveaux individuels.

2.5 Discussion

Dans ce chapitre, nous avons analysé la coopération dans les SMA et avons montré comment l'utilisation des relations de dépendance et de la communication interviennent dans le processus de coopération.

La coopération fait intervenir des éléments appartenant aux niveaux individuel et social au sein d'un agent, et enfin au niveau collectif si cette coopération est considérée par un observateur externe. L'activité de l'agent passe au niveau social, si son activité au niveau individuel devient insuffisante pour lui permettre d'atteindre ses buts ou au contraire si elle offre des résultats susceptibles d'intéresser un autre agent. Les relations de dépendance représentent un moyen pour effectuer ce passage qui doit ensuite être suivi par une mise en relation du niveau social de l'agent avec les niveaux individuel et social d'un autre agent. Ce lien est assuré par la communication. Cette mise en relation entre les niveaux de deux agents est fondamentale pour la mise en place de la coopération.

Chapitre 3

Raisonnement temporel dans les SMA

3.1 Introduction

Pour permettre la résolution de problèmes complexes liés à des systèmes dynamiques et décentralisés à l'aide de techniques multi-agents, il est essentiel que la modélisation de ces problèmes, tienne compte du facteur temporel au niveau du SMA.

Ce facteur doit être pris en compte à la fois au niveau individuel et au niveau social :

- au niveau individuel, l'agent doit avoir à sa disposition des connaissances temporelles relatives aux actions qu'il peut entreprendre, aux événements qu'il perçoit de l'environnement. Il doit également gérer ses croyances et ses intentions au cours du temps ;

- dans la section 2.3.7, nous avons montré comment l'activité de l'agent passe du niveau individuel au niveau social quand ce dernier a besoin d'interagir avec les autres. Ainsi, si le temps est pris en compte au niveau individuel, il doit l'être également au niveau social.

L'objectif du présent chapitre, est d'étudier les représentations temporelles dans les SMA. Tout d'abord, nous présentons quelques travaux qui constituent une base théorique de la représentation du temps. Ensuite, nous montrons comment ces travaux vont permettre de définir des concepts utilisés dans les SMA. Ces concepts sont situés soit au niveau individuel, soit au niveau social. Nous terminons par une discussion en mettant l'accent sur les avantages que présente la prise en compte explicite du temps au niveau social.

3.2 Représentation du temps

La modélisation d'un système dynamique, nécessite une représentation explicite du temps dans le modèle. Cette représentation peut être symbolique ou numérique.

3.2.1 Représentations symboliques

Dans ces approches, le temps est considéré par rapport à une suite totalement ordonnée d'états.

Calcul de situations

Le *calcul de situations* proposé par McCarthy et Hayes [MH69] est une des premières tentatives de prise en compte de l'évolutif dans la représentation d'un système. Une *situation* correspond à un état du monde à un moment donné. Elle n'a pas de durée. La transition d'un état à un autre est causée par les *événements* et les *actions* qui se produisent. On obtient ainsi, une suite totalement ordonnée de situations.

Les événements permettent la transition d'un état à un autre, quand ils se produisent. Quant à la définition temporelle d'un événement, nous allons voir dans les sections suivantes que les avis diffèrent et qu'il n'existe pas de définition standard.

Les actions produisent des événements lorsqu'elles sont exécutées et conduisent ainsi à un changement de situation.

Outre l'impossibilité de représenter des changements continus et la non-linéarité du temps, la limitation principale du formalisme de calcul de situations est due à l'impossibilité de prendre en compte des actions simultanées. En effet, ceci correspond à l'observation de plusieurs événements simultanément et donc à plusieurs situations à un moment donné. Cette limitation rend le calcul de situations inadapté dans un contexte multi-agent où plusieurs agents peuvent agir en même temps en exécutant des actions simultanées [LS92].

C'est au cours des années 80, que les chercheurs ont commencé à utiliser les logiques temporelles réifiées pour représenter le temps. Ces logiques combinent la richesse d'expression des logiques modales [HC68] et l'efficacité des logiques de premier ordre [Rei87]. En effet, le paramètre temporel pouvant être séparé, il devint possible alors de séparer les aspects temporels des aspects atemporels dans le raisonnement. Des axiomes sont alors exclusivement dédiés à des inférences de natures temporelles.

L'*instant* comme primitive temporelle

Pour McDermott [McD82], la primitive temporelle est l'*instant*. Chaque *état* est alors caractérisé par sa date d'occurrence qui est un réel et ceci dans

un contexte où chaque réel est considéré comme une date valide (*"le temps est infini et non circulaire"*).

A partir des états, il définit également la notion de *chroniques* qui ne sont autre que des historiques possibles du monde (un ensemble totalement ordonné d'états allant jusqu'à l'infini). Une chronique admet toujours des branchements vers les futurs possibles et non vers le passé car ce dernier peut ne pas être connu, mais de toute façon, il est unique.

Par rapport aux chroniques, un *intervalle* est défini comme un ensemble d'états totalement ordonné et convexe. Les états et les chroniques sont considérés comme des notions importantes uniquement dans la mesure où ils représentent des périodes pendant lesquelles des *propositions* peuvent changer de valeur de vérité et des *événements* peuvent se produire. Ainsi, une *proposition* est définie par l'ensemble des états dans lesquels la proposition admet la valeur de vérité vraie et un *événement* est identifié par l'ensemble d'intervalles durant lesquels il se produit. Le plus petit intervalle dans cet ensemble correspond à l'occurrence de l'événement. A partir de ces définitions, l'auteur définit les *changements continus* dans le temps, et pour cette fin, il utilise le terme de *"fluent"*.

Plus tard, la logique de McDermott a été critiquée par Allen qui établit que les modèles utilisant l'instant ne sont ni intuitifs ni adaptés à la représentation des changements (états) dans le monde. Son argument est basé sur le fait qu'un événement ne peut pas être marqué par un instant correspondant à son occurrence, car cet événement peut toujours être décomposé en une suite d'événements et ainsi le temps correspondant à cet événement doit également être décomposable.

L'*intervalle* comme primitive temporelle

Allen propose alors une logique temporelle du premier ordre où l'*intervalle* est considéré comme primitive [All83]. L'instant étant par définition un élément non décomposable, il affirme qu'il doit être remplacé par l'intervalle pour marquer un événement car un intervalle peut toujours être décomposé en sous-intervalles.

Partant de ces considérations, Allen définit ensuite 13 relations d'ordonnancement pouvant exister entre deux intervalles dont l'ensemble de disjonctions permet de représenter toutes les contraintes temporelles symboliques possibles. Les relations de la figure 3.1 et leurs inverses permettent d'illustrer ces 13 relations entre deux intervalles I et J.

Afin de définir le rôle que jouent les événements et les actions dans cette logique, l'auteur définit le concept d'*occurrence* [All84] pour les *événements* (prédicat OCCUR) et les *processus* (prédicat OCCURRING). Tout comme pour McDermott, un événement, se produit durant le plus petit intervalle possible, c'est à dire qu'il ne peut pas y avoir de sous-intervalles pendant lesquels l'événement peut se produire. Un processus se déroule durant un

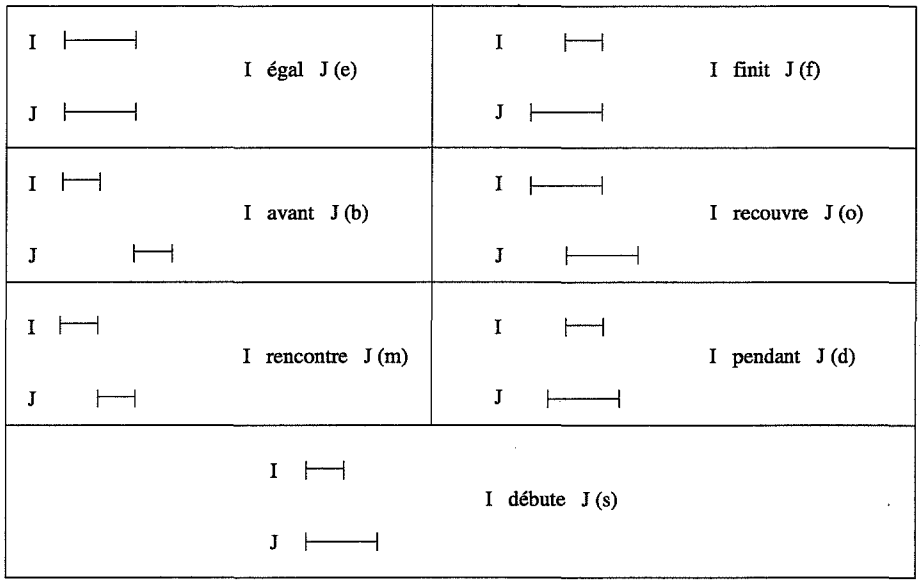


Figure 3.1 : Les treize relations entre deux intervalles I et J : e, b, m, s, f, o, d et leurs inverses : bi, mi, si, fi, oi, di.

intervalle si c'est le cas pour au moins un de ses sous-intervalles. De plus, l'auteur définit une *action* par le fait que son exécution entraîne une occurrence d'événement.

Enfin, l'auteur présente des résultats concernant la modélisation des événements composites, la causalité, les croyances et les intentions, ainsi que des éléments nécessaires à la résolution de problèmes de planification.

Discussion

La logique d'Allen a été critiquée plus tard pour son inadaptation à la représentation des changements continus [Gal90]. En effet, considérons un objet C qui a la possibilité de bouger sur un certain support (une table par exemple). Soit X une position quelconque sur ce support. Si l'objet C est en mouvement continu sur le support pendant un intervalle de temps I et si l'on considère la valeur de vérité de la proposition p = *"C est en X"* pendant l'intervalle I, il faut trouver un sous intervalle i de I où cette proposition tient. Or il n'en existe point, car sinon cela signifie que C est à l'arrêt pendant tout l'intervalle i (C est en position X), ce qui contredit le fait que le mouvement est continu. Ainsi, nous sommes incapables de représenter le fait que l'objet C est en position X à un moment donné si l'on ne dispose que d'intervalles comme primitives temporelles. Ceci devient plus simple, en utilisant une logique où l'instant est une primitive temporelle telle que celle proposée par McDermott.

A noter par ailleurs, que les deux formalismes décrits ci-dessus partagent avec les autres approches symboliques de la représentation du temps telles que les logiques modales par exemple, d'une part, leur lourdeur pour être implémentés et d'autre part, leur inadéquation à l'expression de contraintes numériques. En effet, il est possible de poser des contraintes temporelles entre événements et états, ce qui revient à ordonner des intervalles (Allen) ou des instants (McDermott). Cependant, il reste impossible à l'aide de ces formalismes d'enrichir les contraintes temporelles par des délais comme le permettent les approches numériques. Par exemple, il est impossible de représenter une contrainte spécifiant qu'un événement $e1$ se produit n unités de temps *avant* ou *après* un événement $e2$.

3.2.2 Approches numériques

Les approches numériques ont comme référence l'échelle temporelle absolue par rapport à laquelle sont indexées les primitives, instants ou intervalles. En 91, Allen [All91] s'intéresse à définir les dates d'occurrence d'événements dans différentes situations : si les événements sont instantanés et l'on connaît leurs dates d'occurrence, il est possible de représenter tous les ordonnancements et toutes les durées en utilisant la droite réelle comme un axe temporel. Si par contre, les dates d'occurrence ne sont pas connues et que les événements sont ordonnés chronologiquement selon leurs occurrences, alors il est possible d'utiliser pour un événement son rang comme étant une date fictive. Dans le cas où la date d'occurrence correspond à un intervalle d'incertitude, il est possible d'établir un ordre partiel sur les événements. Cet ordre est défini par une relation de distance entre deux événements encadrée par un intervalle déduit des deux intervalles relatifs aux événements. Enfin, si les événements ne sont pas instantanés et leurs occurrences correspondent à des durées, il est possible d'utiliser des méthodes d'ordonnancement classiques issues de la recherche opérationnelle telles que la méthode "*PERT*" ou une des méthodes de type "*FLOW SHOP*" [AF90].

Les approches numériques offrent en plus de la représentation des événements et des états dans le temps, la possibilité d'ordonner ces éléments sur une échelle temporelle numérique et d'exprimer des délais entre eux. Par exemple, il est possible d'exprimer quand un événement se produit exactement dans le temps (par rapport à une origine), mais également comment il se produit par rapport à d'autres événements : en même temps, n unités de temps plus tôt ou n unités de temps plus tard.

3.3 Connaissances temporelles au niveau individuel

Si l'importance de la prise en compte des aspects temporels relatifs aux concepts d'*action*, de *connaissances*, de *croyance* et d'*intention* est un fait, il est important de préciser toutefois, que ces concepts ne peuvent pas être considérés dans l'absolu. En effet, une action est toujours exécutée dans un *environnement* et par un *agent* ayant un ensemble de croyances et un ensemble d'intentions. Si le raisonnement temporel porte uniquement sur l'agent lui même, il se situe au *niveau individuel.* D'ailleurs avant l'apparition des techniques d'intelligence artificielle distribuée, un agent était toujours considéré seul dans son environnement, même si le terme agent n'était pas nécessairement le plus employé. Dans les sections suivantes, nous allons décrire quelques travaux qui ont constitué la base théorique pour la définition des différents concepts cités ci-dessus.

3.3.1 Croyances

L'étude des croyances d'un agent conduit et de manière naturelle à l'étude des aspects temporels permettant d'analyser la dynamique de ces croyances. En effet, rien que la notion de *persistance* d'une croyance relève déjà du temps. Plus généralement, on peut affirmer, tout comme dans [IS92], qu'il y a un temps où une croyance a lieu et un temps relatif au sujet sur lequel porte cette croyance. On peut considérer l'exemple suivant : "*l'agent A croit (**maintenant**) qu'une action a été effectuée **hier***".

Les logiques modales sont souvent utilisées pour formaliser les croyances [Kri63]. Ces logiques sont basées sur le modèle des mondes possibles [Hin62]. La croyance est définie par un opérateur de croyance [WF94] vérifiant les axiomes définissant la notion de rationalité chez un agent [Doy91] comme les axiomes KD45 :

- axiome K : un agent croit que "$p \supset q$" alors si l'agent croit que p, il croira que q aussi ;
- axiome D : un agent croit que p donc il ne croit pas que $\neg p$;
- axiome 4 : un agent est au courant de ce qu'il croît ;
- axiome 5 : un agent est au courant de ce qu'il ne croît pas ;

Dans une logique modale, le temps n'est pas exprimé explicitement, mais un ensemble d'opérateurs modaux permettent de se déplacer sur une échelle temporelle. Ainsi, il devient plus difficile de gérer la persistance des croyances : si une croyance est valide à un *moment donné* alors *pendant combien de temps* l'est elle ? Sans cette information, la base de croyances peut contenir des incohérences si celles-ci ne sont pas mises à jour correctement.

3.3.2 Intentions

Dans cette section, nous allons présenter quelques formalismes logiques utilisés pour formaliser l'intention. Ce concept nécessite la prise en compte des aspects temporels et de ce fait, sa formalisation utilise souvent des *logiques temporelles.*

Ces formalismes utilisent généralement un ensemble de *logiques modales.* Principalement, il existe trois types d'approche :

- l'approche par logiques du premier ordre : c'est le moyen le plus intuitif pour étendre le calcul de prédicat, en introduisant un argument supplémentaire représentant une valeur temporelle. Plus tard, le problème a été réexaminé et au lieu d'augmenter l'arité du prédicat pour considérer la valeur de vérité d'un fait à un instant donné, un fait est considéré comme une progression de son état dans le temps et sa valeur de vérité est examinée à cet instant. Le *calcul de situation* (Cf. section 3.2) fait partie de ce type d'approches ;

- l'approche par logiques modales a été utilisée à l'origine pour exprimer les notions de *possibilité* et de *nécessité.* Ces logiques utilisent le *modèle des mondes possibles* et sont caractérisées par la définition d'un certain nombre d'*opérateurs modaux* permettant l'accès à un monde possible et la transition d'un monde à un autre. Les notions de possibilité et de nécessité ont été réexaminées dans un contexte temporel ayant un rapport avec les notions de *prédiction* et les *futurs possibles.* Pour exploiter pleinement le concept du temps dans les logiques modales, des "*connectifs*" ont été utilisés permettant de considérer un fait dans le futur ou dans le passé, par exemple, l'expression Fp (F est un connectif, p est un fait) signifie que p sera vrai à un certain temps dans le futur.

  La logique de branchements temporels [BAPM83] est une logique modale basée sur une structure temporelle arborescente où un instant peut être suivi (sur une échelle temporelle) par plusieurs instants possibles. Cette arborescence représente tous les états possibles dans le futur. Une branche de l'arborescence est vue comme un scénario possible qui risque de se dérouler réellement dans le futur. Il est clair, qu'il existe plusieurs scénarios possibles dans le futur, mais qu'il n'en existe qu'un seul qui se déroule réellement, de ce fait, le temps est considéré comme étant linéaire dans le passé. Ainsi, pour un fait donné p, est construit un arbre représentant les scénarios dans le futur où p persiste. A l'aide d'un ensemble d'opérateurs temporels, il est possible de considérer la persistance de p dans différentes régions de l'arbre. Par exemple ([Sin94], pp. 24–25), à un instant donné t_0 dans un scénario S, on peut dire que Fp tient (F est un opérateur) si p est vrai à un certain instant t dans le futur, dans le scénario S. L'opérateur A est

utilisé pour faire référence à tous les scénarios à partir de l'instant présent (l'instant où la formule est considérée). De cette manière, on peut considérer des formules telles que AFp pour vérifier si p est vrai à un instant donné dans le futur et ceci dans tous les scénarios à partir de l'instant présent. Ce type de logique est souvent utilisé, notamment par Rao et Georgeff [RG91a, RG91b, RG93] et Singh [SA93, Sin94] dans leurs travaux sur l'intention.

- l'approche par logiques réifiées est apparue à l'origine dans un besoin de considérer des "*atomes*" (des objets ayant la valeur de vérité vraie ou fausse) comme des objets d'études. Pour répondre à ce besoin, les prédicats sont devenus des fonctions pouvant être les arguments d'autres prédicats. Par exemple, l'expression *State(machine, ok)* qui intuitivement, exprimait que *machine* est en parfait état, devient *True(State(machine, ok))* où l'expression précédente devient un simple terme et le prédicat *True* donne la valeur de vérité de ce terme. L'avantage d'une telle transformation est dû au fait, qu'il est possible d'étendre la *notion de vérité* à une *notion d'incertitude* (assez réaliste dans un contexte de modélisation de systèmes complexes), comme par exemple dans l'expression : $Possibly\_False(State(machine, ok), c)$ où c représente un coefficient de certitude. Les logiques de McDermott et d'Allen présentées précédemment appartiennent à cette classe de logiques.

3.3.3 Evénements et actions

Dans un contexte où un ensemble d'agents sont susceptibles d'*agir* dans un environnement dynamique, il est important d'avoir un modèle des actions qui peuvent être entreprises car ces actions peuvent être exécutées simultanément par plusieurs agents. Il faut alors étudier les effets possibles des actions sur l'environnement en terme de changement.

Actions et changements d'états

Dans [Geo86], l'auteur définit les actions comme étant à l'origine d'événements dont l'occurrence permet la transition d'un état à un autre. En effet, il définit un état du monde comme étant un ensemble d'objets avec un ensemble de relations et de fonctions les liant. Une séquence d'états est appelée "*histoire du monde*". Un état n'a pas de durée et le passage et l'évolution du temps sont visibles uniquement lors de changements d'états causés par l'occurrence d'un événement. Tout comme pour Allen et McDermott, un *type d'événement* est défini par un ensemble de séquences d'états représentant toutes les occurrences possibles de ce type d'événement dans toutes les situations possibles.

Indépendance d'actions

A l'inverse des approches classiques où un événement (ou une action) est modélisé(e) de façon à affecter uniquement certaines relations du monde, l'auteur de [Geo86] insiste sur le fait que pour tenir compte de la possibilité d'entreprendre des actions simultanées, la modélisation de ces actions doit affecter un ensemble plus large de relations du monde. En effet, si une action exécutée par un agent, affecte un ensemble de relations, d'autres relations peuvent être affectées par d'autres actions exécutées par d'autres agents. Pour remédier à ce problème, une notion d'*indépendance d'actions* est proposée : pour chaque action, sont identifiées les variables affectées par cette action et les prédicats admettant ces variables en argument. De cette manière, deux actions peuvent être entreprises simultanément, s'il n'existe pas de prédicat admettant un ensemble de variables en argument et qui soit affecté par les deux actions.

Dans un but de planification [Geo84], une *loi de persistance* est définie, spécifiant que si un prédicat ne doit pas être affecté par une action, et ceci en tenant compte de tous les événements que peut provoquer cette action (comme par exemple, le fait déplacer un cube A cause le déplacement d'un cube B si les deux cubes sont liés par une contrainte physique), alors la valeur de vérité de ce prédicat est préservée après la transition d'un état à un autre.

Actions primitives et actions globales

Quelques années plus tard, Lin et Shoham [LS92] proposent un modèle différent autorisant la concurrence d'actions dans le *calcul de situations*. Il précisent que le modèle proposé par Georgeff est incomplet dans la mesure où il ne permet pas de lister explicitement toutes les actions qui permettent de passer d'une situation donnée à une autre. Dans leur modèle, ils proposent les notions d'*actions primitives* et d'*actions globales*. L'exécution d'une action globale signifie que les actions primitives faisant partie de cette action s'exécutent d'une manière concurrente. Afin d'évaluer ce modèle, un critère formel appelé *critère de complétude épistémologique* est proposé dans un contexte où les actions sont supposées être déterministes, c'est à dire que l'exécution d'une action permet d'une manière unique de passer d'une situation donnée à une autre situation. Une théorie d'actions déterministes est dite "épistémologiquement complète", si pour une situation bien déterminée, elle permet de prédire une description complète de cette situation suite à l'exécution d'une action.

3.4 Connaissances temporelles au niveau social

Dans le chapitre 2, nous avons signalé que lors d'un raisonnement social, un agent tient compte des autres dans le système. Pour cela, il utilise une description externe qui contient des informations relatives à ces agents. En général, l'agent considère ces informations sous forme de croyances et dans ce cas, ses croyances portent sur les croyances d'un autre agent ou plus généralement sur son état mental.

3.4.1 Croyances sur les autres

Dans ce contexte, il est très important de pouvoir exprimer explicitement le temps au niveau des croyances, car les croyances d'un agent sur celles d'un autre doivent être cohérentes avec ces dernières, surtout si celles-ci sont sujettes à des évolutions permanentes dans le temps [Sho90]. Par exemple, "***aujourd'hui***, *l'agent A croit que la* ***semaine dernière*** *l'agent B a cru que* ***dans deux semaines*** *une action a allait être exécutée*".

Dans [IS92], les auteurs traitent ce problème en proposant une logique modale des croyances définie à l'aide d'un opérateur modal B [1] auquel on a rajouté un certain nombre de propriétés pour assurer la cohérence avec les propriétés associées à la notion de croyance classique (axiomes KD45).

Afin de définir la notion de persistance des croyances, l'opérateur modal L [2] est introduit : "*un agent croit qu'un fait est vrai au temps* $t+1$ *si et seulement si il apprend que ce fait est vrai au temps* t *ou s'il croyait que ce fait était vrai au temps* t *et à cet instant il n'a pas appris que ce fait est faux*".

La composition des deux opérateurs B et L permet de définir dans un espace multidimentionnel des régions où les croyances persistent. Un exemple simple est celui d'un agent qui **apprend** au temps t_1 qu'une propriété p [est, sera ou était] vrai au temps t_2, alors il s'agit de savoir quelles sont les deux périodes de temps où l'agent peut **croire** pendant la première période que la propriété p est vraie durant la deuxième période. Ceci correspond à une région dans un espace à deux dimensions. Des exemples plus complexes permettent de définir des régions dans des espaces de dimensions supérieures.

Les croyances d'un agent sur les autres sont inférées soit à partir d'autres croyances, soit à partir d'informations communiquées. Ainsi, l'expression du temps dans la communication est nécessaire, si le temps est exprimé également au niveau des croyances.

---

[1] $B_a^{t_1} B_b^{t_2} \phi$: signifie que l'agent a croit à l'instant t_1 que l'agent b croit ϕ à l'instant t_2.

[2] $L_a^t \phi$: signifie que l'agent a a appris que ϕ est vraie à l'instant t.

3.4.2 Communication

L'utilisation des actes de langage dans la communication, considère l'acte de communiquer comme l'exécution d'une action (cf. section 2.4.2). Or, des formalismes ont déjà été proposés, permettant l'expression du temps dans les actions (cf. section 3.3.3). De ce fait, ces formalismes peuvent être utilisés pour exprimer le temps dans les actes de langages [Sin94, CL95]. L'énonciation d'un acte de langage avec succès dépend des conditions de satisfaction de l'acte en fonction de son but illocutoire. Par exemple, pour un but assertif, l'action est exécutée avec succès si le contenu propositionnel de l'acte est vrai, **à l'instant même** où l'acte a été énoncé par le locuteur. Pour un but directif, l'action est exécutée avec succès si le contenu propositionnel de l'acte devient vrai **à un instant donné dans le futur** après l'énonciation de l'acte.

Une expression explicite du temps au niveau des actes de langage permet l'utilisation de contraintes temporelles dans la communication. Par exemple, un agent pourra envoyer une requête à un autre agent, lui précisant que sa réponse ne doit parvenir après l'expiration d'un délai ou dès que possible. A noter, que l'envoi d'un requête correspond à un *acte interrogatif* : un acte directif qui demande qu'un acte assertif soit exécuté. L'introduction du temps dans la communication entre deux agents nécessite une gestion des contraintes temporelles de part et d'autre.

3.4.3 Gestion des contraintes temporelles

L'introduction des contraintes dans la communication nécessite une gestion au niveau social de ces contraintes. En effet, un agent qui reçoit une requête d'exécution d'une tâche avec un délai à respecter doit tenir compte de ce délai au niveau individuel certes, mais il doit également s'engager auprès de l'agent expéditeur de la requête, à exécuter la tâche avant que ce délai n'expire. La gestion d'une contrainte dépend de sa nature : immédiatement, à partir d'une certaine date, dès que possible, avant l'expiration d'un délai, périodiquement. Quelle que soit la nature de cette contrainte, les deux agents impliqués dans la communication doivent réagir d'une manière cohérente en cas de respect de cette contrainte ou non :

- comment l'allocutaire doit gérer ces activités au niveau individuel par rapport à cette contrainte (influence de cette contrainte sur le niveau individuel de l'allocutaire) ;

- comment le locuteur doit tenir compte de cette contrainte : est-ce qu'il peut considérer qu'elle sera respectée et que la tâche associée sera exécutée à temps avant même d'avoir les moyens de le vérifier ? Par exemple, si le locuteur doit construire un plan correspondant à la suite des opérations après l'exécution de cette tâche, il doit considérer que

la tâche a été exécutée à temps et avec succès. Et s'il s'avère plus tard que la contrainte est violée, il doit avoir les moyens de réviser ses plans et ses croyances par rapport à cette tâche.

Dans [FDV96], les auteurs proposent une description formelle de ce type de contraintes à l'aide d'une logique déontique permettant ainsi de raisonner sur des délais et détecter ceux qui introduisent des incohérences dans le système. Il définissent également la notion d'obligation sous contrainte. Cette notion a déjà été étudiée dans [Kro95].

3.5 Discussion

Dans ce chapitre, nous avons présenté un certain nombre de travaux sur le raisonnement temporel dans les systèmes multi-agents. Il est clair que certains des modèles proposés ont été initialement étudiés dans un contexte mono-agent, tels que par exemple, les modèles de croyances, d'intentions. Avec l'apparition des nouvelles techniques multi-agents, quelques uns de ces modèles ont été adaptés pour répondre à un contexte distribué où un ensemble d'agents peuvent entreprendre des activités simultanément. Néanmoins, dans certains aspects qui sont propres au domaine multi-agent, le raisonnement temporel est encore dans un état embryonnaire. En effet, à notre connaissance, il n'existe pas encore de modèles formels où les aspects temporels et dynamiques dans le raisonnement social soient pris en compte. Le modèle des réseaux de dépendances présenté dans le chapitre 2, ne permet pas non plus de représenter les différentes évolutions des relations de dépendance entre agents au cours du temps.

D'une manière générale, tous les concepts situés au niveau social, ne permettent pas dans leurs formalisations, de représenter explicitement le facteur dynamique. Ce facteur est d'une importance capitale et sa prise en compte dans la modélisation des systèmes multi-agents permet de faire de la prédiction, dans un contexte où les agents sont contraints à fonctionner constamment avec des informations incomplètes ou incertaines.

Dans le chapitre 5, nous proposons un modèle de société d'agents basé sur les réseaux de dépendances et où le facteur dynamique dans les dépendances est pris en compte d'une manière explicite.

Le chapitre suivant présentera le domaine d'application choisi pour valider ce modèle de société. Dans ce domaine, qui est la supervision de systèmes industriels, la plupart des approches doivent tenir compte d'une manière explicite du temps dans la représentation des connaissances, ainsi que dans les algorithmes proposés.

Chapitre 4

Supervision des systèmes dynamiques

4.1 Introduction

La complexité croissante des systèmes industriels accentue l'importance du problème de leur *supervision*. La supervision consiste à déterminer les modes de fonctionnement du système supervisé (figure 4.1, à partir d'un modèle de ce dernier et des informations qu'il émet. Les modes de fonctionnement peuvent être normaux, dégradés, défaillants, critiques ... Ayant détecté un mode de fonctionnement anormal, des actions correctrices doivent être menées afin que le système retrouve son fonctionnement normal.

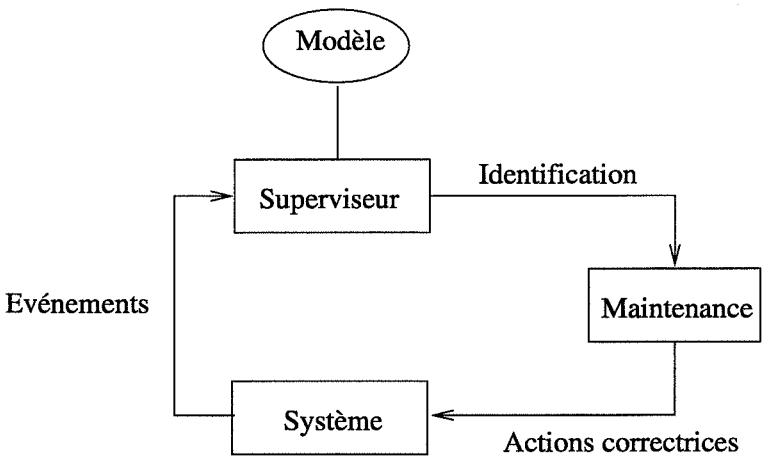


Figure 4.1 : Une architecture simplifiée d'un système de supervision.

Dans ce chapitre, nous donnons tout d'abord une définition des trois composantes du processus de supervision. Nous présentons ensuite, les pro-

blèmes que l'on y trouve. Par la suite, nous présentons et comparons un certain nombre d'approches en reconnaissance de forme et en intelligence artificielle. Nous terminons par une discussion où l'on montre l'importance du facteur temporel dans la supervision et comment cette dernière permet de montrer des aspects intéressants quand elle est utilisée comme domaine d'application dans un système multi-agent.

4.2 Composantes principales de la supervision

Le processus de supervision admet trois composantes principales que l'on peut retrouver dans un système de supervision :

- *Diagnostic.* Nous parlons de diagnostic lorsqu'il s'agit d'analyser des informations correspondant à un fonctionnement qui s'est déroulé dans le passé. Cette méthode est utilisée généralement pour des systèmes qui ne nécessitent pas une intervention immédiate après un dysfonctionnement. Par exemple, la fabrication d'une pièce mécanique nécessite une période de test. Pendant cette période, des mesures sont recueillies en cours de fonctionnement de la pièce. Ces mesures seront analysées par la suite pour déterminer si la pièce en question présente des défaillances pendant la période de test ;

- *Suivi.* Nous parlons de suivi lorsque le système supervisé présente des modes de fonctionnement critiques, comme dans le cas de la supervision d'un réseau électrique, il est important de pouvoir déterminer les dysfonctionnements pendant le fonctionnement du système. Si le système présente une défaillance ou une anomalie, cette anomalie est détectée et identifiée au plus tôt. Ainsi, il devient possible de prendre rapidement des mesures de réparation, afin d'éviter ou de limiter les dégâts qui risquent d'être occasionnés si le système continue à fonctionner dans un mode de fonctionnement anormal ;

- *Prédiction.* Dans certains cas, le suivi n'est pas suffisant car même si une anomalie est détectée à temps, il est déjà trop tard pour réagir. Ainsi, il devient important de pouvoir estimer le fonctionnement prévu du système dans un futur proche, afin de permettre l'anticipation des anomalies. Dans ce dernier cas, nous parlerons de *prédiction.* Par exemple, dans une centrale nucléaire, il est préférable de prédire un dépassement de température de façon à pouvoir réagir rapidement, plutôt que d'attendre que ce dépassement ait lieu avant de réagir.

La supervision est ainsi définie par ces trois composantes qui sont le diagnostic, le suivi et la prédiction. Chacune des composantes permet de déterminer le mode de fonctionnement du système dans une période donnée : dans le **passé** pour le diagnostic, le **présent** pour le suivi et enfin, le **futur**

pour la prédiction. De ce fait, le *facteur dynamique* devient un élément essentiel dans un problème de supervision.

4.3 Problèmes liés à la supervision

Dans un contexte de supervision, il est indispensable de tenir compte d'un certain nombre de problèmes. Ces problèmes se situent au niveau du système supervisé, au niveau du système superviseur et/ou au niveau de l'opérateur humain, responsable du traitement et de l'analyse des résultats produits par le système superviseur :

4.3.1 Masquage

Au cours de la supervision, le système supervisé doit être en mesure de transmettre au système superviseur des informations relatives à son fonctionnement. La *nature* des ces informations, leur *fréquence* de transmission, ainsi que l'équipement permettant cette transmission peuvent constituer de sérieux problèmes, empêchant le système superviseur de produire des résultats exacts. En effet, la défaillance d'un capteur permettant de détecter les occurrences d'un type d'événement, peut induire le système superviseur en erreur. Cette défaillance peut également *masquer* d'autres événements, si ce capteur est placé en amont par rapport à d'autres capteurs.

4.3.2 Engorgement

Le système superviseur est sensé donner des résultats sur le fonctionnement du système supervisé et ceci en fonction des informations reçues en entrée. Si les informations en entrée sont trop nombreuses, le système superviseur risque de ne pas pouvoir traiter la totalité de ces informations. On parle souvent de *problèmes d'engorgement.* Si les informations en entrée sont imprécises ou incomplètes suite à des *problèmes de masquage* par exemple, le système superviseur n'est pas en mesure d'établir avec précision le mode de fonctionnement du système. Dans un contexte de suivi, l'opérateur humain chargé d'observer et d'analyser les résultats provenant par le système superviseur, est souvent submergé par une cascade de messages, dès qu'une anomalie se présente. On parle souvent de *problèmes d'avalanche.*

4.3.3 Modélisation

Par rapport au choix d'une modélisation du système supervisé, la richesse d'expression du modèle utilisé entre souvent en concurrence avec l'efficacité en temps de réponse du système de supervision. En effet, dans un problème de supervision, d'une part, on exige l'efficacité en temps de réponse car le système supervisé évolue dans le temps et le système superviseur

est contraint de respecter des délais. D'autre part, la *fiabilité* du système superviseur est cruciale. Ce dernier facteur est fortement lié à la richesse d'expression du modèle utilisé pour modéliser le système physique. Ceci est réalisable en assurant la *complétude* des connaissances ainsi que leur *cohérence*. Ceci implique l'existence d'une base de connaissances assez riche, mais malheureusement contribue dans la plupart des cas à l'augmentation de la complexité des algorithmes mis en place. Ajoutons à cela le problème de faisabilité : en effet, la complexité croissante des systèmes actuels ne fait qu'accroître le volume des connaissances le modélisant. Il devient alors de plus en plus difficile de construire des modèles complets modélisant des systèmes industriels. En fait, la vraie difficulté est d'être en mesure de fournir une preuve de la complétude de tels modèles. Afin de résoudre ce problème les chercheurs ont adopté des méthodes où seuls des modèles partiels et incomplets sont utilisés tout en garantissant un minimum de fiabilité.

Dans les sections qui suivent, nous nous efforcerons de décrire un certain nombre d'approches pour la supervision, sans toutefois prétendre couvrir tout ce domaine de recherche [Pau81].

4.4 Approches en reconnaissance des formes

Comme leur nom l'indique, ces approches sont basées sur la notion de "*forme*"[Dub90]. Ayant un certain ensemble de *forme-références* (modes de fonctionnement connus du système), la problématique est de déterminer le maximum de ressemblance entre une forme observée (mode de fonctionnement à reconnaître) et une forme-référence. Ces forme-références sont en général des vecteurs à n dimensions si l'on se place dans un espace vectoriel de dimension n. Elles définissent un ensemble de M *classes* (C_1, ...,C_M) correspondant à différents modes de fonctionnement. Pour une forme observée, il s'agit de déterminer la classe à laquelle elle appartient. Le fait que cette forme observée représente un état du système, permet de classer son comportement parmi les modes possibles définis par les classes.

Dans certains cas, l'ensemble des classes n'est pas connu et avant de procéder à un classement des échantillons, des méthodes permettent de déterminer un certain nombre de partitions dans un ensemble d'échantillons.

Dans ce qui suit, nous présenterons essentiellement deux catégories d'approches : les unes permettent de dégager un ensemble de classes à partir d'un ensemble d'échantillons initial. Les autres permettent de classer des nouveaux échantillons sachant l'ensemble des classes.

4.4.1 Analyse de données

Cette méthode fait partie de la première catégorie citée ci-dessus [DLPT82]. Elle est un exemple de méthode d'analyse factorielle [Rao64]. Son objet est

d'essayer de réduire la dimension de l'espace de représentation de manière à obtenir un sous-espace où les données sont mieux représentées. Ayant un ensemble de données représentées par des vecteurs de $\mathcal{R}^d$ $(x_1, ..., x_n)$, il s'agit de trouver un nouveau sous-espace $\mathcal{R}^{d'}$ avec $d' < d$ où la base de ce sous-espace représente les informations les plus importantes concernant les données. Le critère à minimiser est celui de la moyenne des carrés des distances entre les points et leur projection dans ce sous-espace, c'est à dire un critère d'inertie. Ce critère s'écrit :

$$J = \frac{1}{n} \sum_{i=1}^{n} d^2(x_i, x_i^p)$$

En introduisant la matrice de variance-covariance :

$$\hat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^t$$

Le critère J s'écrit alors:

$$J = \frac{1}{n} \sum_{i=1}^{n} (||x_i||^2) - \sum_{k=1}^{d'} u_k^t M \hat{\Sigma} M u_k$$

Où M représente la métrique utilisée et les u_i forment la base du sous-espace recherché. En tenant compte du fait que $u_k^t M u_m = 0$ pour $k \neq m$ et que $u_k^t M u_k = 1$ et en posant $v_i = M^{1/2} u_i$ et $\Sigma' = M^{1/2} \hat{\Sigma} M^{1/2}$, on se ramène à un problème de calcul des valeurs propres et des vecteurs propres de la matrice Σ'. Les $u_i = M^{-1/2} v_i$ sont alors les vecteurs propres de la matrice $\hat{\Sigma} M$. Pour choisir la base du sous-espace recherché, on choisit les d' vecteurs propres correspondant aux valeurs propres dans l'ordre décroissant car ils sont les plus représentatifs des données. En fait, chacun de ces vecteurs constitue une caractéristique forte des données de l'ensemble d'apprentissage. La projection d'un nouvel échantillon dans le nouveau sous-espace permet de déterminer ses caractéristiques en fonction des coordonnées obtenues. Cette nouvelle représentation des données permet la distinction de plusieurs régions dans l'espace correspondant chacune à une classe.

4.4.2 Nuées dynamiques

Comme la méthode précédente, cette méthode [DS76] vise également à rechercher des partitions dans l'ensemble d'apprentissage. Ces partitions sont des classes et correspondent physiquement à autant de modes de fonctionnement. Ayant un ensemble d'apprentissage composé de n vecteurs, l'objectif est de déterminer M classes parmi l'ensemble d'apprentissage. Contrairement à la précédente, il ne s'agit pas de trouver un nouvel espace pour représenter les données, mais d'appliquer un algorithme basé sur le calcul de centre de gravité, appelé *algorithme de coalescence*:

1 Choisir M centres pour les classes qu'on désire obtenir. En général, ces centres sont choisis aléatoirement dans l'ensemble d'apprentissage.

2 Créer une nouvelle partition en associant chaque vecteur de l'ensemble d'apprentissage à la classe ayant le centre le plus proche.

3 Calculer les centres de gravité des nouvelles classes obtenues.

4 Répéter les étapes 2 et 3 afin d'obtenir un minimum du critère choisi. Le critère le plus utilisé est celui de l'erreur quadratiques. Minimiser ce critère revient à minimiser la dispersion à l'intérieur de chaque classe : on cherche à grouper le plus de points possibles autour des centres (zones denses).

5 Réviser le nombre de partitions obtenues en éclatant les classes trop denses et en fusionnant des classes trop pauvres. Une classe est éclatée si la variance de la composante la plus dispersée est trop forte. A l'inverse, deux classes sont regroupées en une seule si leurs centres de gravité sont trop rapprochés.

La difficulté majeure dans une telle méthode est de pouvoir associer à chacune des classes découvertes, un mode de fonctionnement pour le système physique. De plus, il faut disposer d'un long historique de fonctionnement (ensemble d'apprentissage de grande taille) pour que les partitions trouvées correspondent réellement à des modes de fonctionnement.

4.4.3 Approche probabilistique

Dans cette approche, les classes C_i sont connus et de plus, on suppose une connaissance totale des lois de probabilités régissant les observations. Chaque vecteur x de la classe C_i obéit à la loi de probabilité multidimentionnelle $f(x/C_i)$. Les probabilités a priori $P(C_i)$ sont connues également avec $\sum_{i=1}^{M} P(C_i) = 1$. Un nouvel échantillon v doit être affecté à une des classes C_1, ..., C_M. Une règle de décision permet alors de décider comment classer l'échantillon [Cho57]. Cette règle est basée sur la notion de coût. Le coût $C(C_i/C_j)$ est le coût de classer un vecteur v appartenant réellement à la classe C_j dans C_i. A ce coût est associé la probabilité de la classe C_j sachant v, c'est à dire la probabilité a posteriori $P(C_j/v)$. On a: $P(C_j/v) = \dfrac{f(v/w_j)P(w_j)}{\sum_{i=1}^{M} f(v/w_i)P(w_i)}$, en utilisant la règle des coûts 0, 1 où $C(C_i/C_i) = 0$ et $C(C_i/C_j) = 1$ $\forall i \neq j$. Il est clair qu'au fur et à mesure de l'arrivée des échantillons des nouvelles classes peuvent se former et

donc vouloir toujours classer un échantillon dans une des M classes constituerait une erreur de classification. Pour cette raison, une classe C_0 appelée classe de rejet est prévue pour classer les échantillons qui ne peuvent pas être classés dans les autres classes. On pose $C(C_0/C_j) = c_r \ \forall j$.

Pour i=0, on a $C_0(v) = \sum_{j=1}^{M} c_r P(C_j/v) = C_r$.

Pour i=1..M, on a $C_i(v) = \sum_{j=1, j\neq i}^{M} P(C_j/v) = 1 - P(C_i/v)$.

Le risque conditionnel minimum est défini par:

$$C^*(v) = Min_{i=1..M} C_i(v) = 1 - Max_{i=1..M} P(C_i/v)$$

La règle de décision est alors la suivante :

$$\begin{cases} v \text{ est associé à } C_i \text{ si } C^*(v) \leq C_r \\ v \text{ est associé à } C_0 \text{ (rejeté) si } C^*(v) > C_r \end{cases}$$

Cette règle est appelée la *règle de Bayes* ou du *coût minimum*. Dans un contexte réel, il est difficile de connaître totalement la loi de probabilité, on cherche alors à l'estimer.

4.4.4 K plus proches voisins

A lieu d'estimer la loi de probabilité régissant les classes, certaines méthodes utilisent des propriétés de l'espace de représentation de l'ensemble d'apprentissage, comme par exemple la distance entre deux points. En effet, la méthode des *k plus proches voisins* ($kPPV$) tient compte de la notion de *voisinage*. Ayant un ensemble de N échantillons de vecteurs formes bien classées (chacun admet une étiquette faisant référence à la classe à laquelle il appartient), il s'agit de classer un nouveau vecteur forme v. Pour ce faire, on détermine ses k plus proches voisins ($kPPV$=$\{v_1, ..., v_k\}$) [Tom76] parmi les échantillons de bases, ensuite une règle de décision permet de classer le nouveau vecteur forme (lui associer une étiquette). Le calcul des plus proches voisins suppose le choix d'une fonction de distance à l'avance comme la distance euclidienne, de Manhatan ou la distance du max. Ce choix est en général fonction de la nature des données. Le choix de k doit aussi être étudié avec précaution. En général, on prend k nettement inférieur à N, le nombre d'échantillons de base. La règle de décision pour classer v est la suivante :

$$v \in C_c \text{ avec } c = max_i(k_i)$$

avec k_i voisins appartenant à C_i, k_i est éventuellement nul et $\sum_i k_i = k$.

En général, cette règle est utilisée dans le cas de deux classes et dans ce

cas, k est impair ($k_1 \neq k_2$) pour éviter tout conflit. Si le nombre de classes est supérieur à deux classes, il est toujours possible de se ramener à deux classes en réappliquant la méthode sur les classes qui présentent un conflit ($k_i = k_j$).

L'utilisation de cette méthode, suppose l'existence d'un ensemble important de mesures, ce qui est souvent très difficile à obtenir car ces mesures peuvent être coûteuses.

4.4.5 Réseaux de neurones

De même que dans la précédente, on dispose d'un ensemble initial d'échantillons bien classés (ensemble d'apprentissage). Au lieu d'appliquer la règle des *kPPV* pour classer un nouvel échantillon, on choisit arbitrairement p vecteurs dans l'espace engendré par l'ensemble des classes. Ces vecteurs sont appelés *prototypes*. A l'aide d'un algorithme d'apprentissage par un réseau de neurones [KMSK91], les prototypes seront *guidés* pas à pas vers leurs positions optimales (zones de forte densité de probabilité). Pour classer un nouvel échantillon, c'est la règle de *1PPV* qui sera appliquée utilisant l'ensemble des prototypes comme référence au lieu de l'ensemble d'apprentissage initial. Pour permettre de discerner les différentes classes, un algorithme permet de représenter les prototypes dans un espace à deux dimensions sous forme d'un "*maillage*" appelé *carte* où deux "cellules" (prototypes) ont tendance à être activées par le même type de signaux en entrée [Koh84]. Ainsi, un nouvel échantillon aura tendance à activer des cellules voisines sur la carte et il est facile de le classer selon l'étiquette des cellules activées.

L'avantage de cette méthode sur la précédente et le gain qu'elle apporte en temps de calcul car la règle de *1PPV* est appliquée sur l'ensemble des p prototypes au lieu des N échantillons de base, ce qui réduit le nombre de distances à calculer, car p est nettement inférieur à N. Un autre avantage est celui de représenter l'ensemble des classes dans un espace à deux dimensions contre n dimensions dans le cas précédent.

4.4.6 Aspects temporels des ces approches

Les approches que nous venons de présenter, ont toutes besoin d'un ensemble d'apprentissage. Ne disposant pas de modèles explicites, cet ensemble constitue en quelque sorte un modèle implicite des modes de fonctionnement du système supervisé. Un mode de fonctionnement est représenté par une région dans cet espace d'apprentissage. La construction d'un tel ensemble suppose une longue série de tests et de validations afin de s'assurer de la qualité des données recueillies.

Outre la difficulté de construire ces ensembles notamment quand il s'agit d'une application où les tests sont coûteux en ressources et en temps, la faiblesse majeure de telles approches est leur incapacité de représenter la

composante dynamique du système surveillé. La seule information que ces échantillons révèlent sur l'évolution du système, réside dans la formation au cours du temps de nouvelles partitions dans l'ensemble d'apprentissage. Ceci se traduit par la détection éventuellement de nouveaux modes de fonctionnement. Néanmoins, nous ne disposons d'aucune *information temporelle* concernant cette évolution : chaque échantillon correspond à une "*image instantanée*" du comportement du système au moment où cet échantillon a été créé, mais il n'existe aucun lien précisant des relations telles que les relations de précédence entre la création des différents échantillons. Ainsi, des échantillons provenant de tests assez anciens, mélangés à des échantillons plus récents dans l'ensemble d'apprentissage, peuvent être considérés comme des parasites car ils ne sont plus les plus représentatifs de la dynamique du système. Un autre inconvénient qu'entraîne l'absence d'une telle information, est la quasi impossibilité de faire de la prédiction, n'ayant pas un modèle où l'évolution du comportement du système est représentée explicitement dans le temps.

Ces problèmes montrent la nécessité de représenter explicitement l'information temporelle et de pouvoir la manipuler, ce qui a donné naissance à des approches de supervision en intelligence artificielle.

4.5 Approches en intelligence artificielle

La difficulté croissante de modéliser des systèmes industriels par des modèles mathématiques a donné naissance à des modèles où la connaissance est modélisée sous forme symbolique et/ou numérique. La supervision de tels systèmes doit faire appel à des approches bien adaptées à ce genre de modèles. Dans les sections suivantes, nous présenterons un certain nombre d'approches utilisant des modèles où le *facteur temporel* est explicitement représenté. De cette manière et contrairement aux approches présentées précédemment, il est possible de représenter l'évolution du comportement du système surveillé comme étant des séquences qui se déroulent dans le temps.

4.5.1 Systèmes experts

Les systèmes experts étaient largement utilisés pour établir un diagnostic des systèmes statiques. Par la suite, ils ont été adaptés aux systèmes dynamiques, permettant ainsi, la prise en compte explicite du facteur temporel. De ce fait, il est devenu possible de *suivre* des systèmes dynamiques en temps réel.

Les connaissances déductives expriment explicitement les liens existants entre les événements observables et les situations à identifier. De plus, des contraintes temporelles numériques et/ou symboliques peuvent être exprimées sur les événements. Ces contraintes constituent un moyen pour limiter les temps de réponse.

La prise en compte du facteur temporel associe à chaque information, une valeur temporelle. De ce fait, il est nécessaire de prévoir un mécanisme d'oubli, permettant de supprimer les faits qui sont anciens et ne correspondent plus à la dynamique du système. Une difficulté supplémentaire provient du fait que de tels systèmes exigent souvent la complétude des connaissances que les experts sont incapables de fournir. Or, dans un contexte de système dynamique, il n'est pas rare de devoir suivre le système en se basant sur des connaissances incomplètes et parfois incertaines. Par ailleurs, nous devons signaler le problème d'inconsistance qui est commun à tous les systèmes à base de connaissance. En effet, les connaissances issues d'experts peuvent non seulement être incomplètes mais également inconsistantes. Afin d'éliminer les inconsistances, une étape de validation est indispensable.

Cependant, la simplicité du modèle, la facilité de représenter les connaissances, la garantie des temps de réponse font que les systèmes experts demeurent très employés dans les applications industrielles :

- le système *Alexip* [CB95] a été développé à l'*Institut Français du Pétrole* pour le suivi de processus de raffinement et de processus pétrochimiques ;
- le système *Chronos software* [ALA98] a été développé chez *France Télécom* pour la supervision du réseau *Transpac*. Une nouvelle approche est en cours d'étude dans le cadre du projet *GASPAR* pour parvenir plus facilement à la mise à jour des expertises face à des évolutions que présente le système supervisé ;
- chez *Sollac*, le projet *Sachem* [ALA98] vise à superviser un haut-fourneau en utilisant une approche système expert.

L'inconvénient majeur de tels systèmes provient du fait que la plupart d'entre eux nécessitent une connaissance complète du système supervisé et les experts sont souvent incapables de fournir une telle connaissance. De plus, comme ces systèmes permettent l'expression de contraintes temporelles, le *mécanisme d'oubli* devient plus important : les informations temporelles relatives à certains faits permettent d'invalider ces faits et ils doivent donc être supprimés de la base.

4.5.2 Approches par reconnaissance de scénarios

Dans ces approches, l'évolution du système se refère à des scénarios qui ont pour objet de distinguer certains types de comportements. Un système sera modélisé par des *scénarios normatifs* modélisant les modes de ses fonctionnements normaux et des *scénarios de dysfonctionnement.* Durant son évolution, on essayera de détecter l'écart entre les observations et les scénarios normatifs et si écart il y a , faire des appariements avec les scénarios

de dysfonctionnement. On parle ainsi de *reconnaissance de scénarios.* La reconnaissance d'un scénario permet de donner une information sur la nature du comportement du système. Cette reconnaissance est plutôt adaptée au *suivi* de systèmes dynamiques.

Une des difficultés majeures dans cette approche est due au fait que lors du suivi, on ne dispose que d'un ensemble plus ou moins limité de scénarios préprogrammés. Ainsi il devient difficile d'assurer la complétude des connaissances dont on dispose.

Un scénario est représenté par un graphe (appelé graphe de contraintes temporelles) où les nœuds correspondent aux événements et les arcs indiquent les contraintes temporelles entres ces différents événements.

Nous distinguons deux types de contraintes : les *contraintes simples* auxquelles sera associé un graphe simple et les *contraintes disjonctives* auxquelles sera associé un graphe disjonctif. Un exemple de contraintes simples entre deux événements **i** et **j** sera de la forme $D(i,j) = [a_{ij}, b_{ij}]$, un exemple de contraintes disjonctives sera de la forme: $D(i,j) = [a^1_{ij}, b^1_{ij}] \cup ... \cup [a^n_{ij}, b^n_{ij}]$.

Un algorithme de minimalisation de graphes permet de donner un graphe minimal qui restera équivalent au premier en terme de contraintes temporelles. Cet algorithme a principalement pour objet la détection d'une incohérence dans le graphe initial. Un graphe est incohérent si et seulement si le graphe minimal correspondant possède au moins une contrainte vide ($D(i,j) = \phi$).

Il existe trois approches basées sur la reconnaissance de scénarios. Toutes ces approches utilisent le modèle de scénarios, seuls les algorithmes diffèrent d'une approche à une autre.

Reconnaissance de motifs

Dans l'approche par reconnaissance de motifs dans une chaîne de caractères [L94], l'évolution du système est supposée être une sorte de séquence linéaire d'événements. De ce fait, les graphes sont linéaires et simples.

Le problème de reconnaissance est associé à un problème de reconnaissance de motifs dans une chaîne de caractères, pour laquelle il existe des algorithmes en temps linéaire. L'algorithme de Knuth, Morris et Pratt [KMP77] a été adapté pour faire du suivi de systèmes dynamiques : au fur et à mesure de l'arrivée des événements, le système superviseur tente de reconnaître un ou plusieurs scénarios. Ce type de reconnaissance est dit *en ligne.*

Propagation de contraintes

La reconnaissance par propagation de contraintes [DGG93] utilise la notion de chronique. Une chronique correspond à un scénario et donc à un graphe de contraintes temporelles entre classes d'événements. Au fur et à mesure de l'évolution du système, les instances possibles d'une chronique

seront créées et gérées sous forme d'arborescences par le systèmes superviseurs. Un événement peut appartenir à plusieurs chroniques à la fois. Dans une instance, un événement est associé à une fenêtre temporelle qui contient les instants valides pour une occurrence de cet événement. Ces fenêtres vont être réduites au fur et à mesure de la propagation des contraintes. Quand un événement se produit, sa date d'occurrence doit appartenir à la fenêtre qui lui est associée et dans ce cas, cette fenêtre est satisfaite. Une chronique est reconnue si toutes ses fenêtres (relatives à tous les événements) sont satisfaites.

La reconnaissance de chronique a été utilisée dans le projet *AUSTRAL* [LK97] pour l'analyse et la gestion de séquences d'alarmes issues de stations dans un réseau de distribution d'électricité en France. Les chroniques correspondent à la modélisation de cas de panne. Quand une panne est présente sur l'une des stations, la reconnaissance d'une chronique permet d'identifier la panne.

Le projet *GASPAR* [ALA98] a également utilisé la reconnaissance de chroniques. Ce projet vise à gérer des séquences d'alarmes issues d'équipements dans un réseau de télécommunications où un ensemble chroniques modélisent un ensemble dysfonctionnements.

Les algorithmes utilisés dans cette approche conviennent uniquement à une tâche de suivi et sont basés sur le logiciel *IxTeT* [Dou94][Dou96]. Leur complexité de l'ordre de $O(n^2)$ où n correspond au nombre de nœuds dans le graphe d'une chronique.

Comparaison de graphes

Dans cette approche, la reconnaissance d'un scénario est basée sur la comparaison de deux graphes de contraintes temporelles [Fon96]. Le premier graphe est celui de la *session*, il correspond à l'observation d'un comportement du système supervisé pendant une période de temps. Une session peut être construite à partir d'informations données par des experts ou à partir des occurrences d'événements provenant directement du système. Le second graphe correspond à un scénario.

La comparaison des deux graphes permet d'établir différents *modes reconnaissance*. En effet, le graphe de la session ne contient pas toujours tous les événements du scénario. De plus, ces événements peuvent ne pas être datés: à une occurrence d'événement peut correspondre un intervalle d'incertitude. En fonction de la précision des informations contenue dans la session, cette dernière peut *satisfaire totalement* ou *partiellement* le scénario, elle peut être *totalement* ou *partiellement compatible* avec le scénario.

A noter, que cette approche permet plusieurs types de reconnaissance: les sessions peuvent être construites a priori, et la comparaison des graphes permet d'établir un diagnostic sur des comportements passés. Les sessions peuvent être construites au fur et à mesure de l'arrivée des événements,

dans ce cas, la comparaison du graphe de la session courante avec le scénario permet de suivre le comportement du système.

La complexité algorithmique relative à la comparaison de graphes et la propagation des contraintes est en $O(n^3)$.

4.5.3 Réseaux temporels probabilistes

Un réseau probabiliste est un doublet (graphe, distribution de probabilité associée). Parmi ces réseaux, on distingue les réseaux Markoviens (graphes non orientés) et les réseaux Bayésiens (graphes orientés acycliques).

Dans un réseau bayesien, les sommets des graphes sont des variables aléatoires et pour chaque variable, on associe un ensemble de variables qui ont une influence directe sur cette variable. Cet ensemble sera appelé l'ensemble des "causes". Les variables sans causes représentent les racines du graphe. Ainsi, chaque arc du graphe relie deux variables dont l'une (origine de l'arc) est la cause de l'autre (extrémité de l'arc).

Chaque variable peut prendre un nombre fini de valeurs correspondant aux instantiations possibles. La distribution de probabilité associée permet de calculer la probabilité de chaque instantiation [SG94]. Un "état élémentaire" est défini par la donnée d'un fait ("variable=valeur") et d'une date. Pour chaque état élémentaire $X_i(t)$, on associera l'ensemble de ses causes $(X_j(t))$ se présentant au même temps t et celles se présentant antérieurement $(X_j(t-1))$ au temps $t-1$. Ainsi, on peut représenter sous la forme d'un réseau Bayesien ordinaire la probabilité $P(X(t)/X(t-1)$ (probabilité de l'état du monde au temps t sachant son état au temps $t-1$. Finalement en calculant $P(X(0))$ (racine du graphe), $P(X(1)/X(0))$, ..., $P(X(n)/X(n-1))$, on peut représenter les évolutions possibles du monde entre les instants 0 et n. Pour faire du suivi de systèmes, on est amené à calculer les probabilités

$$P(X_i(TempsPresent) = Anomalies_i/EnsembleObservations)$$

Pour faire du diagnostic, on est amené à calculer les configurations les plus probables du monde étant données les observations.
Enfin, pour faire de la prédiction, il suffit de dérouler l'algorithme jusqu'à l'instant voulu dans le futur.

La limite principale d'une telle approche provient de l'absence de possibilité de représenter les variables au sens des logiques du premier ordre. En effet, les états élémentaires correspondent à des variables d'ordre 0 et de ce fait, il est difficile d'utiliser cette approche dans un problème de supervision de grande taille.

4.5.4 Evénements discrets

Dans cette approche un système sera vu à travers ses états possibles [CT94]. Ainsi, son comportement sera défini comme une séquence d'états,

le système part d'un état initial S_0 et termine dans un état final S_n pendant l'intervalle de temps $[t(S_0), t(S_n)]$. Un scénario sera défini comme une séquence ou un chemin dans le graphe d'états étiqueté par les événements faisant en sorte que le système transite d'un état à un autre. Ce chemin, aura une signification particulière et est capable de qualifier le comportement du système. Un système sera décrit par le couple (S, E) où S est l'ensemble des états possibles du système et E est l'ensemble d'événements permettant de passer d'un état à un autre dans S.

Un diagnostic sera décrit par $((S, E), (OBS_1, ..., OBS_n))$ où OBS_1, ..., OBS_n sont les états possibles correspondant aux faits observés. Il consiste à trouver un scénario qui part d'un état $S1$ "cohérent" avec OBS_1 et arrive à un état S_m cohérent avec OBS_m, en passant par des états intermédiaires cohérents respectivement avec OBS_2, ..., OBS_{m-1}.

L'existence d'un moyen (heuristiques) permettant d'affecter une probabilité aux états possibles du système, permettra de dégager les chemins les plus probables dans le graphe des états et donc les scénarios les plus probables, on parle ainsi de *diagnostics préférés.* Les algorithmes de calculs de tels diagnostics utilisés sont du type "*recherche de focalisation*" dans un graphe d'états construit dynamiquement. Ils sont a priori très coûteux et dépendent fortement des heuristiques de focalisation utilisées.

La prise en compte du temps d'une manière explicite, est souvent nécessaire dans un contexte de systèmes dynamiques. Des contraintes temporelles peuvent exister entre les différentes transitions d'un état à un autre. Pour ce faire, le formalisme d'automate temporel peut être utilisé. Ce formalisme a été utilisé pour la modélisation et la supervision du réseau Transpac [BCD+96][Roz97].

4.5.5 Graphes causaux

Cette approche est basée sur le raisonnement suivant : ayant un ensemble d'observations et un ensemble de relations liant causes et effets. Il est possible de déduire l'ensemble des causes ayant conduit à l'ensemble des observations (effets). Ce type de raisonnement est assez naturel et est appelé *raisonnement abductif.* En effet, ayant établi un fait B, et la relation "$A \longrightarrow B$" (A est une cause de B), nous pouvons inférer que A est possible.

Le raisonnement abductif est assez adapté pour le diagnostic [PR90]. Un défaut est modélisé par un ensemble M de formules respectant les formats suivants :
$C \longrightarrow E$: C cause E
$C \wedge C \longrightarrow E$: C et C causent E
$C \wedge \alpha \longrightarrow E$: C peut causer E sous certaines conditions non spécifiées, représentées par la condition abstraite α. L'ensemble de formules M peut être représenté par un graphe orienté où un nœud correspond à une cause ou un effet et un arc lie une cause à un effet.

Un ensemble d'observations Ψ, contient généralement des *observations positives* (E) et des *observations négatives* ($\neg E$). Les observations positives confirment la présence d'un symptôme et les observations négatives confirment l'absence d'autres symptômes.

Un diagnostic consiste à établir une *explication* de Ψ. Cette explication correspond à un ensemble de formules (disjonction de conjonctions) qui, en utilisant les formules de M permettent d'établir les observations positives de Ψ, tout en restant cohérentes avec les observations négatives.

Naturellement, dans le cadre du diagnostic de systèmes dynamiques, le facteur temporel doit être exprimé au niveau des formules. Pour ce faire, les observations peuvent être datées et une contrainte temporelle peut être posée au niveau d'une relation de causes à effets.

Le système DIAPO [PR97], un système de surveillance d'une pompe dans une centrale nucléaire chez EDF (Electricité de France), utilise un raisonnement abductif et un raisonnement temporel.

La limite actuelle du système est due au fait, qu'il est parfois incapable de gérer plusieurs cas de panne qui se produisent simultanément. En effet, pour limiter la complexité combinatoire du modèle, les défauts sont tous considérés indépendemment : la présence de deux défauts implique une conjonction de leurs effets qui n'est pas exprimée explicitement dans le modèle. S'il n'existe pas d'interaction entre les deux défauts et s'ils n'engendrent pas, dans ce cas, de nouveaux effets non décrits par le modèle, le diagnostic sera correct. Dans le cas contraire, le diagnostic peut échouer.

4.5.6 Graphes d'influence

Tout comme l'approche précédente, cette approche est basée sur le principe de causalité [Dav83]. Elle utilise en plus, la physique qualitative pour modéliser le système supervisé. Cette modélisation se fait à l'aide d'un graphe orienté appelé *graphe d'influence*. Les nœuds du graphe représentent des variables continues modélisant des évolutions continues du systèmes (changement de température, de pression). L'évolution de ces variables est décrite numériquement. Les arcs décrivent l'influence existant entre les différentes variables. Un graphe d'influence est une structure qui peut être enrichie en fonction des informations dont on dispose. Les arcs peuvent préciser la direction d'une influence (variable origine admet de l'influence sur la variable extrémité), tout comme ils peuvent correspondre à des fonctions dynamiques complexes.

Les graphes d'influence sont des outils bien adaptés pour le filtrage d'alarmes. En effet, les alarmes issues du système suite à une anomalie sont souvent dépendantes les unes des autres. Cette dépendance se manifeste sous forme de causalité. L'opérateur humain face à une *avalanche* d'alarmes, doit analyser ces alarmes afin de détecter l'alarme correspondant au défaut source. De cette manière, il identifie le défaut et en même il le localise physi-

quement dans le système, et ceci en examinant les variables concernées par l'alarme en question. Face à une liste d'alarmes, un graphe d'influence permet de construire un arbre dont la racine constitue l'alarme source et dont les branches constituent des chemins de propagation de cette alarme. Une alarme source correspond à un *défaut primaire* qui est lié directement à une panne sur la machine. Une alarme faisant partie d'une branche correspond à un *défaut secondaire*. La construction du graphe d'influence passe par l'observation d'une évolution anormale des variables et l'objectif est de localiser la variable dont la déviation peut causer la déviation des autres variables.

Il est toujours important de localiser le défaut source lors d'un diagnostic. Ceci permet aux opérateurs de prendre les mesures nécessaires afin de remettre le système en état de marche. Malheureusement, la propagation des défauts, nécessite souvent de tester les chemins de propagation dans le graphe d'influence. Des tests locaux entre une variable et ses antécédents peuvent être effectués en évaluant la consistance entre les arcs (fonctions dynamiques) et l'*état* des variables. Ces tests sont plus ou moins complexes, selon la définition des arcs et la définition de l'état d'une variable.

Ce type d'approche convient naturellement au suivi de processus continus tels les processus chimiques ou nucléaires. Le système *TIGER* a été développé pour le suivi d'une turbine à gaz [CDT89], un de ses modules (Ca En) [BTM94] utilise un graphe causal pour détecter les anomalies, ainsi que pour interpréter les alarmes quand une anomalie se présente.

4.5.7 Aspects temporels de ces approches

Les différentes approches présentées permettent de se rendre compte des travaux menés dans le cadre de la supervision de systèmes dynamiques. De tels systèmes sont capables d'évoluer et de passer d'un état à un autre au cours du temps. Un tel phénomène accorde au facteur temporel un rôle crucial dans la supervision. En effet, chaque approche a sa propre représentation du temps et des contraintes temporelles et cette représentation la rend plus adaptée à tel ou tel type de supervision (suivi, diagnostic, prédiction).

Une autre caractéristique importante de ces approches est qu'elles sont toutes basées sur la notion de *graphes d'états* (exception faite de l'approche par systèmes experts). Un système industriel présente un ensemble d'états et son évolution se traduit par un chemin de transitions dans le graphe d'états. Malheureusement, dans le cas de systèmes dynamiques et complexes, la construction d'un tel graphe est quasi impossible et même si l'on parvenait à le construire, le volume d'un tel graphe poserait des problèmes d'efficacité des algorithmes utilisés. Les approches par réseaux probabilistiques et celle à base d'événements discrets, reposent toutes les deux sur la notion d'un graphe d'état. Par souci d'efficacité et pour réduire les problèmes de modélisation, les approches par reconnaissance de scénarios ont adopté une autre technique. Au lieu d'avoir un seul graphe traduisant tous les états pos-

sibles du système, seuls un certain nombre de sous-graphes seront considérés. L'avantage d'une telle technique est double :

- d'une part, les algorithmes utilisés se dérouleront sur ces sous-graphes qui sont de plus petite taille que le graphe initial, d'où le gain en terme de performance.
- d'autre part, il est plus facile de modéliser seulement des parties du graphe initial qu'une modélisation globale.

L'inconvénient majeur de telles techniques provient du fait que le modèle ainsi proposé devient incomplet et partiel. Il est alors impératif de s'assurer de la fiabilité des systèmes de supervision basés sur de tels modèles.

4.6 Discussion

Dans ce chapitre, nous avons présenté un certain nombre d'approches pour la supervision, tout en mettant l'accent sur la nécessité d'exprimer explicitement le facteur temporel dans ces approches, surtout quand il s'agit de superviser un système dynamique.

Pour cette raison, la supervision de systèmes dynamiques devient un domaine d'application bien adapté pour traiter le temps dans les systèmes multi-agents. En effet, tous les aspects temporels qu'on retrouve au niveau de la supervision, doivent au moins se trouver aux niveaux individuels des agents (section 3.3) : l'agent doit manipuler et raisonner sur des informations temporelles concernant le système supervisé. Si de plus, une coopération est mise en place afin de permettre aux agents de mener à bien la tâche de supervision, certains aspects temporels doivent être traités aux niveaux sociaux des agents (section 3.4) : les agents doivent s'échanger des informations temporelles portant sur le système supervisé et dans certains cas, il sont soumis à des contraintes temporelles (section 3.4.3).

Inversement, lorsque la supervision est effectuée par un ensemble d'agents, le modèle utilisé doit être adapté à un contexte distribué. Cette adaptation doit tenir compte des problèmes énoncés dans la section 4.3, afin de contribuer à leur résolution (cf. chapitre 8).

Deuxième partie

Société d'agents temporels

Introduction

L'objectif de cette partie est de traiter les aspects temporels dans un système multi-agent. Pour ce faire, nous proposons un modèle d'organisation, un modèle d'interaction et enfin, un modèle d'agent adapté au modèle de société. Tous ces modèles intègrent des caractéristiques temporelles :

- dans le premier chapitre, nous présentons un modèle d'organisation basé sur les relations de dépendance. Dans ce modèle, le facteur temporel est pris en compte dans le raisonnement social des agents ;
- dans le deuxième chapitre, nous présentons un modèle d'interaction basé sur les actes de langage. Le temps est représenté explicitement à la fois dans l'expression et la sémantique du langage. Un ensemble de protocoles d'interaction assure le contrôle des conversations entre les agents ;
- enfin, dans le dernier chapitre, nous présentons un modèle d'agent temporel où le temps est pris en compte dans chaque fonction de l'agent. Un tel modèle est bien adapté pour résoudre des problèmes où le facteur dynamique est crucial.

Dans cette partie, le modèle d'organisation définit les différents éléments du niveau social qui doivent être pris en compte par l'agent. Le modèle d'agent définit les éléments du niveau individuel. Quant au modèle d'interaction, il assure un lien entre les niveaux individuels et sociaux des différents agents.

Chapitre 5

Modèle d'organisation temporelle

5.1 Introduction

Afin d'accomplir une tâche complexe à l'aide d'une société d'agents, il est essentiel que chaque agent sache quelles sont les tâches qui lui sont allouées et comment il doit les accomplir tout en restant cohérent avec les autres membres de la société. Pour ce faire, nous définissons le *domaine de responsabilité individuelle* d'un agent. Il constitue l'ensemble des tâches que cet agent est capable d'accomplir.

Cette définition ne suffit pas pour assurer la coordination des différentes tâches effectuées par plusieurs agents. En effet, en se limitant à cette définition, un agent peut fonctionner isolément et il devient absurde de parler d'une société d'agents puisque le *niveau social* est absent. Le *domaine de responsabilité commune* est défini comme étant l'ensemble des tâches globales dont l'accomplissement dépend de plus d'un seul agent.

Il est important de préciser que le domaine de responsabilité commune se situe au niveau social ou au niveau collectif. En effet, si l'on considère un agent devant exécuter une tâche afin qu'un autre agent puisse accomplir une tâche de plus haut niveau, le domaine de responsabilité commune entre les deux agents relève du niveau social. D'un point de vue externe, si l'on considère la tâche de haut niveau comme étant un but global pour plusieurs agents, ce domaine se situe au niveau collectif.

Le succès d'une de ces tâches, dépend de l'exécution avec succès d'un ensemble de tâches appartenant aux domaines de responsabilité individuelle des agents. Les relations pouvant exister entre ces différentes tâches, permettent de déduire des *relations de dépendances* entre les agents. Afin de gérer des tâches appartenant au domaine de responsabilité commune, les agents concernés sont amenés à raisonner les uns sur les autres en utilisant leurs relations de dépendance.

La nature dynamique des tâches, ainsi que les contraintes temporelles qui peuvent exister entre elles, rendent dynamiques les dépendances entre agents. Au fur et à mesure de l'évolution des tâches, des relations de dépendances vont être considérées par les agents. La durée de prise en compte de ces relations est liée directement à l'exécution des tâches. De ce fait, il est important d'exprimer la composante dynamique de l'ensemble des relations de dépendance. Ceci permettra à un agent de connaître la durée de validité d'une dépendance qu'il a avec un autre agent.

Dans ce chapitre, nous définissons les domaines de responsabilité des agents et montrons comment ces agents vont utiliser leurs relations de dépendances pour interagir afin de résoudre un problème commun. A travers cette étude, nous mettons l'accent sur les aspects dynamiques situés au niveau social. Ceux situés au niveau individuel seront étudiés dans le chapitre 7.

5.2 Définitions préliminaires

Dans le cadre de notre étude, nous considérons une société formée de n agents, elle sera représentée par l'ensemble : $A = \{a_1, ..., a_n\}$.

5.2.1 Ressources

Les agents ont à leur disposition un ensemble de q *ressources*: $R = \{r_1, ..., r_q\}$. Une ressource peut correspondre à une entité physique faisant partie de l'environnement ou alors à une connaissance que l'agent est libre d'utiliser au cours de son raisonnement

5.2.2 Tâches

Les ressources peuvent être utilisées par un ensemble de p *tâches* qui peuvent être accomplies par l'ensemble des agents: $T = \{t_1, ..., t_p\}$. Une tâche peut être exécutée par un seul agent à la fois. Une tâche t_i ayant besoin des ressources $r_{i_1}, ..., r_{i_j}$ pour son exécution sera notée par $t_i(r_{i_1}, ..., r_{i_j})$. Elle définit une application de T dans $\mathcal{P}(R)$ où $\mathcal{P}(R)$ représente l'ensemble des parties de R. Cette application correspond à la dépendance de type Tâche-Ressource définie dans le chapitre 2.

Pour illustrer ces notions ainsi que celles qui seront définies dans les sections suivantes, nous utilisons un exemple du monde des blocs, que nous allons développer tout au long du chapitre. Nous avons trois blocs A, B et C et leurs positions sont indiquées sur la figure 5.1. Dans ce contexte, les trois blocs représentent les ressources et un agent peut effectuer seulement deux types de tâches sur ces blocs :

- *Empiler(X, c_i, c_j)*, déplace le bloc X (A, B ou C) de la colonne c_i au sommet de la colonne c_j ;

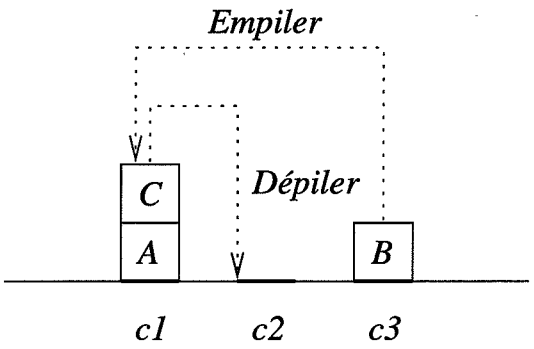


Figure 5.1 : Les positions respectives des trois blocs avec deux types de tâches possibles : *Empiler* et *Dépiler*.

- *Dépiler(X)*, dépile le bloc X vers la première colonne libre.

Relations entre tâches

La définition de tâches complexes crée des relations entre les différentes tâches. Dans le cadre de notre travail, deux types de relations nous intéressent :

- *Tâche/sous-tâche.* Une tâche peut admettre une ou plusieurs sous-tâches. La relation entre une tâche t_i et ses sous-tâches sera définie par l'application s définie comme suit :
  $s : T \longrightarrow \mathcal{P}(T)$
  où $\mathcal{P}(T)$ représente l'ensemble des parties de T. Ainsi, $s(t_i)$ définit l'ensemble des sous-tâches de la tâche t_i. Afin qu'une tâche soit accomplie, il est nécessaire que toutes ses sous-tâches le soient également. De la même façon, on peut définir l'ensemble des tâches dont t_i fait partie : $a(t_i) = \{t_j | t_i \in s(t_j)\}$.
  D'après cette définition, il est facile de remarquer que plusieurs tâches peuvent se partager la même sous-tâche. La relation tâche/sous-tâche correspond à la dépendance de type Tâche-Tâche définie dans le chapitre 2. Une tâche t_i est dite *élémentaire* si elle n'admet pas de sous-tâches ($s(t_i) = \phi$). Dans l'exemple, les tâches *Empiler* et *Dépiler* sont les seules tâches élémentaires ;

- *Opposition.* Deux tâches t_i et t_j sont opposées si elles ne peuvent être exécutées simultanément. Cette opposition sera notée par $t_i \leftrightarrow t_j$. La définition exacte de cette relation dépend du problème à résoudre. Dans notre exemple, il est impossible d'exécuter au même instant, deux tâches élémentaires faisant intervenir la même colonne ou le même bloc. Nous avons ainsi, $Empiler(A, c1, c2) \leftrightarrow Empiler(C, c3, c2)$.

Pour illustrer la relation tâche/sous-tâche dans notre exemple, nous considérons les deux tâches de haut niveau t_{ABC} et t_{CBA}. La première consiste à déplacer les trois blocs pour les ranger dans n'importe quelle colonne et ceci dans l'ordre suivant : A est à la base de la pile, ensuite B est posé sur A et enfin C est au sommet. La deuxième consiste à placer les blocs dans l'ordre inverse. Ces deux tâches peuvent être décomposées en une succession de plusieurs tâches élémentaires :
$s(t_{ABC}) = \{Dépiler(C), Empiler(B, c3, c1), Empiler(C, c2, c1)\}$,
$s(t_{CBA}) = \{Dépiler(C), Empiler(B, c3, c2), Empiler(A, c1, c2)\}$.
Nous remarquons que ces deux tâches admettent une sous-tâche commune qui est $Dépiler(C)$ d'où t_{ABC}, $t_{BCA} \in a(Dépiler(C))$.

Contraintes temporelles entre tâches

Le fait qu'une tâche t_i soit en cours d'exécution sur un intervalle I, sera noté par $[t_i]_I$. Le début de l'exécution de cette tâche sera noté $d(t_i)$ et la fin de son exécution sera noté $f(t_i)$. Naturellement, l'exécution de cette tâche est pris en charge par un agent a_u, nous définissons les deux opérateurs : $[\,]_I$ et $:<>_t$, exprimant par $a_u : [t_i]_I$, que la tâche t_i est en cours d'exécution par l'agent a_u sur l'intervalle I et par $a_u :< t_i >_t$, qu'à l'instant t, la tâche a fini d'être exécutée par cet agent. Nous pouvons ainsi écrire :

$$\boxed{a_u : [t_i]_I \Longleftrightarrow \forall J \mid J\{e, s, f, d\}I, a_u : [t_i]_J}$$

Cette formule précise que si une tâche est en cours d'exécution sur un intervalle I, elle l'est également sur tout sous-intervalle J de I. A noter l'utilisation des symboles définissant les treize positions pouvant exister entre deux intervalles. Ces symboles sont introduits dans le chapitre 3. La formule suivante permet de définir l'intervalle de temps correspondant à l'exécution totale de la tâche :

$$\boxed{a_u :< t_i >_t \Longleftrightarrow a_u : [t_i]_{[d(t_i),t]} \ \wedge \ \forall I \mid [d(t_i), t]\{e, s, f, d\}I, \neg(a_u : [t_i]_I)}$$

La relation tâche/sous-tâche et la relation d'opposition entre tâches définissent d'une manière naturelle un ensemble de contraintes temporelles sur le déroulement de chacune de ces tâches. Par exemple, la relation existant entre la tâche t_{ABC} et ses sous-tâches exige d'une part que les sous-tâches soient exécutées dans un certain ordre et d'autre part, nous ne pouvons dire que t_{ABC} a été exécutée que si les trois sous-tâches ont été exécutées auparavant. En supposant, qu'une tâche élémentaire prend exactement une unité de temps ($f(t_i) - d(t_i) = 1$), nous pouvons représenter sur un graphe les délais minimal et maximal entre le début et la fin de chacune des tâches (figure 5.2). Les nœuds correspondent aux débuts ou fins de tâches, les arcs sont étiquetés d'intervalles dont les bornes correspondent aux délais minimum et maximum à respecter entre les deux nœuds, début et fin des différentes tâches. Quand il n'existe pas de contrainte entre deux nœuds, l'arc

correspondant portera l'étiquette $[-\infty, +\infty]$. Pour exprimer des contraintes de précédence sans délai (plus tard, plus tôt), nous utilisons les valeurs $+\infty$ et $-\infty$.

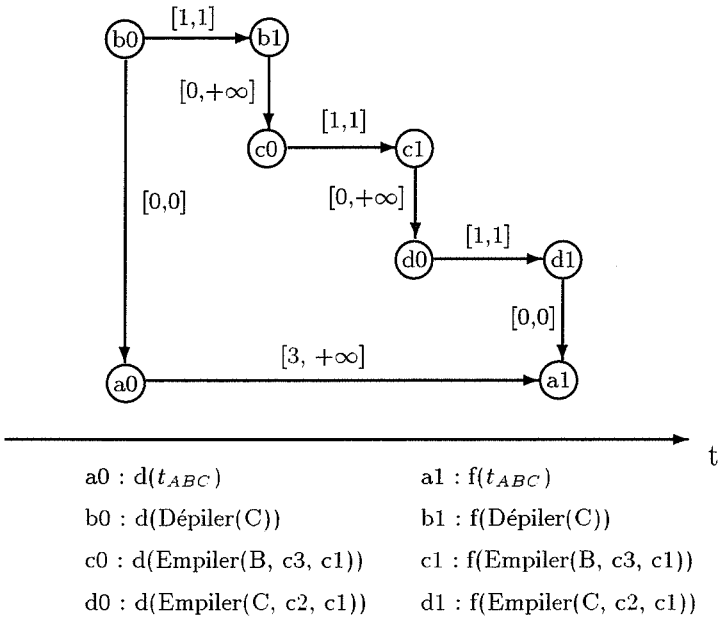


Figure 5.2 : Les délais minimal et maximal entre le début et la fin de la tâche t_{ABC} et ses sous-tâches.

D'une manière générale, les contraintes temporelles entre les nœuds ne correspondent pas uniquement à des contraintes de précédences. Par exemple, une tâche qui consiste en l'assemblage d'une pièce mécanique peut être décomposée en plusieurs sous-tâches qui peuvent être exécutées dans un ordre différent, selon la disponibilité des ressources.

5.2.3 Domaines de responsabilité

La réalisation d'une tâche par plusieurs agents nécessite une coordination entre eux. Cette coordination consiste à définir pour chacun des agents comment il doit exécuter un certain nombre de tâches, afin que la tâche globale soit réalisée. Cette définition représente les différentes responsabilités de l'agent [1].

[1] Par abus de langage, nous parlerons indifféremment de *responsabilité* ou *domaine de responsabilité*

Domaine de responsabilité individuelle

Le *domaine de responsabilité individuelle* est constitué d'un ensemble de tâches dont un agent est *responsable*. L'accomplissement de ces tâches est indépendant des autres tâches ou des autres agents. Un agent exécutera une tâche appartenant à son domaine de responsabilité individuelle, soit pour atteindre un but personnel, soit pour participer, dans un processus de coordination, à l'accomplissement d'une tâche plus complexe.

Nous définissons la responsabilité individuelle d'un agent par l'application $r : A \longrightarrow \mathcal{P}(T)$.

Le concepteur du système doit définir pour chaque agent, l'ensemble des tâches qu'il pourra exécuter. Cette définition peut évoluer au cours du fonctionnement et ceci en fonction de la possibilité de répartition de tâches. L'affectation des domaines de responsabilité aux agents est souvent basée sur un critère et ceci en fonction du problème à résoudre. Par exemple, si l'on souhaite superviser les machines-outils d'un atelier de production par une société d'agents, la répartition des tâches de surveillance peut être basée sur la distribution physique des machines dans l'atelier ou alors sur sa structure fonctionnelle hiérarchique (l'atelier est vu comme un ensemble de composants formés eux mêmes de plusieurs composants).

A noter, que l'intersection de deux domaines de responsabilité individuelle n'est pas forcément vide. Deux agents peuvent avoir une même tâche dans leurs domaines de responsabilité individuelle. L'intérêt d'un tel recouvrement est de pouvoir trouver une alternative quand un agent est momentanément incapable d'exécuter une tâche appartenant à son domaine de responsabilité individuelle.

Domaine de responsabilité commune

L'affectation des domaines de responsabilité individuelle reste insuffisante pour mettre en place une coopération afin d'accomplir une tâche complexe. En effet, un agent doit être conscient que l'exécution d'une tâche appartenant à son domaine de responsabilité individuelle, est destinée à résoudre une tâche de plus haut niveau, qui correspond à un but global qu'il faut atteindre. En l'absence de cette considération, l'agent en question est capable d'exécuter d'autres tâches sans se rendre compte qu'elles peuvent empêcher l'accomplissement d'une tâche globale.

Pour un agent a_u, nous définissons le domaine de responsabilité commune comme suit, $cr : A \longrightarrow \mathcal{P}(T)$:

$$\boxed{t \in cr(a_u) \Longleftrightarrow t \notin r(a_u) \ \wedge \ \exists t_i \in s(t) \mid t_i \in r(a_u)}$$

Pour illustrer cette idée, revenons à notre exemple où deux agents a_1 et a_2 doivent exécuter la tâche T_{CBA}, avec l'hypothèse suivante : a_1 admet

dans son domaine de responsabilité individuelle toutes les tâches élémentaires pouvant manipuler le bloc A (on dira que a_1 est responsable du bloc A) et a_2 est responsable des blocs B et C. Afin d'exécuter cette tâche, les deux agents doivent envisager une coopération. Un scénario possible est le suivant : l'agent a_2 s'engage à exécuter les tâches *Dépiler(C)* et *Empiler(B,c3,c2)* et l'agent a_1 s'engage à exécuter la tâche *Empiler(A,c1,c2)*. D'après la définition des domaines de responsabilité individuelle des deux agents et la définition d'un domaine de responsabilité commune, nous avons : $T_{CBA} \in cr(a_1) \cap cr(a_2)$.

Le point essentiel à retenir de cet exemple, est que la nécessité d'une coopération a poussé les agents à se coordonner pour exécuter indépendamment chacun un ensemble de sous-tâches de façon à accomplir une tâche de plus haut niveau. Cette volonté à coopérer se place au niveau social et elle prouve que les agents possèdent une sorte de *responsabilité commune* [Jen92b].

Ayant défini les domaines de responsabilité individuelle et commune, les contraintes temporelles entre tâches agissent à la fois au niveau individuel et au niveau social. En effet, dans notre exemple, lorsque l'agent a_2 exécute la tâche *Dépiler(C)*, il doit tenir compte du moment où il commence son exécution et de la durée de cette exécution. A ce niveau, l'agent a_2 est seul concerné par ces contraintes. Mais si par contre, on place l'exécution de cette tâche dans le contexte d'exécution de la tâche t_{CBA}, ces contraintes concernent à présent l'agent a_1 également, car l'agent a_2 est incapable d'exécuter la totalité des sous-tâches à lui tout seul et l'ordre d'exécution des trois sous-tâches doit être respecté par les deux agents. Cette distinction entre les deux niveaux joue un rôle important lors de l'étude des interactions entre agents qui sera détaillée dans le chapitre 6.

5.3 De la responsabilité commune vers les dépendances

L'existence des différentes relations entre tâches présentées dans la section 5.2.2, montrent que les tâches admettent des dépendances. Ces dépendances se traduisent par des *dépendances fonctionnelles* entre agents [LCd96]. Le concepteur du système ayant défini l'ensemble des tâches, ainsi que les domaines de responsabilité individuelle et commune des agents, chaque agent est capable de déterminer ses dépendances avec les autres. Ces dépendances sont exprimées sous forme d'un réseau, appelé *réseau de dépendances*. Ce réseau permet à chaque agent de connaître les agents dépendants de lui et ceux dont il dépend. Il est important de signaler que ces relations de dépendance relèvent du niveau social, elles interviennent dans le raisonnement d'un agent sur les autres.

Les deux relations entre tâches qui nous intéressent et que nous avons

définies dans la section 5.2.2, sont la relation *tâche/sous-tâche* et la relation d'*opposition*. A partir de ces deux relations, nous pouvons définir deux types de relation de dépendances entre agents: la première est exprimée en terme de *besoin* et l'autre est exprimée sous forme d'*aide*. Mais avant d'analyser plus en détail ces deux relations, nous présentons également un troisième type de dépendance exprimé cette fois-ci sous forme de *concurrence*. En effet, cette dépendance est due au recouvrement autorisé entre les domaines de responsabilité individuelle des agents. L'évolution de l'exécution des tâches dans le temps, rend les relations de dépendances dynamiques. Elles sont tantôt actives tantôt inactives. Cette notion de *dépendance active* constitue un moyen permettant à l'agent de connaître *à chaque instant*, les agents qu'il doit contacter pour leur demander de l'aide ou au contraire pour leur proposer ses services. De cette manière, un agent possède un réseau de dépendances composé de deux parties principales : la première est un graphe *statique* [ASB97, ABS97] représentant toutes les dépendances avec les autres relatives aux différentes tâches, et la deuxième, est une instantiation *dynamique* du premier graphe où seules les dépendances actives sont représentées.

Dans les sections suivantes, nous définissons chacune des relations et montrons tous les aspects dynamiques qui les caractérisent.

5.3.1 Relation de concurrence

L'autorisation d'éventuels recouvrements entre les domaines de responsabilité individuelle des agents, introduit la notion de *concurrence* entre deux agents a_u et a_v ayant la même tâche t_i dans leurs domaines de responsabilité individuelle $(t_i \in r(a_u) \cap r(a_v))$.

Définition statique

Cette concurrence est exprimée comme étant une dépendance entre les deux agents et est définie par une relation $\mathcal{C}$ comme suit :

$$\boxed{a_u \mathcal{C}_{t_i} a_v \Longleftrightarrow t_i \in r(a_u) \cap r(a_v)}$$

La relation $\mathcal{C}$ est une relation d'équivalence entre les agents. Elle est déduite des propriétés de l'intersection ensembliste.

Définition dynamique

La relation de concurrence sur une tâche est utile uniquement dans un contexte où une tâche doit être exécutée ou si elle est déjà en cours d'exécution. De ce fait, il est intéressant de pouvoir considérer cette relation sur un intervalle de temps : la relation de concurrence sur la tâche t_i entre les agents a_u et a_v est active sur un intervalle I, si et seulement si la tâche est

en cours d'exécution par l'un des deux agents et ceci pendant un intervalle J contenant I. Ceci peut être exprimé par la formule suivante (l'opérateur $\oplus$ définit la disjonction exclusive) :

$$a_u \mathcal{C}_{t_i}(I) a_v \Longleftrightarrow a_u \mathcal{C}_{t_i} a_v \ \wedge \ (a_u : [t_i]_J \ \oplus \ a_v : [t_i]_J) \ \wedge \ I \ \{d, s, f, e\} \ J$$

Continuité dans le temps

La règle de continuité dans le temps d'une concurrence est donnée par la formule suivante :

$$a_u \mathcal{C}_{t_i}(I) a_v \ \wedge \ (a_u : [t_i]_J \ \oplus \ a_v : [t_i]_J) \ \wedge \ I\{s, d, f\}J \Longrightarrow a_u \mathcal{C}_{t_i}(J) a_v$$

Exemple 5.1 : Relation $\mathcal{C}$.

Une relation de concurrence entre les deux agents a_1 et a_2 peut être exprimée par le fait qu'ils se partagent en plus, la responsabilité du bloc B. Dans ce cas, les deux agents vont avoir dans leurs réseaux de dépendances, respectivement les relations suivantes : $a_1 \mathcal{C}_{Empiler(B,c3,c1)} a_2$ et $a_2 \mathcal{C}_{Empiler(B,c3,c1)} a_1$. Si la tache *Empiler(B, c3, c1)* est en cours d'exécution par l'un des deux agents, ces deux relations deviennent actives et le restent pendant toute la durée d'exécution de cette tâche.

5.3.2 Relation de besoin

Le fait qu'une tâche t_i appartenant au domaine de responsabilité d'un agent a_u ($t_i \in r(a_u)$) admette une sous-tâche t_j ($t_j \in s(t_i)$) appartenant au domaine de responsabilité d'un autre agent a_v ($t_j \in r(a_v)$), crée une relation de dépendance entre les deux agents, exprimée en terme de besoin. En effet, pour exécuter la tâche t_i, l'agent a_u aura besoin des services de l'agent a_v pour exécuter la sous-tâche t_j.

Définition statique

Cette relation de dépendance, notée par $\mathcal{D}$, est définie comme suit :

$$a_u \mathcal{D}_{t_j}^{t_i} a_v \Longleftrightarrow t_i \in r(a_u) \ \wedge \ t_j \in r(a_v) \cap s(t_i) \ \wedge \ t_j \notin r(a_u)$$

Définition dynamique

La définition que nous avons donnée de la relation $\mathcal{D}$ est en fait, une définition *statique* et elle va servir à la construction des réseaux de dépendances de chacun des agents.

Pourtant, une telle relation de dépendance, est utilisée seulement dans un contexte bien précis : il s'agit de savoir quand est-ce que cette dépendance est *active*, c'est à dire quand est-ce qu'un agent aura effectivement besoin de tenir compte de cette dépendance. En effet, d'une part, son activation

dépend des déroulements des tâches: un agent ne tiendra pas compte de cette dépendance si aucune des tâches n'a besoin d'être exécutée et d'autre part, cette activation dépend de la terminaison des tâches. Une dépendance active de la forme $a_u \mathcal{D}^{t_i}_{t_j} a_v$ n'est plus active à partir du moment où la sous-tâche t_j a été exécutée. Nous représentons le fait qu'une dépendance est active sur un intervalle I par $a_u \mathcal{D}^{t_i}_{t_j}(I) a_v$. Cette activation est définie par la formule suivante:

$$\boxed{a_u \mathcal{D}^{t_i}_{t_j}(I) a_v \Longleftrightarrow a_u \mathcal{D}^{t_i}_{t_j} a_v \ \wedge \ a_v : [t_j]_J \ \wedge \ a_u : [t_i]_K \ \wedge \ I = J \cap K}$$

Continuité dans le temps

Une dépendance active sur un intervalle I, reste active sur tout intervalle où les tâches t_i et t_j sont en cours d'exécution simultanément:

$$\boxed{a_u \mathcal{D}^{t_i}_{t_j}(I) a_v \ \wedge \ a_u : [t_i]_J \ \wedge \ a_v : [t_j]_K \ \wedge \ I\{d\}J \cap K \Longrightarrow a_u \mathcal{D}^{t_i}_{t_j}(J \cap K) a_v}$$

Cette dépendance cesse d'être active, si l'agent a_v a fini d'exécuter la sous-tâche t_j à l'instant t. Nous rappelons que I^- et I^+ permettent l'accès respectivement aux bornes inférieure et supérieure de l'intervalle I:

$$\boxed{a_u \mathcal{D}^{t_i}_{t_j}(I) a_v \ \wedge \ a_v :< t_j >_t \Longrightarrow (\forall J \mid a_u \mathcal{D}^{t_i}_{t_j}(J) a_v \Longrightarrow J\{s, d, f, e\}[I^-, t])}$$

Exemple 5.2 : Relation $\mathcal{D}$.

Si l'agent a_1 doit exécuter la tâche t_{CBA}, il dépendra de l'agent a_2 concernant l'exécution des deux premières sous-tâches *Dépiler(C)* et *Empiler(B, c3, c2)*. Donc, nous pouvons déduire les dépendances suivantes:

$a_1 \mathcal{D}^{t_{CBA}}_{Dépiler(C)} a_2$ et $a_1 \mathcal{D}^{t_{CBA}}_{Empiler(B,c3,c2)} a_2$.

Par contre, si l'agent a_1 devait exécuter la tâche t_{ABC}, il serait complètement dépendant en terme de besoin de l'agent a_2 concernant l'exécution de toutes les sous-tâches. A noter, que l'agent a_2 est capable d'exécuter la tâche t_{ABC} sans aucune aide de l'agent a_1, on dira que l'agent est *autonome* vis-à-vis de cette tâche (cf. chapitre 2).

5.3.3 Relation d'aide

Un agent a_u utilisera cette dépendance dans l'objectif d'aider un autre agent a_v à anticiper et interrompre ou empêcher l'exécution d'une tâche, car celle-ci est en opposition avec une tâche en cours d'exécution par l'agent a_u.

Définition statique

Cette dépendance est représentée par la relation $\mathcal{H}$ définie comme suit:

$$\boxed{a_u \mathcal{H}^{t_i}_{t_j} a_v \Longleftrightarrow t_i \leftrightarrow t_j \ \wedge \ t_i \in r(a_u) - r(a_v) \ \wedge \ t_j \in r(a_v) - r(a_u)}$$

D'après cette définition, il est facile de montrer que la relation $\mathcal{H}$ est *commutative*.

Définition dynamique

Tout comme pour la relation $\mathcal{D}$, l'utilisation de la relation de dépendance de type $\mathcal{H}$, dépend du cours d'exécution des tâches. En effet, il est inutile qu'un agent s'intéresse à un instant donné à cette dépendance, si au moins une des deux tâches n'est pas en cours d'exécution. Pour cette raison, nous utilisons également la notion de dépendance active sur un intervalle I, représentée par $a_u \mathcal{H}^{t_i}_{t_j}(I) a_v$. L'activation d'une dépendance est définie par la formule suivante :

$$\boxed{a_u \mathcal{H}^{t_i}_{t_j}(I) a_v \Longleftrightarrow a_u \mathcal{H}^{t_i}_{t_j} a_v \ \wedge \ (a_u : [t_i]_J \ \vee \ a_v : [t_j]_J) \ \wedge \ I\{s, d, f, e\}J}$$

Continuité dans le temps

Tout comme pour la relation $\mathcal{D}$, nous définissons pour la relation $\mathcal{H}$, la continuité dans le temps :

$$\boxed{a_u \mathcal{H}^{t_i}_{t_j}(I) a_v \ \wedge \ (a_u : [t_i]_J \ \vee \ a_v : [t_j]_J) \ \wedge \ I\{s, d, f, e\}J \Longrightarrow a_u \mathcal{H}^{t_i}_{t_j}(J) a_v}$$

Cette dépendance cesse d'être active à partir du moment où l'une des deux tâche t_i ou t_j a fini d'être exécutée :

$$\boxed{a_u \mathcal{H}^{t_i}_{t_j}(I) a_v \wedge (a_u :< t_i >_t \vee a_v :< t_j >_t) \Longrightarrow (\forall J | a_u \mathcal{H}^{t_i}_{t_j}(J) a_v \Longrightarrow J\{b, o, m, s, d, f, e\}[I^-, t])}$$

Contrairement à la relation $\mathcal{D}$, nous faisons intervenir l'exécution de la tâche t_i par l'agent a_u, ce qui n'était pas nécessaire avec une dépendance de type $\mathcal{D}$ car la terminaison de la tâche t_i était conditionnée par la terminaison de la sous-tâche t_j et donc la tâche t_j devait forcément se terminer avant. Dans une dépendance exprimée par la relation $\mathcal{H}$, les deux tâches peuvent a priori être indépendantes. De plus, aucune contrainte temporelle ne peut être exprimée sur leurs exécutions, ni sur leurs terminaisons.

Exemple 5.3 : Relation $\mathcal{H}$.

Dans notre exemple, l'agent a_1 aura dans son réseau de dépendances la relation suivante : $a_1 \mathcal{H}^{Empiler(A,c1,c2)}_{Empiler(B,c3,c2)} a_2$, car les deux tâches en question font intervenir la même colonne $c2$ et par conséquent, on a $Empiler(A, c1, c2) \leftrightarrow Empiler(B, c3, c2)$.

L'exécution de l'une de ces deux tâches, rend la relation active pendant toute la durée de cette exécution.

5.4 Relation de pouvoir

Dans une situation de concurrence, deux agents capables d'exécuter une même tâche vont se servir d'une *relation de pouvoir* leur permettant de choisir celui qui va exécuter la tâche. Cette notion de pouvoir est introduite dans le but général de répartir la charge de travail entre les agents. A noter,

que notre définition du pouvoir diffère de celle qu'on trouve dans [Cas90] où le fait qu'un agent a_u dépende d'un autre agent a_v implique que l'agent a_v a du pouvoir sur lui.

5.4.1 Définition statique

Nous définissons la relation de *pouvoir* entre deux agents par la relation $>_{t_i}$ comme suit : $a_u >_{t_i} a_v$ implique que si la tâche t_i doit être exécutée, c'est l'agent a_v qui doit s'en charger.

De la même manière, nous pouvons définir la relation réciproque notée $<_{t_i}$ comme suit :

$$\boxed{a_u <_{t_i} a_v \Longleftrightarrow a_v >_{t_i} a_u}$$

La relation $>_{t_i}$ définit une *relation d'ordre* sur l'ensemble des agents :

- elle est réflexive car si un agent a_u est le seul responsable d'une tâche t_i, dans tous les cas, c'est lui même qui sera chargé de cette tâche et ceci ne contredit pas le résultat $a_u >_{t_i} a_u$;
- elle est antisymétrique. En effet, d'après la définition de la relation $>$, si l'on a à la fois $a_u >_{t_i} a_v$ et $a_u >_{t_i} a_v$, le premier résultat implique que l'agent a_v sera chargé de l'exécution de t_i et le deuxième implique que c'est l'agent a_u qui en sera chargé. Or ceci n'est possible que si a_u et a_v représentent tous les deux le même agent ;
- elle est transitive, car si l'on a les deux résultats suivants : $a_u >_{t_i} a_v$ et $a_v >_{t_i} a_w$ d'après le premier, c'est l'agent a_v qui sera chargé de la tâche t_i, mais en utilisant le deuxième résultat, c'est finalement l'agent a_w qui sera chargé de cette tâche. D'où on obtient le même résultat que si on avait $a_u >_{t_i} a_w$.

Cet ordre est partiel car on ne peut comparer (par la relation $>_{t_i}$) que des agents admettant la tâche t_i dans leurs domaines de responsabilité individuelle. L'ordre défini par cette relation est important car il permet déterminer l'agent qui a le moins de pouvoir vis-à-vis d'une tâche (section 5.5).

5.4.2 Définition dynamique

La relation de pouvoir permet de désigner l'agent qui sera chargé d'une tâche, mais à un moment donné les agents a_u et a_v responsables de la même tâche t_i n'ont aucun moyen leur permettant de savoir quelle est la relation valide parmi $a_u >_{t_i} a_v$ et $a_v >_{t_i} a_u$. Pour ce faire, ils vont utiliser un critère spécifique dépendant en général du contexte de travail dans lequel ces agents sont plongés.

Une relation de pouvoir sous la forme $a_u >_{t_i} a_v$ ne dure pas éternellement. En effet, cette relation dépend de deux paramètres: d'une part, de la durée d'exécution de la tâche t_i et d'autre part, du critère choisi, comme la charge de travail qui varie au fur et à mesure de l'exécution des tâches. Ces deux paramètres rendent la relation de pouvoir *dynamique*. Par la suite, cette relation sera considérée comme une fonction du temps valide sur un intervalle donné. Nous pouvons ainsi écrire $a_u >_{t_i} (I)a_v$ et ceci afin de tenir compte de cet opérateur dans un raisonnement temporel.

5.5 Concurrence et pouvoir

La présence d'une relation de concurrence dans les réseaux de dépendances de deux agents a_u et a_v, signifie que quand il va falloir exécuter la tâche t_i, les deux agents doivent se mettre d'accord sur celui qui sera chargé de cette exécution. C'est dans ce contexte, que la relation de pouvoir décrite dans la section précédente va servir comme critère. En effet, si $a_u >_{t_i} a_v$, c'est l'agent a_v qui exécutera la tâche et inversement. Les définitions des relations $\mathcal{C}$ et $>$, nous permettent d'établir le résultat suivant: l'agent qui a le moins de pouvoir doit se charger de l'exécution d'une tâche dans un contexte de concurrence. Ceci peut s'exprimer par la formule suivante:

$$\boxed{\forall a_v \mid a_u \mathcal{C}_{t_i}(I)a_v \ \wedge \ J\{si, di, fi, o, m\}I \ \wedge \ a_v >_{t_i} (J)a_u \Longrightarrow a_u : [t_i]_I}$$

A l'inverse, si l'agent a_v est en train d'exécuter la tâche t_i sur un intervalle K et que la relation $a_u \mathcal{C}_{t_i} a_v$ est active sur un intervalle I contenu dans K, nous pouvons en déduire que l'agent a_u a du pouvoir sur l'agent a_v sur un intervalle J contenant la borne inférieure de I:

$$\boxed{a_v : [t_i]_K \wedge a_u \mathcal{C}_{t_i}(I)a_v \wedge I\{e, s, f, d\}K \Longrightarrow J\{si, di, fi, o, m\}I \wedge a_u >_{t_i} (J)a_v}$$

Exemple 5.4 : Relations $\mathcal{C}$ et $>$.

Dans cet exemple, les agents a_1 et a_2 se partagent toujours le bloc B et ils doivent exécuter la tâche t_{ABC} (cf. section 5.2.2). Il est plus raisonnable que l'agent a_2 exécute les deux tâches *Dépiler(C)* et *Empiler(C, c2, c1)* et que l'agent a_1 exécute la tâche *Empiler(B, c3, c2)*, plutôt que l'agent a_2 car ce dernier est déjà surchargé par l'exécution de deux tâches. Dans un tel contexte, il est possible de prendre comme critère, la charge de travail de chacun des agents à l'instant t où une tâche commune doit être exécutée. Ainsi, dans l'exemple, nous avons le résultat suivant: $a_2 >_{Empiler(B,c3,c1)} ([t, t'])a_1$. Une règle doit permettre de calculer l'intervalle $[t, t']$. Par exemple, nous pouvons établir que t' correspond à la date à laquelle la tâche *Empiler(B, c3, c2)* a fini d'être exécutée.

5.6 Conjonction des relations de dépendance

Lors de la résolution d'un problème un agent va consulter son réseau de dépendances et utiliser ses relations de dépendance avec les autres, d'une part pour résoudre des problèmes qu'il est capable de résoudre partiellement et d'autre part, pour proposer de l'aide aux autres ou leur éviter des efforts inutiles dans certains cas (utilisation de la relation $\mathcal{H}$). Cependant, les agents ne doivent pas se contenter d'utiliser leurs réseaux de dépendances indépendamment les uns des autres. En effet, l'existence de plusieurs relations de dépendance entre un agent et les autres (conjonction de relations), permet à cet agent d'inférer d'autres relations de dépendance entre les autres. Ce type d'inférence, augmente la faculté de raisonnement social chez les agents.

Dans les sections suivantes, nous allons étudier toutes les conjonctions possibles entre trois agents. Nous les avons regroupées dans trois catégories :

- *Conjonction* $\mathcal{CC}$. Correspond à la conjonction de deux relations de concurrence ;
- *Conjonction* $\mathcal{CX}$. Correspond à la conjonction d'une relation de concurrence et d'une relation d'aide ou de besoin ;
- *Conjonction* $\mathcal{XX}$. Correspond à la conjonction de deux relations de besoin ou d'aide.

Nous avons choisi de traiter la conjonction $\mathcal{CC}$ séparément, car elle présente des propriétés différentes. La conjonction d'une relation de besoin et d'une relation d'aide ne permet pas de déduire une troisième relation, elle ne sera donc pas étudiée.

Par ailleurs, le lecteur trouvera dans l'annexe A, la démonstration des formules utilisées dans les sections suivantes.

5.6.1 Conjonction $\mathcal{CC}$

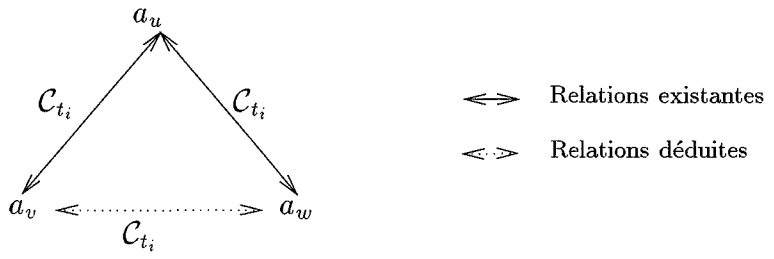


Figure 5.3 : Un agent ayant la même relation de concurrence avec deux agents, en déduit que les deux autres sont liés par la même relation.

Un agent a_u ayant une relation de concurrence sur une tâche t_i avec deux agent a_v et a_w, peut en déduire une relation de concurrence entre les deux agents (figure 5.3) :

$$\boxed{a_u\mathcal{C}_{t_i}a_v \ \wedge \ a_u\mathcal{C}_{t_i}a_w \Longrightarrow a_v\mathcal{C}_{t_i}a_w}$$

L'agent a_u active la relation $a_v\mathcal{C}_{t_i}a_w$, si l'une des deux relations $a_u\mathcal{C}_{t_i}a_v$ et $a_u\mathcal{C}_{t_i}a_w$ est active, mais pas l'autre :

$$\boxed{a_u\mathcal{C}_{t_i}(I)a_v \ \oplus \ a_u\mathcal{C}_{t_i}(I)a_w \Longrightarrow a_v\mathcal{C}_{t_i}(I)a_w}$$

L'agent a_u tient compte de l'activation de ces relations de la manière suivante :

- si les relations $a_u\mathcal{C}_{t_i}a_v$ et $a_v\mathcal{C}_{t_i}a_w$ sont actives, la tâche t_i est cours d'exécution par l'agent a_v (le raisonnement est le même pour l'agent a_w). Quand la tâche est exécutée, l'agent a_u sait qu'il sera informé ainsi que l'agent a_w du résultat de l'exécution par l'agent a_v ;
- si les deux relations $a_u\mathcal{C}_{t_i}a_v$ et $a_u\mathcal{C}_{t_i}a_w$ sont actives, la tâche t_i est cours d'exécution par l'agent a_u lui même. Il est tenu d'informer les deux agents du résultat de l'exécution.

5.6.2 Conjonction $\mathcal{CX}$

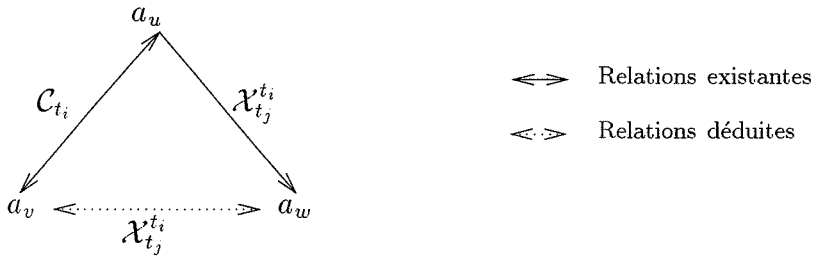


Figure 5.4 : Une relation de concurrence entre deux agents leur permet d'avoir le même type de relation avec d'autres agents.

Un agent a_u ayant une relation de concurrence sur une même tâche t_i avec un agent a_v et une autre relation $\mathcal{X}$=$\mathcal{D}$ ou $\mathcal{H}$ avec un agent a_w, lui permet de déduire que l'agent a_v admet la même relation $\mathcal{X}$ avec l'agent a_w (figure 5.4) :

$$\boxed{a_u\mathcal{C}_{t_i}a_v \ \wedge \ a_u\mathcal{X}_{t_j}^{t_i}a_w \Longrightarrow a_v\mathcal{X}_{t_j}^{t_i}a_w}$$

De plus, si l'agent a_u a du pouvoir sur l'agent a_v vis-à-vis de la tâche t_i et la dépendance $a_u\mathcal{C}_{t_i}a_v$ est active sur un intervalle I, cela signifie que la tâche t_i est en cours d'exécution par l'agent a_v. L'agent a_u peut en déduire que la dépendance $a_v\mathcal{X}^{t_i}_{t_j}a_w$ sera activée sur un sous-intervalle J de I par l'agent a_v :

- pour $\mathcal{X}$=$\mathcal{D}$, l'intervalle J correspond à la durée de l'exécution de la sous-tâche t_j ;
- pour $\mathcal{X}$=$\mathcal{H}$, I=J car la tâche t_i est en cours d'exécution et $t_i \leftrightarrow t_j$.

Ne connaissant pas le sous-intervalle J, l'agent a_3 active cette relation sur tout l'intervalle I :

$$a_u\mathcal{C}_{t_i}(I)a_v \wedge a_u\mathcal{X}^{t_i}_{t_j}a_w \wedge a_u >_{t_i} (J)a_v \wedge J\{o,m,di,e\}I \Longrightarrow a_v\mathcal{X}^{t_i}_{t_j}(I)a_w$$

Nous allons examiner l'utilisation de cette formule lors d'un raisonnement social par l'agent a_u :

- l'activation de la relation $a_v\mathcal{X}^{t_i}_{t_j}a_w$, permet à l'agent a_u de savoir qu'il ne doit pas activer la relation $a_u\mathcal{X}^{t_i}_{t_j}a_v$ si besoin est ;
- l'activation la relation $a_u\mathcal{C}_{t_i}a_v$ permet à l'agent a_u de savoir que l'agent a_v va l'informer du résultat de l'exécution de la tâche t_i.

Exemple 5.5 : Conjonction $\mathcal{CH}$.

Nous nous plaçons dans le contexte suivant, l'agent a_1 est responsable du bloc C, l'agent a_2 est responsable des blocs A et C et enfin l'agent a_3 est responsable des blocs A et B.

Dans ce contexte, l'agent a_1 peut déduire les relations suivantes :

$$a_1\mathcal{C}_{Dépiler(C)}a_2 \text{ et } a_1\mathcal{H}^{Dépiler(C)}_{Empiler(B,c3,c2)}a_3.$$

En réécrivant la première formule, nous obtenons :

$$a_1\mathcal{C}_{Dépiler(C)}a_2 \wedge a_1\mathcal{H}^{Dépiler(C)}_{Empiler(B,c3,c2)}a_3 \Longrightarrow a_2\mathcal{H}^{Dépiler(C)}_{Empiler(B,c3,c2)}a_3.$$

Supposons à présent, que la tâche *Dépiler(C)* est en cours d'exécution par l'agent a_2 sur un intervalle de temps I ($a_1 >_{Dépiler(C)} (I)a_2$). Ceci aura pour effet d'activer la relation $a_1\mathcal{C}_{Dépiler(C)}a_2$. D'après la deuxième formule, la relation déduite $a_2\mathcal{H}^{Dépiler(C)}_{Empiler(B,c3,c2)}a_3$ devient active également. Cette activation permet à l'agent a_1 de savoir qu'il ne doit pas activer la relation $a_1\mathcal{H}^{Dépiler(C)}_{Empiler(B,c3,c2)}a_3$ et donc qu'il ne doit pas informer l'agent a_3 que la tâche *Dépiler(C)* est en cours d'exécution, car ce dernier est déjà mis au courant par l'agent a_2.

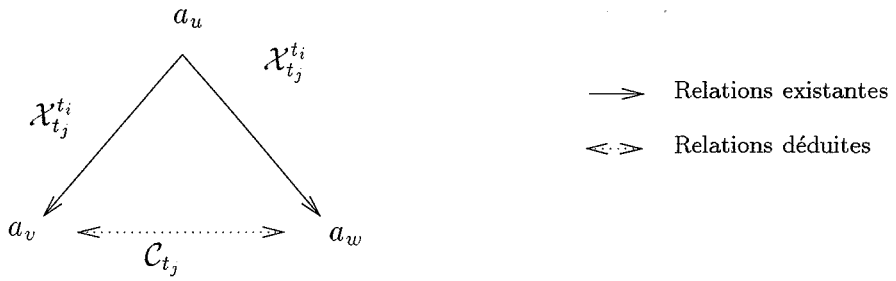


Figure 5.5 : Avoir la même relation de dépendance avec deux agents (aide ou besoin), implique une relation de concurrence entre eux.

5.6.3 Conjonction $\mathcal{XX}$

Si un agent admet la même relation de besoin ou d'aide avec deux agents, il peut déduire qu'il existe une relation de concurrence entre eux (figure 5.5) :

$$\boxed{a_u \mathcal{X}_{t_j}^{t_i} a_v \ \wedge \ a_u \mathcal{X}_{t_j}^{t_i} a_w \Longrightarrow a_v \mathcal{C}_{t_j} a_w}$$

De plus, si une des deux dépendances devient active, l'agent a_u peut inférer que la relation déduite le devient également :

$$\boxed{a_u \mathcal{X}_{t_j}^{t_i} a_v \ \wedge \ a_u \mathcal{X}_{t_j}^{t_i}(I) a_w \Longrightarrow a_v \mathcal{C}_{t_j}(I) a_w}$$

Dans un raisonnement social, l'agent a_u tient compte des ces relations actives comme suit :

- l'activation de la relation $a_v \mathcal{C}_{t_j} a_w$ permet à l'agent a_u de savoir qu'il ne doit pas activer aucune relation faisant intervenir la tâche t_j car cette dernière est déjà en cours d'exécution ;
- l'activation de la relation $a_u \mathcal{X}_{t_j}^{t_i}(I) a_w$ lui permet de savoir qu'il sera informé du résultat de l'exécution de la tâche t_j par l'agent a_w.

Exemple 5.6 : Conjonction $\mathcal{HH}$.
Nous nous plaçons dans le contexte suivant, l'agent a_1 est responsable des blocs A et B, l'agent a_2 des blocs B et C et enfin, l'agent a_3 est responsable des blocs C et A. On suppose par ailleurs que l'agent a_1 doit exécuter la tâche T_{CBA} et que l'agent a_3 est en train d'exécuter la tâche *Dépiler(C)*.
Dans ce contexte, l'agent a_1 admet dans son réseau $\mathcal{H}$ les relations :

$$a_1 \mathcal{H}_{Dépiler(C)}^{Empiler(B,c3,c2)} a_3 \text{ et } a_1 \mathcal{H}_{Dépiler(C)}^{Empiler(B,c3,c2)} a_2.$$

En utilisant la première formule, nous pouvons écrire :

$$a_1 \mathcal{H}_{Dépiler(C)}^{Empiler(B,c3,c2)} a_3 \ \wedge \ a_1 \mathcal{H}_{Dépiler(C)}^{Empiler(B,c3,c2)} a_2 \Longrightarrow a_2 \mathcal{C}_{Dépiler(C)} a_3$$

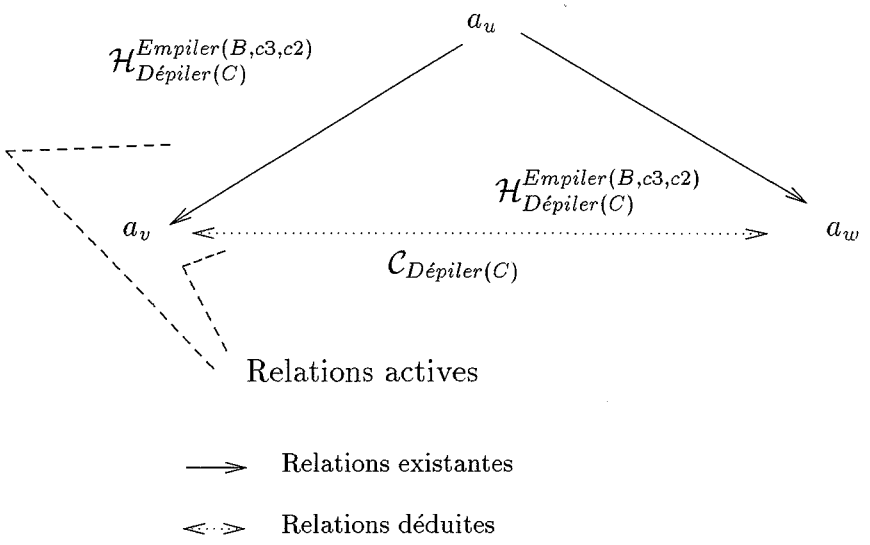


Figure 5.6 : Avoir la même relation d'aide avec deux agents, implique une relation de concurrence entre eux.

Le fait que l'agent a_3 soit en train d'exécuter la tâche *Dépiler(C)*, implique que la relation $a_1\mathcal{H}^{Empiler(B,c3,c2)}_{Dépiler(C)}a_3$ devient active et de ce fait, la relation $a_2\mathcal{C}_{Dépiler(C)}a_3$ le devient aussi d'après la troisième formule (figure 5.6) :

$$a_1\mathcal{H}^{Empiler(B,c3,c2)}_{Dépiler(C)}(I)a_3 \ \wedge \ a_1\mathcal{H}^{Empiler(B,c3,c2)}_{Dépiler(C)}a_2 \Longrightarrow a_2\mathcal{C}_{Dépiler(C)}(I)a_3$$

Comme l'agent a_1 doit exécuter la tâche T_{CBA}, il doit donc activer une des deux relations $a_1\mathcal{D}^{T_{CBA}}_{Dépiler(C)}a_2$ et $a_1\mathcal{D}^{T_{CBA}}_{Dépiler(C)}a_2$. Comme la relation $a_2\mathcal{C}_{Dépiler(C)}a_3$ est active, l'agent a_1 sait que la tâche *Dépiler(C)* est en cours d'exécution et qu'il n'a pas besoin d'activer aucune des deux relations ci-dessus. Il sait également qu'il sera informé du résultat de l'exécution de tâche, car la relation $a_1\mathcal{H}^{Empiler(B,c3,c2)}_{Dépiler(C)}a_3$ est active.

5.7 Organisation temporelle

Les définitions des différentes relations décrites dans les sections précédentes ont permis de dégager un certain nombre de correspondances entre ces relations. Dans les deux tables suivantes, nous précisons leurs types, les liens entre elles, ainsi que les relations existant entre les tâches concernées dans le cas où la relation fait intervenir plus d'une tâche. Nous rappelons que la fonction r définit le domaine de responsabilité d'un agent, s définit l'ensemble des sous-tâches d'un tâche et a est la fonction réciproque (section 5.2).

Relation	$a_u \mathcal{C}_{t_j} a_v$	$a_u \mathcal{C}_{t_i}(I) a_v$	$a_u \mathcal{D}^{t_i}_{t_j} a_v$	$a_u\ \mathcal{D}^{t_i}_{t_j}(I)\ a_v$
Type	Statique	Dynamique	Statique	Dynamique
Déduite de	r	$\mathcal{C}$, $a_u : [t_i]_I$ et $a_v : [t_i]_I$	r, a et s	$\mathcal{D}$, $a_u : [t_i]_I$ et $a_v : [t_j]_I$
Relation entre tâches	$\times \times \times$	$\times \times \times$	Tâche/Sous-tâche	Tâche/Sous-tâche

Relation	$a_u\ \mathcal{H}^{t_i}_{t_j}\ a_v$	$a_u\ \mathcal{H}^{t_i}_{t_j}(I)\ a_v$	$a_u\ >_{t_i}(I)\ a_v$
Type	Statique	Dynamique	Dynamique
Déduite de	$\leftrightarrow$	$\mathcal{H}$, $a_u : [t_i]_I$ et $a_v : [t_j]_I$	Activité des agents
Relation entre tâches	Opposition	Opposition	$\times \times \times$

5.7.1 Niveaux individuel et social au sein de l'agent

Les différents liens entre relations conduisent à la définition des interactions entre les niveaux individuel et social d'un agent (figure 5.7). Au *niveau*

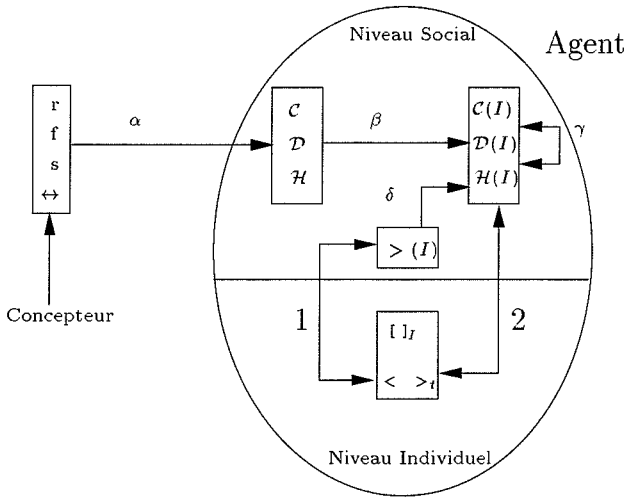


Figure 5.7 : Interactions entre les niveaux individuel et social au sein d'un agent.

individuel, on retrouve les différentes activités de l'agent en terme d'exécution de tâches ($: [\,]_I$ et $:<>_t$). Au *niveau social* et on retrouve la relation de pouvoir, les relations de dépendance statiques (concurrence, besoin et aide), ainsi que leurs instantiations (dépendances actives) :

- (α). Les dépendances statiques sont construites à partir des domaines

de responsabilité et les relations entre tâches, qui sont des paramètres fixés par le concepteur ;

- (β, γ). Les dépendances actives sont déduites des différentes dépendances statiques ainsi que d'autres dépendances actives (cf. section 5.6) ;
- (δ). La relation de pouvoir joue un rôle dans l'activation des dépendances (section 5.6.2) ;
- (1). L'utilisation de la relation de pouvoir dépend de l'activité des agents à un instant donné ;
- (2). L'activité des agents joue également un rôle dans l'activation de leurs dépendances. En effet, une dépendance ne deviendra active que si la ou les tâches concernées doivent être exécutées ou sont déjà en cours d'exécution.

Ainsi, l'activité des agents est un facteur déterminant de leurs relations de pouvoir et de dépendances actives. Ceci représente l'impact du niveau individuel chez un agent sur son niveau social, en d'autres termes, ceci montre comment l'activité individuelle d'un agent peut déterminer son rôle social en terme de dépendance et de pouvoir. Inversement, l'activation d'une dépendance au sein d'un agent se traduit par l'exécution des tâches par l'un et/ou l'autre des agents. La relation de pouvoir détermine également l'activité des agents à un moment donné: en cas de concurrence sur une tâche, elle détermine l'agent qui sera chargé de cette tâche.

5.7.2 Définition de l'organisation

La définition des différentes relations de pouvoir et de dépendance est un moyen permettant de définir la notion d'*organisation*. L'organisation sous sa forme statique sera représentée par l'ensemble des graphes des relations $>$, $\mathcal{C}$, $\mathcal{D}$ et $\mathcal{H}$:

$$\boxed{\mathcal{O} = \{>, \mathcal{C}, \mathcal{D}, \mathcal{H}\}}$$

Naturellement, cette organisation n'existe nulle part dans le système, car aucun des agents ne connaît parfaitement toutes les relations de dépendance entre les uns et les autres. Elle représente uniquement une image de l'ensemble des agents dans un contexte d'étude où elle est observée de l'extérieur. De cette manière, le concepteur pourra étudier l'impact de son choix du paramètre $>$ et du paramètre r qui définit les domaines de responsabilité (a et s étant fixes et dépendants de la composition des tâches en sous-tâches) sur l'organisation (figure 5.8).

Il est évident que l'activité des agents est dynamique et ceci affecte leur organisation à travers les activations et les désactivations des différentes

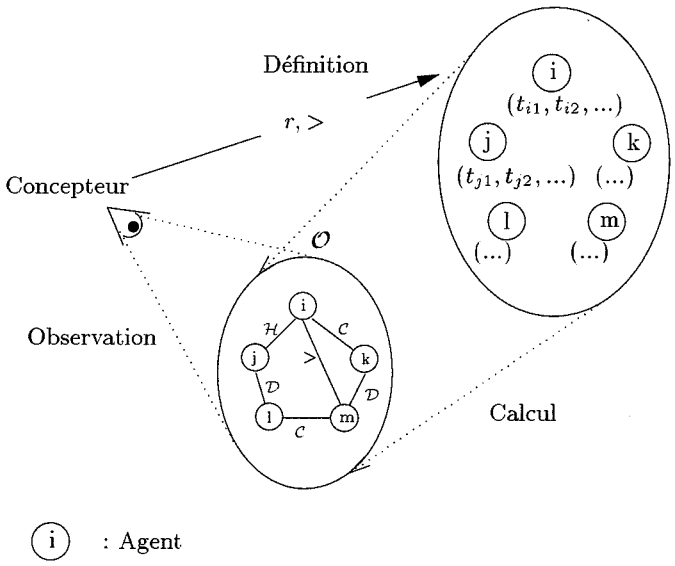


Figure 5.8 : Pour différents choix de $r$et $>$, le concepteur peut obtenir des organisations différentes.

dépendances au cours du temps. Il convient donc de définir une composante dynamique de l'organisation qui est définie par l'organisation sous sa forme statique mais également par les graphes des différentes relations de pouvoir et de dépendances actives sur un intervalle donné I :

$$\boxed{\mathcal{O}(I) = \{\mathcal{O},\ >(I),\ \mathcal{C}(I),\ \mathcal{D}(I),\ \mathcal{H}(I)\}}$$

Encore une fois, cette représentation dynamique de l'organisation sera utilisée dans le but de contrôler son évolution qui dépend essentiellement de l'activité individuelle de chacun des agents. Nous verrons dans le chapitre 6, que les interactions entre agents ont un impact direct sur leurs activités individuelles. Ainsi, l'organisation $\mathcal{O}(I)$ peut être considérée comme un résultat indirect des interactions qui ont lieu au cours du temps (figure 5.9).

5.8 Discussion

Dans ce chapitre, nous avons présenté le modèle d'une société d'agents dédiée à la résolution de problèmes complexes où les tâches qui doivent être exécutées peuvent être décomposées en plusieurs sous-tâches. Cette décomposition conduit à un ensemble de contraintes temporelles qui, même exprimées sous leur forme la plus simple comme étant des contraintes de précédences, régissent souvent l'exécution des tâches. Cette dynamique des tâches doit être prise en compte par l'ensemble des agents et ceci en fonction

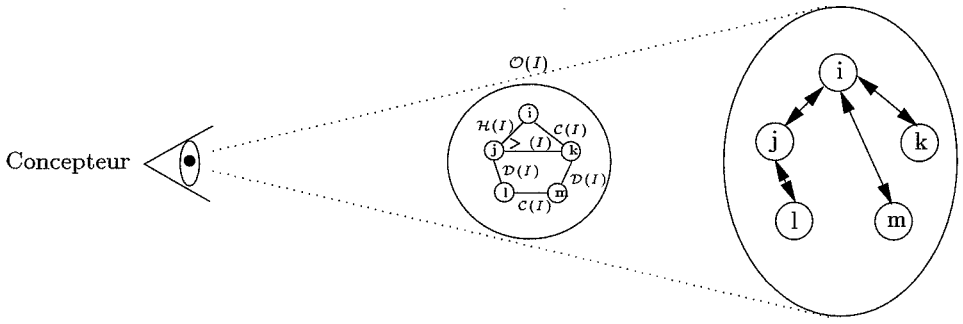


Figure 5.9 : L'organisation $\mathcal{O}(I)$ est définie par les instances actives des différentes relations.

de leurs domaines de responsabilité. Elle est prise en compte à deux niveaux au sein de l'agent :

- au niveau individuel, il s'agit de prendre en compte essentiellement les durées d'exécution des tâches, les différentes contraintes temporelles entre une tâche et ses sous-tâches ou entre deux tâches opposées ;
- au niveau social, il s'agit de prendre en compte les contraintes posées par l'exécution de plusieurs sous-tâches par plusieurs agents, afin d'effectuer une tâche de haut niveau.

La définition des différents domaines de responsabilité permettent aux agents de déduire des relations de dépendance avec les autres. Ces relations sont représentées par un réseau de dépendances local à chaque agent. En fonction de leurs activités individuelles en terme d'exécution de tâches et en fonction de leurs relations de pouvoir, les dépendances deviennent actives et permettent ainsi aux agents d'avoir un raisonnement temporel sur leurs relations de dépendance, ainsi que sur celles des autres. Les relations de pouvoir sont déterminées en utilisant un critère connu par l'ensemble des agents, il dépend généralement de leur activité individuelle. Les différentes relations de dépendance et de pouvoir permettent de définir l'organisation à la fois dans sa composante statique et dans sa composante dynamique. L'organisation constitue essentiellement un moyen permettant d'observer l'évolution des interactions entre agents, mais elle permet également de choisir certains paramètres tels que la définition des domaines de responsabilité. Ainsi, notre modèle de société d'agents est basé essentiellement sur la notion de réseau de dépendances, ainsi que sur l'étude de la dynamique des différentes dépendances.

Chapitre 6

Modèle d'interaction temporelle

6.1 Introduction

Dans le chapitre précédent, nous avons décrit le modèle d'organisation d'une société d'agents basée sur la notion de réseau de dépendances temporelles. Ces réseaux permettent aux agents de connaître leurs relations de dépendance avec les autres afin d'interagir avec eux lors de la résolution de problèmes complexes.

Le présent chapitre, a pour objet de décrire les moyens de communication mis à disposition des agents, afin d'exprimer leurs besoins d'une manière claire et compréhensible par l'ensemble des agents.

Nous commençons par la description du *langage d'interaction* $\mathcal{L}$ connu par l'ensemble des agents car ceux-ci communiquent par *envoi de messages* formant une expression respectant la *grammaire de ce langage* $\mathcal{L}$. Ce langage permet l'expression de contraintes temporelles au niveau des messages.

Ensuite et afin que les échanges de messages soient interprétés uniformément par l'ensemble des agents, nous définissons une sémantique associée au langage. Cette sémantique est basée sur les actes de langage.

Enfin et dans le but de contrôler les échanges de messages, différents protocoles d'interaction sont définis. En effet, dans une conversation un agent ayant reçu un message doit savoir comment répondre à ce message, sinon des réponses spontanées de part et d'autre, risquent d'entraîner la conversation dans une impasse ou dans un état de dégénérescence. Nous définissons d'abord trois *protocoles élémentaires*: le *protocole de requête*, le *protocole d'information* et enfin le *protocole d'interruption de tâches.* A partir de ces protocoles, peuvent être définis des *protocoles plus complexes* tels que le *protocole de négociation.*

6.2 Langage d'interaction $\mathcal{L}$

La communication par envoi de message exige que les messages soient interprétés de la même façon par l'ensemble des agents, sans cela, les conversations peuvent présenter des incohérences entraînant des anomalies dans le fonctionnement de la société. Pour éviter ce problème, nous avons défini un langage d'interaction $\mathcal{L}$ commun à tous les agents. Un message envoyé ou reçu doit être une expression respectant la grammaire du langage $\mathcal{L}$.

6.2.1 Grammaire du langage

La grammaire du langage $\mathcal{L}$ est définie comme suit, sous forme *BNF* :

<Msg> := '('<Communication>')' <Content> '('<*Context*>')'
<Communication> := <*date*> <*sender*> <*receiver*>
<Content> := '('**REQUEST**')' '('<something> <*timeout*>')'
| '('**RESPOND**')"('<answer> <*date*>')'
| '('**INFORM**')"('<fact> <*date*>')'
| '('**REFRAIN** <*Context*>')'
<something> := **INFORMATION-ABOUT** <*info*>
| **PERFORMED** <*task*>
<answer> := **PERFORMED** <*result*>
| **FAILED**
| <*others*>
<fact> := <answer> <*task*>
| <*task*> '↔' <*task*>

Les champs écrits en italique correspondent aux symboles terminaux. Dans un message, ils correspondent à des identificateurs (*sender, receiver, context, task, result*), des dates ou des délais (*date, timeout*). Les champs écrits en lettres capitales correspondent aux mots clés du langages. Les autres champs sont des symboles non terminaux de la grammaire.

Le champ <Msg> représente le corps du message. Un message est constitué de trois champs : le champ <Communication> contenant la date d'émission du message, l'identificateur de l'expéditeur et celui du destinataire. Le champ <Context> indique le contexte de conversation. Ce contexte sera utilisé dans tous les messages concernant une même conversation entre plusieurs agents. Le champ <Content> contient l'information pertinente qui doit être traitée par l'agent destinataire.

Dans un message de requête (mot clé REQUEST), le champ <*task*> précise la tâche à exécuter et le champ <*timeout*> précise le délai associé à cette tâche sous forme d'un intervalle. La borne inférieure de cet intervalle correspond à la date à partir de laquelle la tâche peut être exécutée. Sa borne supérieure correspond au nombre maximum d'unités de temps que doit prendre l'exécution de cette tâche.

Dans un message de réponse à une requête (mot clé RESPOND), le champ <answer> indique si l'exécution d'une tâche s'est effectuée avec succès (mot clé PERFORMED) et dans ce cas un champ <*result*> permet de donner le résultat de l'exécution. Le contenu de ce champ dépend du type de la tâche exécutée et les résultats qu'elle fournit. Dans le cas d'un échec, le mot clé FAILED est utilisé. Dans tous les cas, la date (champ <*date*>) à laquelle le succès ou l'échec de la tâche à exécuter est précisée. Le champ <*others*> est utilisé quand le message représente une réponse à une requête d'information (mot clé INFORMATION-ABOUT).

Quand le message est utilisé pour informer sur un fait (mot clé INFORM), ce fait (champ <*fact*>) peut indiquer un fait ou alors il peut avoir le même contenu qu'une réponse (champ <answer>) et dans ce cas, la tâche est précisée car dans le cas d'une réponse le contexte de conversation permet de retrouver de quelle tâche il s'agit.

6.2.2 Informations temporelles dans un message

Un message envoyé ou reçu peut contenir trois types d'informations temporelles :

- la date d'émission. Cette date est insérée par l'expéditeur avant l'émission du message. Elle permet à l'agent destinataire de traiter les messages dans leur ordre d'émission qui peut différer de l'ordre de réception ;

- une date est incluse dans le contenu propositionnel :

  - dans le cas d'une requête, un délai précise la date limite à laquelle l'agent expéditeur doit recevoir une réponse à cette requête, cette date va servir à la gestion des buts de l'agent destinataire ;
  - dans le cas d'une réponse à une requête, l'agent expéditeur précise la date à laquelle la tâche répondant à la requête a été exécutée.

    Dans le cas d'une information à propos d'un fait comme la nouvelle position d'un bloc par exemple, cette date est celle à laquelle l'information a été établie.

Exemple 6.1 : Message de requête.

Dans l'exemple sur le monde des blocs, le champ < *task* > peut être instancié par *Dépiler(C)*, le champ <*Context*> par *ctxt*1, le champ <*date*> par *d*, le champ <*sender*> par *a*1, le champ <*receiver*> par *a*2 et enfin le champ <*timeout*> par [5, 10]. Ces instantiations donnent lieu au message suivant :

$$(d\ a_1\ a_2)(REQUEST)(PERFORMED\ Dépiler(C)\ [5,10])(ctxt1)$$

6.3 Sémantique associée au langage $\mathcal{L}$

Dans cette section, il s'agit d'étudier la sémantique associée à chacun des termes du langage $\mathcal{L}$ en s'appuyant sur la théorie des actes de langage. Nous nous inspirons du travail mené par E. Alphonse dans son travail de DEA où sont définis les quatre actes illocutoires représentés sous la forme $< F, p >$ où F désigne la force illocutoire et p le contenu propositionnel. Les forces illocutoires primitives sont déterminées par le but illocutoire : *assertif*, *directif*, *engageant* ou *déclaratif*. A noter, que le but illocutoire *expressif* n'a pas été abordé, n'ayant pas d'utilité cruciale dans notre contexte. Cette sémantique va permettre de définir les changements que produit une communication dans les états mentaux du locuteur et de l'allocutaire. D'une part, nous allons définir l'acte perlocutoire d'un acte illocutoire pour l'allocutaire et d'autre part, nous allons préciser la vision de cet acte perlocutoire par le locuteur.

6.3.1 Définitions préliminaires

Dans cette section, nous donnons quelques définitions qui seront utilisées par la suite.

L'instant t_{now} définit l'instant présent où une proposition ou une propriété a été établie ou considérée.

Prédicats

- Le prédicat DONE admet deux paramètres, le premier est une action ou une suite d'actions, ce qui correspond dans notre contexte à une tâche et le deuxième est un intervalle de temps. DONE(ϕ, I) est vrai si et seulement si la réalisation de l'action ϕ se termine durant l'intervalle I ;

- Le prédicat CAN est un prédicat de capacité, contrairement à l'opérateur $>$ défini dans le chapitre 5, il ne définit pas la relation de pouvoir entre deux agents vis-à-vis d'une tâche (cf. section 2.3.6), mais le pouvoir d'un agent à exécuter une tâche ou dans un cadre général le pouvoir de rendre une propriété vraie : CAN(a_u, p) est vrai si et seulement si l'agent a_u est capable d'entreprendre une action ou une suite d'actions permettant de rendre p vrai ;

- Le prédicat *failed* précise si une action a échoué. *failed*(a) est vrai si et seulement si la réalisation de l'action a a échouée ;

- Le prédicat *is-true-at-date* admet deux paramètres : une proposition et une date. *is-true-at-date(p, d)* est vrai si et seulement si p est vraie à la date d. De plus, s'il est établi à une date d qu'une proposition est vraie à une date d_0, alors ce résultat peut être établi à n'importe quel

date d' ultérieure à d :
is-true-at-date(is-true-at-date(p, d_0), d) $\Longleftrightarrow \forall$ *d'* $\geq$ *d, is-true-at-date(is-true-at-date(p, d_0), d').*

Opérateurs

- L'opérateur BEL est un opérateur de croyance, il admet intuitivement deux paramètres, l'identificateur de l'agent qui croît et la proposition sur laquelle porte la croyance. Par exemple, BEL(a_u, p) signifie que l'agent a_u croît que p est vraie ;

- L'opérateur INT est un opérateur d'intention, il admet le même type de paramètre que l'opérateur BEL. INT(a_u, p), signifie que l'agent a_u a l'intention de rendre p vraie dans le futur. En pratique, tout but de l'agent correspond à une intention ;

- L'opérateur S-COMMIT symbolise l'engagement d'un agent envers un autre agent pour rendre une propriété vraie. Cette notion d'engagement est celle de l'engagement social évoquée dans la section 2.2.2. S-COMMIT(a_u, a_v, p) signifie que l'agent a_u s'engage envers l'agent a_v de rendre p vrai dans le futur.

Fonctions

- La fonction *results-when-performed* rend le résultat de la réalisation d'une action ou une suite d'actions. Ce résultat peut être une proposition ;

- La fonction *getTask* permet de retrouver une tâche, à partir d'un contexte de conversation.

6.3.2 Actes de langage de base

A partir des quatre buts illocutoires, il est possible de définir quatre actes de langage : *informer*, *commander*, *s'engager* et *déclarer*. Dans notre contexte, nous n'aurons besoin que des actes *informer* et *commander*, en voici la définition :

Informer

Le but illocutoire *assertif* permet la définition de l'acte de langage défini par le performatif **informer**. Le fait qu'un agent a_u *informe* un agent a_v du contenu propositionnel p (*informer*(a_u, a_v, p)) correspond à l'acte illocutoire $< assertif,\ p >$ et signifie que l'agent a_u souhaite informer l'agent a_v que le contenu propositionnel p est vrai au moment de l'énonciation de l'acte.

Sémantique La sémantique associée à cet acte, qui constitue également ses conditions de succès, est la suivante :

$$BEL(a_u, p) \wedge BEL(a_u, \neg BEL(a_v, p)) \wedge INT(a_u, BEL(a_v, p)).$$

Acte perlocutoire Nous définissons l'acte perlocutoire de cet acte pour l'agent a_v, comme étant la satisfaction de l'intention du locuteur et donc BEL(a_v, p). De plus, l'agent a_v croît les conditions de satisfaction de l'acte, puisqu'il l'a reconnu et donc nous avons :

$$BEL(a_u,\ BEL(a_u, p) \wedge BEL(a_u, \neg BEL(a_v, p)) \wedge INT(a_u, BEL(a_v, p))).$$

De son coté, le locuteur raisonne sur l'acte perlocutoire et établit le résultat suivant :

$$BEL(a_u,\ BEL(a_v,\ p)).$$

Commander

Le but illocutoire *directif* conduit à la définition de l'acte de langage défini par le performatif **commander.** L'acte illocutoire $< directif,\ p >$ d'une communication entre un agent a_u et un autre agent a_v signifie que l'agent a_u *commande* l'agent a_v de rendre le contenu propositionnel p vrai ($commander(a_u,\ a_v,\ p)$) où $p = DONE(a,\ I)$, a étant une action.

Sémantique La sémantique de cet acte est la suivante :

$$BEL(a_u, CAN(a_v, p)),\ INT(a_u, p),\ INT(a_u, \text{S-COMMIT}(a_v, a_u, p)).$$

Acte perlocutoire L'acte perlocutoire de cet acte est le suivant :

$$S\text{-}COMMIT(a_v,\ a_u,\ p).$$

L'agent a_v croît les conditions de satisfactions de l'acte :

$$BEL(a_v,\ BEL(a_u, CAN(a_v, p)),\ INT(a_u, p),\ INT(a_u,\ \text{S-COMMIT}(a_v, a_u, p))).$$

Après raisonnement sur l'acte perlocutoire, l'agent a_u peut établir le résultat suivant :

$$BEL(a_u,\ S\text{-}COMMIT(a_v,\ a_u,\ p)).$$

6.4 Protocoles d'interaction élémentaires

Le langage d'interaction constitue un moyen commun pour que les agents formulent leurs besoins et s'échanger des messages compréhensibles et interprétés d'une manière cohérente par tous. Néanmoins, le langage d'interaction n'est pas suffisant pour mener des conversations avec succès. Les agents doivent par ailleurs avoir un moyen leur permettant de réagir avec précision face à un message reçu et de cette manière, il n'y a plus de confusion dans l'interprétation des messages. Pour ce faire, nous utilisons un ensemble de protocoles d'interaction comme un moyen de contrôle des interactions et une garantie de leur succès. Ces protocoles sont les briques de base de toute conversation qui peut être composée de plusieurs messages. C'est pour cette raison que ces protocoles sont dits *élémentaires*. Nous en avons identifié trois: le *protocole de requête*, le *protocole d'information*, le *protocole d'interruption de tâches*. A noter, que ces protocoles ne sont pas exhaustifs et qu'il est possible d'avoir d'autres types de protocole et cela en fonction des besoins en terme de communication.

6.4.1 Protocole de requête

Le protocole de requête est utilisé uniquement pour indiquer au destinataire que l'expéditeur d'un message de requête attend une réponse à cette requête (figure 6.1). Il est important de rappeler que le mot clé REQUEST indique uniquement au destinataire qu'il doit suivre le protocole de requête pour réagir suite à la réception du message. La sémantique associée à une requête se trouve au niveau du protocole et non au niveau du langage. En effet, le mot clé REQUEST aurait pu être remplacé par un autre mot clé sans pour autant empêcher l'agent destinataire de traiter le message comme un message de requête.

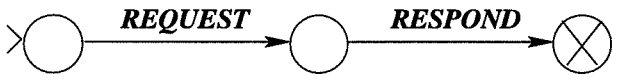


Figure 6.1 : Protocole de requête.

Ce protocole indique à l'allocutaire qu'une requête doit nécessairement avoir une réponse, ainsi, ce protocole peut être défini comme suit :

REQUEST := *commander*; RESPOND	RESPOND := *informer*.

L'opérateur ';' définit la succession de deux actions: $a;b$ correspond à la succession des actions a et b.

Message de requête

Un agent a_v recevant un message de requête d'un agent a_u, l'interprète selon les termes qui constituent le message :

(d a_u a_v)(REQUEST)(INFORMATION-ABOUT p I)(ctxt)
(d, a_u, a_v)(REQUEST)(PERFORMED task I)(ctxt)

Dans le premier cas, le message est interprété comme suit :

commander(a_u,a_v,DONE(informer(a_v,a_u,p) $\vee$ informer(a_v,a_u,$\neg p$), I)).

L'effet perlocutoire de l'acte est :

S-COMMIT(a_v, a_u, DONE(informer(a_v, a_u, p) $\oplus$ informer(a_v, a_u, $\neg p$), I)).

Cet acte signifie que l'agent a_v s'engage à exécuter l'action *informer* dans le futur et informer dans tous les cas la valeur de vérité de p. La définition de cet acte perlocutoire assure la bonne application du protocole de requête car l'exécution de l'action *informer* correspond à la partie RESPOND dans la définition du protocole. Il est important de rappeler que S-COMMIT indique qu'un agent s'engage à exécuter une action dans le futur, mais DONE contrôle le délai autorisé (intervalle I) pour l'exécution de cette action.

Dans le second cas, le message est interprété comme suit :

commander(a_u, a_v, DONE(informer(a_v, a_u, results-when-performed(task)) $\vee$ informer(a_v, a_u, failed(task)), t)).

L'effet perlocutoire est :

S-COMMIT(a_v, a_u, DONE(informer(a_v, a_u, *results-when-performed*(task)) $\oplus$ informer(a_v, a_u, *failed*(task)), t)).

Cet acte signifie que l'agent a_v s'engage à exécuter l'action *informer* et informer des résultats de l'exécution d'une tâche dans le cas où elle se termine avec succès ou indiquer que l'exécution a échoué dans le cas contraire. Dans les deux cas, cet acte n'engage pas l'allocutaire à exécuter la tâche en question.

Message de réponse

A la réception d'une réponse à une requête, le message est interprété en fonction des éléments le constituant :

(d a_v a_u)(RESPOND)(PERFORMED <result> t)(ctxt)
(d a_v a_u)(RESPOND)(FAILED t)(ctxt)

Dans le premier cas, le message sera interprété comme suit :

informer(a_v,a_u,*is-true-at-date(results-when-performed(getTask(ctxt)), t)*)
results-when-performed(getTask(ctxt))=< *result* >.

L'acte perlocutoire qui en résulte est :

BEL(a_u, *is-true-at-date(results-when-performed(getTask(ctxt)), t)*).

Dans le deuxième cas, le message sera interprété comme suit :

informer(a_v, a_u, *is-true-at-date(failed(getTask(ctxt)), t)*).

L'acte perlocutoire correspondant est :

BEL(a_u, *is-true-at-date(failed(getTask(ctxt)), t)*).

Exemple 6.2 : Requête.

Pour illustrer le protocole de requête, nous reprenons l'exemple du monde des blocs. Supposons que l'agent a_1 est responsable des blocs A et B et l'agent a_2 est responsable du bloc C. Afin d'exécuter la tâche T_{CBA}, l'agent a_1 consulte son réseau de dépendances. L'existence de la relation $a_1 \mathcal{D}^{T_{CBA}}_{Dépiler} a_2$, permet à l'agent a_1 d'envoyer le message de requête suivant à l'agent a_2 :

$$(d\ a_1\ a_2)(REQUEST)(PERFORMED\ Dépiler(C)\ [t_{now}, +\infty])(ctxt)$$

Ce message signifie que l'agent a_1 a entamé le protocole de requête et il s'attend à une réponse de l'agent a_2. Il est interprété comme suit par l'agent a_2 :

commander(a_1, a_2, DONE(*informer*(a_2, a_1, *results-when-performed*(*Dépiler*(C)))
∨ *informer*(a_2, a_1, *failed*(*Dépiler*(C))), $[t_{now}, +\infty]$)).

L'acte perlocutoire correspondant est :

S-COMMIT(a_2, a_1, DONE(informer(a_2, a_1, *results-when-performed*(*Dépiler*(C)))
∨ informer(a_2, a_1, *failed*(*Dépiler*(C))), $[t_{now}, +\infty]$)).

L'agent a_2 est tenu après l'exécution de la tâche Dépiler(C) d'exécuter l'action *informer* assurant l'application du protocole de requête, ce qui entraîne l'envoi du message :

$$(d\ a_2\ a_1)(RESPOND)(PERFORMED\ t)(ctxt)$$

Pour l'agent a_1, ce message s'interprète de la manière suivante :

informer(a_2, a_1, *is-true-at-date(results-when-performed(getTask(ctxt)), t)*).

L'acte perlocutoire correspondant est le suivant :

BEL(a_1, *is-true-at-date(results-when-performed(getTask(ctxt)), t)*).

Ceci permet à l'agent a_1 de savoir que la tâche Dépiler(C) a été exécutée avec succès et qu'il n'y a pas de résultats à transmettre (le champ <*result*> est vide), il indique également la fin du protocole de requête entre les deux agents.

6.4.2 Protocole d'information

Le protocole d'information est utilisé dans un contexte où un agent ayant établi des résultats, désire les communiquer à un autre. En utilisant ce protocole, l'envoi d'un message ne nécessite pas une réponse du destinataire (figure 6.2).

Figure 6.2 : Protocole d'information.

Ce protocole indique à l'allocutaire que le message reçu contient une information et qu'il n'est pas tenu de répondre. Ce protocole peut être défini comme suit :

INFORM := *informer*

Un message d'information admet la structure suivantes :

(d a_u a_v)(INFORM)(p t)(ctxt)

A la réception d'un message d'information, l'allocutaire l'interprète de la manière suivante :

informer(a_u, a_v, is-true-at-date(p, t))

L'acte perlocutoire de cet acte est :

BEL(a_v, is-true-at-date(p, t)).

De plus l'agent a_u peut établir le résultat suivant :

BEL(a_u, BEL(a_v, is-true-at-date(p, t))).

Exemple 6.3 : Information.

En gardant les mêmes domaines de responsabilité que dans l'exemple 6.2, supposons que l'agent a_1 a déplacé le bloc B de la colonne $c3$ vers la colonne $c2$ à l'instant t_0. L'agent a_2 n'étant responsable que du bloc C, il lui est impossible de percevoir cet événement. Afin de lui permettre la connaissance des positions des blocs, l'agent a_1 peut vouloir l'informer du déplacement en question en lui envoyant le message suivant :

$$(d\ a_1\ a_2)(INFORM)(PERFORMED\ Dépiler(B)\ t_0)(ctxt)$$

Ce message indique à l'agent a_2 qu'il ne doit pas envoyer une réponse selon le protocole d'information. Il sera interprété comme suit :

$$informer(a_1,\ a_2,\ \text{is-true-at-date}(\textit{results-when-performed}(\text{Dépiler(B)}),\ t_0).$$

L'acte perlocutoire correspondant est :

$$\text{BEL}(a_2,\ \text{is-true-at-date}(\textit{results-when-performed}(\text{Dépiler(B)}),\ d')).$$

6.4.3 Protocole d'interruption de tâches

Le protocole d'interruption de tâches est utilisé dans un contexte où des tâches devant être exécutées par d'autres agents, ne doivent plus être exécutées suite à un changement de contexte. Les agents ayant envoyé les requêtes doivent envoyer des messages demandant l'interruption des tâches en cours d'exécution. L'agent ayant reçu un message d'interruption d'une tâche, ne doit pas envoyer une réponse (figure 6.3) selon le protocole d'interruption de tâches, mais doit réappliquer le protocole avec d'autres agents à qui il a délégué des sous-tâches de la tâche qui lui a été déléguée et qui doit être interrompue. Ainsi, l'application de ce protocole est un processus récursif. Dans la figure 6.3, nous avons représenté uniquement une étape de ce protocole entre deux agents.

Figure 6.3 : Protocole d'interruption de tâches.

La définition du protocole est la suivante :

REFRAIN := *commander*

Un message d'interruption de tâche admet la structure suivante :

$$(\text{d}\ a_u\ a_v)(\text{REFRAIN}\ ctxt)(ctxt1)$$

Un agent a_v recevant un message d'interruption de tâches d'un agent a_u, doit s'assurer de l'interruption de toutes les sous-tâches. Pour ce faire, nous introduisons l'action *stop*, permettant l'interruption de l'exécution d'une autre action. L'exécution de $stop(a_u, \Phi)$ implique que l'agent a_u a arrêté l'exécution de l'action Φ. Si l'action admet une suite d'actions plus élémentaires, ces actions doivent être interrompues également :

$stop(a_u, \Phi) \Longrightarrow \forall \phi, a_v \mid \phi \in s(\Phi) \cap r(a_v)$, $commander(a_u, a_v, \text{DONE}(stop(a_v, \phi), [t_{now}, +\infty]))$.

Ainsi, l'agent a_v peut interpréter le message d'interruption de tâche comme suit :

$commander(a_u, a_v, \text{DONE}(stop(a_v, \text{getTask}(ctxt)), [t_{now}, +\infty]))$.

Ceci oblige l'agent a_v à interrompre la tâche correspondant au contexte $ctxt$ à partir du moment où le message a été reçu (t_{now}).

L'acte perlocutoire de cet acte est le suivant :

$\text{S-COMMIT}(a_v, a_u, \text{DONE}(stop(a_v, \text{getTask}(ctxt)), [t_{now}, +\infty]))$.

De plus, l'agent a_u peut établir le résultat suivant :

$\text{BEL}(a_u, \text{S-COMMIT}(a_v, a_u, \text{DONE}(stop(a_v, \text{getTask}(ctxt)), [t_{now}, +\infty])))$.

Exemple 6.4 : Interruption d'une tâche.

Les trois agents a_1, a_2 et a_3 sont responsables respectivement des blocs A, B et C. Si l'agent a_3 est chargé de la tâche T_{CBA}, il doit envoyer des requêtes aux agents a_2 et a_1 pour exécuter respectivement les sous-tâches *Empiler(B, c3, c2)* et *Empiler(A, c1, c2)*. Supposons que l'agent a_3 a déjà exécuté la sous-tâche *Dépiler(C)* et avant que l'agent a_2 n'exécute la sous-tâche *Empiler(B, c3, c2)*, l'agent a_1 a déplacé le bloc A sur le bloc C en exécutant la tâche *Empiler(A, c1, c2)*. L'agent a_3 ayant perçu cet événement (empilement du bloc A sur le bloc C), en déduit que la tâche T_{CBA} ne peut plus être exécutée telle qu'elle a été planifiée. Alors il envoie un message d'interruption de la sous-tâche *Empiler(B, c3, c2)* à l'agent a_2 :

$$(d\ a_3\ a_2)(REFRAIN\ ctxt)(ctxt1)$$

Ce message sera interprété comme suit par l'agent a_2 :

$commander(a_3, a_2, \text{DONE}(stop(a_2, \text{getTask}(ctxt)), [t_{now}, +\infty]))$.

L'effet perlocutoire correspondant est le suivant :

S-COMMIT(a_2, a_3, DONE($stop(a_2$, getTask($ctxt$)), $[t_{now}, +\infty]$)).

L'agent a_3 après raisonnement sur l'acte perlocutoire peut établir le résultat suivant :

BEL(a_3, S-COMMIT(a_2, a_3, DONE($stop(a_2$, getTask($ctxt$)), $[t_{now}, +\infty]$))).

Ce résultat lui permet de s'assurer que la tâche *Empiler(B, c3, c2)* va être interrompue dans le futur.

Le contexte de conversation *ctxt* est le même utilisé lors de l'envoi de la requête par l'agent a_3. Il permet à l'agent a_2 de savoir qu'il s'agit d'interrompre la sous-tâche *Empiler(B, c3, c2)*. Cette sous-tâche étant une tâche élémentaire, le protocole d'interruption de tâches s'arrête.

6.5 Protocoles d'interaction plus complexes

La définition des trois protocoles élémentaires permet de définir des protocoles plus complexes composés de plusieurs de ces protocoles élémentaires.

6.5.1 Grammaire générique

Pour ce faire, nous introduisons les opérateurs '.' et ';' et définissons une grammaire génératrice de tous les protocoles possibles :

P ⟶ P ; E
| P . E
| E
E ⟶ REQ-P
| INF-P
| REF-P

REQ-P, INF-P et REF-P représentent respectivement le protocole de requête, le protocole d'information et le protocole d'interruption de tâches.

Opérateur ';'

L'opérateur ';' indique un choix entre deux protocoles à suivre : $p_1 ; p_2$ indique qu'il est possible de suivre soit le protocole p_1 soit le protocole p_2.

En désignant par *entry(p)*, l'état initial d'un protocole p et *exit(p)*, l'ensemble de ses états finaux, le protocole P défini par $P = P_1$; P_2 admet comme état initial *entry(P)*, la fusion des états initiaux des protocoles P_1 et P_2 notée par $entry(P_1) \vee entry(P_2)$. Il admet comme état final, la réunion des états finaux des deux protocoles $exit(P) = exit(P_1) \cup exit(P_2)$ (cf. figure 6.4).

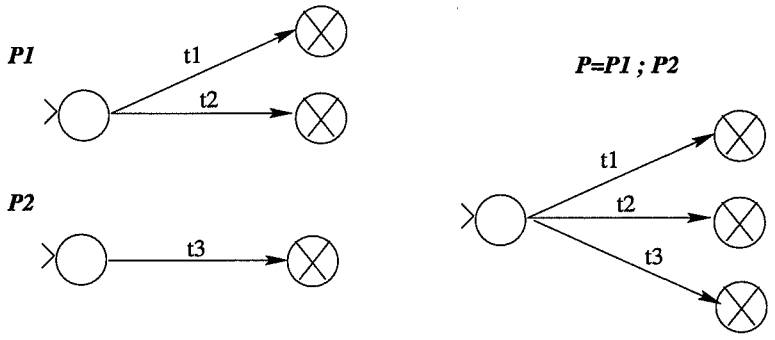


Figure 6.4 : Opérateur de choix: $P = P_1 \ ; \ P_2$.

Opérateur '.'

L'opérateur '.' désigne la concaténation de deux protocoles: ayant un protocole P défini par $P = P_1.P_2$, nous avons *entry(P)=entry(P_1)*, *exit(P)= exit(P_2)* et enfin le ou les états finaux du protocole P_1 constituent l'état initial du protocole P_2 (cf. figure 6.5). Donc, lors du suivi du protocole P le

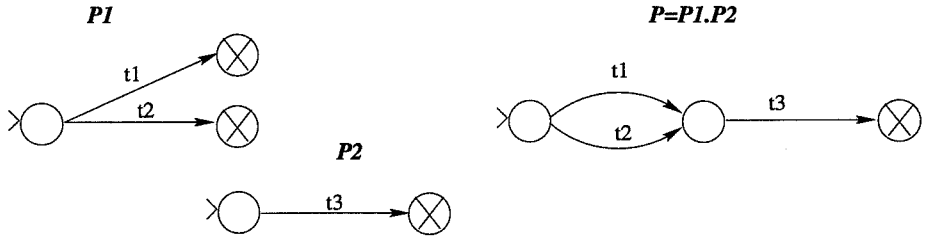


Figure 6.5 : Opérateur de concaténation: $P = P_1.P_2$.

passage par un état final du protocole P_1 est considéré comme un passage par l'état initial du protocole P_2.

Distributivité des opérateurs

La distributivité des opérateurs ; et . est définie comme suit:
$p_1.(p_2; p_3) \Longleftrightarrow p_1.p_2; p_1.p_3$
Cette grammaire permet de définir des protocoles complexes ayant comme atomes les trois protocoles élémentaires. Nous allons en voir un exemple dans ce qui suit.

6.5.2 Protocole de négociation

Le protocole de négociation est essentiellement utilisé pour résoudre des conflits entre plusieurs agents. Les agents peuvent avoir des conflits de res-

sources ou de buts (cf. section 2.2.3). Dans ce cas, ils doivent essayer de se convaincre mutuellement ou alors trouver un compromis. La plupart des protocoles sont basés sur des techniques d'argumentation [PJ96]. Dans notre étude, un conflit provient de deux tâches liées par la relation d'opposition ($\leftrightarrow$) et qui doivent être exécutées par deux agents. Ces agents sont capables d'évaluer un choix d'une tâche en se basant sur un système de pondérations. Nous supposons l'existence d'une fonction *poids* permettant de donner à chaque instant, le poids d'une tâche en cours d'exécution. Cette fonction dépend de la nature des tâches et le contexte dans lequel elles se déroulent. Nous adoptons une technique d'argumentation pour définir un protocole de négociation : un agent a_u détectant un conflit de buts avec un autre agent a_v va lui demander une argumentation de son choix. L'agent a_v donne une réponse en argumentant son choix. Après évaluation des choix des deux agents, l'agent a_u peut soit demander à l'agent a_v d'interrompre sa tâche si son choix a plus de poids que celui de l'agent a_v, soit informer l'agent a_v de son choix dans le cas contraire. Dans ce dernier cas, l'agent a_v confirme l'interruption de sa tâche.

Le protocole de négociation *NEG-P* peut être défini en fonction des trois protocoles élémentaires comme suit (cf. figure 6.6) :

NEG-P=REQ-P.(REF-P;INF-P.INF-P)

Exemple 6.5 : Protocole de négociation.

Afin d'illustrer le protocole de négociation, nous reprenons l'exemple 6.2. L'agent a_1 admet un but qui consiste à exécuter la tâche T_{CBA}. Supposons que l'agent a_2 souhaite déplacer le bloc B de la colonne $c3$ vers la colonne $c2$, il envoie alors une requête à l'agent a_1 lui demandant le déplacement de ce bloc :

$$(d_1\ a_2\ a_1)(REQUEST)(PERFORMED\ Dépiler(B)\ [t_{now}, +\infty])(ctxt)$$

L'agent a_1 détecte que cette tâche est en opposition avec la tâche T_{CBA}, il déduit qu'il y a un conflit. Il répond par un message de contenu vide afin de terminer le protocole de requête :

$$(d_2\ a_1\ a_2)(RESPOND)(\quad d_2)(ctxt)$$

L'agent a_1 enclenche le protocole de négociation et demande des informations sur le choix de l'agent a_2 :

$$(d_3\ a_1\ a_2)(REQUEST)(\text{INFORMATION-ABOUT } ctxt\ T)(ctxt1)$$

L'agent a_2 répond qu'il désire exécuter la tâche T_{BC} (C est posé sur B) :

$$(d_4\ a_2\ a_1)(RESPOND)(T_{BC}\ d_4)(ctxt1)$$

En supposons que la tâche T_{CBA} admet plus de poids que la tâche T_{BC}, l'agent a_1 envoie une demande d'interruption de la tâche T_{BC} à l'agent a_2 :

$$(d_5\ a_1\ a_2)(REFRAIN\ ctxt)(ctxt1)$$

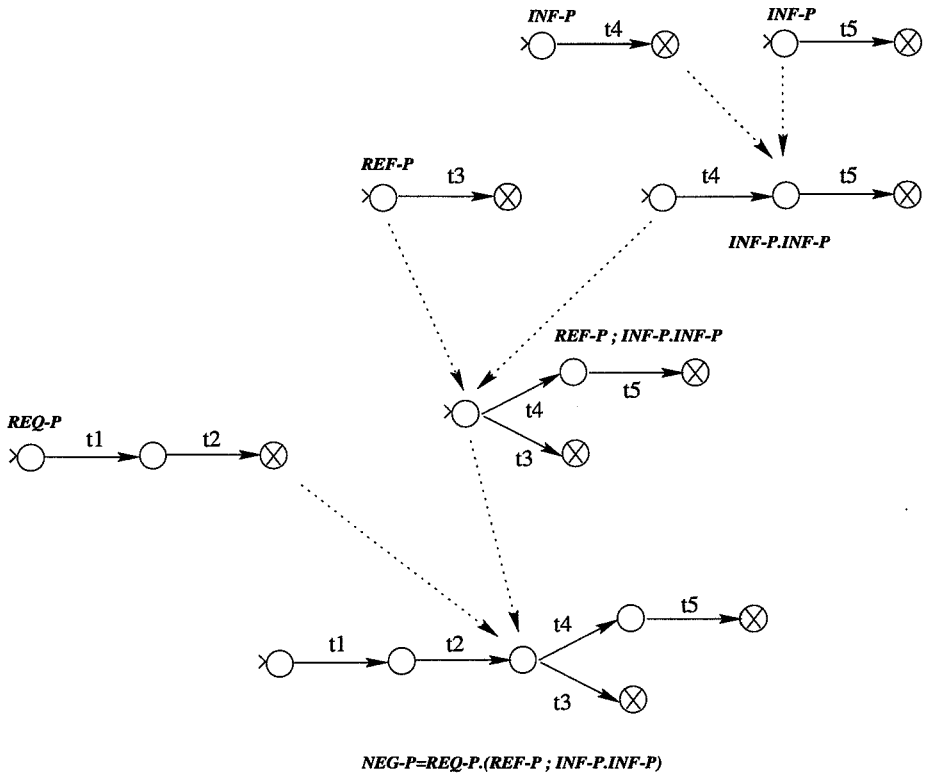


Figure 6.6 : Définition du protocole de négociation en fonction des trois protocoles élémentaires.

6.6 Discussion

Dans ce chapitre, nous avons proposé un modèle d'interaction basé sur un ensemble de protocoles élémentaires constituant les briques de base pour construire des protocoles plus complexes tels que le protocole de négociation. Lors d'une conversation, il est important que les deux interlocuteurs interprètent les messages échangés d'une manière uniforme et cohérente. Pour cette raison, à chaque protocole élémentaire est associée une sémantique basée sur les actes de langage. Nous avons utilisé deux actes correspondant aux deux performatifs : *commander* et *informer*. Nous avons défini pour chacun de ces actes : la sémantique, ainsi que l'acte perlocutoire. Ceci constitue un moyen permettant à l'allocutaire de reconnaître l'intention du locuteur lors d'une conversation. Le langage d'interaction, ainsi que l'ensemble des protocoles proposés permettent aux agents d'exprimer leurs besoins d'envoyer des requêtes, d'informer ou de demander l'interruption d'une tâche.

De plus, le langage d'interaction permet d'exprimer des contraintes temporelles représentant le délai qu'il ne faut pas dépasser pour répondre à une

requête par exemple. Ces contraintes sont prises en compte également au niveau de la définition des sémantiques associées aux différents protocoles. L'intérêt apporté par l'introduction de cette dimension temporelle est la bonne adaptation pour manipuler des tâches où les contraintes temporelles jouent un rôle essentiel entre elles.

Dans le chapitre suivant, nous allons voir que le modèle d'agent proposé, tient compte de ces contraintes dans tous les types de traitement qu'un agent est amené à faire. Ainsi, le modèle d'interaction constitue un moyen d'échanger les contraintes temporelles entre les différents agents impliqués dans une coopération et donc dans une conversation.

Chapitre 7

Modèle d'agent temporel

7.1 Introduction

Le présent chapitre vise à décrire le modèle d'un agent temporel qui intègre les modèles d'organisation et d'interaction présentés dans les deux chapitre précédents.

Dans notre contexte, les agents évoluent dans un environnement dynamique où ils doivent accomplir des tâches. Les durées, ainsi que l'ordre d'exécution des tâches peuvent varier dans le temps.

Le modèle sera présenté en détaillant chacune des fonctions au sein de l'agent. Pour ce faire, une distinction sera faite entre deux niveaux : le *niveau individuel* et le *niveau social.*

A noter que ces fonctions permettent la prise en compte des contraintes temporelles liées au domaine d'application et dont l'importance se manifeste au niveau de l'exécution des différentes tâches et plus spécifiquement au niveau de la gestion des buts. Cette prise en compte des aspects dynamiques permet à l'agent de fonctionner dans un contexte où des contraintes temporelles sont posées sur les durées des tâches ainsi que sur leur succession.

7.2 Architecture de l'agent

Nous proposons une architecture composée d'un *état mental*, d'un ensemble de *descriptions externes* relatives aux autres agents et d'un ensemble de fonctions. Ces éléments vont permettre à l'agent de fonctionner dans une société dont le modèle est proposé dans les chapitres 5.

La figure 7.1 illustre les flots de contrôle et les flots de données entre les différentes fonctions et structures. Chacune d'entre elles sera décrite en détail dans les sections 7.4 et 7.5, en précisant si elle intervient au niveau individuel ou au niveau social au sein de l'agent.

L'*état mental* de l'agent contient toutes les informations intervenant dans le raisonnement de l'agent.

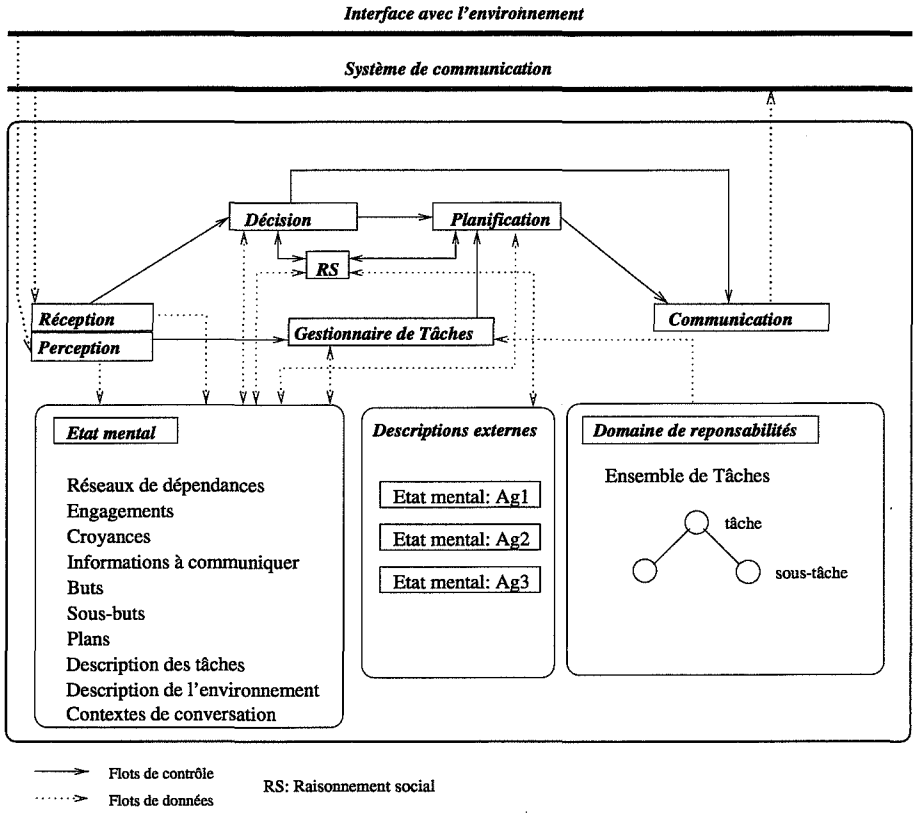


Figure 7.1 : Architecture d'un agent temporel

L'ensemble des *descriptions externes* consiste en une liste d'états mentaux des autres agents que l'agent construit localement. Chaque état mental correspond à l'image d'un agent.

L'interface de l'agent avec l'extérieur est assurée par trois fonctions :

- *perception.* Nous rappelons que l'environnement d'un agent est formé des autres agents, ainsi que des ressources accessibles par cet agent. La fonction de perception assure la détection des changements de l'état des ressources dans l'environnement. Ces changements se présentent sous forme d'événements qui constituent les entrées de cette fonction. Elle génère en sortie le type de l'événement qui s'est produit ainsi que sa date d'observation ;

- *réception.* Cette fonction assure la réception des messages (entrées) des autres agents. La syntaxe des messages respecte la grammaire du langage $\mathcal{L}$ proposé dans le chapitre 6. En fonction du type de message, cette fonction génère en sortie un *engagement* ou une *croyance* (cf. section 7.5.2) ;

- *communication.* En entrée, cette fonction reçoit une structure contenant les informations nécessaires à la formation d'un message écrit en langage $\mathcal{L}$. Elle génère le message et provoque en sortie l'envoi du message via la couche de communication vers un ou plusieurs destinataires.

En interne, l'agent doit pouvoir gérer des tâches qui sont sous sa responsabilité. Le *gestionnaire de tâches* assure cette fonction. Il admet en entrée toute information générée par la fonction de perception. En sortie, il génère des informations concernant les délais à respecter dans l'exécution des différentes tâches.

Lors de la réception d'un message ou de la perception d'un événement, une décision est prise par la *fonction de décision.* Cette fonction est chargée essentiellement de la gestion des buts.

A chaque tâche devant être exécutée correspond un but. Si la tâche admet des sous-tâches (cf. section 5.2.2), une *fonction de planification* décompose le but en plusieurs sous-buts correspondant chacun à une sous-tâche. Pour permettre l'exécution de la tâche, ainsi que de l'ensemble de ses sous-tâches, cette fonction génère un plan d'exécution relatif au but : ce plan précise comment vont être exécutées les sous-tâches qui ne sont pas sous la responsabilité de l'agent. Cette fonction est chargée également de l'exécution des plans en utilisant les délais générés par le gestionnaire de tâche.

Enfin, une fonction dédiée au raisonnement social est invoquée par la fonction de décision et la fonction de planification avant chaque création de but. Cette fonction permet la mise à jour des descriptions externes.

7.3 Etat mental

L'*état mental* désigne toute la partie dynamique des différentes représentations et connaissances d'un agent. Il admet la structure suivante :

```
(MENTAL-STATE
   :dep-C <DEP-NET>
   :dep-D <DEP-NET>
   :dep-H <DEP-NET>
   :commitments <S-COMMIT-L>
   :belief-list <BELIEF-L>
   :out-msg-list <MSG-ENTRY-L>
   :goal-list <GOAL-L>
   :sub-goal-list <SUB-GOAL-L>
   :plan-list <PLAN-L>
   :task-list <TASK-L>
   :env-desc <EVENT-L>
   :context-list <CTXT-L>)
```

Cette structure modélise les éléments suivants : trois réseaux de dépendances de types $\mathcal{C}$ (:dep-C), $\mathcal{D}$ (:dep-D) et $\mathcal{H}$ (:dep-H), la liste des engagements sociaux (:commitments), une liste d'informations (:out-msg-list) utilisable par la fonction de communication (section 7.5.3), une liste de croyances (:belief-list) utilisable par la fonction de décision (section 7.4.3), la liste des buts (:goal-list), la liste des plans associés aux buts (:plan-list), la liste des sous-buts (:sub-goal-list), la description des tâches (:task-list), la description de l'environnement (:env-desc) et enfin, la liste des contextes de conversation.

Dans les sections suivantes, nous décrivons chacun de ces éléments.

7.3.1 Réseaux de dépendances

Un réseau de dépendances admet la structure suivante :

(DEP-NET
:type <*dep-id*>
:dep-list <DEP-L>)

Ce réseau admet un type (:type) de valeur $\mathcal{C}$, $\mathcal{D}$ ou $\mathcal{H}$, selon les modèles décrits dans le chapitre 5. Il admet également la liste des dépendances de ce type avec les autres (:dep-list).

Le type DEP-L est défini comme suit :

<DEP-L> ::= '('<DEP>*')'

(DEP
:active <*value*>
:agent <*ag-id*>
:local-task <*task-id*>
:remote-task <*task-id*>)

La structure DEP contient un champ indiquant si la dépendance est active ou non (:active), l'identificateur de l'agent concerné par la dépendance (:agent), ainsi que les identificateurs des deux tâches faisant l'objet de cette dépendance (:local-task, :remote-task).

7.3.2 Engagements sociaux

Suite à la réception d'un messages de requête, une information est inscrite dans l'état mental (:commitments). Cette information est considérée comme un engagement social envers l'agent ayant envoyé la requête. La liste

des engagements sociaux est définie par le type S-COMMIT-L :

<S-COMMIT-L> ::= '('<S-COMMIT>*')'

Un engagement social est défini comme suit :

(S-COMMIT
:agent <*ag-id*>
:task <*task-id*>
:context <*context*>
:delay <*timeout*>)

Un agent s'engage auprès d'un autre agent (:agent) pour exécuter une tâche (:task) ou donner une information sur la création d'un contexte de conversation (:context), en respectant un délai (:delay).

Un engagement admet deux formes, correspondant respectivement à une requête d'exécution de tâche et à une requête d'information :

- dans le premier cas, l'engagement est de la forme :

  (S-COMMIT
  :agent a_u
  :task T
  :context *NULL*
  :delay $[d_{min}, d_{max}]$

  Il admet la sémantique suivante (cf. section 6.3.1 pour la définition des différents prédicats, opérateurs et fonctions) :

  S-COMMIT(self, a_u, DONE(T, $[d_{min}, d_{max}]$))

  S-COMMIT(self, a_u, DONE(informer(self, a_u, results-when-performed(task)) $\vee$ informer(self, a_u, failed(task)), $[d_{min}, d_{max}]$))

  Cette sémantique signifie que l'agent (self) s'engage à exécuter la tâche T et à informer l'agent a_u des résultats de l'exécution si la tâche est exécutée avec succès. Si elle échoue l'agent a_u est également informé ;

- dans le second cas, l'engagement est sous la forme :

  (S-COMMIT
  :agent a_u
  :task *NULL*
  :context *ctxt*
  :delay $[d_{min}, d_{max}]$

Il admet la sémantique suivante :

S-COMMIT(self, a_u, DONE(informer(self, a_u, get-desc($ctxt$)), $[d_{min}, d_{max}]$))

L'agent (self) s'engage à informer l'agent a_u, de la description (*get-desc*) associée au contexte de conversation *ctxt*.

7.3.3 Croyances

Les informations provenant de l'extérieur par l'intermédaire d'un message de réponse à une requête ou un message d'information sont considérées comme des nouvelles croyances. La liste des croyances est définie par le type BELIEF-L :

<BELIEF-L> ::= '('<BELIEF>*')'

Une croyance est définie comme suit :

(BELIEF
:agent <*ag-ig*>
:task <*task-id*>
:status <*value*>
:results <*task-results*>
:date <*date*>
:context <*context*>
:when <*date*>)

Une croyance concerne un agent (:agent), à propos d'une tâche (:task). Cette tâche a échouée ou alors a été exécutée avec succès (:status) à la date :date et dans ce cas, elle admet éventuellement des résultats (:results). Le champ :context précise le contexte de conversation relatif à cette croyance et le champ :when indique quand cette croyance a été communiquée.

7.3.4 Informations échangées

La liste :out-msg-list contient toutes les informations que l'agent désire communiquer aux autres agents. Cette liste est définie comme suit :

<MSG-ENTRY-L> ::= '('<MSG-ENTRY>*')'

Une entrée de la liste admet la structure suivante :

(MSG-ENTRY
:agent <*ag-ig*>
:type <*keyword*>
:task <*task-id*>

:results	<*task-results*>
:delay	<*timeout*>
:date	<*date*>
:context	<*context*>
:info	<*info-id*>
:msg-date	<*date*>

Cette structure contient tous les éléments nécessaires pour la construction d'un message en langage $\mathcal{L}$ (cf. section 6.2).

7.3.5 Buts

La notion de but est liée à l'exécution des tâches par les agents. A chaque but est associée une tâche et un *délai*. Un but est *satisfait* si la tâche associée a été exécutée avec succès tout en respectant le délai autorisé. La liste des buts est définie par le type GOAL-L :

<GOAL-L> ::= '('<GOAL>*')'.

Le type GOAL modélisant un but, est défini comme suit :

(GOAL

:id	<*goal-id*>
:type	<*value*>
:satisfied	<*value*>
:satifiable	<*value*>
:subgl-list	<GOAL-ID-L>
:exec-agent	<*ag-id*>
:res-agent	<*ag-id*>
:task	<*task-id*>
:delay	<*timeout*>
:conversation	<*context*>)

Le type GOAL-ID-L est défini comme suit :

<GOAL-ID-L> ::= '('<*goal-id*>*')'

La structure GOAL contient l'identificateur du but (:id), la liste des identificateurs des sous-buts (:subgl-list), l'identificateur de l'agent chargé de l'exécution de la tâche associée au but (:exec-agent), l'identificateur de l'agent concerné par le résultat de l'exécution de cette tâche (:res-agent), l'identificateur de la tâche (:task), le délai associé à la tâche (:delay), le contexte de

conversation associé au but (:conversation), un champ indiquant si le but est satisfait (:satisfied) et un champ indiquant s'il est satisfaisable (:satisfiable).

Types de but

Pour un agent a_u, nous distinguons trois types de buts (:type) :

- un but est dit *interne*, si l'agent est chargé de l'exécution de la tâche et il est également l'agent concerné par le résultat de cette exécution (:exec-agent = :res-agent = a_u) ;
- un but est dit *externe*, si l'agent est chargé de l'exécution de la tâche et un autre agent est concerné par le résultat de cette exécution (:exec-agent = a_u, :res-agent $\neq a_u$) ;
- un but est dit *dépendant*, si l'agent est concerné par le résultat de l'exécution de la tâche et cette exécution doit être effectuée par un autre. En fait, le choix de l'autre agent dépend des dépendances qui existent entre les deux agents, ce qui explique pourquoi le but est dit dépendant (:res-agent = a_u, :exec-agent $\neq a_u$).

La distinction entre ces trois types est utile lors de la gestion des buts (section 7.4.3).

Sémantique d'un but

A noter, que les buts et les sous-buts (section suivante) modélisent les intentions de l'agent. En reprenant les définitions données dans la section 6.3.1, nous pouvons associer à un but relatif une tâche T et un délai I la sémantique suivante :

$$INT(self,\ DONE(T,\ I))$$

Cette sémantique est également valable pour un sous-but.

7.3.6 Sous-buts

Nous avons vu dans la section 5.2.2 qu'une tâche de haut niveau peut être décomposée en plusieurs sous-tâches. Le but associé à cette tâche doit être décomposé en plusieurs sous-buts où chaque *sous-but* est associé à une sous-tâche.
La liste des sous-buts est définie par le type SUB-GOAL-L :

<SUB-GOAL-L> ::= '('<GOAL>*')'

Un sous-but admet la structure suivante :

(SUB-GOAL
:id <*goal-id*>
:up-level <*goal-id*>
:subgl-list <GOAL-ID-L>
:exec-agent <*ag-ig*>
:sub-task <*task-id*>
:delay <*timeout*>
:conversation <*context*>)

Un sous-but est toujours attaché à un seul but de haut niveau (:up-level). Pour un agent donné, les sous-buts sont dépendants ou internes, ce qui explique l'existence du seul champ (:exec-agent) qui correspond à l'identificateur de l'agent chargé de l'exécution de la tâche associée au sous-but. Les autres champs ont été décrits dans la section précédente.

7.3.7 Plans

La décomposition d'un but en sous-buts, nécessite l'élaboration d'un plan d'exécution des différentes sous-tâches associées aux différents sous-buts.

La liste de plans est définie par le type PLAN-L :

<PLAN-L> ::= '('<PLAN>*')'

Un plan admet la structure suivante :

(PLAN
:goal <*goal-id*>
:entry-list <ENTRY-L>)

Un plan contient l'identificateur du but associé à la tâche (:goal), une liste d'entrées (:entry-list) où chaque entrée contient des informations permettant l'exécution d'une sous-tâche.

Le type ENTRY-L est défini comme suit :

<ENTRY-L> ::= '('<ENTRY>*')'

Une entrée admet la structure suivante :

(ENTRY
:sub-task <*task-id*>
:delay <*timeout*>
:dep-agent <*ag-id*>)

Elle contient l'identificateur de la sous-tâche (:sub-task), le délai associé (:delay) et l'identificateur de l'agent chargé de l'exécution de cette sous-tâche (:dep-agent).

7.3.8 Description des tâches

La description des tâches élémentaires dépend du domaine d'application. Une tâche de haut niveau est décomposée en une liste de sous-tâches tout en précisant les contraintes de précédences entre les différentes sous-tâches.

La liste des tâches est définie par le type TASK-L :

<TASK-L> ::= '('<TASK>*')'

Une tâche admet la structure suivante :

(TASK
:id <*task-id*>
:condition <*precondition*>
:active <*value*>
:last <*date*>
:last-results <*task-results*>
:sub-task-list <TASK-ID-L>
:constraint-list <CONS-L>)

La description d'une tâche admet donc un identificateur (:id), une précondition qui doit être satisfaite pour que la tâche puisse être exécutée (:condition). Cette précondition peut être modélisée par un prédicat du premier ordre. Une tâche est dans un état actif ou inactif (:active). Le champ :last donne la dernière date d'exécution de la tâche avec succès et le champ :last-results donne les derniers résultats d'exécution avec succès de la tâche. Enfin, la description d'une tâche contient la liste des identificateurs de ses sous-tâches (:sub-task-list) et la liste des contraintes temporelles entre ces sous-tâches (:constraint-list). Si la tâche est élémentaire, ces deux dernière listes sont vides. En voici la définition :

<TASK-ID-L> ::= '('<*task-id*>*')'
<CONS-L> ::= '('<CONS>*')'

<CONS> ::= '('<*task-id*> [d_{min}, d_{max}] <*task-id*>')'

Une contrainte précise les durées minimale (d_{min}) et maximale d_{max} à respecter entre l'exécution de deux sous-tâches.

A noter, que l'échec ou le succès d'une tâche, n'est pas exprimé au niveau de la description d'une tâche, mais au niveau du but ou sous-but relatif à cette tâche (champs :satisfiable et :satisfied).

Les tâches constituent le domaine de responsabilité individuelle d'un agent. Donc un agent est capable d'exécuter toutes les tâches appartenant à son domaine de responsabilité :

$\forall$ T $\in$ r(self), $\exists$ I | CAN(self, DONE(T, I))

7.3.9 Description de l'environnement

La description de l'environnement est donnée à travers les *événements* qui se produisent au fil du temps. De ce fait, la description de l'environnement est en évolution permanente.

Une liste d'événements est définie par le type EVENT-L :

<EVENT-L> ::= '('<EVENT>*')'

Nous définissons un événement comme suit :

(EVENT
:id <*event-id*>
:p <*proposition*>
:d_{obs} <*date*>)

Un événement admet un identificateur (:id), la dernière date d'observation (:d_{obs}) et une proposition (:p) décrivant un état du monde, comme par exemple : *p*="*le bloc C est en colonne c3*". Si la proposition admet la valeur vraie sur un intervalle I, alors la borne inférieure de cette intervalle correspond à la date d'observation de l'événement.

En reprenant la définition du prédicat *is-true-at-date* donnée dans la section 6.3.1, la date d'observation : d_{obs} est définie comme suit :

$$\begin{cases} \text{is-true-at-date}(p, : d_{obs}) = true \\ \forall d \mid \text{is-true-at-date}(p, d) = true \Longrightarrow : d_{obs} \leq d \end{cases}$$

Quand un événement se produit, la proposition qui lui est associée devient vraie. L'ensemble des propositions qui sont vraies à un instant donné, définissent la description de l'environnement à cet instant.

7.3.10 Contextes de conversation

La liste des contextes de conversations contient tous les contextes de conversation relatifs aux buts gérés par l'agent. Cette liste est définie par le type CTXT-L :

<CTXT-L> ::= '('<CONTEXT>*')

Un contexte de conversation admet la structure suivante :

(CONTEXT
:id <*ctxt-ig*>
:desc <*info-id*>
:agent <*ag-id*>
:date <*date*>

Un contexte admet un identificateur (:id), une description (:desc) précisant le contexte de création du but correspondant, l'identificateur de l'agent à l'origine du contexte et enfin la date de sa création (:date). Si le contexte provient d'un message reçu d'un autre agent, cette date correspond à la date de réception du message. Les fonction *get-desc* et *get-date* permettent respectivement de donner la valeur des champs :desc et :date.

7.3.11 Exemple

Dans l'exemple du monde des blocs, l'agent a_1 est responsable des blocs A et B, l'agent a_2 est responsable des blocs B et C et enfin l'agent a_3 est responsable des blocs A et C. Nous supposons par ailleurs que l'agent a_1 doit exécuter la tâche T_{CBA} et nous nous proposons de décrire son état mental :

- l'attribution des différentes responsabilités va permettre aux agents de construire leurs réseaux de dépendances. Nous avons choisi de représenter le réseau $\mathcal{D}$ de l'agent a_1 :

(DEP-NET
:type $\mathcal{D}$
:dep-list
(:active *false*
:agent a_2
:local-task T_{CBA}
:remote-task *Dépiler(C)*)

(:active *false*
:agent a_3

:local-task T_{CBA}
:remote-task *Dépiler(C)*))

- l'ensemble des buts de l'agent $a1$ est formé d'un seul but $g_{T_{CBA}}$ concernant la tâche T_{CBA} :

(GOAL
:id $g_{T_{CBA}}$
:type *dependant*
:satisfied *false*
:satifiable *true*
:subgl-list ($g_{Dépiler(C)}$ $g_{Empiler(B,c3,c2)}$ $g_{Empiler(A,c1,c2)}$)
:exec-agent *NULL*
:res-agent a_1
:task T_{CBA}
:delay $[t_{now}, +\infty]$
:conversation *ctxt1*)

La valeur du champ (:exec-agent) est nulle car cette tâche doit être effectuée par plusieurs agents. La valeur infinie du délai signifie qu'il n'y a pas de contraintes sur la tâche T_{CBA} et qu'elle peut être exécutée à n'importe quel instant dans le futur ;

- le but $g_{T_{CBA}}$ se décompose en trois sous-buts :

(SUB-GOAL
:id $g_{Dépiler(C)}$
:up-level $g_{T_{CBA}}$
:subgl-list *NULL*
:exec-agent a_3
:sub-task *Dépiler(C)*
:delay $[t_{now}, +\infty]$
:conversation *ctxt11*)

(SUB-GOAL
:id $g_{Empiler(B,c3,c2)}$
:up-level $g_{T_{CBA}}$
:subgl-list *NULL*
:exec-agent a_2
:sub-task *Empiler(B, c3, c2)*
:delay $[t_{now}, +\infty]$

:conversation *ctxt12*)

(SUB-GOAL
:id $g_{Empiler(A,c1,c2)}$
:up-level $g_{T_{CBA}}$
:subgl-list *NULL*
:exec-agent a_1
:sub-task *Empiler(A, c1, c2)*
:delay $[t_{now}, +\infty]$
:conversation *ctxt13*)

Les identificateurs des agents (a_2 et a_3) qui peuvent satisfaire les deux premiers sous-buts sont déterminés par la consultation du réseau type $\mathcal{D}$. Si plusieurs agents peuvent exécuter la même tâche, la relation de pouvoir est utilisée comme critère de sélection. C'est l'agent qui a le moins de pouvoir vis-à-vis de cette tâche qui se chargera de son exécution ;

- l'unique plan correspondant à la décomposition du but $g_{T_{CBA}}$ en trois sous-buts est défini comme suit :

(PLAN
:goal $g_{T_{CBA}}$
:entry-list
((ENTRY
:sub-task *Dépiler(C)*
:delay $[t_{now}, +\infty]$
:dep-agent $a3$)

(ENTRY
:sub-task *Empiler(B, c3, c2)*
:delay $[t_{now}, +\infty]$
:dep-agent $a2$)))

Ce plan contient uniquement deux entrées car l'agent a_1 est responsable du sous-but $g_{Empiler(A,c1,c2)}$ (:exec-agent=a_1). Par ailleurs, le choix des agents n'est pas arbitraire, en effet, l'agent utilise son raisonnement social pour choisir parmi plusieurs agents capables d'effectuer la même tâche (cf. section 7.5.5) ;

- les tâches élémentaires sont du type : Dépiler(X) ou Empiler(X, Y, Z) où $X \in \{A, B, C\}$ et $Y, Z \in \{c1, c2, c3\}$. Afin de simplifier, l'identificateur d'une tâche élémentaire sera le même que son type. Pour

pouvoir exécuter ces tâches, il faut respecter la contrainte suivante : un bloc ne peut pas être déplacé s'il n'est pas libre. Cette contrainte peut être modélisée par un prédicat et constitue une précondition pour les sous-tâches élémentaires. La tâche T_{CBA} est définie comme suit :

(TASK
:id T_{CBA}
:condition *NULL*
:active *false*
:last d_n
:sub-task-list
(*Dépiler(C), Empiler(B, c3, c2), Empiler(A, c1, c2)*)
:constraint-list
((*Dépiler(C)* [0, $+\infty$] *Empiler(B, c3, c2)*)
(*Empiler(B, c3, c2)* [0, $+\infty$] *Empiler(A, c1, c2)*)))

- la description de l'environnement à la date d_0 (aucun bloc n'a été déplacé) est donnée par les trois propositions suivantes :

p1="A est en colonne c3 et en deuxième position"
p2="B est en colonne c3 et en première position"
p3="C est en colonne c1 et en première position"

Ces propositions correspondent à trois événements :

(EVENT
:id *e1*
:p *p1*
:d_{obs} d_0)

(EVENT
:id *e2*
:p *p2*
:d_{obs} d_0)

(EVENT
:id *e3*
:p *p3*
:d_{obs} d_0)

- la liste des contexte de conversation est la suivante :

(CONTEXT
:id *ctxt1*

:desc *internal goal, external constraint*
:agent a_1
:date d)

(CONTEXT
:id *ctxt11*
:desc *dependant sub-goal, external constraint*
:agent a_1
:date d)

(CONTEXT
:id *ctxt12*
:desc *dependant sub-goal, external constraint*
:agent a_1
:date d)

(CONTEXT
:id *ctxt13*
:desc *internal goal, external constraint*
:agent a_1
:date d)

7.4 Niveau individuel

L'exécution des tâches sous la responsabilité d'un agent, nécessite souvent la prise en compte des autres agents. Cependant, un agent doit pouvoir effectuer un certain nombre de tâches indépendamment des autres. Il doit être capable de percevoir son environnement : connaître les occurrences d'événements produisant des changements dans l'environnement. Il doit également pouvoir gérer des tâches qui sont en cours d'exécution et qui appartiennent à son domaine de responsabilité individuelle. Le processus de décision est également propre à l'agent : il prend seul la décision de créer et de poursuivre un but, de croire à un événement ou un fait. Afin de remplir cet ensemble d'activités individuelles, un agent possède un ensemble de fonctions dédiées à ces activités : la perception, la gestion des tâches, la décision et la planification.

7.4.1 Perception

Cette fonction se charge de la perception des événements provenants de l'environnement. Ces événements concernent uniquement les tâches qui sont sous la responsabilité de l'agent (événements provenants de capteurs particuliers). Le domaine de responsabilité sera utilisé pour la sélection des

capteurs relatifs à un agent. Tout événement perçu, est inscrit dans l'état mental dans une structure EVENT (cf. section 7.3.9).

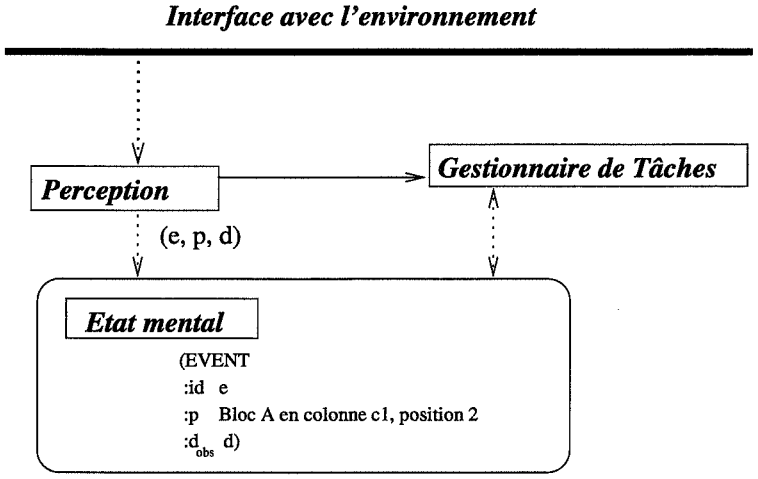


Figure 7.2 : La fonction de Perception

Dans cette structure, la date d'observation (d_{obs}) correspond à la date à laquelle l'événement a été perçu par la fonction de perception. Cette date diffère de la *date d'occurrence* où l'événement a eu lieu réellement (d_{occ}). Nous précisons qu'il n'existe aucun moyen de connaître la date d'occurrence d'un événement et que la seule information qu'on détient est sa date d'observation délivrée par un capteur. Par ailleurs, la date de prise en compte d'un événement appelée *date de traitement*, notée d_t, est la date à laquelle l'événement est pris en compte. Nous avons la relation suivante :

$$d_{occ} \leq d_{obs} \leq d_t.$$

La distinction de ces trois dates pose les problèmes suivants :

- une imprécision des capteurs aura tendance à augmenter l'écart entre d_{occ} et d_{obs}. Si cet écart est assez important, l'ordre d'occurrence des événements risque de ne plus correspondre à la réalité et le raisonnement de l'agent sur ces événements est faussé ;

- quelle que soit la précision des capteurs, il existe un écart minimum non nul entre d_{occ} et d_{obs} d'un événement. Dans un contexte où la fréquence d'occurrence d'événements est assez élevée, cet écart conduit au masquage d'une ou de plusieurs occurrences (figure 7.3).

L'importance accordée à ce type de problème dépend du domaine d'application traité. En effet, dans un contexte où les occurrences d'événements doivent être contraintes dans le temps, le masquage ou le retard d'une observation peut être fatal pour la bonne conduite d'un processus tel que la

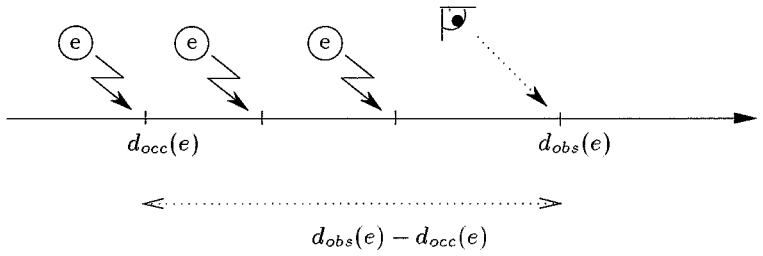


Figure 7.3 : Plusieurs occurrences d'un même événement peuvent être masquées à cause de l'écart entre une date d'occurrence et une date d'observation.

supervision d'un système industriel par exemple. S'il est impossible d'avoir des améliorations au niveau des capteurs, les agents peuvent contourner le problème, s'ils sont capables de raisonner en présence d'informations incertaines.

Exemple 7.1 : Masquage et retard d'événement.

Nous reprenons l'exemple tiré du monde des blocs (figure 5.1) avec un agent a_1, responsable des blocs A et B, et un agent a_2, responsable des blocs B et C. L'événement typique que nous considérons est la perception d'un bloc qui vient se poser sur un autre bloc. Par exemple, si l'agent a_2 pose le bloc C sur le bloc B (*Empiler(C, c1, c3)*), l'agent a_1, étant responsable du bloc B, perçoit cet événement à l'aide d'un capteur placé à bord du bloc B qui est chargé de lui transmettre le type de l'événement (*Empilement du bloc C sur le bloc B*) et sa date d'observation. Supposons que l'agent a_2 a fini de déplacer le bloc C à l'instant t (d_{occ}) et supposons qu'à cause de l'imprécision des capteurs, l'agent a_1 ne prend connaissance de cet événement qu'à l'instant $t + \Delta$. Si l'agent a_1 tente de déplacer le bloc B à l'instant $t + \frac{\Delta}{2}$, il aura la surprise de voir que le bloc C est au sommet de la pile et qu'il est impossible de déplacer le bloc B.

Un autre problème peut se poser également si la durée $\Delta = d_{obs} - d_{occ}$ est assez longue pour permettre plusieurs déplacements des blocs avant que l'agent a_1 ne perçoive l'événement. Si par exemple, l'agent a_2 déplace le bloc C sur le bloc B, ensuite il dépile le bloc C vers la colonne $c2$ et enfin il déplace le bloc A sur le bloc B pendant la durée Δ, à l'instant $t + \Delta$, l'agent a_1 aura uniquement perçu le premier événement car les autres auront été masqués par cet événement. De ce fait et à cet instant, il lui sera impossible de déplacer le bloc A de la colonne $c1$, car ce bloc occupe à présent la colonne $c3$.

7.4.2 Gestion des tâches

Le gestionnaire de tâches se charge de la gestion des tâches (cf. section 7.3.8) qui sont sous la responsabilité individuelle de l'agent.

Activation d'une tâche

Une tâche doit être exécutée, si ses préconditions (cf. section 7.3.8) sont vérifiées. Une précondition représente souvent la réalisation d'un état dans l'environnement. L'état de l'environnement étant décrit à travers les événements qui s'y produisent, l'observation d'un événement, peut provoquer l'exécution d'une tâche qui devient active : par exemple, dans la surveillance d'un processus chimique, la tâche qui consiste à diminuer la température du produit en réaction peut admettre comme précondition un dépassement de température ou de pression qui peut être modélisé par un événement.

Un événement peut également empêcher l'exécution d'une tâche: par exemple, dans le monde des blocs, un agent est incapable de déplacer un bloc X qui n'est pas libre. Un agent détecte qu'un bloc X n'est pas libre, s'il observe un événement du type (*Empilement du bloc Y sur le bloc X*).

Exécution des sous-tâches

Un ensemble de sous-tâches n'est pas toujours exécuté par un seul agent. Le plan (section 7.3.7) relatif à la tâche globale indique comment une sous-tâche doit être exécutée quand le but relatif à cette sous-tâche est *dépendant*.

L'agent chargé de l'exécution de cette sous-tâche doit avoir le délai associé à cette sous-tâche. Le gestionnaire de tâches vérifie le respect des contraintes temporelles entre les sous-tâches déjà exécutées et quand une sous-tâche doit être exécutée, il inscrit le délai associé à cette sous-tâche dans l'entrée correspondante du plan (figure 7.4).

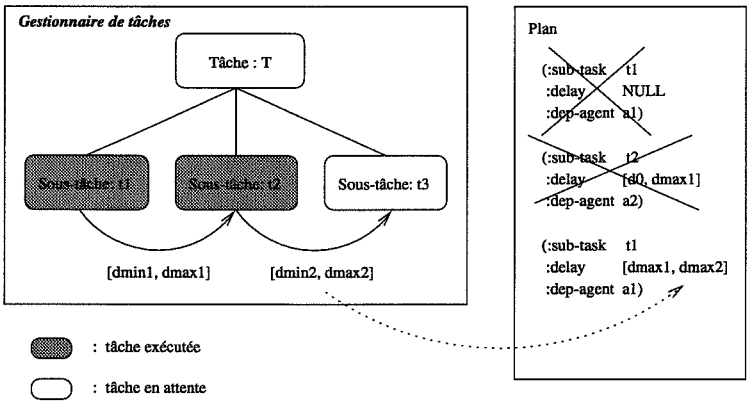


Figure 7.4 : Gestion des délais relatifs aux sous-tâches.

Raisonnement hypothétique

Quand un délai relatif à une sous-tâche n'est pas respecté, cela signifie l'échec de l'exécution de cette sous-tâche

Cependant, cela reste insuffisant car une sous-tâche étant parfois exécutée par un autre agent, ce dernier peut l'avoir exécutée tout en respectant le délai associé, mais sa réponse parvient tardivement à l'agent ayant envoyé la requête. Si la réponse ne respecte pas le délai associé à l'exécution d'une sous-tâche, le gestionnaire de tâches ne peut se mettre en attente de cette réponse car cela empêche le suivi des autres sous-tâches et le calcul de leurs délais. Le gestionnaire de tâches doit donc être capable de continuer son suivi des sous-tâches même en cas de réponse tardive. Pour ce faire, nous adoptons une stratégie selon laquelle le gestionnaire de tâches utilise un *raisonnement hypothétique* (figure 7.5) : quand une réponse ne parvient pas à temps, le

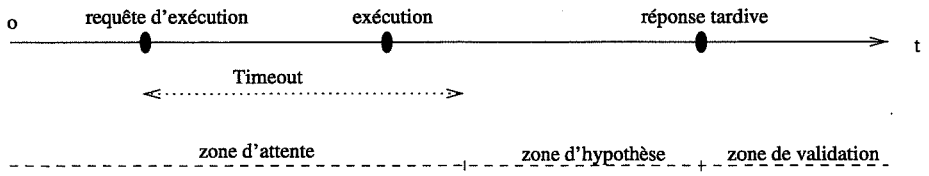


Figure 7.5 : Les zones d'hypothèse et de validation au cours d'un raisonnement hypothétique

gestionnaire de tâches **émet l'hypothèse** que la sous-tâche en question a été exécutée tout en respectant son délai et continue le suivi de l'exécution des autres sous-tâches. Le sous-but correspondant à la sous-tâche n'étant pas supprimé, le but global ne peut être satisfait. Il ne le sera que lorsque ce sous-but sera vérifié. Ainsi, le gestionnaire de tâches peut suivre l'exécution des autres sous-tâches et calculer leurs délais. Une réponse étant assurée par le protocole de requête (cf. section 6.4.1), lui permet par la suite, de vérifier la validité de son hypothèse, grâce à la date d'exécution de la sous-tâche inscrite dans la réponse. Si la sous-tâche a effectivement été exécutée avec succès, le sous-but correspondant étant satisfait, est supprimé. Le but global est satisfait également. Dans le cas contraire, le but correspondant ne peut plus être satisfait, le but global non plus. Dans ce cas, toutes les sous-tâches correspondant aux autres sous-buts doivent être interrompues. Les sous-buts en question sont supprimés et le but global l'est également.

Exemple 7.2 : Robot jongleur.

Pour illustrer le raisonnement hypothétique, nous allons reprendre notre exemple des blocs en y apportant quelques modifications. Nous tiendrons compte de ces modifications uniquement dans cette section. A présent, les trois blocs ne sont plus rangés dans des colonnes, mais peuvent rouler sur deux rails dans deux sens opposés. Sur un rail, les blocs ne peuvent se déplacer que dans un seul sens. Nous considérons deux agents a_1 et a_2. Chacun peut garder uniquement un seul bloc de son côté. En d'autres termes, un des trois blocs est en permanence sur un des deux rails (pensez à un jongleur où les agents a_1 et a_2 sont assimilés aux deux mains du jongleur). Nous nous plaçons dans le contexte illustré par la figure 7.6. Pour

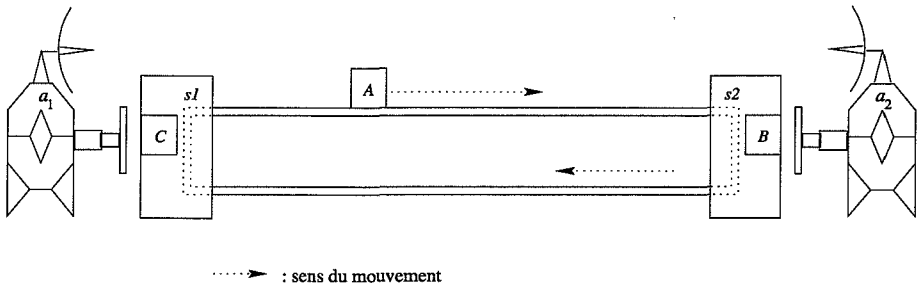


Figure 7.6 : Les blocs peuvent rouler sur deux rails dans deux sens opposés.

garder l'équilibre, chacun des deux agents dès l'envoi d'un bloc doit demander à l'autre agent d'envoyer le bloc qu'il possède. Ce dernier doit confirmer l'envoi du bloc en envoyant un message pour répondre à la demande. Ainsi, dès que l'agent a_1 envoie le bloc A, il demande à l'agent a_2 d'envoyer le bloc B. En supposant qu'un bloc met x unités de temps pour traverser un rail, l'agent a_1 ne peut pas attendre la réponse de a_2 plus de x unités de temps, car si la réponse est en retard pour une raison quelconque, le bloc B arrivera avant que l'agent a_2 n'ait eu le temps d'envoyer le bloc C. Ainsi l'agent a_1 doit utiliser un raisonnement hypothétique, lui permettant, en cas de retard d'une réponse, de supposer qu'un bloc a été envoyé du coté de l'agent a_2 et d'envoyer un bloc de son coté. Dans ce cas, si le bloc est effectivement envoyé par l'agent a_2 avec un retard de confirmation, l'agent a_1 aura sauvé la situation en envoyant un bloc et pourra par la suite vérifier que son hypothèse est valide quand il recevra la confirmation de l'agent a_2. Si par contre l'agent a_2 n'avait pas envoyé de bloc, l'hypothèse est invalide et un problème de surcharge se produira du coté de l'agent a_2. A noter, que le problème produit est indépendant de l'agent a_1. Son hypothèse n'était qu'une tentative de sauver la situation en cas de retard d'une réponse.

Dans un cadre, plus général le raisonnement hypothétique peut être utilisé pour éviter des problèmes résultants d'un retard d'une information au sujet de l'arrivée d'un événement ou de l'exécution d'une tâche. Ce raisonnement est admis uniquement dans un contexte où il n'y a pas de perte d'informations et que les agents sont toujours assurés de recevoir les informations tardivement ou de les inférer quand elles sont perdues.

7.4.3 Décision

Les informations reçues de l'extérieur, soit sous forme d'événements ou sous formes de messages des autres agents, doivent permettre la mise à jour des différents éléments dans l'état mental. Dans certains cas, le traitement des ces informations nécessite la création de nouveaux buts. La fonction de décision gère la création et la suppression des buts.

Traitement des événements

Comme nous l'avons signalé dans la section 7.4.2, l'observation d'un événement permet ou empêche l'exécution de certaines tâches.

Si une tâche T doit être exécutée (:condition=vrai), la fonction de décision construit un but relatif à cette tâche et lui associe un nouveau contexte de conversation. Une décomposition de ce but en sous-buts est réalisée par la fonction de planification (cf. section 7.4.4), si nécessaire. Ce but admet la structure suivante :

(GOAL

:id	g_T
:type	*NULL*
:satisfied	*false*
:satifiable	*true*
:subgl-list	*NULL*
:exec-agent	*NULL*
:res-agent	*me*
:task	T
:delay	*NULL*
:conversation	*new1*)

Les champs :type, :subgl-list et :exec-agent seront complétés par la fonction de planification (section 7.4.4). Le champ :delay est NULL car aucune contrainte n'est associée à l'exécution de cette tâche.

Traitement des engagements

Un engagement social pris suite à la réception d'une requête d'exécution de tâche nécessite la création d'un but. Pour un engagement social de la forme (S-COMMIT :agent=a_u :task=T :context=*NULL* :delay=$[d_{min}, d_{max}]$), le but externe associé est le suivant :

(GOAL

:id	g_T
:type	*external*
:satisfied	*false*
:satifiable	*true*
:subgl-list	*NULL*
:exec-agent	*me*
:res-agent	a_u
:task	T
:delay	$[d_{min}, d_{max}]$

:conversation *new1*)

Si l'engagement est de la forme (S-COMMIT :agent=a_u :task=*NULL* :context=*old1* :delay=$[d_{min}, d_{max}]$), une structure de message est construite et est ajoutée à la liste :out-msg-list dans l'état mental :

(MSG-ENTRY

:agent	a_u
:type	*RESPOND*
:task	*NULL*
:results	*NULL*
:delay	*NULL*
:date	*get-date(old1)*
:context	*new1*
:info	*get-desc(old1)*

Traitement d'une croyance

Une croyance (BELIEF) porte soit sur l'exécution d'une sous-tâche, soit sur un événement :

- dans le premier cas, si la sous-tâche est exécutée avec succès alors sa description est mise à jour permettant ainsi au gestionnaire de tâches la prise en compte de cette exécution dans l'exécution de la tâche globale. Le but associé à la sous-tâche est satisfait. Si la sous-tâche a échoué, sa description est également mise à jour, permettant au gestionnaire de tâches de déduire l'échec de la tâche globale. Le but associé à la sous-tâche devient non satisfaisable ;

- dans le second cas, la description de l'environnement est mise à jour. Le gestionnaire de tâche pourra tenir compte de ce changement dans l'environnement pour vérifier si de nouvelles tâches peuvent être exécutées.

Traitement et gestion des buts

Un agent à un moment donné, gère plusieurs buts avec leurs sous-buts. Dans un contexte général, les contraintes temporelles entre les sous-tâches peuvent être fixées de façon à permettre que deux sous-tâches soient exécutées simultanément par deux agents différents. La fonction de décision utilise le délai associé à chacun des buts comme étant une priorité, la priorité d'un but sera donc inversement proportionnelle à son délai. Ainsi, les tâches relatives aux buts doivent être traitées par le gestionnaire de tâches

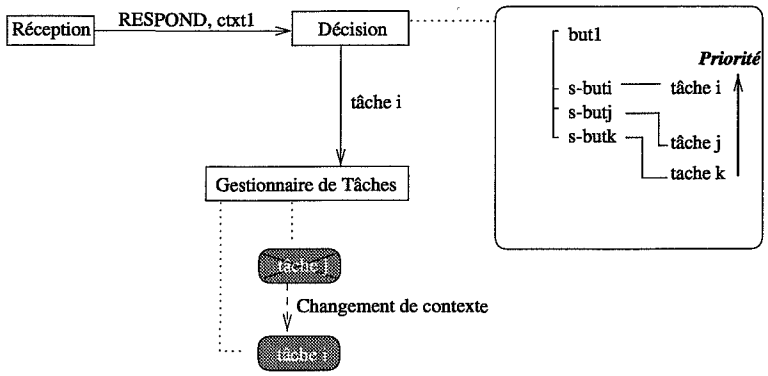


Figure 7.7 : Changement de contexte lors de la réception d'une information concernant un but plus prioritaire

en respectant cet ordre de priorité : parmi les tâches actives, la tâche admettant le plus petit délai sera traitée en priorité. Cependant, si une tâche t_i est en cours d'exécution par un autre agent a_v, un agent a_u peut traiter une tâche t_j moins prioritaire, sachant que dès qu'une réponse concernant la tâche t_i parvient de l'agent a_v, l'agent a_u doit interrompre la tâche t_j dans la mesure du possible et traiter cette réponse (figure 7.7).

Un but peut être satisfait, non satisfait mais satisfaisable ou non satisfaisable :

- la satisfaction d'un but global dépend de la satisfaction de l'ensemble des sous-buts et la satisfaction d'un but n'admettant pas de sous-buts, dépend du succès de l'exécution de la tâche correspondante. Le but d'un but satisfait dépend de son type :
  - si le but est interne ou dépendant alors il est supprimé tout simplement ;
  - s'il est externe alors l'agent concerné est notifié et ensuite il est supprimé. Dans ce cas, la fonction de décision ajoute une entrée à la liste :out-msg-list dans l'état mental. Cette entrée sera traitée par la fonction de communication (section 7.5.3) ;
- un but non satisfait mais qui est encore satisfaisable est gardé dans la liste des buts ;
- si un but est non satisfaisable, cela signifie que tous les buts de plus haut niveaux le sont également :
  - si le but est interne, il est supprimé et la tâche associée au but est interrompue et redevient inactive ;

- si le but est externe, il subit le même traitement qu'un but interne et en plus l'agent concerné est notifié ;
- enfin si le but est dépendant, des requêtes d'interruption de tâches doivent être envoyées aux agents devant exécuter les sous-tâches avant la suppression du but.

7.4.4 Planification

La fonction de planification est chargée de la création et de l'exécution de plans.

Création d'un plan

A la création d'un but g, la fonction de planification teste la tâche relative au but :

- si c'est une tâche élémentaire, alors le but ne doit être décomposé et aucun plan n'est créé. Si le but n'admet pas de type (cf. section 7.4.3) alors il est interne et le champ :type est mis à jour ;
- si la tâche admet des sous-tâches et que l'une d'entre elles n'appartient pas au domaine de responsabilité individuelle de l'agent, alors le but g est dépendant (mise à jour du champ :type) et un plan relatif au but est créé. Ce plan admet autant d'entrées que de tâches devant être exécutées par d'autres agents. A la création, les entrées d'un plan sont incomplètes : les délais sont mis à jour au fur et à mesure par le gestionnaire de tâches et le choix des agents pour exécuter les sous-tâches est déterminé à l'aide d'un raisonnement social (cf. section 7.5.5). Un sous-but est créé pour chacune des sous-tâches et est ajouté dans la liste des sous-buts :subgl-list du but g. Si la sous-tâche doit être exécutée par l'agent lui même, le but est interne, sinon il est dépendant. Un sous-but subit ensuite le même traitement que le but g.

Exécution d'un plan

La fonction de planification s'occupe également de l'exécution des plans : quand une entrée d'un plan est complété par le gestionnaire de tâches, cette entrée doit être exécutée et ensuite supprimée. Un plan est supprimé si toutes ses entrées le sont également.

L'exécution d'une entrée consiste à créer à partir des informations stockées dans l'entrée du plan, une entrée de type MSG-ENTRY et la rajouter à la liste :out-msg-list dans l'état mental. Cette entrée est ensuite traitée par la fonction de communication (section 7.5.3).

7.5 Niveau Social

Le niveau social est constitué des fonctions permettant à l'agent de communiquer avec les autres (reception et communication) et d'une fonction dédiée au raisonnement social. Ce fonction utilise les informations contenues dans l'état mental et dans les descriptions externes. Ces dernières étant mises à jour exclusivement par cette fonction, appartiennent également au niveau social.

7.5.1 Description externe

La représentation d'un autre agent est appelée *description externe* [DM91]. Dans notre modèle d'agent, une description externe correspond à une image incomplète (état mental) de l'agent concerné que l'agent maintient localement. Un agent admet une liste d'états mentaux relatifs à tous les agents connus. Ces états mentaux seront mis à jour au fur et à mesure des interactions et suite à un raisonnement social.

Les informations dans cette structure correspondent à des croyances sur les autres. Par exemple, si la description externe d'un agent a_u contient dans son réseau de dépendances $\mathcal{D}$ la relation $a_u \mathcal{D}^{T1}_{T2} a_v$, cette information admet la sémantique suivante pour l'agent (me) : BEL(self, $a_u \mathcal{D}^{T1}_{T2} a_v$).

Une description externe est définie comme suit :

<EXTERNAL-DESC> ::= '('<MENTAL-STATE>*')'

La mise à jour de cette structure est le résultat du raisonnement social (section 7.5.5). Dans notre étude, et pour valider le modèle de société décrit dans le chapitre 5, seuls sont mis à jour les domaines de responsabilité individuelle des agents et leurs réseaux de dépendance :

- les domaines de responsabilité individuelle sont connus dès le départ, l'agent utilisera les informations qui s'y trouvent pour construire ses trois réseaux de dépendances ($\mathcal{C}$, $\mathcal{D}$ et $\mathcal{H}$;
- les réseaux de dépendance des autres sont mis à jour, d'une part, par l'utilisation la commutativité des relations de dépendance et d'autre part, par l'utilisation d'un raisonnement social :
  - dans le premier cas, l'agent (self) utilise les résultats suivants :
    $\forall a_u \exists T \mid self\ \mathcal{C}_T a_u \Longleftrightarrow a_u \mathcal{C}_T\ self$
    $\forall a_u \exists T1, T2 \mid self\ \mathcal{H}^{T1}_{T2} a_u \Longleftrightarrow a_u \mathcal{H}^{T2}_{T1}\ self$
  - dans le second cas, les relations de dépendance des autres sont déterminées à l'aide de conjonctions de relations de dépendance de

l'agent (cf. section 5.6) ou alors suite à la réception d'un message de requête d'exécution d'une tâche (cf. section suivante).

7.5.2 Réception

Cette fonction est chargée de la réception des messages provenants des autres agents. Sa mission principale est de vérifier qu'un message respecte le format du langage $\mathcal{L}$ (cf. chapitre 6) et ensuite, traduire ce message en une structure du type MSG-ENTRY contenant toutes les informations nécessaires pour le traitement de son contenu. Les messages seront traités selon la date d'émission la plus ancienne (:msg-date) :

- si le message reçu est une requête, la fonction de réception transforme la structure relative à la requête en un *engagement social* (S-COMMIT) et l'insère dans la liste des engagements sociaux afin d'être traité par la fonction de décision ;
- si le message reçu est une réponse ou est destiné à informer l'agent d'un événement, la fonction de réception transforme la structure relative au message en une *croyance* (BELIEF) afin d'être traitée par la fonction de décision.

7.5.3 Communication

La fonction de communication est chargée de l'envoi des messages aux autres agents, le format des messages respecte la grammaire définissant le langage $\mathcal{L}$. L'envoi des messages est effectué suite à la création et l'ajout d'une entrée dans la liste :out-msg-list dans l'état mental. La fonction de communication transcode cette information vers une expression du langage $\mathcal{L}$:

(<date> <agent-id> <agent-id>)(<type>)(<contenu>)(<contexte>).

Le message est ensuite envoyé par la couche de communication vers l'agent destinataire. Rappelons que le champ <contenu> contient toujours une *variable temporelle* représentant un *délai* ou une *date*.

7.5.4 Gestion des contextes de conversation

Dans les sections précédentes, nous avons montré les règles d'utilisation des contextes de conversation, sans préciser comment ces contextes sont gérés au sein de l'agent. Chaque contexte est créé par un seul agent et sa création correspond à un début de conversation. La création est réalisée soit par la fonction de décision, soit par la fonction de planification. Dans le premier cas, l'agent désire communiquer une information à un autre agent, il lui envoie le contexte de conversation pour lui permettre de converser au

sujet de cette information. Dans le deuxième cas, l'agent a besoin d'envoyer une requête d'exécution de tâches ou une requête d'information à un autre agent, il lui envoie alors le contexte de conversation en sachant que l'autre utilisera ce même contexte dans sa réponse à la requête. Les contextes de conversation apparaissent à la fois dans les messages échangés et dans les entrées relatives aux buts localisées dans l'état mental de l'agent. Quand un agent reçoit un message avec un contexte de conversation, ce contexte sera utilisé à chaque création d'un but ou une communication provoquée par l'arrivée de ce message. La suppression d'un but de haut niveau entraîne la suppression du contexte correspondant dans l'état mental.

7.5.5 Raisonnement social

Un agent admet une fonction dédiée au raisonnement social décrit dans le chapitre 5. Toutes les informations déduite lors d'un raisonnement social servent à mettre à jour les réseaux de dépendances dans l'état mental et dans les descriptions externes (section 7.5.1).

Le raisonnement social permet également de compléter les entrées dans les plans : à la création, dans chaque entrée d'un plan, il faut désigner un agent qui sera chargé de l'exécution de la sous-tâche correspondante. S'il existe plusieurs agents capables d'exécuter la sous-tâche, la fonction dédiée au raisonnement social applique les résultats établis dans la section 5.5.

7.6 Discussion

Dans ce chapitre, nous avons décrit le modèle d'un agent adapté à une société dont les caractéristiques concordent avec celles présentées dans le chapitre 5. L'architecture est composée de plusieurs fonctions. Au sein de l'agent, nous distinguons deux niveaux : le niveau individuel et le niveau social. Les fonctions du niveau individuel sont chargées de la perception d'événements, la décision, la planification et enfin la gestion de tâches. Celles du niveau sociale sont chargées de la réception et l'envoi de messages, ainsi que du raisonnement social.

La description externe contient toutes les informations qui relèvent du niveau social. Ces informations vont être utilisées lors d'un raisonnement social qui s'appuient essentiellement sur la conjonction des différentes relations de dépendance (cf. section 5.6).

Un point essentiel, présent au niveau de toutes ces fonctions est la prise en compte du facteur dynamique lié au domaine d'application ou à l'organisation des agents.

Troisième partie

Système *STARS*

Introduction

Dans cette partie, nous présentons la supervision des systèmes dynamiques par reconnaissance de scénarios temporels, comme étant un domaine d'application bien adapté aux modèles de société et d'agent, proposés dans la partie précédente. Dans ce domaine, la prise en compte des contraintes temporelles qui régissent le système supervisé est essentielle :

- le premier chapitre, présente le modèle des scénarios, ainsi que leurs différents modes de reconnaissance. Une extension au modèle est proposé pour permettre le suivi de scénarios dans un contexte distribué ;
- le deuxième chapitre, présente une instantiation du modèle de société où les agents sont chargés de la supervision par reconnaissance des scénarios. L'accent sera mis sur le raisonnement social entrepris par les agents pendant le processus de supervision ;
- le dernier chapitre, présente une implémentation des modèles présentés dans la partie précédente et la présente partie : une société d'agents temporels pour la reconnaissance de scénarios (*Système STARS*).

Chapitre 8

Reconnaissance de scénarios temporels

8.1 Introduction

Dans la partie précédente, nous avons présenté un modèle de société d'agents pour effectuer des tâches où les contraintes temporelles jouent un rôle essentiel. Dans la première partie, nous avons vu que la supervision des systèmes dynamiques est un domaine d'application où la prise en compte du temps est cruciale. Un ensemble d'agents chargé de la supervision d'un système dynamique doit tenir compte des contraintes temporelles qui régissent le système en question. Pour cela, l'approche par reconnaissance de scénarios a été choisie car elle permet une représentation explicite des contraintes temporelles et permet de montrer les caractéristiques principales d'une société dont le modèle a été décrit dans la deuxième partie.

Dans ce chapitre, nous décrivons le modèle des scénarios temporels et leur construction. Nous décrivons également la reconnaissance de scénarios et ses différents modes.

Enfin, nous proposons une extension du modèle des scénarios en vue de les reconnaître dans un environnement distribué où un scénario peut admettre un ou plusieurs sous-scénarios. Un ensemble d'agents sera chargé du suivi et de la reconnaissance de ces sous-scénarios.

8.2 Modélisation par scénarios temporels

Le suivi d'un système industriel nécessite une modélisation de son comportement présentant les différents modes de fonctionnement possibles. Nous appelons *scénario*, une représentation d'un mode de fonctionnement particulier du système à surveiller. Le déroulement d'un scénario correspond au fonctionnement réel du système physique pendant une période de temps variable. Avant de décrire les différents modes de reconnaissance d'un scénario,

nous allons d'abord décrire sa structure et les différents aspects temporels qui le caractérisent.

8.2.1 Structure d'un scénario

La structure principale d'un scénario est un *graphe de contraintes temporelles* [DMP91] où un nœud correspond à une *variable d'instant* dont la valeur correspond à la date d'occurrence d'un *événement*. Un arc correspond à une *contrainte temporelle* pouvant exister entre deux variables d'instants. Un ensemble de *préconditions* permettent l'*activation* du scénario : un scénario est activé en vue d'être reconnu si les préconditions qui lui sont associées sont satisfaites. Ces préconditions peuvent être modélisées par un ensemble de prédicats. Un exemple de scénario temporel peut être illustré par la figure 8.1.

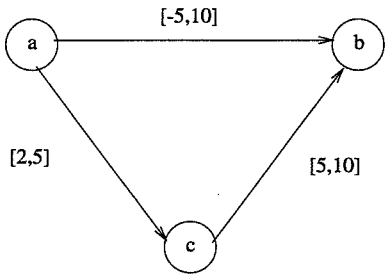


Figure 8.1 : Le graphe d'un scénario temporel

Un nœud du graphe correspond toujours à un événement instantané : si un événement n'est pas instantané, il devra être représenté par deux nœuds. Par exemple, si l'on s'intéresse uniquement à la position où une porte peut être considérée comme fermée, alors l'événement correspondant peut être représenté par un seul nœud. Il se produit quand la porte atteint cette position. Si par contre, on considère la fermeture comme une succession de deux événements où le premier (respectivement deuxième) correspond à la position où la porte est ouverte (respectivement fermée), cette situation peut être représentée par deux nœuds avec une contrainte temporelle entre les deux, précisant les durées maximale et minimale d'une fermeture de porte.

Une contrainte temporelle permet de préciser les écarts au plus tôt et au plus tard entre deux occurrences d'événements. Ceci peut être représenté par un intervalle. Par exemple, dans la figure 8.1, la contrainte précise que l'événement b doit se produire au minimum cinq unités de temps et au maximum dix unités de temps après l'occurrence de l'événement c. Si l'événement b se produit en dehors de cet intervalle, on dit que la contrainte est violée. Une contrainte contenant une valeur négative et une autre positive (figure 8.1), signifie que les événements peuvent se produire dans n'importe quel ordre :

l'événement b peut se produire cinq secondes avant a, en même temps ou deux secondes après. Le déroulement d'un scénario correspond à un chemin dans le graphe où l'activation d'un nœud signifie l'occurrence de l'événement correspondant. Il est évident que la présence de contraintes donne au scénario une durée variable. De plus, l'ordre des occurrences n'est pas figé. Ceci montre le pouvoir d'expression de ce formalisme notamment quand il s'agit de modéliser les comportements d'un système dynamique sujet à des évolutions en permanence.

A noter que, dans le cadre d'une modélisation par scénarios temporels, il n'est pas possible de représenter des processus continus, seuls des événements discrets peuvent être pris en compte.

8.2.2 Contraintes symboliques/numériques

Dans la section précédente, nous avons décrit les contraintes temporelles d'un graphe comme étant des contraintes numériques. Cependant, la modélisation d'un mode de fonctionnement n'est pas toujours évidente. En effet, lors de la modélisation, nous n'avons pas toujours des contraintes entre événements instantanés.

Evénements instantanés/non instantanés

Considérons l'événement O correspondant à l'ouverture d'une porte (la porte est en position "ouvert") et l'événement D correspondant au déplacement d'un chariot d'un point A à un point B. L'occurrence de D, signifie que le chariot est au point B. Dès lors que l'on veut poser une contrainte temporelle entre les deux événements comme par exemple, : *la porte doit être ouverte cinq à dix secondes avant le déplacement du chariot* (voir figure 8.2), une telle représentation devient problématique : nous n'avons aucune infor-

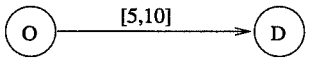


Figure 8.2 : Représentation par deux nœuds de deux événements non instantanés.

mation sur la position du chariot avant que la porte ne soit complètement ouverte. Tout ce que l'on sait, c'est que le chariot a atteint le point B, cinq à dix secondes après l'ouverture complète de la porte. Or cet événement n'est pas instantané. Pour pouvoir le positionner dans le temps par rapport à d'autres événements, nous avons besoin de représenter explicitement le début et la fin de cet événement par deux événements instantanés. Nous représentons ainsi, le déplacement du chariot par deux événements où le premier correspond au départ du point A ($D1$) et le deuxième à l'arrivée au point B ($D2$). La représentation donnée par la figure 8.3 décrit la même

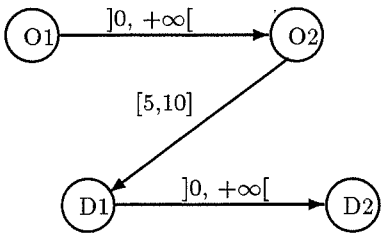


Figure 8.3 : Représentation par quatre nœuds de deux événements non instantanés.

contrainte temporelle où l'événement "ouverture d'une porte" est également représenté par deux événements instantanés (O1, O2). Des contraintes implicites sont également introduites précisant que le début d'un événement doit se produire avant sa fin ($[0, +\infty[$). En fait, ces contraintes pourraient être précisées si l'on détenait des informations sur la durée d'ouverture de la porte ou la durée que prend le déplacement du chariot, ce qui n'était pas possible dans la première représentation.

Contraintes d'intervalles/instants

L'exemple décrit ci-dessus montre comment représenter un type de contrainte entre deux événements non instantanés. En fait, il en existe autant que de relations possibles entre intervalles [All84] : en associant un intervalle à chaque événement non instantané, on peut contraindre ces événements de treize façons possibles.

Dans l'exemple, si on associe respectivement les deux intervalles $I = [O1, O2]$ et $J = [D1, D2]$ aux deux événements (ouverture de la porte et déplacement du chariot), la contrainte entre ces deux événements peut être exprimée par : I *avant*$[5,10]$ J. Ceci correspond à une des treize positions relatives entre deux intervalles définies par Allen, en augmentant cette relation par une contrainte numérique.

Afin de représenter ces relations au sein de graphes de contraintes décrits ci-dessus, il faut pouvoir transformer les contraintes entre intervalles en des contraintes entre instants tout en préservant le pouvoir d'expression des contraintes initiales.

Il existe seulement trois types de contrainte possibles entre deux instants i et j : $i = j$, $i < j$ ou alors $i > j$. Naturellement, il est possible d'exprimer des contraintes qui sont une composition des trois décrites ci-dessus, comme par exemple, $i <= j$. Par ailleurs, il est possible de vouloir représenter qu'il n'existe aucune contrainte temporelle entre deux variables d'instant. Dans ce cas, la contrainte est exprimée sous la forme : $i <=> j$, on dit qu'il existe une *contrainte universelle* entre ces deux variables. Ainsi, en assimilant les instants i et j à deux nœuds e_i et e_j, les trois contraintes de base, ainsi que

les contraintes composées peuvent être représentés numériquement comme le montre la figure 8.4.

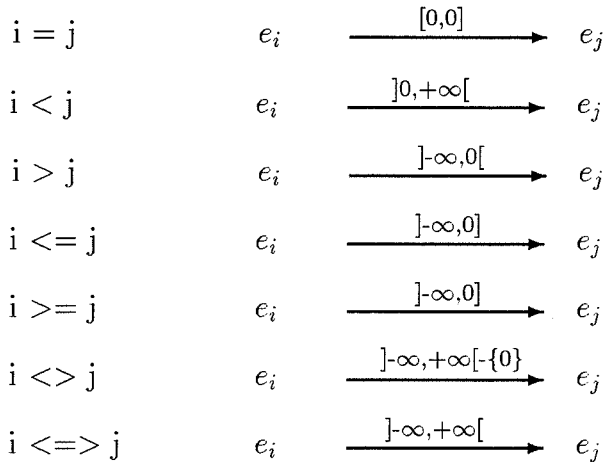


Figure 8.4 : Représentation numérique des contraintes symboliques entre instants.

La figure 8.5 permet de représenter les transformations de toutes les contraintes symboliques possibles entre intervalles en contraintes numériques entre instants. Les variables d'instant I^- et I^+ accèdent au début et à la fin d'un intervalle I.

8.2.3 Construction d'un scénario temporel

La construction d'un scénario temporel passe par les étapes suivantes :

- avoir un récit décrivant le fonctionnement devant être modélisé par le scénario. Ce récit est fourni généralement par un expert du domaine. Il n'existe pas de forme standard pour un récit : il peut correspondre à un texte ou à une liste de données ...
- extraire les événements à partir du récit ;
- extraire les contraintes temporelles entre les différents événements ;
- construire le graphe de contraintes du scénario à partir des événements et des contraintes temporelles.

Afin d'expliciter la structure d'un scénario et la manière dont il est construit pour représenter le fonctionnement d'un système, nous allons utiliser l'exemple du fonctionnement d'une machine dans un atelier de production. Celui-ci est modélisé par le scénario "`machine`" obtenu à partir du récit suivant recueilli auprès d'experts du domaine. Les contraintes temporelles qui vont donner lieu aux événements du scénario sont indexées par [n°] :

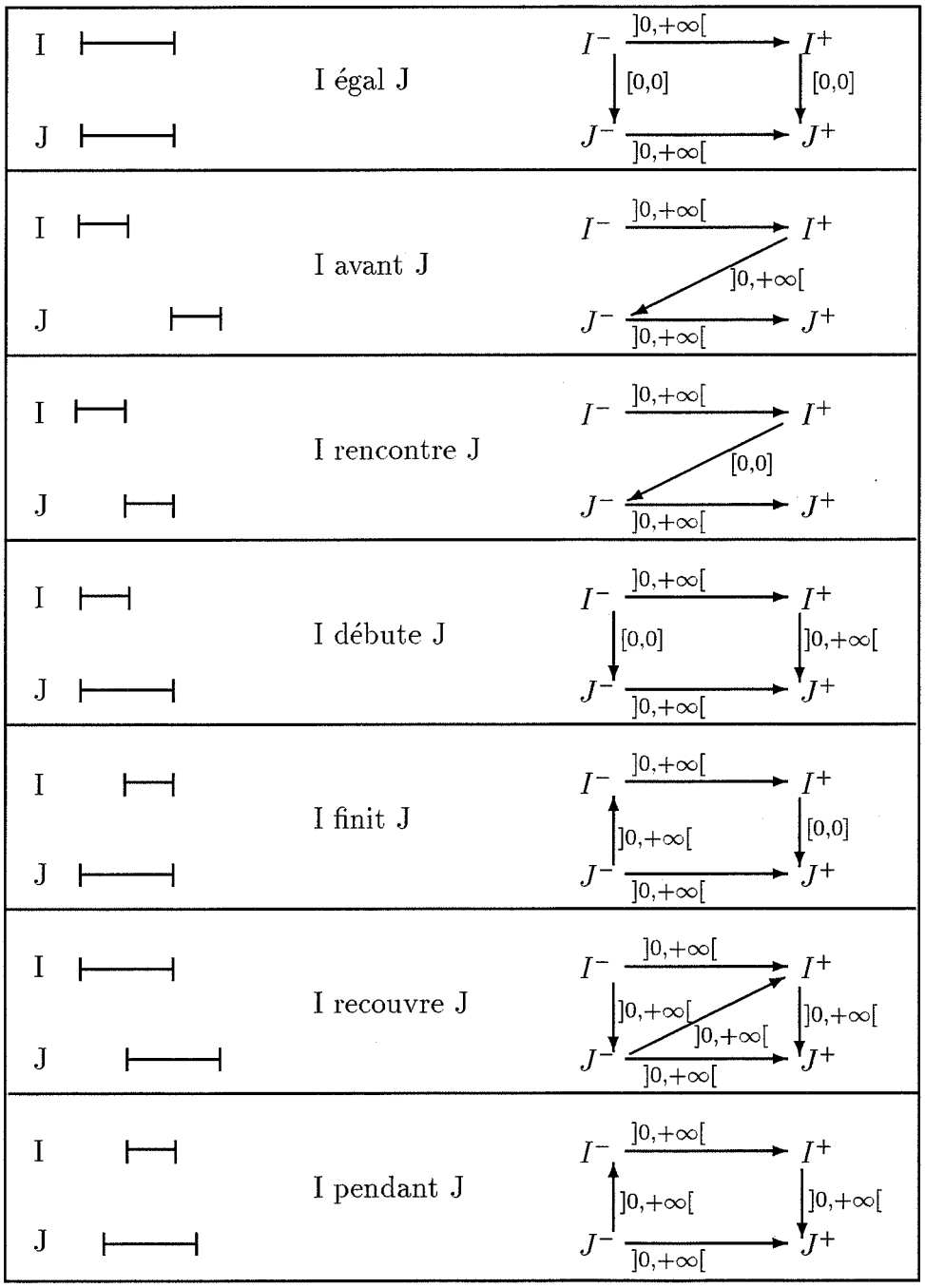


Figure 8.5 : Représentation des contraintes entre intervalles par des contraintes entre instants. En considérant ces relations et leurs inverses, il est possible de représenter toutes les positions relatives entre les deux intervalles I et J.

- " La machine est libre "
- " Puis une pièce se présente et la machine devient occupée " [1]
- " Le temps d'usinage normal d'une pièce est inférieur à 2mn " [2]
- " Après la fin de l'usinage la machine redevient libre " [3]
- " Pendant toute la durée de l'usinage le moteur de la machine doit fonctionner parfaitement " [4]

Les événements possibles sont représentés dans la table suivante (la variable t_i correspond à la date d'occurrence d'un événement) :

Evénement	**Intervalle**
machine.état = occupé	$I = [t_0, t_1]$
machine.fonction = usinage	$J = [t_2, t_3]$
machine.état-moteur = ok	$K = [t_4, t_5]$

Les contraintes temporelles enrichies par des délais que l'on peut exprimer sur ces événements sont les suivantes (début(I) et fin(I) accèdent respectivement aux bornes inférieure et supérieure de l'intervalle I) :

début(I)	{égal}	début(J)	[1]
début(J)	{]0, 120]avant}	fin(J)	[2]
fin(I)	{égal}	fin(J)	[3]
J	{pendant}	K	[4]

Nous en déduisons le graphe de contraintes du scénario exprimé, représenté dans la figure 8.6. Les contraintes implicites qui peuvent être déduites des autres contraintes sont représentées également.

8.3 Reconnaissance de scénarios temporels

L'objectif final d'une modélisation par scénarios du comportement d'un dispositif est le suivi et la supervision de ce dernier et ceci en utilisant un ensemble d'*algorithmes de reconnaissance*. Ces algorithmes sont basés sur la propagation des différentes contraintes temporelles dans un graphe.

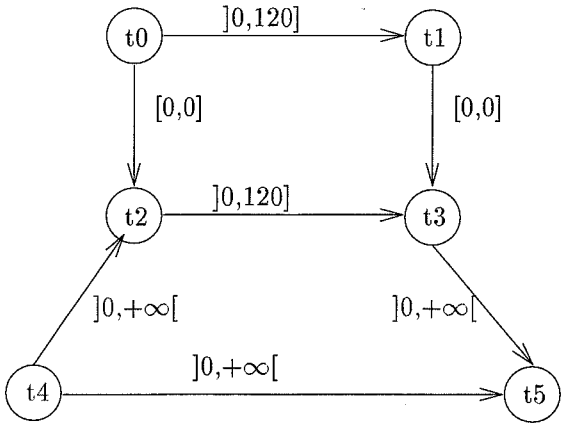


Figure 8.6 : Graphe du scénario "`machine`".

8.3.1 Détection d'incohérence

Il est à noter également, que lors de la construction d'un scénario, l'ajout de nouveaux nœuds et de nouvelles contraintes peut entraîner des *incohérences* dans le graphe du scénario.

Une incohérence a lieu quand les contraintes entre les noeuds sont posées de sorte qu'aucune instantiation possible des variables d'instant ne permet de satisfaire l'ensemble de ces contraintes. Afin de détecter une incohérence dans le graphe, un *algorithme de minimalisation* permet d'obtenir à partir d'un graphe initial, le graphe minimal équivalent exprimant les mêmes contraintes temporelles initiales. Ce graphe est unique et la présence d'une contrainte vide dans ce graphe, signale la présence d'une incohérence. Un exemple de graphe incohérent est donné dans la figure 8.7. L'incohérence

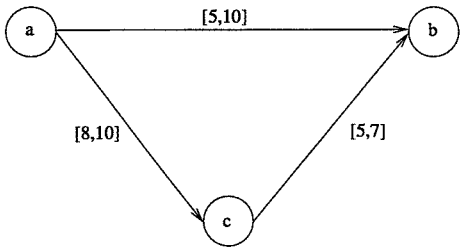


Figure 8.7 : Un graphe de contrainte incohérent.

provient du fait que le chemin minimum entre a et b prend 13 secondes en passant par c, alors que la contrainte entre a et b spécifie que b peut se produire au plus tard 10 secondes après a. Le graphe minimal d'un tel graphe contient une contrainte vide.

8.3.2 Suivi de scénario

La démarche générale à suivre pendant la supervision peut être illustrée par la figure 8.8. Le comportement observé du système correspond à une collection d'événements dans l'ordre de leurs occurrences. Ces événements sont datés si l'on dispose de capteurs capables de déterminer les dates d'observation. Dans le cas contraire, un intervalle d'incertitude lie deux événements, précisant l'écart au minimum et au maximum de leurs occurrences. Nous rappelons que dans les deux cas, cette collection d'événements est appelée la *session* et par la suite ce terme sera utilisé pour désigner un comportement observé. Si l'on souhaite qualifier un comportement observé, une comparaison de la session avec les *scénarios normatifs* modélisant un bon fonctionnement permet de détecter toute déviation d'un comportement de référence souhaité. Si un écart est détecté, un appariement avec les scénarios de mauvais fonctionnements permet d'identifier le type d'anomalie responsable de cet écart et de donner sous forme d'un diagnostic l'état de fonctionnement du système surveillé. A noter qu'il existe plusieurs *modes de reconnaissance* possibles. Par la suite, nous allons décrire ces différents modes et les différents types de reconnaissance associés. Cependant, nous ne donnons les détails des algorithmes sachant que vous en trouverez une description plus détaillée dans [Fon93] ou dans [Fon96].

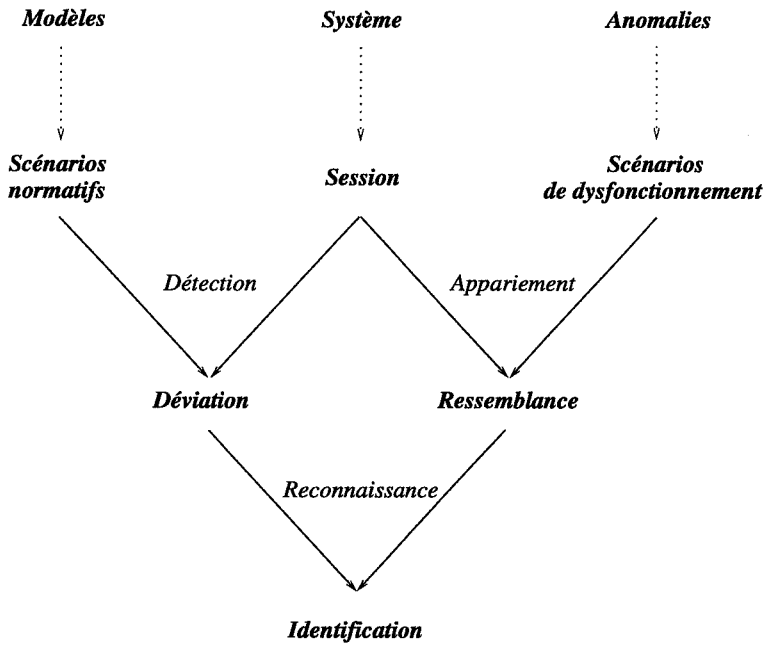


Figure 8.8 : La démarche générale suivie lors de la supervision [Fon93].

Mode *hors ligne*

Le mode *hors ligne* est utilisé en général quand la session est construite à partir d'informations données par un expert du domaine. Les événements sont liés par des contraintes d'intervalle et la session correspond à un graphe de contraintes similaire à celui du scénario qualifié comme étant un *comportement de référence.*

La supervision est effectuée à posteriori et a pour objectif de *qualifier* un comportement du système observé dans le passé. Cette qualification est établie en comparant le graphe du scénario avec celui de la session déduite des experts. Ces comparaisons vont donner lieu à plusieurs types de reconnaissance :

- la *compatibilité* d'une session avec un scénario est établie si la *fusion* de leurs graphes donnent un graphe *cohérent* ;
- la *satisfaction* d'un scénario par une session est établie si le graphe de la session est *inclus* dans celui du scénario ;
- une compatibilité ou une satisfaction est dite *totale* si la session comporte tous les événements du scénario. Dans le cas contraire, elle est *partielle* ;
- le *rejet* d'un scénario signifie que le comportement observé du système n'est pas celui modélisé par le scénario. Si le scénario rejeté modélise un bon fonctionnement, cela signifie soit que le système présente un bon fonctionnement différent de celui modélisé par le scénario ou alors il présente un mauvais fonctionnement et dans ce cas, il va falloir reconnaître un scénario de mauvais fonctionnement afin d'identifier l'anomalie présente. Si le scénario rejeté modélise un mauvais fonctionnement, cela signifie que le système présente une autre anomalie qui n'est pas modélisée par le scénario rejeté et dans ce cas, une reconnaissance d'un autre scénario de mauvais fonctionnement permet de résoudre le problème.

Pendant la supervision, les scénarios actifs ont trois status possibles : ils peuvent être rejetés, reconnus partiellement ou reconnus totalement. Dans un mode *hors ligne* une reconnaissance partielle peut être due à un manque d'informations sur les événements.

Mode *en ligne*

Contrairement au mode hors ligne, le mode *en ligne* ne se limite pas à expliquer tout simplement un phénomène qui s'est produit dans le passé, mais permet en plus de détecter les anomalies quand elles se présentent en temps courant. Ceci permet de réagir à temps afin de supprimer une

anomalie avant qu'elle n'occasionne d'importants dégâts et parfois afin de permettre l'anticipation d'autres anomalies.

Le mode en ligne consiste à suivre un scénario à partir d'événements ordonnés et datés. Quand le système de supervision reçoit un événement, sa date d'occurrence doit satisfaire la contrainte correspondante dans le graphe du scénario. Donc le mode en ligne utilise uniquement la satisfaction comme critère de reconnaissance. Quand le scénario est en cours de suivi, la session courante doit satisfaire partiellement le scénario (tous les événements ne sont pas encore arrivés). Quand tous les événements se sont produits et qu'aucun n'a violé une contrainte dans le graphe du scénario, on parlera d'une satisfaction totale.

Le mode en ligne est plutôt utilisé pour essayer de reconnaître des scénarios de bon fonctionnement, il permet de suivre le système, de s'assurer qu'il est dans un état (*bon*) à un moment donné et de détecter au plus tôt sa transition dans un état jugé mauvais (rejet du scénario). Le rejet d'un scénario de bon fonctionnement doit être accompagné de déclenchement de scénarios de mauvais fonctionnement afin d'identifier le ou les problèmes car les pannes peuvent être multiples.

8.3.3 Interprétation de la reconnaissance

Le raisonnement lors de la reconnaissance d'un scénario nous mène à poser deux questions: la première est de savoir si une reconnaissance partielle d'un scénario est suffisante ou non, la deuxième est de savoir comment tenir compte de cette reconnaissance lors d'un raisonnement temporel, surtout si le système de supervision gère des scénarios auxquels on peut appliquer plusieurs modes de supervision (hors ligne, en ligne, ou mixte).

Reconnaissance totale ou partielle ?

Concernant la première question, il suffit de définir un résultat exigé (satisfaction totale, satisfaction partielle, compatibilité totale, compatibilité partielle) pour chaque scénario lors de sa création. Si le résultat d'une reconnaissance ne correspond pas au résultat exigé, ce résultat sera jugé insuffisant. Mais dans un contexte réel est-ce toujours vrai ? En effet, nous pouvons rencontrer des situations complexes où les événements n'ont pas la même importance et leurs occurrences n'ont pas la même sémantique: une reconnaissance partielle signifie que les événements ne se sont pas produits en totalité, il faut savoir quels sont parmi ces événements ceux qui garantissent que cette reconnaissance partielle permet de qualifier le comportement observé. Ainsi, une reconnaissance partielle n'aura pas toujours la même signification, selon les événements qui se produisent. Afin de qualifier d'une manière plus précise un comportement observé conduisant à une reconnaissance partielle, il est possible d'avoir une couche d'analyse plus

fine de cette reconnaissance. Cette couche peut être construite autour d'un système à base de règles, qui, en fonction du scénario reconnu partiellement et les événements qui se sont produits permet d'évaluer la reconnaissance en question.

Raisonnement temporel

Concernant la deuxième question, il est plus difficile de donner une réponse car en effet, la reconnaissance d'un scénario signifie que le système s'est comporté selon le scénario depuis son activation et jusqu'à sa reconnaissance. Plus tard, nous n'aurons aucune garantie que cette reconnaissance soit encore valide : le système peut avoir un comportement différent de celui décrit par le scénario reconnu.

Pour résoudre ce problème, nous introduisons la notion d'*espace de validité* d'un scénario. Cet espace correspond à la période de temps pendant laquelle la reconnaissance d'un scénario reste valide. En effet, la reconnaissance d'un scénario peut devenir obsolète suite à la reconnaissance d'un autre scénario ayant une sémantique contradictoire. L'espace de validité d'un scénario peut avoir une durée constante, mais sa durée peut également dépendre de la satisfaction d'une condition particulière. Par exemple, dans un atelier de production, si un dispositif doit être démarré une fois tous les jours, l'espace de validité du scénario modélisant le démarrage peut avoir une durée constante d'une journée. Par contre, la reconnaissance d'un scénario $S1$ modélisant une baisse de puissance du moteur d'une machine invalide la reconnaissance antérieure d'un scénario S modélisant le régime normal du moteur en question. Ainsi, l'espace de validité du scénario S est conditionné par la reconnaissance du scénario $S1$, plus éventuellement celles d'autres scénarios modélisant des mauvais fonctionnements du moteur.

8.4 Adaptation des scénarios à un contexte distribué

La supervision par reconnaissance de scénarios est effectuée en général par un *système de suivi de scénarios* (*SSS*). Ce système est chargé de capter les événements provenant du système à surveiller et en fonction de ces événements, des scénarios seront activés pour être éventuellement reconnus par la suite. Une architecture simplifiée du système (figure 8.9) permet de voir que le flux d'information converge vers le *SSS*. Celui-ci est considéré comme une sortie du système dans la mesure où il offre les résultats de reconnaissance permettant ainsi aux opérateurs responsables de la surveillance de connaître l'état de fonctionnement du système physique.

Une telle architecture a l'avantage de ne pas être très coûteuse en terme d'écriture de logiciel, mais pose un certain nombre de problèmes liés es-

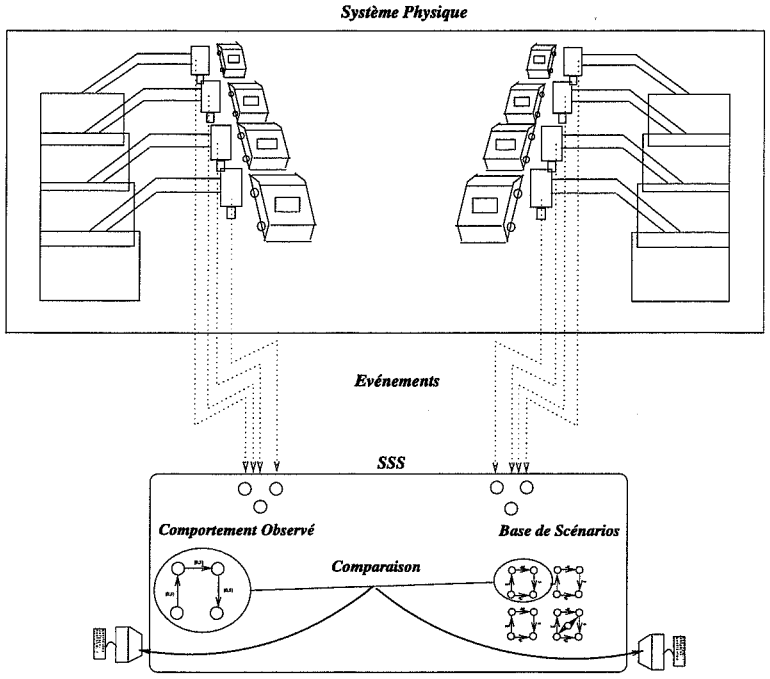


Figure 8.9 : Architecture simplifiée d'un système de supervision centralisé

sentiellement à la distribution physique du système surveillé. Celui-ci est souvent constitué de plusieurs *composants* ou *sous-systèmes* admettant des interactions entre eux afin de remplir une certaine tâche. La surveillance d'un tel dispositif par un système de supervision centralisé, exige que les informations sous forme d'événements soient transmises de chacun des composants vers le système superviseur. Cette démarche risque de poser des problèmes d'engorgement et de masquage (chapitre 4).

L'utilisation d'un système de supervision décentralisé contribue à la résolution de ces problèmes. En effet, en concevant un système de supervision formé de plusieurs unités responsables chacune de la supervision d'une partie du système (figure 8.10), il devient possible de faire des traitements parallèles. De cette manière, la charge de travail globale du système de supervision est scindée en différentes charges moins importantes réparties sur l'ensemble des unités. En distribuant les traitements, chaque unité aura besoin d'une partie des informations sur le système surveillé et cette partie concerne uniquement le secteur dont elle est responsable (cf. section 5.2.3) d'où la diminution du risque d'engorgement. La décentralisation des unités de supervision peut diminuer les problèmes de masquage. En effet, les informations sont transmises du composant vers l'unité responsable de sa surveillance qui peut être située sur le même site géographique. Ainsi, le problème de transmission des informations est plus simple que dans le cas

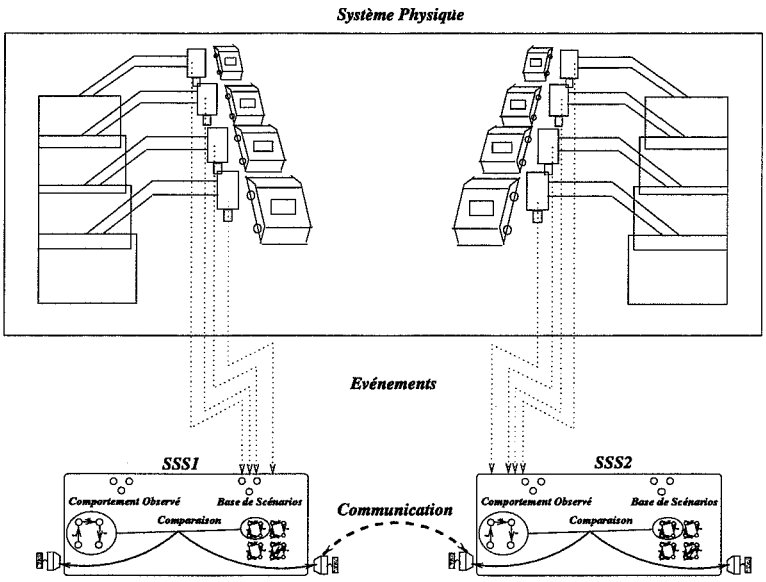


Figure 8.10 : Architecture simplifiée d'un système de supervision décentralisé

précédent. Dans le chapitre 9, nous présenterons plus en détail le processus de distribution et le critère d'attribution des responsabilités de chacune des unités. Dans ce chapitre, notre intérêt porte uniquement sur les changements devant être apportés à la structure des scénarios pour la rendre mieux adaptée à une architecture distribuée.

8.4.1 La relation scénario/sous-scénario

La structure du système physique étant constituée de plusieurs composants, ces derniers ne sont pas indépendants. Il existe des interactions entre eux, se situant à la fois au niveau structurel et fonctionnel. D'une part, un composant est formé de plusieurs sous-composants et d'autre part, un composant est chargé d'effectuer une tâche de haut niveau et un sous-composant effectue une sous-tâche de la tâche globale.

Pour modéliser les modes de fonctionnement d'un tel système, il est important et cela pour rendre la modélisation plus facile, qu'un scénario puisse admettre un ou plusieurs *sous-scénarios*. Il correspond ainsi au déroulement d'une tâche dans un certain mode de fonctionnement et les sous-scénarios correspondent aux déroulements des différentes sous-tâches. D'un point de vue de modélisation, la reconnaissance d'un sous-scénario est représentée comme étant un événement et correspond nœud dans le graphe de contraintes du scénario de haut niveau.

L'organisation des scénarios en sous-scénarios est plutôt adaptée à une

supervision en ligne. En effet, lors du suivi du scénario de haut niveau, quand une occurrence d'un événement correspondant à la reconnaissance d'un sous-scénario est attendue avec un certain délai, le sous-scénario doit être reconnu avant ce délai, sinon la contrainte est violée, ce qui cause le rejet du scénario. Donc, le sous-scénario doit être suivi en parallèle et c'est sa reconnaissance qui est contrainte dans le temps.

8.4.2 Modélisation par scénarios et sous-scénarios

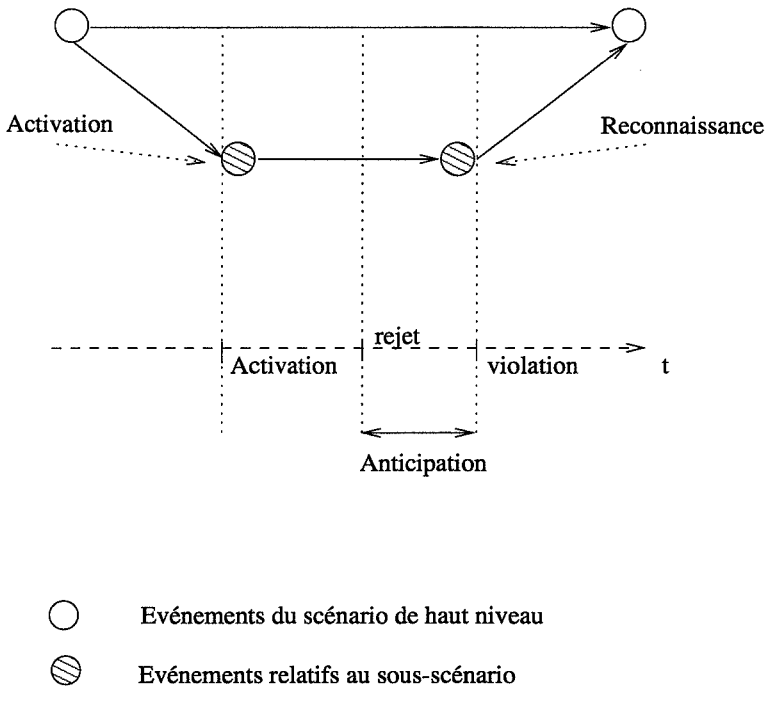


Figure 8.11 : Anticipation du rejet d'un scénario de haut niveau suite au rejet d'un sous-scénario.

Le déroulement d'un scénario prend un temps variable selon l'occurrence des événements et l'ordre dans lequel ils se produisent. Cette durée peut être représentée par un intervalle borné par deux variables d'instant. La borne inférieure correspond à l'activation du scénario et la borne supérieure à sa reconnaissance. D'un point de vue de modélisation, un sous-scénario est assimilé à deux événements instantanés représentés par deux nœuds dans le graphe de contraintes. L'occurrence du premier événement correspond à l'activation du sous-scénario et celle du dernier à sa reconnaissance. Naturellement, si le sous-scénario est rejeté, la contrainte relative au dernier événement ne sera pas respectée et ceci causera le rejet du scénario de plus haut niveau. Il est possible au moment du rejet du sous-scénario, d'anticiper

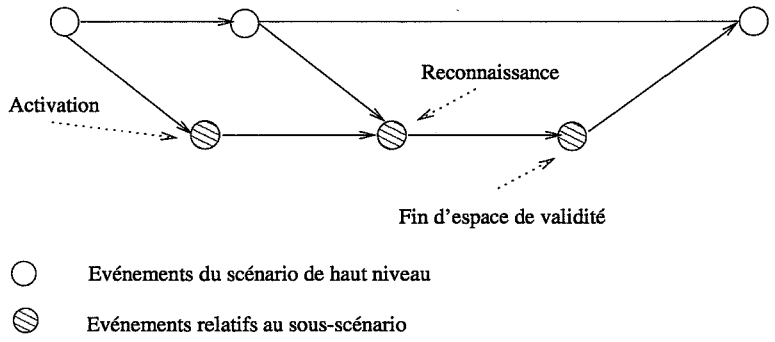


Figure 8.12 : Représentation par trois événements instantanés d'un sous-scénario.

et ne pas attendre que la contrainte soit violée pour savoir que le scénario de haut niveau est rejeté (figure 8.11).

Dans certains contextes, il est important de pouvoir s'assurer que la reconnaissance d'un scénario n'est pas invalidée pendant une certaine durée qui est fixée par une contrainte dans le graphe du scénario. Dans ce cas, nous avons besoin de trois événements instantanés pour représenter un sous-scénario. Le premier correspond à l'activation du scénario, le second correspond à sa reconnaissance et par conséquent au début de son espace de validité et le dernier correspondra à la fin de cet espace (figure 8.12).

Dans la section suivante, nous donnons un exemple de modélisation d'un sous-scénario du scénario "`machine`" décrit dans la section 8.2.3.

8.4.3 Construction d'un sous-scénario

Pour illustrer la notion de sous-scénario que vous venons d'introduire, reprenons le scénario "`machine`" décrit dans la section 8.2.3. Nous considérons que l'événement instantané *début* de l'événement abstrait "machine.état-moteur=ok" correspond à la reconnaissance d'un scénario de bon fonctionnement "`moteur`" [Fon96] qui est un sous-scénario du scénario "`machine`". L'événement *fin* correspond à la fin de l'espace de validité du scénario "`moteur`" après reconnaissance. Ce scénario est construit à partir du récit suivant par les experts :

- " Le moteur est démarré "
- " Puis le starter du moteur est en position "on" " [1].
- " La température de l'eau doit atteindre une valeur minimum de 40°C au plus 30s après le démarrage du moteur " [2].
  " Elle doit alors rester supérieure à 40°C au moins aussi longtemps que le moteur fonctionne " [3].

- " La température de l'huile doit atteindre une valeur minimale de 85°C au plus 35s après le démarrage du moteur " [4].
  " Elle doit alors rester supérieure à 85°C au moins aussi longtemps que le moteur fonctionne " [5].

- " La vitesse-moteur doit être redevenue comprise entre 1000t/mn et 1500t/mn au plus 50s après le démarrage du moteur " [6], " alors même que les températures d'eau et d'huile sont à leurs niveaux respectifs " [7,8].
  " De plus, le starter doit être en position "off" entre 10 et 30s avant que la vitesse-moteur ne soit redevenue comprise entre 1000t/mn et 1500t/mn " [9].
  " Enfin, la vitesse-moteur doit rester inférieure à 1500t/mn au moins aussi longtemps que le moteur fonctionne " [10].

Les événements sont représentés dans le tableau suivant :

Evénement	**Intervalle**
moteur.état=marche	$I' = [i_0, i_1]$
moteur.starter=on	$J' = [i_2, i_3]$
eau.température≥40	$K' = [i_4, i_5]$
huile.température≥85	$L' = [i_6, i_7]$
1000≤moteur.vitesse≤1500	$M' = [i_8, i_9]$

Les contraintes temporelles indexées par [n°] dans le récit sont exprimées par des contraintes mixtes d'instants et d'intervalles :

début(I')	{avant}	début(J')	[1]
début(K')	{[0,30]après}	début(I')	[2]
K'	{finit, est recouvert par}	I'	[3]
début(L')	{[0,35]après}	début(I')	[4]
L'	{finit, est recouvert par}	I'	[5]
début(M')	{[0,50]après}	début(I')	[6]
M'	{pendant}	K'	[7]
M'	{pendant}	L'	[8]
J'	{[10,30]avant}	M'	[9]
M'	{finit, est recouvert par}	I'	[10]

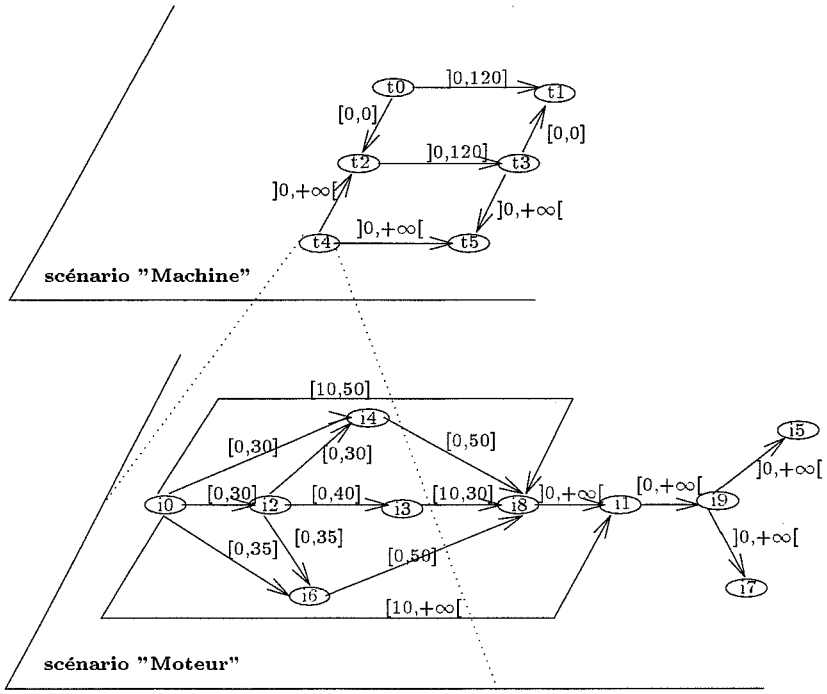


Figure 8.13 : Graphe de contraintes des scénarios "machine/moteur"

Le graphe de contraintes obtenu est représenté dans la figure 8.13. Comme pour le scénario "`machine`" certaines contraintes représentées sont des contraintes implicites. Dans ce graphe, il aurait été possible de représenter l'événement correspondant à l'activation du scénario "`moteur`" par un nœud dans le graphe, mais ce nœud aurait été lié par une contrainte universelle à tous les autres nœuds, ce qui n'apporte aucune richesse d'expression supplémentaire. Pour cette raison, nous l'avons supprimer.

8.5 Discussion

Dans ce chapitre, nous avons présenté les aspects essentiels de la modélisation des modes de fonctionnement d'un système industriel par un ensemble de scénarios. Ce formalisme est bien adapté à des systèmes dynamiques présentant des changements d'état au cours du temps.

La distribution de la supervision de façon à avoir plusieurs unités de supervision où chacune est chargée du suivi d'un sous-ensemble des scénarios, permet d'éviter les problèmes d'engorgement et les problèmes de masquage.

La nouvelle structure des scénarios permet un parallélisme au niveau du suivi d'un scénario et d'un sous-scénario. En effet, dans notre exemple, dans

le scénario "`machine`", quand une pièce à usiner se présente devant la machine, avant de commencer une phase d'usinage, nous devons s'assurer que le moteur fonctionne parfaitement et ceci correspond à la reconnaissance du scénario "`moteur`" représenté par le nœud 4 dans le graphe du scénario "`machine`". Donc cela signifie que si le scénario "`machine`" est suivi par une certaine unité, le scénario "`moteur`" doit être suivi parallèlement par une autre unité. Ce parallélisme permet d'augmenter les performances globales du système de supervision face à la complexité algorithmique de la reconnaissance des scénarios ($O(n^3)$) : la structuration d'un scénario en plusieurs sous-scénarios permet de diminuer le nombre de nœuds dans son graphe (n) lors de la propagation des contraintes.

Le chapitre suivant concerne l'instantiation d'une société d'agents dans laquelle les agents sont chargés de la supervision par reconnaissance de scénarios, d'un système dynamique et distribué.

Chapitre 9

Application à la supervision

9.1 Introduction

L'objet du présent chapitre est une instantiation du modèle de société décrit dans la deuxième partie de ce manuscrit à la supervision par reconnaissance de scénarios (cf. chapitre 8).

Cette instantiation nous permettra d'illustrer toutes les propriétés importantes décrites dans le modèle de la société : notamment l'utilisation des relations de dépendance par les agents et leur raisonnement temporel sur ces relations.

Tout d'abord, nous décrivons le système à superviser. A travers cet exemple, nous définissons l'ensemble des ressources et des tâches à effectuer par l'ensemble des agents. Par la suite, nous définissons les relations de dépendance qui résultent de la définition des domaines de responsabilité des agents. Différents cas de fonctionnement du système seront choisis afin de décrire l'évolution des différentes relations de dépendance. Ces cas permettront également de montrer comment les agents vont coopérer pour mener à bien la supervision d'un tel système.

9.2 Définition de l'exemple

L'exemple que nous avons choisi, consiste à superviser une flotte de bus dans le cadre du transport en commun. Les bus ont chacun une ligne à parcourir et ils doivent desservir les différents arrêts qui se trouvent sur cette ligne. Afin de simplifier l'exemple, nous ne considérons que trois bus qui doivent desservir une ligne. Afin de superviser cette ligne et les bus qui s'y trouvent, nous distinguons deux niveaux de scénarios :

- les scénarios du niveau *bus* servent à suivre le fonctionnement d'un bus indépendemment de la ligne et des autres bus qui s'y trouvent ;
- les scénarios du niveau *ligne* servent à suivre le fonctionnement sur

une ligne. Ils doivent tenir compte de la position de chacun des bus sur la ligne, ainsi que de leur vitesse relative.

Afin de superviser le comportement de chacun des bus, les scénarios des niveaux *bus* et *ligne* doivent être suivis.

9.2.1 Scénarios du niveau bus

A ce niveau, nous distinguons six scénarios :

- le scénario *Moteur* décrit le bon fonctionnement du moteur depuis son démarrage jusqu'à son arrêt ;
- le scénario *Marche* permet de détecter que le bus est en état de marche et qu'il respecte la vitesse limite autorisée ;
- les scénarios *Chargement* et *Déchargement* décrivent un arrêt pour, respectivement, embarquer ou déposer des passagers ;
- les scénarios *Ouverture* et *Fermeture* permettent de détecter un dysfonctionnement d'ouverture et de fermeture des portes du bus.

Outre les scénarios *Moteur* et *Marche*, les autres scénarios admettent un espace de validité nul;

Moteur

Le scénario *Moteur* est celui décrit dans la section 8.4.3. Ce scénario sera un sous-scénario des scénarios *Marche*, *Chargement* et *Déchargement.*

Marche

La vitesse du bus doit être comprise entre 0 et V_{max} et ceci pendant au moins 5 secondes. Tant que la vitesse est comprise entre 0 et V_{max}, le moteur du bus fonctionne parfaitement.

A partir de cette description, nous pouvons extraire les événements suivants :

Evénement	Intervalle
$0 <$ *bus.vitesse*$\leq V_{max}$	$[t_0, t_1]$
bus.état-moteur $=$ *ok*	$[t_2, t_3]$

Pour un événement non instantané associé à l'intervalle $[t_i, t_j]$, nous désignons par i et j les deux événements instantanés correspondants.

Les contraintes temporelles sont les suivantes :

t_1	{[5, +∞]après}	t_0	[1]
$[t_0, t_1]$	{pendant}	$[t_2, t_3]$	[2]

L'ensemble des événements ainsi que les contraintes temporelles exprimées entre eux permettent de déduire le graphe de contraintes de la figure 9.1.

La condition d'activation du scénario est l'observation de l'événement 0 (la vitesse du bus passe de la valeur nulle à une valeur positive). Le nœud correspondant dans le graphe sera noté dans un graphisme différent.

L'espace de validité du scénario s'arrête, si le moteur présente un dysfonctionnement (événement 3 : fin de l'espace de validité du scénario *Moteur*), si le bus est à l'arrêt (événement 1), ou si sa vitesse dépasse V_{max} (événement 1).

A noter que le scénario *Marche* admet le scénario *Moteur* comme sous-scénario. En effet, l'événement 2 correspond à une reconnaissance du scénario *Moteur* et l'événement 3 correspond à la fin de son espace de validité.

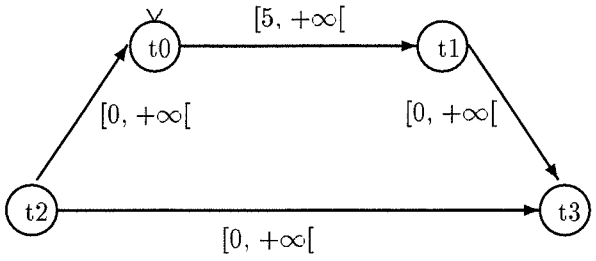


Figure 9.1 : Graphe du scénario *Marche*. L'observation de l'événement 0 permet l'activation du scénario.

Chargement

Le chargement consiste en un arrêt du bus, qui doit durer au moins une minute et au plus cinq minutes. Pendant ce temps, il y aura une ouverture suivie d'une fermeture des portes. Pendant l'arrêt, le moteur continue à fonctionner parfaitement.

Les événements qui interviennent dans ce scénario sont les suivants :

Evénement	**Intervalle**
bus.vitesse = 0	$[t_6, t_7]$
bus.état-moteur = ok	$[t_2, t_3]$
porte.état = ouvert	$[t_4, t_5]$

Les contraintes temporelles entre ces événements sont données dans la table suivante :

t_7	{[60, 300]après}	t_6	[1]
$[t_6, t_7]$	{pendant}	$[t_2, t_3]$	[2]
$[t_4, t_5]$	{pendant}	$[t_6, t_7]$	[3]

Le graphe de contraintes correspondant est représenté dans la figure 9.2.

La condition d'activation correspond à l'occurrence de l'événement 6 (la vitesse du bus passe d'une valeur positive à la valeur nulle).

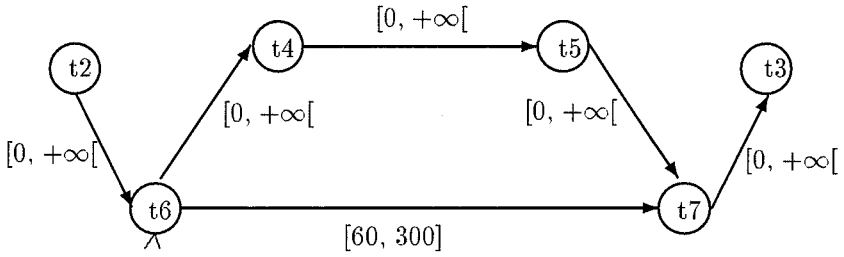


Figure 9.2 : Graphe du scénario *Chargement*. L'observation de l'événement 6 permet l'activation du scénario.

Déchargement

Le scénario *Déchargement* est similaire au scénario *Chargement* sauf qu'une demande d'arrêt doit être effectuée auparavant par l'un des passagers à l'intérieur du bus. Cette demande cesse d'être prise en compte à partir du moment où le bus est à l'arrêt.

Nous retrouvons les mêmes événements et les mêmes contraintes que le scénario précédent et en plus nous avons l'événement suivant :

Evénement	**Intervalle**
bus.arrêt = demandé	$[t_8, t_9]$

Cet événement est lié aux autres événements par la contrainte suivante :

$[t_8, t_9]$	{rencontre}	$[t_6, t_7]$	[1]

Le graphe du scénario *Déchargement* est exprimé par la figure 9.3.

La condition d'activation du scénario correspond à l'occurrence de l'événement 8 (activation du bouton de demande d'arrêt).

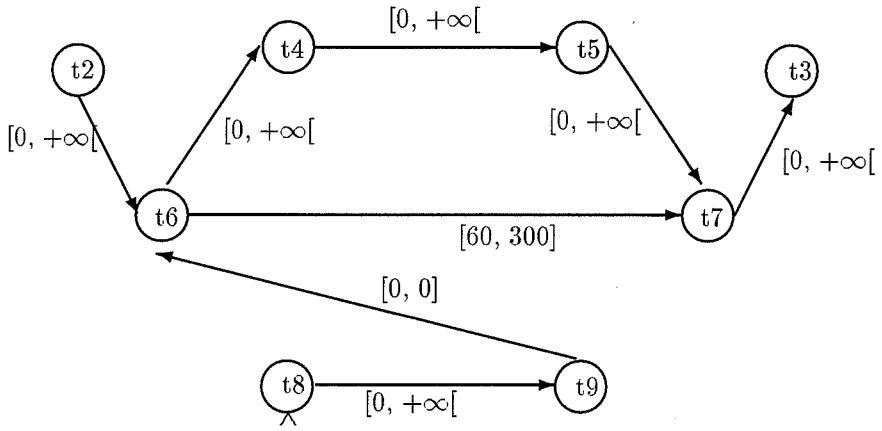


Figure 9.3 : Graphe du scénario *Déchargement*. L'observation de l'événement 8 active le scénario.

Ouverture/Fermeture

Après l'activation de la commande d'ouverture (respectivement de fermeture), une porte doit s'ouvrir (respectivement se fermer) au maximum dix secondes plus tard.

Les scénarios *Ouverture* et *Fermeture* décrivent respectivement l'ouverture et la fermeture d'une porte.

Les événements qui interviennent dans ces deux scénarios sont les suivants :

Evénement	**Intervalle**
porte.état = ouvert	$[t_4, t_5]$
porte.état = fermé	$[t_{10}, t_{11}]$
porte.ouverture = demandé	$[t_{12}, t_{13}]$
porte.fermeture = demandé	$[t_{14}, t_{15}]$

Les contraintes temporelles entre les différents événements sont les suivantes :

$[t_{12}, t_{13}]$	{finit}	$[t_{10}, t_{11}]$	[1]
$[t_{14}, t_{15}]$	{finit}	$[t_4, t_5]$	[5]

Les deux graphes des scénarios *Ouverture* et *Fermeture* sont représentés respectivement par les figures 9.4 et 9.5.

La condition d'activation du scénario *Ouverture* est l'occurrence de l'événement 12 (activation de la commande d'ouverture). Celle du scénario *Fer-*

meture est l'occurrence de l'événement 14 (activation de la commande de fermeture).

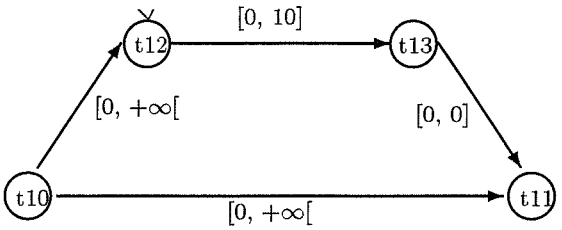


Figure 9.4 : Graphe du scénario *Ouverture*. L'observation de l'événement 12 sert à activer le scénario.

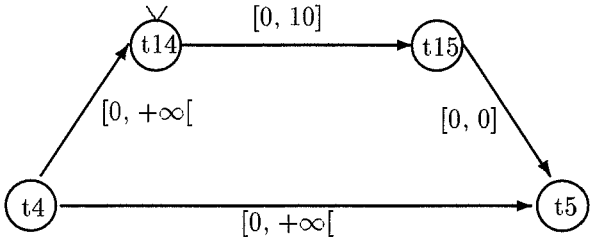


Figure 9.5 : Graphe du scénario *Fermeture*. L'observation de l'événement 14 sert à activer le scénario.

9.2.2 Scénarios du niveau ligne

La ligne que nous proposons de superviser est divisée en trois tronçons : AB, BC et CD. Les points A et D constituent les terminus de la ligne. Au niveau d'une ligne, nous distinguons seulement quatre scénarios. Les scénarios *Terminus A* et *Terminus D*, décrivent les règles de fonctionnement des terminus A et D. Les scénarios *Aller* et *Retour* décrivent le parcours d'un bus d'un terminus à un autre dans un sens ou dans l'autre.

Tous les scénarios du *niveau ligne* admettent des espaces de validité nuls.

Terminus A/D

Un bus à l'arrêt dans un des deux terminus, ne peut pas quitter le terminus avant qu'un autre bus vienne d'entrer ou sortir du dernier tronçon de la ligne. Ce bus est tenu de quitter le terminus au maximum 30 secondes après qu'un autre bus ne soit entrer ou sorti du dernier tronçon. Par ailleurs, l'attente d'un bus à un terminus ne peut dépasser dix minutes. Par exemple, si un bus est au terminus D de la ligne, il doit quitter sa position au plus

tard 30 secondes après le passage d'un autre bus par le point C, dans un sens ou dans l'autre.

Les scénarios *Terminus A* et *Terminus D* sont similaires. Nous en décrivons donc un seul, *Terminus D*.

Deux événements interviennent dans ce scénario. Le premier indique qu'un bus entre ou quitte le tronçon CD, le deuxième indique que le bus est à l'arrêt au terminus D.

Evénement	**Intervalle**
ligne.tronçon = CD	$[t_{16}, t_{17}]$
ligne.terminus = D	$[t_{20}, t_{21}]$

Les contraintes temporelles entre ses événements sont les suivantes :

t_{16}	{[0, 30]avant}	t_{21}	[1]
t_{21}	{[0, 600]après}	t_{20}	[2]
t_{16}	{après}	t_{20}	[3]

Le graphe relatif au scénario est représenté par la figure 9.6.

La condition d'activation du scénario correspond à l'occurrence de l'événement 20.

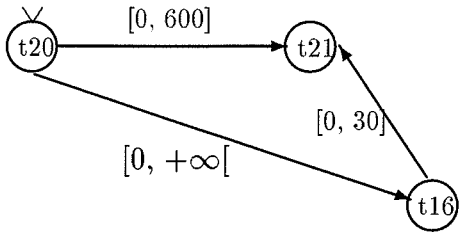


Figure 9.6 : Graphe du scénario *Terminus D*. L'observation de l'événement 20 sert à activer le scénario.

Aller/Retour

Pour un bus donné, un aller simple consiste à partir d'un terminus pour arriver à un autre terminus. Le parcours d'un tronçon doit durer entre cinq et quinze minutes.

Les événements et les contraintes temporelles pour un aller simple du terminus A au terminus D, sont respectivement décrites dans les deux tables suivantes :

Evénement	**Intervalle**
ligne.tronçon = CD	$[t_{16}, t_{17}]$
ligne.tronçon = AB	$[t_{18}, t_{19}]$
ligne.tronçon = BC	$[t_{24}, t_{25}]$
ligne.terminus A = ok	$[t_{26}, t_{27}]$
ligne.terminus D = ok	$[t_{28}, t_{29}]$

t_{27}	{égal}	t_{18}	[1]
t_{19}	{[300, 900]après}	t_{18}	[2]
t_{25}	{[300, 900]après}	t_{24}	[3]
t_{17}	{[300, 900]après}	t_{16}	[4]
$[t_{16}, t_{17}]$	{rencontre}	$[t_{28}, t_{29}]$	[5]
$[t_{18}, t_{19}]$	{rencontre}	$[t_{24}, t_{25}]$	[6]
$[t_{24}, t_{25}]$	{rencontre}	$[t_{16}, t_{17}]$	[7]

Le graphe du scénario *Aller* est représenté par la figure 9.7. A noter que le scénario *Aller* admet les scénarios *Terminus A* et *Terminus D* comme deux sous-scénarios. Les événements 26 et 28 correspondent à l'activation de ces deux sous-scénarios et les événements 27 et 29 correspondent à leurs reconnaissance.

La condition d'activation du scénario *Aller* correspond à l'occurrence de l'événement 27.

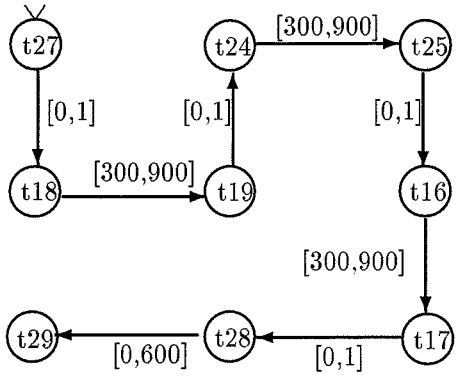


Figure 9.7 : Graphe du scénario *Aller*. L'observation de l'événement 26 sert à activer le scénario.

9.3 Définition des ressources et des tâches

Les scénarios définis ci-dessus constituent l'ensemble de ressources pour un ensemble d'agents responsables de la supervision des trois bus, ainsi que de la ligne. L'ensemble des ressources relatives à un bus b_i est R_i={*Moteur, Chargement, Déchargement, Ouverture, Fermeture, Terminus A, Terminus B, Aller, Retour* }. L'ensemble total de ressources R est défini par: $R = R_1 \cup R_2 \cup R_3$.

A l'ensemble des scénarios correspond une seule tâche: reconnaître un scénario.

Il existe entre les différentes tâches de reconnaissance deux types de relation : la relation tâche/sous-tâche et la relation d'opposition.

- La décomposition d'une tâche en sous-tâches est la même que celle de la décomposition du scénario correspondant en sous-scénarios. En effet, la reconnaissance d'un scénario nécessite la reconnaissance de tous ses sous-scénarios, en respectant les contraintes temporelles décrites par le graphe du scénario. Ces contraintes sont les mêmes que celles existant entre la tâche et ses sous-tâches ;

- La relation d'opposition entre deux scénarios spécifie que l'un ne peut être suivi si la reconnaissance de l'autre est encore valide. La figure 9.8 permet de rendre compte des relations d'opposition et des relations scénario/sous-scénario entre les différents scénarios. Par abus de langage, nous parlerons indifféremment de tâches de reconnaissance de scénarios et de scénarios.

9.4 Relations de dépendances

L'affectation des domaines de responsabilité aux agents leur permet de déterminer leurs relations de dépendance (cf. section 5.3) en tenant compte de la relation scénario/sous-scénario et de la relation d'opposition (figure 9.8). Nous considérons dans notre exemple, six agents responsables du suivi de ces scénarios pour un bus donné:

$$A = \{a_1,\ a_2,\ a_3,\ a_4,\ a_5,\ a_6\}$$

Les domaines de responsabilité sont définis comme suit :
$r(a_1) = \{Aller,\ Retour,\ Terminus\ A\}$
$r(a_2) = \{Aller,\ Retour,\ Terminus\ D\}$
$r(a_3) = \{Aller,\ Retour,\ Marche\}$
$r(a_4) = \{Chargement,\ Déchagement\}$
$r(a_5) = \{Ouverture,\ Fermeture,\ Moteur\}$
$r(a_6) = \{Moteur\}$

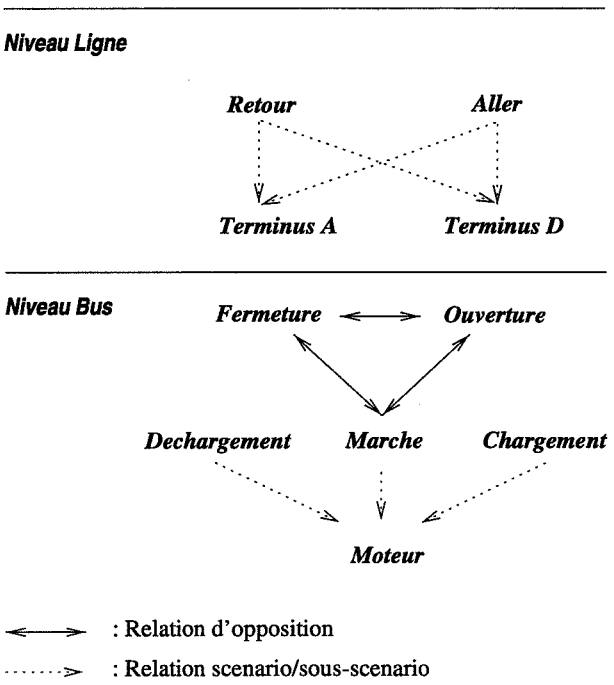


Figure 9.8 : Relations d'opposition et scénario/sous-scénario.

Les réseaux de dépendances (types $\mathcal{C}$, $\mathcal{D}$ et $\mathcal{H}$) construits par les agents sont donnés par la figure 9.9. Nous avons choisi de représenter les réseaux globaux pour éviter de surcharger le chapitre par dix huit figures. Cependant, chacun des agents aura uniquement une partie de ces réseaux dans son état mental. Cette partie concerne uniquement ses dépendances avec les autres, elle est formée du nœud relatif à l'agent, ainsi que tous les nœuds adjacents et les liens entre ces différents nœuds. Les agents qui ne figurent pas dans un des réseaux admettent un réseau vide du même type. Par exemple, l'agent a_5 n'a pas de dépendances en terme de besoin avec aucun des agents. Il est capable de suivre les scénarios *Ouverture*, *Fermeture* et *Moteur* indépendemment des autres et dans ce cas, son réseau $\mathcal{D}$ est vide.

9.5 Relations de pouvoir

Les relations de pouvoir existent entre les trois agents a_1, a_2 et a_3 concernant la reconnaissance des scénarios *Aller* et *Retour* et entre les agents a_5 et a_6 concernant la reconnaissance du scénario *Moteur*.

Une relation de pouvoir entre deux agents est déterminée en fonction de la position du bus sur la ligne à un moment donné. La figure 9.10 permet d'illustrer la relation de pouvoir vis-à-vis des scénarios *Aller/Retour* et

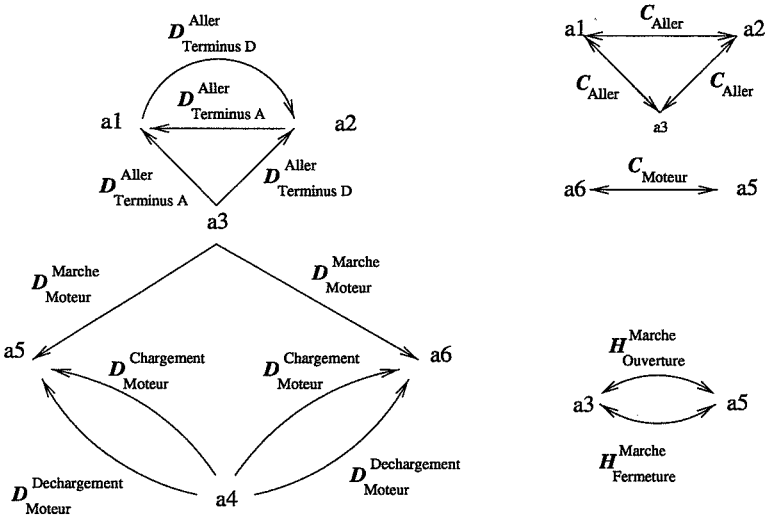


Figure 9.9 : Réseaux globaux de dépendances $\mathcal{C}$, $\mathcal{D}$ et $\mathcal{H}$.

Moteur.

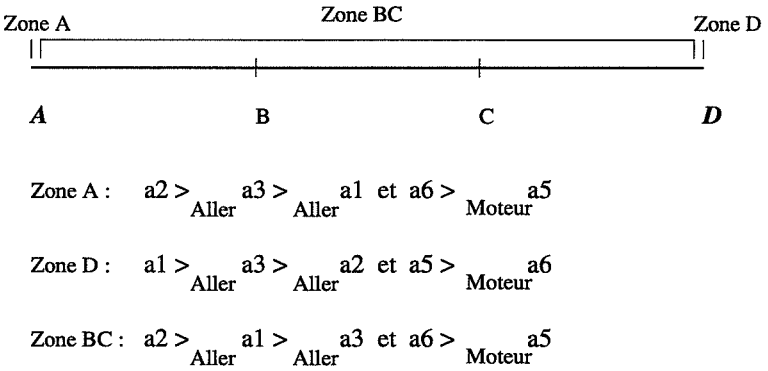


Figure 9.10 : La position du bus qui effectue un aller permet de déterminer les relations de pouvoir entre les agents a_1, a_2 et a_3 vis-à-vis du scénario *Aller* et entre les agents a_5 et a_6 vis-à-vis du scénario *Moteur*.

Par exemple, si le bus se trouve dans la zone BC au moment où on a besoin de suivre le scénario *Aller*, l'agent a_3 ayant le moins de pouvoir, suivra le scénario.

9.6 Raisonnement social

Lors du suivi des différents scénarios, les agents doivent se servir de leurs relations de dépendance avec les autres afin de reconnaître un scénario. Dans

les sections suivantes seront testées différentes situations permettant chacune de mettre en avant un type de dépendance entre deux agents. Ces derniers seront amenés à raisonner sur leurs relations de dépendance et de pouvoir afin d'en suivre les évolutions.

9.6.1 Concurrence et pouvoir

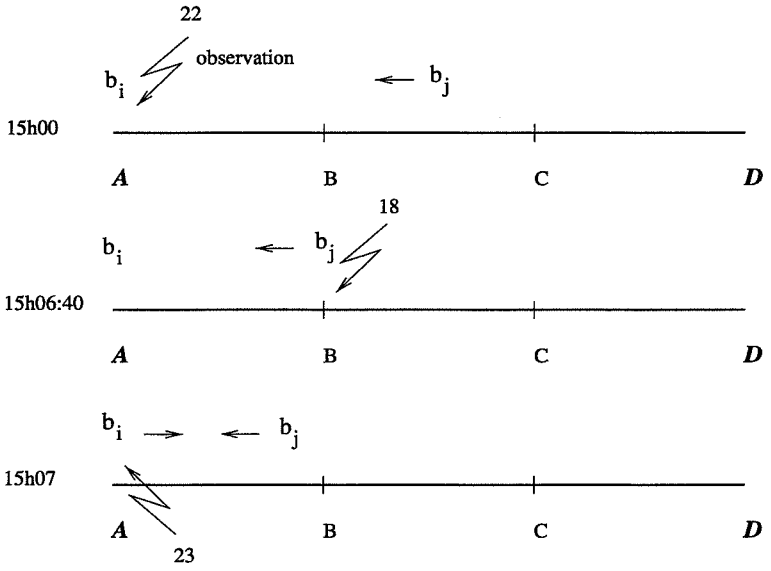


Figure 9.11 : Activation du scénario *Aller* suite à la reconnaissance du scénario *Terminus*.

Nous considérons la situation suivante (figure 9.11) : un bus b_i est à l'arrêt au terminus A depuis 15h00. Un autre bus b_j entre dans le tronçon AB à 15h06:40. Le bus b_i quitte alors le terminus à 15h07 entraînant alors la reconnaissance du scénario *Terminus A* par l'agent a_1.

Cette reconnaissance permet l'activation du scénario *Aller* correspondant (observation de l'événement 27 de la figure 9.7). L'existence des deux relations $a_1 \mathcal{C}_{Aller} a_2$ et $a_1 \mathcal{C}_{Aller} a_3$, pousse l'agent a_1 à informer a_2 et a_3 de cette reconnaissance en leur envoyant les messages suivants :

(d a_1 a_2)(INFORM)(PERFORMED *Terminus A* 15:00:00 15:07:00)(ctxt1)

(d a_1 a_3)(INFORM)(PERFORMED *Terminus A* 15:00:00 15:07:00)(ctxt2)

Ceci aura pour effet, l'activation également du scénario *Aller* chez les agents a_2 et a_3. A noter, qu'un message contient toujours la date d'activation du scénario et la date de sa reconnaissance ou de son rejet.

Pouvoir

La relation de pouvoir illustrée par la figure 9.10 permet de déterminer parmi les trois agents, quel est celui qui doit suivre le scénario *Aller*. En effet, le bus venant de quitter le terminus A, est désormais dans la zone BC. D'après la relation de pouvoir relative à cette zone, l'agent a_3 a le moins de pouvoir, il se chargera donc du suivi du scénario (cf. section 9.5).

Dépendances actives

La figure 9.12 permet d'illustrer les relations actives dans les différents réseaux de dépendances. La règle d'activation est donnée dans la section 5.3.1 :

- Agent a_1 :

$$a_1\mathcal{C}_{Aller}(I)a_3 \Longleftrightarrow a_1\mathcal{C}_{t_i}a_3 \wedge (a_1 : [Aller]_J \oplus a_3 : [Aller]_J) \wedge I\ \{d, s, f, e\}\ J$$

- Agent a_2 :

$$a_2\mathcal{C}_{Aller}(I)a_3 \Longleftrightarrow a_2\mathcal{C}_{t_i}a_3 \wedge (a_2 : [Aller]_J \oplus a_3 : [Aller]_J) \wedge I\ \{d, s, f, e\}\ J$$

- Agent a_3 :

$$a_3\mathcal{C}_{Aller}(I)a_1 \Longleftrightarrow a_3\mathcal{C}_{t_i}a_1 \wedge (a_3 : [Aller]_J \oplus a_1 : [Aller]_J) \wedge I\ \{d, s, f, e\}\ J$$

$$a_3\mathcal{C}_{Aller}(I)a_2 \Longleftrightarrow a_3\mathcal{C}_{t_i}a_2 \wedge (a_3 : [Aller]_J \oplus a_2 : [Aller]_J) \wedge I\ \{d, s, f, e\}\ J$$

Ces formules sont vérifiées car l'agent a_3 suit le scénario *Aller* et les relations de concurrence statiques existent bien dans le réseau de chacun des agents.

La règle de continuité d'une relation de concurrence (section 5.3.1) établit qu'une relation de concurrence active, le reste pendant toute la durée d'exécution de la tâche faisant l'objet de cette concurrence. A partir de ce résultat, nous pouvons déduire que ces relations seront actives pendant toutes la durée de reconnaissance du scénario *Aller*.

Après la reconnaissance ou le rejet du scénario, les deux relations actives dans le réseau de a_3, cessent de l'être. Afin de permettre aux agents a_1 et a_2 de désactiver les relations actives dans leurs réseaux et de mettre à jour les informations relatives au scénario, l'agent a_3 doit les informer du résultat de la reconnaissance.

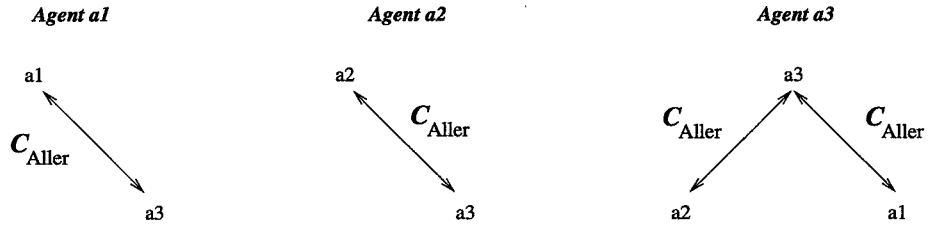


Figure 9.12 : Les relations de concurrence actives dans les réseaux de chacun des trois agents. Elles seront actives pendant toute la durée du suivi du scénario *Aller*.

9.6.2 Besoin

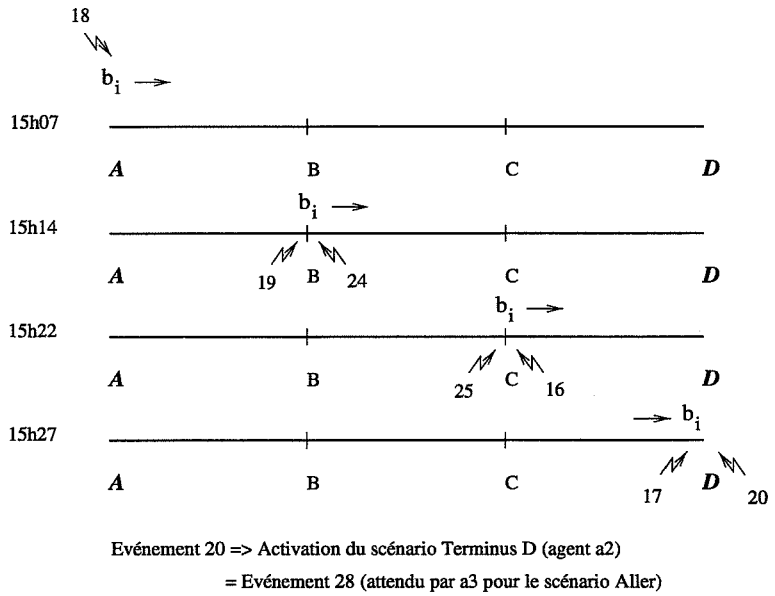


Figure 9.13 : Suivi du scénario *Aller* par l'agent a_3.

Le bus b_i traverse respectivement le tronçon AB en sept minutes, le tronçon BC en huit minutes et enfin le tronçon CD en cinq minutes (figure 9.13). Ce parcours correspond dans l'ordre à l'observation des événements 18, 19, 24, 25, 16 et 17. Les contraintes temporelles dans le graphe sont également respectées (voir figure 9.7).

Il arrive à présent au terminus D d'où l'observation de l'événement 20 et l'activation du scénario *Terminus D* par l'agent a_2 à 15h27.

Afin de continuer le suivi du scénario *Aller*, l'agent a_3, doit s'attendre à l'événement 28 qui correspond à l'activation du scénario *Terminus D*.

En utilisant la relation $a_3 \mathcal{D}^{Aller}_{TerminusD} a_2$ de son réseau de dépendances et en tenant compte des contraintes relatives à l'événement 28 dans le graphe du scénario *Aller*, l'agent a_3 envoie une requête de reconnaissance à l'agent a_2 :

(d a_3 a_2)(REQUEST)(PERFORMED *Terminus D* [15:27:00, 0])(ctxt3)

Ce message informe l'agent a_2, que l'agent a_3 a besoin d'une réponse 0 minute après 15h27. L'agent a_3 sait que l'agent a_2 ne pourra pas respecter cette contraintes, il entame alors un raisonnement hypothétique.

Raisonnement hypothétique

D'après le graphe du scénario *Aller*, l'événement 28 doit être observé en même temps que l'événement 17 observé auparavant (figure 9.7). De ce fait, l'agent a_3 doit utiliser un raisonnement hypothétique concernant l'activation du scénario *Terminus D* car la réponse de a_2 ne peut arriver à temps. Il considère que l'événement 28 s'est produit à temps et se met en attente d'une réponse de l'agent a_2 qui permettra de vérifier si l'événement 29 s'est également produit à temps.

Le bus b_i est à l'arrêt au terminus, un deuxième bus b_k entre dans le tronçon CD à 15h32 (observation de l'événement 16)(figure 9.14). Cinq secondes

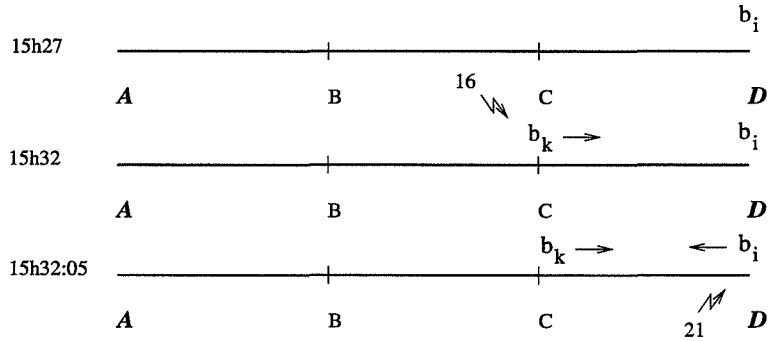


Figure 9.14 : La reconnaissance du scénario *Terminus D* par l'agent a_2 correspond à l'occurrence de l'événement 29 dans le scénario *Aller*, ce qui entraîne sa reconnaissance par l'agent a_3.

plus tard, le bus b_i quitte le terminus (événement 21). L'observation de ce dernier événement permet à l'agent a_2 de reconnaître le scénario *Terminus D*. Il envoie alors une réponse à l'agent a_3 lui précisant la date d'activation et de reconnaissance du scénario :

(d a_2 a_3)(RESPOND)(PERFORMED *Terminus D* 15:27:00 15:32:05)(ctxt3)

Cette réponse permettra dans un premier temps de valider l'hypothèse concernant l'événement 28 et ensuite la reconnaissance du scénario *Aller* par l'agent a_3. A noter, que l'agent a_2 utilise le même contexte de conversation dans sa réponse car la requête et la réponse font toutes les deux partie du même protocole de requête suivi par les deux agents.

L'agent a_3 ayant reconnu le scénario *Aller* informe les agents a_1 et a_2 de cette reconnaissance :

(d a_3 a_1)(INFORM)(PERFORMED *Aller* 15:07:00 15:32:05)(ctxt6)

(d a_3 a_2)(INFORM)(PERFORMED *Aller* 15:07:00 15:32:05)(ctxt7)

Dépendances actives

La table suivante permet d'illustrer les relations actives des trois agents, en fonction des activations et reconnaissances des scénarios, pendant le parcours du bus b_i du terminus A au terminus B.

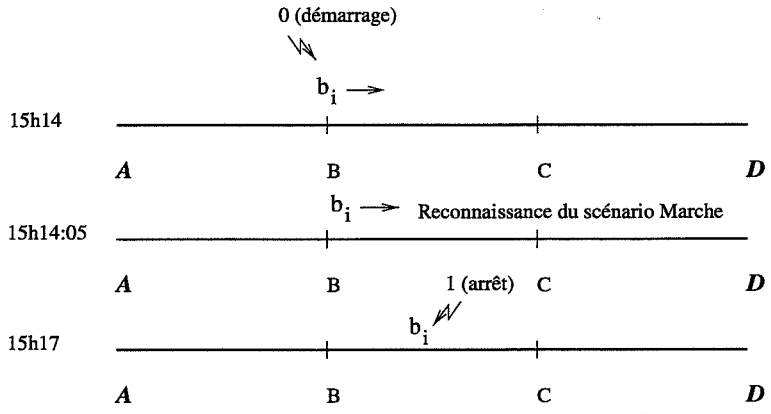
Heure	15h00	15h07	15h27	15h32:05
Agent a_1	Aucune	$a_1 C_{Aller} a_3$	$a_1 C_{Aller} a_3$	Aucune
Agent a_2	Aucune	$a_2 C_{Aller} a_3$	$a_2 C_{Aller} a_3$	Aucune
Agent a_3	Aucune	$a_3 C_{Aller} a_1$ $a_3 C_{Aller} a_2$	$a_3 C_{Aller} a_1$ $a_3 C_{Aller} a_2$ $a_3 \mathcal{D}^{Aller}_{TerminusD} a_2$	Aucune
Scénarios Actifs	Terminus A (a_1)	Aller (a_3)	Terminus D (a_2) Aller (a_3)	Aucun
Scénarios Reconnus	Aucun	Terminus A (a_1)	Aucun	Terminus D (a_2) Aller (a_3)

La dépendance $a_3 \mathcal{D}^{Aller}_{TerminusD} a_2$ reste active pendant toute la durée de reconnaissance du scénario *Terminus D*. L'agent a_3 désactive cette dépendance après la réception de la réponse envoyée par l'agent a_2 (contexte de conversation ctxt3).

La reconnaissance du scénario *Aller*, permet à l'agent a_3, de désactiver les relations $a_3 C_{Aller} a_1$ et $a_3 C_{Aller} a_2$. Les agents a_2 et a_3 désactivent respectivement leurs relations $a_1 C_{Aller} a_3$ et $a_2 C_{Aller} a_3$ après avoir été informés de cette reconnaissance par l'agent a_3 (contextes de conversation ctxt6 et ctxt7).

9.6.3 Aide

La relation d'aide existe uniquement entre les deux agents a_3 et a_5 comme le montre la figure 9.9.



Evénement 0 => Activation du scénario Marche (agent a3)

15h14:05 => Reconnaissance du scénario Marche (agent a3)
=> Début espace de validité
=> Activation relations a3 $H^{\text{Marche}}_{\text{Ouverture}}$ a5 , a3 $H^{\text{Marche}}_{\text{Fermeture}}$ a5

Evénement 1 => Fin espace de validité
=> Désactivation relations a3 $H^{\text{Marche}}_{\text{Ouverture}}$ a5 , a3 $H^{\text{Marche}}_{\text{Fermeture}}$ a5

Figure 9.15 : Reconnaissance du scénario *Marche* par l'agent a_3 et activation de la relation d'aide avec l'agent a_5 viv-à-vis de cette reconnaissance.

Afin de montrer l'utilisation de cette relation par les deux agents, nous nous intéressant au parcours du tronçon BC par le bus b_i. D'après la figure 9.13, ce parcours a lieu entre 15h14 et 15h22. Le bus est à l'arrêt à 15h14, il roule sans interruption jusqu'à 15h17 (figure 9.15). Le scénario *Marche* est activé par l'agent a_3 dès le démarrage du bus à 15h14. Cinq secondes plus tard, l'agent a_3 reconnaît le scénario (voir la description du scénario *Marche*). Son espace de validité étant conditionné par un disfonctionnement du moteur (événement 3), un arrêt du bus ou un dépassement de la vitesse limite (événement t_1), les des deux relations $a_3\mathcal{H}^{Marche}_{Ouverture}a_5$ et $a_3\mathcal{H}^{Marche}_{Fermeture}a_5$ deviennent actives et le restent de 15h14 à 15h17 (arrêt du bus).

L'agent a_3 informe l'agent a_5 de la reconnaissance du scénario *Marche* afin que ce dernier puisse activer ses relations :

(d a_3 a_5)(INFORM)(PERFORMED *Marche* 15:14:00 15:14:05)(ctxt4)

L'activation de ces relations signifie que les scénarios *Ouverture* et *Fermeture* ne peuvent être reconnus pendant toute la durée de l'activation. Par exemple, pendant la marche du bus, rien n'empêche de faire une demande

d'ouverture de porte, ce qui aura pour effet d'activer le scénario *Ouverture*. Or après dix secondes (cf. scénario *Ouverture*) la porte restera fermée, d'où le rejet du scénario *Ouverture*. L'existence d'une dépendance active entre les deux agents permet à l'agent a_5 de savoir qu'il ne doit reconnaître aucun des scénarios *Ouverture* et *Fermeture*, même si des événements permettant l'activation de ces deux scénarios se sont produits.

Quand le bus s'arrête à l'arrêt à 15h17, l'espace de validité du scénario *Marche* est fini. L'agent a_3 déactive ses relations. Il en informe l'agent a_5 afin de lui permettre de désactiver ses relations :

$$(\text{d } a_3\ a_5)(\text{INFORM})(\text{FAILED } \textit{Marche}\ 15{:}17{:}00)(\text{ctxt5})$$

Les relations étant inactives, le suivi des scénarios *Ouverture* et *Fermeture* redevient possible.

9.7 Conjonction des relations de dépendance

Dans la section 5.6, nous avons décrit trois types de conjonction : $\mathcal{CC}$, $\mathcal{CX}$ et $\mathcal{XX}$. Dans cette section, nous montrons l'utilisation d'une conjonction dans chaque catégorie.

9.7.1 Relations de concurrence

Les trois agents a_1, a_2 et a_3 sont en concurrence sur le scénario *Aller*. D'après les résultats présentés dans la section 5.6.1, chacun des trois agents peut déduire qu'il existe une relation de concurrence entre les deux autres vis-à-vis du scénario *Aller* ($u \neq v \neq w \in \{1,2,3\}$) :

$$a_u \mathcal{C}_{Aller} a_v \ \wedge\ a_u \mathcal{C}_{Aller} a_w \Longrightarrow a_v \mathcal{C}_{Aller} a_w$$

En se plaçant dans le contexte de la section 9.6.1 (début du parcours de la ligne AD par le bus b_i), nous allons analyser l'activation des différentes relations pour chacun des trois agents. La figure 9.16 illustre l'activation des relations déduites, les relations initiales étant celles de la figure 9.12. La règle d'activation est donnée dans la section 5.6.1, en voici l'instanciation ($\oplus$: opérateur de disjonction exclusive) :

$$a_u \mathcal{C}_{Aller}(I) a_v \ \oplus\ \neg a_u \mathcal{C}_{Aller}(I) a_w \Longrightarrow a_v \mathcal{C}_{Aller}(I) a_w$$

Agent a_3

Les deux relations $a_3 \mathcal{C}_{Aller} a_1$ et $a_3 \mathcal{C}_{Aller} a_2$ sont actives pendant le suivi du scénario *Aller*. De ce fait, la relation déduite $a_1 \mathcal{C}_{Aller} a_2$ n'est pas active.

Ce résultat permet à l'agent a_3 de savoir qu'il est tenu d'informer les deux agents a_1 et a_3 du résultat de la reconnaissance du scénario car ils n'ont aucun autre moyen de le savoir.

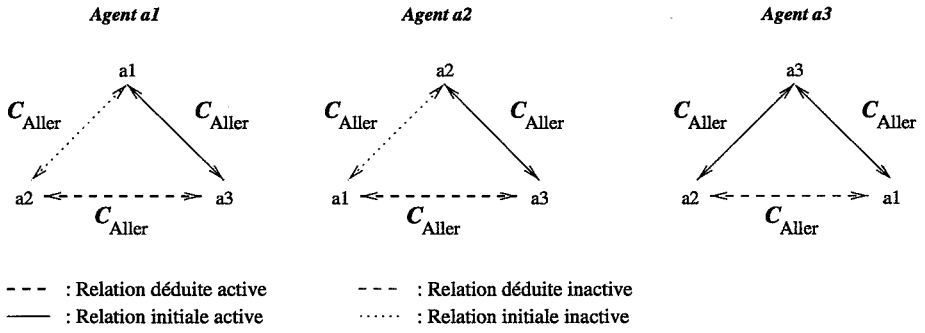


Figure 9.16 : L'activation des relations de concurrence déduites pour chacun des agents.

Agent a_1

La relation $a_1\mathcal{C}_{Aller}a_3$ est active, tandis que la relation $a_1\mathcal{C}_{Aller}a_2$ ne l'est pas. De ce fait, la relation déduite $a_2\mathcal{C}_{Aller}a_3$ est active.

Cette relation active permet à l'agent a_1 de savoir qu'il n'est pas tenu d'informer l'agent a_2, s'il reçoit de l'agent a_3 des informations concernant la reconnaissance du scénario *Aller*. En fait, quand l'agent a_3 lui apprendra que le scénario a été reconnu, l'agent a_1 sait que l'agent a_2 sera informé également.

Agent a_2

L'agent a_2 va suivre le même raisonnement que l'agent a_1 et obtiendra le même résultat car sa relation déduite $a_1\mathcal{C}_{Aller}a_3$ est également active.

9.7.2 Relations de concurrence et de besoin

Dans la section 9.4, nous avons vu que l'agent a_1 admet dans ses réseaux de dépendances les relations $a_1\mathcal{C}_{Aller}a_3$ et $a_1\mathcal{D}^{Aller}_{TerminusD}a_2$.

Afin de montrer comment l'agent a_1 va utiliser la conjonction de ces deux relations dans son raisonnement social, nous reconsidérons le parcours du tronçon BC par le bus b_i.

En instanciant la première formule donnée dans la section 5.6.2 avec les deux relations, l'agent a_1 peut déduire le résultat suivant (cf. figure 9.17) :

$$a_1\mathcal{C}_{Aller}a_3 \wedge a_1\mathcal{D}^{Aller}_{Terminus\ D}a_2 \Longrightarrow a_3\mathcal{D}^{Aller}_{Terminus\ D}a_2$$

D'après la figure 9.12, la relation $a_1\mathcal{C}_{Aller}a_3$ est active pendant toute la durée du parcours du tronçon BC.

Le bus étant dans la zone BC, nous pouvons établir la relation $a_1 >_{Aller} a_3$ (section 9.6.1). Cette relation est valable également pendant toute la durée du parcours du tronçon BC.

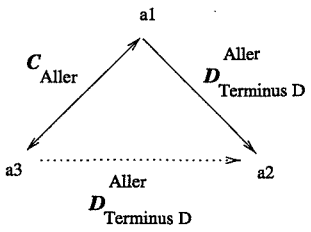


Figure 9.17 : A partir d'une relation de concurrence et une autre de besoin, un agent en déduit une relation besoin.

En instanciant la deuxième formule de la section 5.6.2, avec les deux relations actives ci-dessus, l'agent a_1 peut déduire le résultat suivant :

$$a_1\mathcal{C}_{Aller}(I)a_3 \wedge a_1\mathcal{D}^{Aller}_{Terminus\ D}a_2 \wedge a_1 >_{Aller} (J)a_3 \wedge J\{o,m,di,e\}I \Longrightarrow a_3\mathcal{D}^{Aller}_{Terminus\ D}(I)a_2$$

Grâce à l'activation de la relation $a_3\mathcal{D}^{Aller}_{Terminus\ D}a_2$, l'agent a_1 sait qu'il ne doit pas activer la relation $a_1\mathcal{D}^{Aller}_{TerminusD}a_2$ si besoin est, tant que la première relation est active.

La relation $a_1\mathcal{C}_{Aller}a_3$ étant active, l'agent a_1 sait que l'agent a_3 va l'informer du résultat de la reconnaissance du scénario *Aller*, ce qui aura pour effet de désactiver ces deux relations.

9.7.3 Relations de besoin

Nous nous plaçons dans le contexte de la figure 9.15 : le bus b_i démarre à 15h14 du point B, il parcours une partie du tronçon BC et s'arrête à 15h17.

Pendant ce parcours, un passager désire descendre à l'arrêt suivant. Il appuie alors sur le bouton de demande d'arrêt à 15h16 (événement 8). Le scénario *Déchargement* devient actif et est en cours de suivi par l'agent a_4. Le réseau de dépendances de ce dernier contient les deux relations $a_4\mathcal{D}^{Déchargement}_{Moteur}a_6$ et $a_4\mathcal{D}^{Déchargement}_{Moteur}a_5$. En instanciant la première formule de la section 5.6.3, l'agent a_4 en déduit le résultat suivant :

$$a_4\mathcal{D}^{Déchargement}_{Moteur}a_6 \wedge a_4\mathcal{D}^{Déchargement}_{Moteur}a_5 \Longrightarrow a_5\mathcal{C}_{Moteur}a_6.$$

D'après la règle de pouvoir concernant le suivi du scénario *Moteur*, nous avons $a_5 >_{Moteur} a_6$ (l'agent a_6 suit le scénario *Moteur*). Ainsi, la relation $a_4\mathcal{D}^{Chargement}_{Moteur}a_6$ devient active.

En réecrivant la deuxième formule de la section 5.6.3, l'agent a_4 déduit le résultat suivant :

$$a_4\mathcal{D}^{Déchargement}_{Moteur}(I)a_6 \wedge a_4\mathcal{D}^{Déchargement}_{Moteur}a_5 \Longrightarrow a_5\mathcal{C}_{Moteur}(I)a_6.$$

Lors de la description des scénarios *Chargement* et *Déchargement*, nous avons signalé que l'activation du scénario *Déchargement* entraîne l'activation du scénario *Chargement* du moment où le bus est à l'arrêt (observation de l'événement 6 à 15h17). La reconnaissance de ce dernier nécessite également la reconnaissance du sous-scénario *Moteur*. L'agent a_4 consulte son réseau de dépendances et trouve les deux relations $a_4\mathcal{D}^{Chargement}_{Moteur}a_5$ et $a_4\mathcal{D}^{Chargement}_{Moteur}a_6$. L'activation de la relation déduite $a_5\mathcal{C}_{Moteur}a_6$, empêche l'agent a_4 d'envoyer une nouvelle requête à l'agent a_6 pour reconnaître le sous-scénario *Moteur*. En fait, une reconnaissance de ce sous-scénario va servir à la reconnaissance des scénarios *Déchargement* et *Chargement*.

9.8 Discussion

Dans ce chapitre, nous avons mis l'accent sur l'utilisation des relations de dépendance par un ensemble de six agents responsables de la supervision d'une flotte de trois bus circulant sur une ligne. Les agents utilisent ces relations dans leur raisonnement social afin d'améliorer le processus de supervision. A travers, l'exemple nous avons montré comment les agents utilisent les activations des relations pour déduire des informations relatives à la reconnaissance de certains scénarios. Ces activations leur permettent également de ne pas activer inutilement d'autres relations et de mieux gérer leurs communications.

Les scénarios que nous avons proposés modélisent uniquement des bons fonctionnements. Or dans la section 4.5.2, nous avons signalé qu'un système dynamique est généralement modélisé par un ensemble de scénarios de bon et de mauvais fonctionnement.

En effet, dans certains cas, le rejet d'un scénario de bon fonctionnement ne suffit pas pour localiser et identifier une anomalie. Dans notre contexte, ce n'est pas le cas : le rejet d'un scénario de bon fonctionnement est dû soit au retard d'un événement, soit à l'arrivée d'un événement trop tôt. Pour l'ensemble des scénarios proposés, ce phénomène suffit pour déterminer le problème.

Par exemple, si un bus reste à l'arrêt plus de dix minutes au terminus D, le scénario *Terminus D* est rejeté et cela pour cause, le retard de l'événement 21 (quitter le terminus). Ce rejet cause également le rejet du scénario *Aller*. Un diagnostic au niveau *bus*, permet de révéler que le bus a eu du retard au terminus D. Au niveau *ligne*, le diagnostic permet de révéler que le bus a parcouru normalement la ligne et qu'une anomalie s'est présentée au niveau du terminus D. Dans les deux cas, nous avons assez d'éléments pour expliquer le problème, sans avoir recours à des scénarios de dysfonctionnement.

Une version plus complexe de cet exemple nécessiterait la reconnaissance de scénarios de dysfonctionnement. En effet, dès qu'il s'agit de plusieurs lignes qui se croisent, nous ne pouvons plus tenir compte d'une ligne iso-

lément des autres : deux bus sur deux lignes différentes peuvent effectuer chacun un parcours correct sur sa ligne, mais l'existence d'un croisement entre ces deux lignes peut poser un problème. Dans ce cas, un scénario de dysfonctionnement permet de détecter un éventuel problème. Ce scénario tiendra compte respectivement des deux bus sur les deux lignes.

Dans le chapitre suivant, nous proposons une implémentation d'une société d'agents temporels pour la supervision par reconnaissance de scénarios. L'exemple présenté dans ce chapitre, servira pour illustrer différents aspects de l'implémentation.

Chapitre 10

Implémentation du modèle

10.1 Introduction

Dans le présent chapitre, nous présentons l'architecture d'implémentation d'une société d'agents responsable de la supervision par reconnaissance de scénarios. Cette architecture a conduit au développement du système *STARS* (*Society of Temporal Agents Recognizing Scenarios*).

Le système est réalisé à l'aide d'un ensemble d'objets C++. Ils sont liés par un ensemble de relations (cf. section 10.2). Nous utilisons le produit ILOG-SERVER [ILO95a] pour la modélisation des objets et des relations existant entre ces objets.

Nous présentons tout d'abord, les objets servant à modéliser le système supervisé, c'est-à-dire les classes servant à définir les scénarios temporels et les événements provenant du système supervisé.

Nous présentons ensuite, l'architecture d'un agent. On y trouve l'état mental qui est consulté et mis à jour pendant toute la durée de son fonctionnement. Pour chacun des éléments y figurant, nous décrivons son contexte de création, d'utilisation et de mise à jour ou de suppression.

Par la suite, nous décrivons le cycle général de l'agent, en détaillant chacune des méthodes appelées.

Enfin, nous décrivons la procédure de lancement du système et donnons quelques éléments de description de l'interface réalisée pour visualiser le fonctionnement de l'ensemble des agents. Les éléments de l'interface ont été développés en ILOG-SERVER et ILOG-VIEWS [ILO95b].

Afin d'illustrer les différents objets définis, nous utilisons des diagrammes de classe avec la notation OMT [Rum91].

10.2 Relations entre objets

Le modèle d'objets utilisé dans le développement du système *STARS* permet l'utilisation de deux relations entre objets :

- la relation *possède* entre deux objets A et B spécifie que l'objet A possède l'objet B. La conséquence de cette relation est que l'objet B ne pourra plus être possédé par aucun autre objet (figure 10.1). Pour un objet donné, il est toujours possible de calculer la relation inverse *possédé par*, ce qui permet de retrouver l'objet possesseur ;

Figure 10.1 : Relation *possède* entre deux objets A et B.

- la relation *utilise* entre les deux objets spécifie que l'objet A utilise l'objet B. Contrairement à la relation *possède*, un objet peut être utilisé par plusieurs autres objets (figure 10.2) et de ce fait, la relation inverse *utilisé par* permet de retrouver la liste des utilisateurs de l'objet.

Figure 10.2 : Relation *Utilise* entre deux objets A et B.

L'utilisation des relations et de leurs inverses permet l'accès direct à un objet à partir d'un autre.

Dans les sections suivantes, nous utilisons les termes *utilise*, *utilisé par*, *possède*, *possédé par* en caractères italiques pour faire référence aux deux relations décrites ci-dessus, ainsi qu'à leurs inverses.

10.3 Système supervisé

Les objets modélisant le système supervisé correspondent aux classes définissant les événements et les scénarios et tous les algorithmes nécessaires à leur suivi et leur reconnaissance.

10.3.1 Evénements

Les événements provenant du système supervisé sont modélisés par la classe *SysEvent*. Un *SysEvent* admet un *identificateur* est une *description.*

La *date d'observation* d'un événement est représentée dans chaque scénario où figure cet événement.

10.3.2 Scénario

La classe *Scenario* contient la structure sous forme de graphe de contraintes, la condition d'activation et contient également tous les algorithmes de reconnaissance d'un scénario (modes en ligne et hors ligne).

Une session est modélisée également par la classe *Scenario.*

Un objet de ce type *utilise* un ensemble de *SysEvent* qui font partie de son graphe de contraintes et un ensemble de contraintes modélisées par la classe *Constraint* (figure 10.3).

Un objet *Constraint* contient un intervalle et *utilise* deux *SysEvent.*

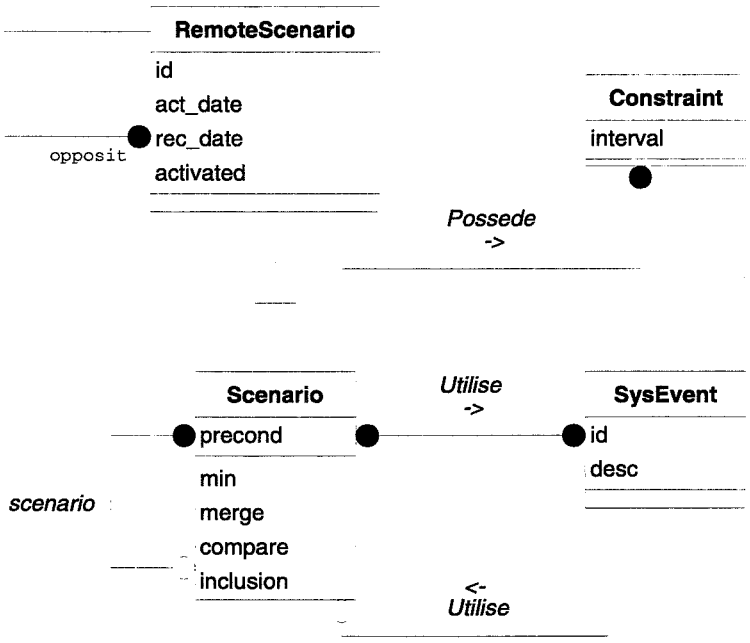


Figure 10.3 : Classe *Scenario.*

Reconnaissance

Les algorithmes de reconnaissance sont modélisés dans la classe *Scenario* :

- la méthode *min* admet en entrée un scénario. En sortie, elle retourne le scénario avec un graphe minimal si ce graphe est cohérent, sinon elle retourne une valeur nulle ;
- la méthode *merge* permet la fusion des graphes de deux scénarios ;
- la méthode *compare* teste l'égalité des événements de deux *Scenario* ;
- la méthode *inclusion* teste l'inclusion entre deux scénarios ;

L'utilisation de ces méthodes permet la définition des différents types de reconnaissance décrit dans chapitre 8 :

- la *compatibilité partielle* entre un scénario S et une session s est réalisée si $min(merge(S,s)) \neq$ NULL ;
- la *compatibilité totale* est réalisée si $compare(s,S) \wedge min(merge(S,s)) \neq$ NULL ;
- la *satisfaction partielle* correspond à $inclusion(s,S)$;
- la *satisfaction totale* correspond à $compare(s,S) \wedge inclusion(s,S)$;

Classe *RemoteScenario*

La classe *RemoteScenario* est une classe générique utilisée pour représenter un scénario dont on ne dispose pas de l'ensemble d'événements, ni de l'ensemble de contraintes. Un *RemoteScenario* sert à représenter un scénario appartenant au domaine de responsabilité d'un autre agent.

Cette classe contient uniquement des informations telles que le nom du scénario, sa dernière date de reconnaissance, son état actuel (actif, non actif) et la date de la dernière activation.

La classe *Scenario* est une sous-classe de la classe *RemoteScenario.*

Relation scénario/sous-scénario

Un sous-scénario est également modélisé par la classe *Scenario.* Pour faire le lien entre un scénario et un sous-scénario, nous utilisons une relation d'utilisation entre un *SysEvent* et un *Scenario.* Ainsi, les relations d'utilisation entre un *Scenario* et un *SysEvent* d'une part, et entre un *SysEvent* et un *Scenario* d'autre part, permettent d'accéder aux sous-scénarios à partir d'un scénario. Inversement, l'utilisation des inverses de ces relations, permet l'accès au scénarios de plus haut niveau à partir d'un scénario (cf. figure 10.4).

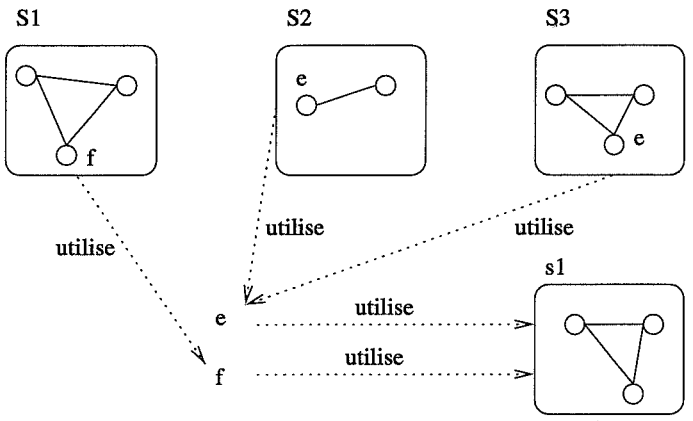


Figure 10.4 : Accès à partir d'un scénario à tous les sous-scénarios et inversement à partir d'un sous-scénario à tous les scénarios pères.

Relation d'opposition

La relation d'opposition est définie au niveau de la classe *RemoteScenario*. En effet, cette classe *utilise* une liste de *RemoteScenario* qui sont en opposition avec le scénario.

10.4 Agent

Un agent est modélisé par la classe *Agent* (figure 10.5). Cette classe *possède* un objet *MentalState* (état mental), une liste d'objets *MentalState* (descriptions externes), un objet *Sensor* (capteur sensoriel) et enfin, un objet *Interaction* (communication).

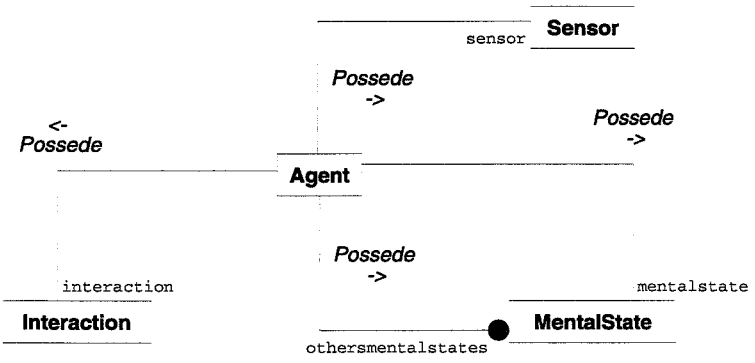


Figure 10.5 : Classe *Agent*.

Le cycle de fonctionnement de l'agent sera décrit dans la section 10.5,

après avoir décrit les classes ci-dessus dans les sections suivantes.

10.4.1 Etat mental

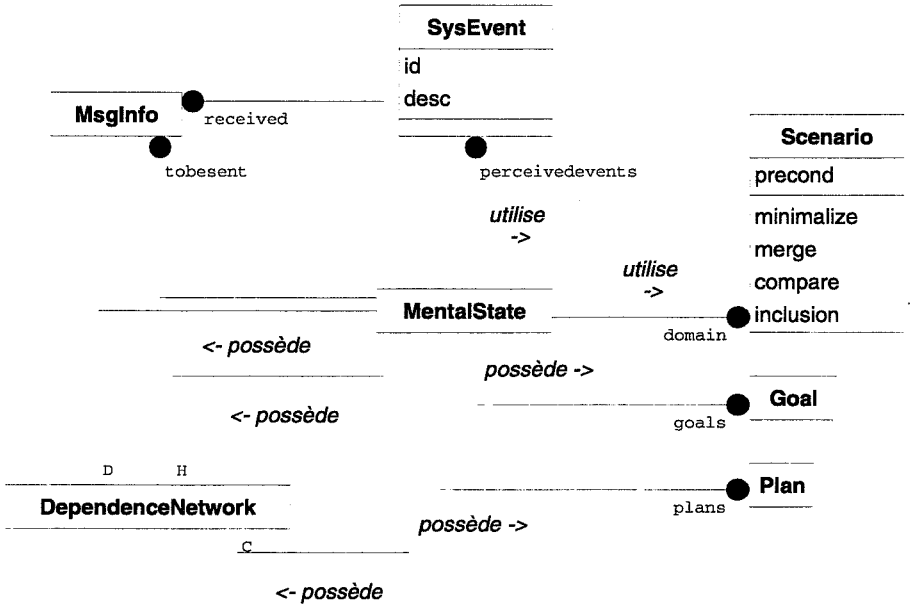


Figure 10.6 : Classe *MentalState*.

L'état mental de l'agent est défini par la classe *MentalState* (figure 10.6). Il est formé des éléments suivants :

- *utilise* une liste de *RemoteScenario* qui constitue le *domaine de responsabilité* de l'agent ;
- *possède* une liste de *Goal*, buts en attente et qui demeurent insatisfaits pour le moment ;
- *possède* une liste de *Plan*, plans associés aux buts en attente ;
- contient une liste de *SysEvent*, événements observés et non encore traités ;
- *possède* trois *DependenceNetwork*, réseaux de dépendances ($\mathcal{C}$, $\mathcal{D}$ et $\mathcal{H}$) ;
- contient deux listes de *MsgInfo*, informations à communiquer ou reçues d'autres agents.

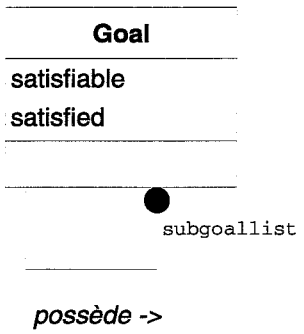


Figure 10.7 : Classe *Goal.*

Buts/sous-buts

Un but est modélisé par la classe *Goal.* Sa structure correspond à celle décrite dans le chapitre 7. Les sous-buts sont également modélisés par cette classe (figure 10.7). Il existe une relation de possession entre un but est ses sous-buts. L'utilisation de la relation inverse, permet de retrouver le but de haut niveau à partir d'un sous-but.

Plans

Un plan est modélisé par la classe *Plan.* Il possède une liste d'entrées modélisées par la classe *PlanEntry.* Ces deux classes admettent les mêmes structures que celles décrites dans le chapitre 7.

Réseaux de dépendances

Les trois réseaux de dépendances $\mathcal{C}$, $\mathcal{D}$ et $\mathcal{H}$ sont modélisés par la classe *DependenceNetwork* qui est une structure de graphe où un nœud central représente l'agent et les autres nœuds représentent les autres agents dont il dépend. Les nœuds sont modélisés par la classe *Node* qui contient un champ identificateur d'un agent. Les arcs sont modélisés par la classe *DepLink* qui contient deux champs identificateurs des scénarios faisant l'objet de la dépendance et un champ précisant si la dépendance est *active.* Un *DepLink utilise* deux *Node.* Un *DependenceNetwork possède* un ensemble de *DepLink* et *utilise* un ensemble de *Node* (figure 10.8).

Les relations d'utilisation permettent de retrouver tous les réseaux de dépendances (y compris ceux dans les descriptions externes) qui utilisent un nœud (tous les agents qui interviennent dans les relations de dépendance d'un agent). Elles permettent également de retrouver tous les arcs partants/arrivants à un nœud (toutes les dépendances relatives à un agent).

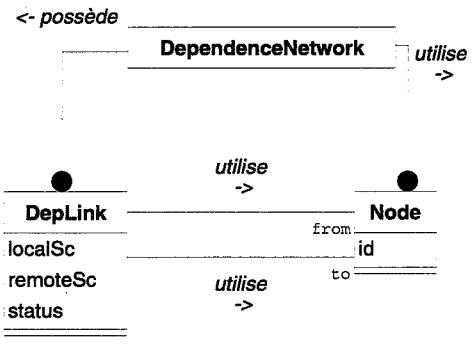


Figure 10.8 : Classe *DependenceNetwork*.

Les réseaux de dépendances sont créés après l'attribution des domaines de responsabilité aux agents.

La connaissance des scénarios et de leurs responsables permet le calcul des réseaux $\mathcal{C}$ et $\mathcal{D}$ (cf. figures 10.9 et 10.10).

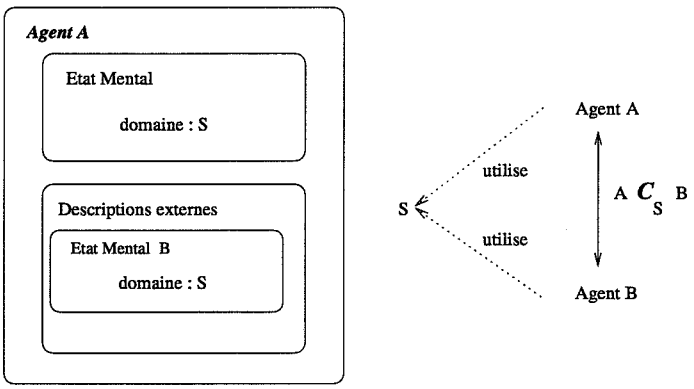


Figure 10.9 : Construction du réseau $\mathcal{C}$.

Le réseau $\mathcal{H}$ est calculé en fonction des relations d'opposition entre les scénarios (cf. figure 10.11).

Informations échangées

Les informations reçues ou à envoyer à d'autres agents, sont définies par deux listes d'objets de type *MsgInfo*. Cette classe contient les mêmes champs que la structure *MSG-ENTRY* décrite dans le chapitre 7. Nous rappelons qu'elle contient toutes les informations nécessaires pour la formation d'un message écrit en langage $\mathcal{L}$ (chapitre 6). Inversement, quand un message est reçu, il est transformé en un objet *MsgInfo* avant traitement.

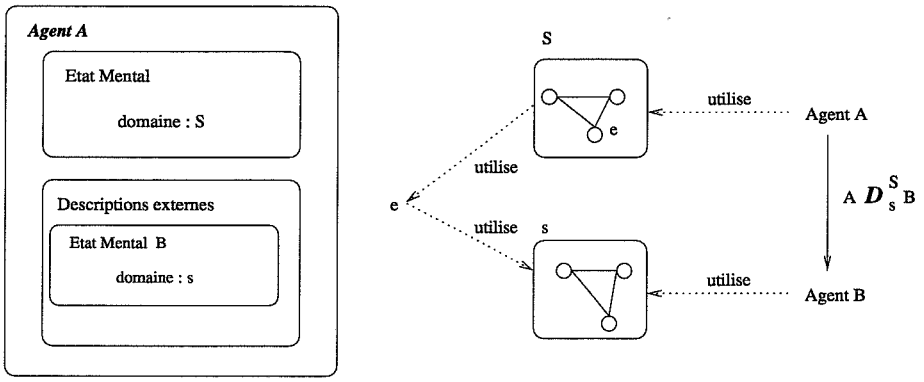


Figure 10.10 : Construction du réseau $\mathcal{D}$.

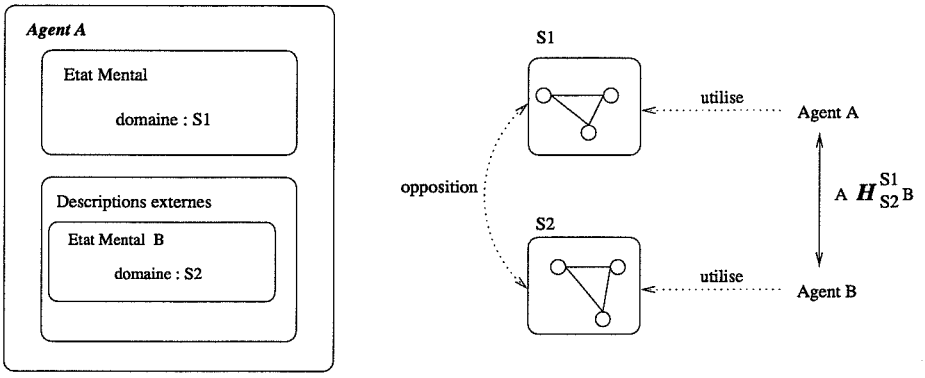


Figure 10.11 : Construction du réseau $\mathcal{H}$.

10.4.2 Descriptions externes

Les descriptions externes sont représentées par une liste de *MentalState*. Dans une description externe, seuls les réseaux de dépendances et la liste des scénarios seront mis à jour. A noter, qu'une description externe pointe sur une liste de *RemoteScenario* et non pas de *Scenario*, car l'agent ne détient pas une information précise sur les scénarios appartenant aux domaines des autres agents.

10.5 Cycle de l'agent

Le cycle de l'agent est défini par l'appel de la méthode *mainLoop* de la classe *Agent*. Cette méthode fait appel à un ensemble de méthodes qui définissent les activités de haut niveau de l'agent. Le cycle sera exécuté indéfiniment après le lancement de l'agent :

- *perceive* permet la perception d'événements provenant de l'environnement (cf. section 10.5.1) ;
- *receive* assure la réception de messages d'autres agents (cf. section 10.5.2) ;
- *decide* modélise la prise de décision après la réception de messages d'autres agents (cf. section 10.5.4) ;
- *send* permet l'envoi de messages à d'autres agents (cf. section 10.5.2).

La planification et le raisonnement social ne figurent pas dans le cycle général. Ils seront utilisés par la méthode *decide*.

10.5.1 Perception

Le système supervisé émet des événements en permanence durant son fonctionnement. Ces événements sont inscrits dans un fichier (environnement) en précisant le type de l'événement et sa date d'observation.

Chaque agent est doté d'un capteur sensoriel qui se charge de détecter les événements en relation avec les scénarios qui sont sous sa responsabilité. Ce capteur est modélisé par la classe *Sensor*. La méthode *perceive* de la classe *Agent* fait appel à la méthode *perceive* de la classe *Sensor* qui scrute l'environnement, filtre les événements observés et rend la liste des *SysEvent* observés qui appartiennent aux scénarios du domaine de responsabilité de l'agent. Cette liste est inscrite dans l'état mental (champ *perceivedevents*).

10.5.2 Communication

La communication avec les autres agents est assurée par la classe *Interaction*.

Réception

La méthode *receive*, récupère les messages reçus, transforme un message en un objet *MsgInfo* et ajoute cet objet à la liste *received* dans l'état mental.

Envoi

Quand un message doit être envoyé à un autre agent, un objet *MsgInfo* est construit, contenant toutes les informations nécessaires à la formation du message. Cet objet est stocké dans la liste *tobesent* de l'état mental. La méthode *send* transforme chaque objet de cette liste en un message en langage $\mathcal{L}$. Ce message est ensuite envoyé à l'agent destinataire.

10.5.3 Raisonnement social

Le raisonnement social n'est pas modélisé par une méthode faisant partie du cycle de l'agent. Un agent fait appel à son raisonnement social quand il s'agit d'utiliser ses relations de dépendance avec les autres. Ce raisonnement permet à un agent d'une part, la mise à jour de l'état d'activation de l'ensemble de ses relations et d'autre part, la mise à jour des relations déduites entre les autres agents (conjonction des relations de dépendance).

Une relation déduite entre deux agents implique la mise à jour des réseaux correspondants dans les descriptions externes relatives aux deux agents.

Quand une relation de dépendance doit être activée, $\mathcal{D}$ par exemple (méthode *checkD*), les conjonctions $\mathcal{CX}$ et $\mathcal{XX}$ sont examinées (méthodes *checkCX*, *checkXX*).

Dans le chapitre 5, nous avons montré que l'activation et la désactivation d'une relation de dépendance correspond au début et à la fin de l'exécution d'une tâche. A chaque tâche en cours d'exécution correspond un but. Ainsi, une relation doit être activée après la création d'un but, elle cesse d'être active après sa suppression.

Dans ce qui suit, nous décrivons les différentes méthodes relatives au raisonnement social. Ces méthodes sont appelées par d'autres méthodes (section 10.5.4).

La méthode *chooseAgent* permet de choisir l'agent qui a le moins de pouvoir vis-à-vis de la reconnaissance d'un scénario. S'il n'y a qu'un seul agent capable de suivre le scénario, il est choisi.

Fonction *chooseAgent*(*Scenario* S)
l = liste agents a_k | self $\mathcal{C}_S a_k$
retourner a_l | $\forall k\ a_k >_S a_l$
FinFonction

La méthode *checkD* teste si une dépendance $\mathcal{D}$ doit être activée, elle utilise le raisonnement entrepris dans une conjonction $\mathcal{XX}$.

Fonction *checkD*(*Scenario s*)
l = liste agents a_i | self $\mathcal{D}_s^S a_i$
si $\exists a_i, a_j$ dans l | $a_i \mathcal{C}_s a_j$ est active
alors retourner faux
sinon retourner vrai
FinFonction

La méthode *switchOnC* permet l'activation des relations $\mathcal{C}$ d'un agent. L'activation des relations déduites par les conjonctions $\mathcal{CX}$ et $\mathcal{CC}$ est également examinée.

Fonction *switchOnC*(*Scenario* S)
l = liste agents a_i | self $\mathcal{C}_S a_i$
Pour chaque a_i dans l **faire**
activer self $\mathcal{C}_S a_i$
checkCX($\mathcal{C}$, S, S, a_i)
checkCC($\mathcal{C}$, S, S, a_i)
FinFonction

La méthode *switchOnD* permet l'activation d'une relation $\mathcal{D}$. L'activation des relations déduites par les conjonctions $\mathcal{XX}$ et $\mathcal{CX}$ est également examinée.

Fonction *switchOnD*(*Scenario* S, *Scenario* s)
activer self $\mathcal{D}_s^S a$
checkXX($\mathcal{D}$, S, s, a)
checkCX($\mathcal{D}$, S, s, a)
FinFonction

La méthode *switchOnH* permet l'activation d'une relation $\mathcal{H}$. L'activation des relations déduites par les conjonctions $\mathcal{XX}$ et $\mathcal{CX}$ est également examinée.

Fonction *switchOnH*(*Scenario* S)
l_1 = liste de *Scenario* S_i | $S_i \leftrightarrow S$
pour chaque S_i dans l_1 **faire**
l_2 = liste de *MentalState* responsables de S_i
pour chaque r_j dans l_2 **faire**
si self $\mathcal{H}_{S_i}^S r_j$ est non active
alors activer self $\mathcal{H}_{S_i}^S r_j$
ajouter entrée dans *mentalstate.tobesent* : S reconnu
checkCX($\mathcal{H}$, S, S_i, r_j)
checkXX($\mathcal{H}$, S, S_i, r_j)
FinFonction

Les trois méthodes suivantes permettent respectivement, la désactivation des relations $\mathcal{C}$, $\mathcal{D}$ et $\mathcal{H}$ relatives à un scéanrio.

Fonction *switchOffC*(*Scenario* S)
l = liste agents a_i | self $\mathcal{C}_S a_i$
pour chaque a_i dans l **faire**
désactiver self $\mathcal{C}_S a_i$
checkCC($\mathcal{C}$, S, S, a_i)
checkCX($\mathcal{C}$, S, S, a_i)
FinFonction

Fonction *switchOffD*(*Scenario* S)
pour chaque a_i, S_j | self $\mathcal{D}^S_{S_j} a_i$ **faire**
désactiver self $\mathcal{D}^S_{S_j} a_i$
checkCX($\mathcal{D}$, S, S_j, a_i)
checkXX($\mathcal{D}$, S, S_j, a_i)
FinFonction

Fonction *switchOffH*(*Scenario* S)
pour chaque S_i, a_j | self $\mathcal{H}^S_{S_i} a_j$ **faire**
désactiver self $\mathcal{H}^S_{S_i} a_j$
checkCX($\mathcal{H}$, S, S_i, a_j)
checkXX($\mathcal{H}$, S, S_i, a_j)
FinFonction

Les méthodes *checkCC*, *checkCX*, *checkXX* vérifient respectivement les activations des relations déduites par une conjonction $\mathcal{CC}$, $\mathcal{CX}$, $\mathcal{XX}$. Ces règles sont données dans la section 5.6.

10.5.4 Décision

La méthode *decide* de la classe *Agent* fait appel à quatre méthodes: *handleEvents* (gestion des événements), *handleReceivedEntries* (gestion des informations communiquées), *handleGoals* (gestion des buts) et *handlePlans* (gestion des plans).

Gestion des événements

La fonction *handleEvents* est définie comme suit:

Fonction *handleEvents*(liste de *SysEvent mentalstate.perveivedevents*)
pour chaque événement e_i dans *mentalstate.perceivedevents* **faire**
l_i=liste de *Scenario* concernés
/* relation utilisation Scenario-SysEvent */
pour chaque scénario S_j dans l_i **faire**
check(S_j, e_i)

supprimer e_i
FinFonction

La fonction *check* vérifie le respect des contraintes temporelles d'un scénario suite à l'observation d'un événement.

Fonction *check*(*Scenario* S_j, *SysEvent* e_i)
si S_j.activated = faux
alors si S_j.precond = vrai
alors a = *chooseAgent*(S_j)
si a = self
alors S_j.activated = vrai
créer *Goal* b_j pour S_j
Plan(b_j)
switchOnC(S_j)
sinon activer self $\mathcal{C}_{S_j}a$
si S_j.activated = vrai
alors mettre à jour graphe de contraintes S_j
si contraintes de S_j ok
alors **si** e_i est le dernier dans S_j
alors S_j est reconnu
si S_j.temporalspace > 0
alors *switchOnH*(S_j)
si b_j n'a pas de sous-buts non satisfaits
alors b_j.satisfied = vrai
/* sinon *Raisonnement hypothétique* en cours ...*/
si S_j *est utilisé par SysEvent* e
alors l_1 = liste *Scenario* concernés
/* relation utilisation Scenario-SysEvent */
pour chaque S_k dans l_1 **faire**
check(S_k, e)
l_2 = liste agents a_l | $a_l \mathcal{D}^S_{S_j}$ self
pour chaque a_l dans l_2 **faire**
ajouter entrée dans *mentalstate.tobesent* :
informer reconnaissance
sinon **si** événement attendu e dans S_j *utilise* scénario S_k
/* S_k est un sous-scénario de S_j */
alors **si** délai ok
alors ajoute délai dans plan de S_j
créer *Goal* b_k pour S_k
Plan(b_k)
sinon /* déclenchement raisonnement hypothétique */
date d'observation de e = fin du délai
sinon S_j est rejeté
b_j.satisfiable = faux
/* tous les buts de plus hauts niveaux sont insatisfaisables */
FinFonction

Traitement des informations reçues

Nous avons signalé plus haut, qu'un message reçu est transformé en un objet *MsgInfo* et est ajouté dans la liste *received* de l'état mental. Nous avons plusieurs types de message et pour chaque type, l'agent doit suivre le traitement associé à ce type.

Fonction *handleReceivedEntries*(liste de *MsgInfo mentalstate.received*)
pour chaque entrée e dans *mentalstate.received* **faire**
cas : e.type = REQUEST
handleRequest(e);
cas : e.type = RESPOND
handleResponse(e);
cas : e.type = INFORM
handleInformation(e);
cas : e.type = REFRAIN
handleRefrainingRequest(e);
FinFonction

Fonction *handleRequest*(*MsgInfo* e)
si e.scenario $\neq$ NULL /* Requête de reconnaissance de scénarios */
alors **si** e.scenario.temporalspace = valide
alors ajouter entrée dans *mentalstate.tobesent*: réponse scénario reconnu
sinon créer *Goal* g pour e.scenario
Plan(g)
sinon /* Requête d'information */
ajouter entrée dans *mentalstate.tobesent*: réponse get-desc(e.context)
FinFonction

Fonction *handleResponse*(*MsgInfo* e)
si e.scenario $\neq$ NULL /* réponse : reconnaissance */
alors **si** e.results = reconnu
alors g = but relatif à e.scenario
g.satisfied = vrai
G = père de g
S = G.scenario
e = événement dans S relatif à e.scenario
/* relation utilisation */
si e est attendu **alors** *check*(S, e)
/* sinon raisonnement hypothétique validé */
l = liste de *Scenario utilisant* e (sauf S)
pour chaque S_i dans l **faire**
check(S_i, e)
sinon g.satisfiable = faux
FinFonction

```
Fonction handleInformation(MsgInfo e)
 si e.scenario ≠ NULL
 alors si e.results = reconnu
        alors si e.scenario est utilisé par événement e
              alors l = liste de Scenario utilisant e
                    pour chaque S_i dans l faire
                          check(S_i, e)
FinFonction
```

```
Fonction handleRefrainingRequest(MsgInfo e)
 b = but relatif à e.context
 pour chaque Goal b_i dans b.subgllist faire
        si b_i est dépendant
        alors ajouter entrée dans mentalstate.tobesent:
              interruption de reconnaissance
              répéter même traitement que b pour b_i
              supprimer b
FinFonction
```

Gestion des buts

La méthode *handleGoals* vérifie quels sont les *Goal* qui doivent être supprimés de la listes *goals* dans l'état mental.

```
Fonction handleGoals(liste de Goal mentalstate.goals)
 pour chaque b_i dans mentalstate.goals faire
   si b_i.satisfiable = faux
   alors si b_i est externe
         alors ajouter entrée dans mentalstate.tobesent : Echec
         pour chaque b_j dans b_i.subgoallist faire
               si b_j est dépendant
               alors ajouter entrée dans mentalstate.tobesent :
                     interruption reconnaissance
         switchOffC(b_i.scenario)
         switchOffD(b_i.scenario)
         switchOffH(b_i.scenario)
         supprimer b_i
   si b_i.satisfied = vrai
   alors si b_i est externe
         alors ajouter entrée dans mentalstate.tobesent : scénario reconnu
         switchOffC(b_i.scenario)
         switchOffD(b_i.scenario)
```

```
        switchOffH(b_i.scenario)
        supprimer b_i
    si b_i.delay a expiré
    alors si SysEvent e attendu dans b_i.scenario utilise un Scenario
        alors date observation de e = fin de b_i.delay
            /* raisonnement hypothétique */
            mise à jour contraintes de b_i.scenario
FinFonction
```

10.5.5 Planification

La fonction de planification est définie par la méthode *Plan* :

```
Fonction Plan(Goal g)
 Créer plan P
 si g.scenario ∈ mentalstate.domain
 alors pour chaque e_i en attente dans g.scenario.events faire
        si e_i utilise scéanrio s_i
        alors  créer entrée ent dans P
 si P.entries = NULL
 alors supprimer P
FinFonction
```

La fonction *handlePlans* est chargée de l'exécution des entrées complétées dans les plans de la liste *plans* dans l'état mental.

```
Fonction handlePlans(liste de Plan mentalstate.plans)
 pour chaque p_i dans mentalstate.plans faire
    si p_i.entries = NULL
    alors supprimer p_i
    sinon pour chaque ent_j dans p_i faire
      si ent_j.delay neq NULL
      alors id = chooseAgent(s_i)
            si id ≠ self
            alors  si checkD(ent_j.scenario) = vrai
                   alors  construire une entrée MsgInfo ent
                          ajouter ent dans mentalstate.tobesent
                          construire but pour ent_j.scenario
                          switchOnD(p_i.scenario, ent_j.scenario)
                   sinon supprimer ent_j
            sinon supprimer ent_j
FinFonction
```

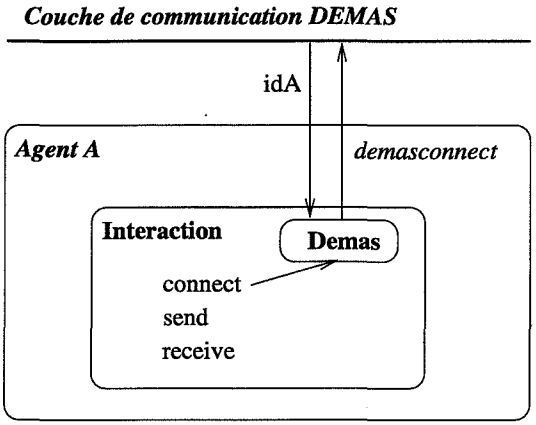


Figure 10.12 : Connexion d'un agent à la couche de communication *Demas*.

10.6 Lancement du système

Un utilisateur a la possibilité de lancer un ensemble d'agents sur différentes machines. A chaque agent est associé un fichier définissant son domaine de responsabilité individuelle. Au lancement, les agents ne peuvent pas s'envoyer de messages. Ils doivent d'abord se connecter à la couche de communication.

Connection

La classe *Interaction* permet à l'agent de se connecter à la couche de communication *Demas* faisant partie d'une plateforme développée au sein du laboratoire (figure 10.12). La couche *Demas* est développée en JAVA. Une interface écrite en C++ permet l'accès à cette couche par des objets écrits en C++.

La méthode *connect* de la classe *Interaction* appelle la méthode *demasconnect* qui fait une demande de connexion auprès de la couche *Demas*. L'agent est connecté et se voit attribuer un numéro d'identification unique (idA).

Présentation

Afin de communiquer avec les autres agents, un agent doit connaître leurs numéros d'identification. Pour ce faire, la couche *Demas* lui offre la possibilité d'envoyer un message à tous les agents connectés sans connaître leurs numéros d'identification. Après sa connexion, chaque agent envoie à tous les autres, un message contenant son identificateur et son numéro d'identification. Les numéros d'identification étant connus, les agents peuvent à présent s'envoyer des messages d'une manière nominative.

10.7 Interface

Toutes les classes décrites plus haut, sont dotées d'un service de notification. Il est modélisé par la classe *NotificationService*. Cette classe permet au niveau de chaque instance créée (*Agent*, *MentalState*, *Goal*, *Plan*, *Scenario*, ...) de notifier toutes les interfaces (*API: Application Programming Interface*) connectées à l'agent. Les *API* peuvent être connectées en local ou à distance.

Au début de chaque cycle de l'agent, on teste si de nouvelles *API* sont connectées à l'agent. Si c'est le cas, elles sont notifiées de l'état actuel de tous les composants de l'agent et par la suite, ces *API* sont notifiées régulièrement quand un changement se produit au niveau des composants.

La notification suit un protocole de communication entre les composants de l'agent et les *API* connectées. En effet, quand l'état d'un objet change, le service de notification au niveau de cet objet, se charge d'envoyer un message à toutes les *API* connectées. Ces dernières interprètent le message et mettent à jour les objets graphiques en fonction du contenu du message (cf. figure 10.13). La description du langage d'interaction entre un agent et une *API* est hors du propos de ce document.

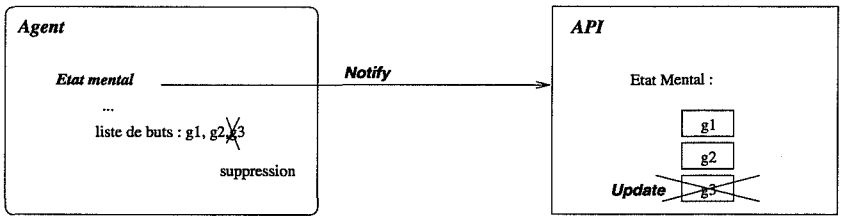


Figure 10.13 : Communication entre un agent et une *API* connectée

Une *API* est modélisée par la classe *AgentView*. Cette classe hérite des classes *NotifiedView* et *AgentViewApplication* (figure 10.14).

La classe *NotifiedView* assure la communication par *socket* entre l'interface et le service de notification (*NotificationService*) de l'agent auquel elle est connectée. La classe *AgentViewApplication* est capable d'interprêter les messages reçus par la classe *NotifiedView* et de mettre à jour les objets graphiques de l'interface. La classe *AgentViewApplication* permet également l'enregistrement d'une session de communication entre un agent et l'interface. Ceci permet de redérouler cette session en se déplaçant sur l'échelle temporelle à différentes vitesses et dans les deux sens. L'annexe D présente un certain nombre de copies d'écrans, résultat du déroulement de l'exemple présenté dans la section suivante.

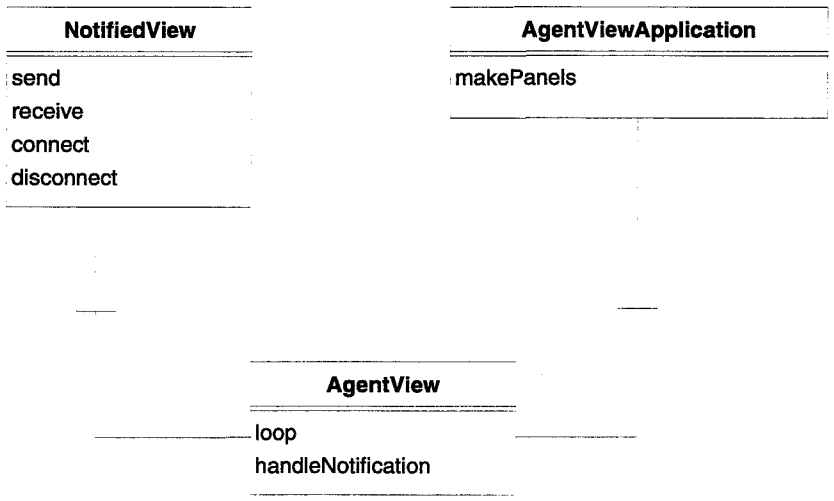


Figure 10.14 : Héritage multiple pour la classe *AgentView*

10.8 Exemple : supervision de la flotte de bus

Dans cette section, nous reprenons l'exemple décrit dans le chapitre précédent afin de montrer le fonctionnement interne des agents au cours du déroulement du scénario *Aller*. Nous nous plaçons dans le contexte de la section 9.6.1.

Le bus b_i est à l'arrêt au terminus A (événement 22).

Traitement de l'événement 22 par a_1

Nous allons suivre l'exécution d'un cycle de l'agent a_1 :

- l'exécution de la méthode *perceive* aura pour effet, de rajouter l'événement 22 dans la liste *perceivedevents* de son état mental ;
- l'exécution de la méthode *receive* n'aura aucun effet, car l'agent n'a pas reçu de nouveaux messages ;
- en exécutant la méthode *decide*, l'agent va exécuter dans l'ordre les méthodes *handleEvents*, *handleReceivedEntries*, *handleGoals*, *handleplans* :
  - la méthode *handleEvents* traite la liste des événements *perceivedevents* qui contient le seul événement 22. $l_i = \{Terminus\ A\}$. Appel de la fonction *check*(*Terminus A*, event 23) :
    - *Terminus A*.activated = faux ;
    - *Terminus A*.precond = vrai ;

* l'appel de la fonction *chooseAgent*(*Terminus A*) rend l'identificateur de l'agent a_1 car il est le seul responsable du scénario *Terminus A* ;
* *Terminus A*.activated <- vrai ;
* un Goal $g_{TerminusA}$ est ajouté à la liste $a_1.mentalsate.goals$;
* *Terminus A* n'admettant pas de sous-scénario, le but $g_{TerminusA}$ n'admet pas de plan, suite à l'appel de la méthode *plan* ;
* l'appel de la fonction *switchOnC* n'aura aucun effet car il n'existe aucun autre agent qui soit en concurrence avec l'agent a_1 sur le scénario *Terminus A* ;
* *Terminus A* étant activé, ces contraintes sont mises à jour : attente des événements 18 et 23 avec un délai de 10 minutes ;
* l'événement 22 n'étant pas le dernier du scénario, ceci termine l'exécution de la fonction *check*. L'événement 22 est supprimé et ceci termine l'exécution de la méthode *handleEvents*.

– l'appel de *handleReceivedEntries* n'aura aucun effet, car aucun message n'a été reçu ;

– appel de la méthode *handleGoals* :
$g_{TerminusA}$.satifiable = vrai
$g_{TerminusA}$.satisfied = faux
donc $g_{TerminusA}$ est maintenu dans la liste mentalstate.goals.

– l'agent n'ayant aucun *Plan* dans la liste *mentalstate.plans*, la méthode *handleplans* n'aura aucun effet. Ceci termine l'exécution de la méthode *decide*.

- la méthode *send* n'a aucun effet car l'agent n'a aucun message à envoyer. L'exécution de cette méthode termine un cycle de l'agent a_1.

Reconnaissance du scénario *Terminus A* par a_1

Nous nous déplaçons sur l'échelle du temps vers 15h07 (l'événement 18 a déjà été observé et traité), le bus b_i quitte le terminus A (événement 23).

Nous allons réexaminer le cycle de l'agent a_1 sachant que les seuls appels de méthodes ayant un effet, sont *check*(*Terminus A*, event 23) et *send* :

check(*Terminus A*, event 23) :

- *Terminus A*.activated = vrai ;
- les contraintes de *Terminus A* sont mises à jour : pas d'événements attendus et l'événement 23 respecte les délais ;
- l'événement 23 est le dernier du scénario ;

- *Terminus A* est reconnu ;
- *Terminus A*.temporalspace = 0 ;
- $g_{TerminusA}$ n'a pas de sous-buts, donc $g_{TerminusA}$.satisfied = vrai ;
- *Terminus A est utilisé* par le *SysEvent* 27 ;
- les *Scenario Aller* et *Retour* sont les seuls scénarios concernés par l'événement 27 ;
- pour chacun de ces scénarios, nous appelons la fonction *check* : *check*(*Aller*, event 27), *check*(*Retour*, event 27).
- la liste l_2 est composée des agents a_2 et a_3 car nous avons : $a_2\mathcal{D}^{Aller}_{Terminus\ A}a_1$, $a_2\mathcal{D}^{Retour}_{Terminus\ A}a_1$, $a_3\mathcal{D}^{Aller}_{Terminus\ A}a_1$ et $a_3\mathcal{D}^{Retour}_{Terminus\ A}a_1$;
- deux entrées *MsgInfo* ent_1 et ent_2 sont construites afin d'informer respectivement les agents a_2 et a_3 de la reconnaissance de *Terminus A*. Ces entrées sont rajoutées à la liste *mentalstate.tobesent* ;

Nous allons examiné le résultat des appels *check*(*Aller*, event 27) et *check*(*Retour*, event 27) :

- *check*(*Aller*, event 27) :
  - *Aller*.activated = faux ;
  - *Aller*.precond = vrai ;
  - l'appel de la fonction *chooseAgent* retourne l'identificateur de l'agent a_3, car il a le moins de pouvoir vis-à-vis de la reconnaissance de *Aller* ;
  - l'agent a_1 active la relation $a_1\mathcal{C}_{Aller}a_3$;
- *check*(*Retour*, event 27) :
  - *Retour*.activated = faux ;
  - *Aller*.precond = faux ;

Appel de la méthode *send* :

- la liste *mentalstate.tobesent* contient deux entrées ent_1 et ent_2 ;
- l'entrée ent_1 est transformée en un message en langage $\mathcal{L}$. Ce message est ensuite envoyé à l'agent a_2 ;
- l'entrée ent_2 est transformée en un message en langage $\mathcal{L}$. Ce message est ensuite envoyé à l'agent a_3 ;

Réception du message d'information par a_2

L'exécution d'un cycle par l'agent a_2 donne le résultat suivant :

- la méthode *perceive* n'aura aucun effet ;
- la méthode *receive* transforme le message envoyé par a_1 en une entrée e de type *MsgInfo* et ajoute cette entrée à la liste *mentalstate.received* ;
- la méthode *decide* :
  - *handleEvents* n'a aucun effet ;
  - *handleReceivedEntries*($\{e\}$) fait appel à *handleInformation*(e) :
    - e.scenario = *Terminus A* ;
    - e.results = reconnu ;
    - e.scenario *est utilisé par SysEvent* 27 ;
    - le *Scenario Aller* est le seul scénario concerné par cet événement ;
    - appel de la fonction *check*(*Aller*, event 27) :
      - *Aller*.activated = faux ;
      - *Aller*.precond = vrai ;
      - *chooseAgent* retourne a_3. L'agent a_2 active la relation $a_2\mathcal{C}_{Aller}a_3$ et ceci termine l'exécution de la méthode *check*.
- les méthodes *handleGoals* et *handlePlans* n'ont aucun effet.

Réception du message d'information par a_3

L'agent a_3 suit les mêmes étapes que l'agent a_2 jusqu'à l'appel de la méthode *check*(*Aller*, event 27) :

- *Aller*.activated = faux ;
- *Aller*.precond = vrai ;
- *chooseAgent* retourne a_3 ;
- *Aller*.activated <- vrai ;
- création d'un *Goal* g_{Aller} ;
- la méthode *Plan* crée un plan P_{Aller} qui admet une seule entrée e_1 relative au sous-scénario *Terminus D* ;
- l'appel de la méthode *switchOnC* aura pour effet d'activer les relations $a_3\mathcal{C}_{Aller}a_1$ et $a_3\mathcal{C}_{Aller}a_2$ dans le réseau $\mathcal{C}$ de a_3. Ces deux relations étant actives, l'appel de la méthode *checkCC* n'activera pas la relation $a_2\mathcal{C}_{Aller}a_3$ dans la description externe de a_2, ni la relation $a_3\mathcal{C}_{Aller}a_2$ dans celle de a_3. L'appel de la méthode *checkCX* n'aura pas d'effets ;

- *Aller* est actif. Ses contraintes sont mises à jour ;
- l'événement 27 n'est pas le dernier dans le scénario ;
- l'événement attendu est le *SysEvent* 18. Il n'utilise aucun *Scenario.* Donc, l'exécution de la méthode *check* est terminé ;
- l'appel de la méthode *handleGoals* concerne uniquement le but g_{Aller} : g_{Aller}.satisfiable = vrai et g_{Aller}.satisfied = faux, donc ce but est maintenu dans la liste *mentalstate.goals* ;
- l'appel de la méthode *handlePlans* concerne le seul plan P_{Aller}. Il contient une seule entrée qui n'admet pas encore de délai, donc cette entrée n'est pas exécutée.

Reconnaissance du scénario *Terminus D*

Nous nous déplaçons sur l'échelle temporelle vers 15h27 (section 9.6.2). L'événement observé est le *SysEvent* 20. Cet événement permet à l'agent a_2 d'activer le scénario *Terminus D.* L'agent a_3 est en attente de l'événement 28 qui correspond à l'activation du scénario *Terminus D.* Ainsi, à la dernière exécution de la méthode *check* par l'agent a_3, le délai [0,0] est ajouté dans l'entrée du plan P_{Aller}. Un *Goal* $g_{Terminus\ D}$ est créé. Ce but est un sous-but du g_{Aller}. Ce sous-but n'admet pas de plan.

L'exécution de *handleGoals* maintient le but et le sous-but dans la liste *mentalstate.goals.*

L'exécution de *handlePlans* permet l'exécution de l'unique entrée dans le plan P_{Aller}. Une entrée *MsgInfo* est ajoutée à la liste *mentalstate.tobesent* et la relation $a_3 \mathcal{D}^{Aller}_{TerminusD} a_2$ est activée. L'entrée du plan est supprimée et le plan l'est également

L'exécution de la méthode *send* permet l'envoi d'un message de requête à l'agent a_2.

L'exécution à nouveau de la méthode *handleGoals* permet de se rendre compte que $g_{Terminus\ D}$.delay a expiré. L'agent a_3 utilise son raisonnement hypothétique et considère que l'événement 28 attendu est arrivé en respectant le délai [0,0].

A présent, l'événement attendu pour le scénario *Aller* est l'événement 29. Ce *SysEvent* correspond à la reconnaissance du *Scenario Terminus D.* Le plan P_{Aller} étant supprimé l'agent se met en attente de la réponse de a_2.

A 15h32:05 le scénario *Terminus D* est reconnu par l'agent a_2. Il envoie un message de réponse à l'agent a_3.

L'agent a_3 reçoit le message (*receive*). Une entrée e_2 est ajoutée à la liste *mentalstate.received*. L'agent a_3 exécute ensuite la méthode *handleResponse* :

- e_2.scenario = *Terminus D* ;
- *Terminus D* est reconnu ;
- $g_{TerminusD}$.satisfied <- vrai ;
- l'appel de la fonction *check*(*Aller*, event 29) établit la reconnaissance du scénario *Aller*. Le but g_{Aller} n'ayant pas de sous-buts non satisfaits ($g_{Terminus\ D}$.satisfied = vrai), il est satisfait ;
- l'appel de la méthode *handleGoals* permet la suppression des buts g_{Aller} et $g_{TerminusD}$. Les relations de dépendance sont désactivées (*switchOffC*, *switchOffD*, *switchOffH*) ;
- l'appel de la méthode *switchOffC*, fait appel à la méthode *checkCC*, qui provoque l'envoi de deux messages aux agent a_1 et a_2 pour les prévenir de la reconnaissance du scénario *Aller*. Ceci aura pour effet, de désactiver leurs relations C.

10.9 Discussion

Dans ce chapitre, nous avons proposé un modèle d'implémentation pour une société d'agents responsable de la supervision par reconnaissance de scénarios. Le raisonnement de l'agent tient compte du facteur temporel, surtout au niveau de l'activation des différentes relations de dépendance. Cette prise en compte est également visible au niveau de la gestion des événements et de la gestion des buts.

Les relations d'utilisation entre objets ont permis l'accès direct à certain nombre d'objets à partir d'autres objets. Ces objets sont souvent la représentation d'autres objets qui appartiennent à d'autres agents. En effet, la classe *RemoteScenario* permet de créer des objets qui correspondent à des scénarios appartenant aux domaines d'autres agents. La classe *MentalState* sert à la fois à modéliser l'état mental de l'agent, ainsi que les descriptions externes.

Pour cette raison, il faut être vigilant au niveau de ces représentations. Une description externe ne doit pas contenir des informations contradictoires avec l'état mental qu'elle représente. Or, la mise à jour de l'état mental diffère de la mise à jour d'une description externe et l'origine des informations est différente. Un agent met à jour son état mental en fonction des scénarios

qui sont actifs et les messages provenant d'autres agents, alors qu'une description externe est mise à jour à partir d'informations initiales telles que la description des scénarios, la conjonction des relations de dépendance et enfin, à partir des messages provenant des autres agents. De ce fait, une relation déduite entre deux agents entraînant la mise à jour de leurs descriptions externes, doit être représentée dans les états mentaux respectifs des deux agents. Afin de garantir la cohérence des différentes représentations, les informations sur les scénarios communiquées aux agents à leur création, doivent être complètes et identiques pour tous les agents.

Chapitre 11

Conclusions et perspectives

Dans ce travail, nous avons étudié les aspects temporels dans un système multi-agent. L'objectif général étant d'intégrer le temps dans le fonctionnement d'une société d'agents, nous avons défini un modèle d'organisation temporelle des agents, un modèle d'interaction temporelle et un modèle d'agent temporel.

11.1 Principales contributions de cette thèse

Dans cette thèse, nous proposons un **modèle d'organisation** basé sur les relations de dépendance et de pouvoir entre agents. Dans ce modèle, les relations de dépendance évoluent au cours du temps et cette évolution est exprimée explicitement par un facteur temporel au niveau de la définition des relations. Nous définissons trois relations : la *relation de concurrence*, la *relation de besoin* et la *relation d'aide*. Un agent utilise ces relations dans son raisonnement social :

- la relation de concurrence permet une bonne répartition des tâches entre agents et permet également de trouver une alternative dans le cas où un agent est incapable momentanément d'exécuter une tâche ;
- la relation de besoin définit une liaison entre les tâches de deux agents. Grâce à cette relation, un agent peut établir dynamiquement un état de l'activité de l'agent qui lui est relié par cette relation ;
- la relation d'aide est définie dans l'objectif de minimiser les communications entre les agents. Elle devient importante dans un contexte où les agents sont chargés d'exécuter des tâches sous contraintes temporelles ;
- la relation de pouvoir définit le critère pour éliminer une situation de concurrence sur une tâche entre deux agents.

L'utilisation des relations de dépendance donne à un agent la possibilité de positionner son activité dans le temps par rapport à celles des autres agents. Au cours d'un processus de coopération, cela permet à un agent d'une part, de prévoir l'évolution des activités des autres et d'autre part, d'adapter son activité en fonction de celles des autres (section 9.7.2).

Les relations de dépendance sont donc une preuve de l'existence de liens entre les activités internes d'un ensemble d'agents impliqué dans une coopération. Ces activités sont souvent contraintes par/dans le temps.

Afin d'utiliser ces liens dans leurs activités sociales, les agents doivent pouvoir communiquer. Ils doivent également pouvoir exprimer à travers le langage d'interaction, toutes les contraintes temporelles relatives à leurs activités internes.

Le **modèle d'interaction** proposé permet l'expression de contraintes temporelles au niveau du langage d'interaction et de sa sémantique. Il est basé sur les actes de langage. L'ensemble de protocoles que nous avons défini, est un moyen de contrôler les conversations menées par les agents.

Après avoir défini les modèles d'organisation et de communication, nous avons proposé un **modèle d'agent** où sont pris en compte les aspects définissant le niveau individuel de l'agent, et les aspects définissant le niveau social au sein de cet agent. Le temps est pris en compte explicitement à la fois au niveau individuel et au niveau social :

- au niveau individuel, on retrouve les contraintes temporelles relatives aux tâches appartenant au domaine de responsabilité individuelle de l'agent ;
- au niveau social, on retrouve les contraintes relatives aux niveau individuel de l'agent et celles relatives aux niveaux individuels d'autres agents. Ces contraintes sont exprimées par les relations de dépendance de l'agent (sections 5.3.1, 5.3.2 et 5.3.3) ;
- la communication est un moyen direct pour mettre en relation les contraintes des niveaux individuels des deux agents.

La définition des modèles d'organisation, d'interaction et d'agent a conduit au développement du système *STARS* qui présente deux propriétés intéressantes. D'une part, il permet à travers l'exemple du chapitre 9 de montrer l'intérêt qu'apporte chacun des modèles précédents grâce au caractère dynamique de la reconnaissance de scénarios temporels, et d'autre part, il permet

de montrer que la distribution du processus de supervision admet un certain nombre d'avantages :

- du point de vue de la modélisation, l'introduction de relations entre scénarios (scénario/sous-scénario, opposition), facilite la modélisation des modes de fonctionnement de systèmes ayant une structure hiérarchique au niveau physique et/ou au niveau fonctionnel ;
- d'un point de vue performance, la structuration des scénarios en sous-scénarios permet de réduire le nombre de nœuds dans les graphes de contraintes. Ce résultat est intéressant sachant que les algorithmes de reconnaissance ont une complexité en $O(n^3)$. La distribution des scénarios contribue également à la résolution de problèmes liés à la supervision tels que les problèmes d'engorgement et de masquage.

11.2 Perspectives

Notre définition du niveau social s'appuie essentiellement sur les relations de dépendance. Dans la définition du modèle d'agent, nous avons utilisé les notions d'*engagement social* et de *responsabilité commune*. Ces notions ne jouent pas un rôle significatif dans le modèle à cause d'une définition stricte de la gestion des buts. En effet, un agent recevant un message de requête d'exécution d'une tâche, n'a qu'une possibilité : chercher à satisfaire le but relatif à la tâche. Ainsi, il est sans objet d'associer un engagement social à un but. Dans certains contextes, les agents peuvent être amenés à refuser ou négocier une requête. Dans ce cas, il est important de tenir compte explicitement des engagements sociaux afin de compléter le modèle social basé sur les relations de dépendance.

Le modèle de communication pourrait également être amélioré. En effet, dans le modèle actuel, nous utilisons seulement deux actes de langage : un *acte directif* et un *acte assertif*. Une extension du modèle devrait permettre l'utilisation des actes *expressif*, *promissif* et *déclaratif*. Le langage d'interaction pourrait également être amélioré, en lui permettant l'expression de contraintes temporelles plus complexes tels que : *périodiquement*, *dès que possible*, *dès qu'un fait est établi* ...

La définition actuelle de la fonction de planification est statique. Ceci est dû à l'unicité de la décomposition d'une tâche. En généralisant le modèle des tâches, celles-ci pourront être construites dynamiquement à partir d'autres sous-tâches. Dans ce cas la planification devient dynamique.

Actuellement, nous avons la même architecture pour tous les agents. De ce fait, ces agents sont capables d'exécuter le même type de tâches. Cette ar-

chitecture devrait être améliorée afin de permettre aux agents d'effectuer des tâches de natures différentes. Par exemple, dans le système *STARS* tous les agents utilisent exclusivement la reconnaissance de scénario pour la supervision. Il existe des situations où d'autres approches sont capables de donner de meilleurs résultats que la reconnaissance de scénarios. En concevant une société multi-modèle où les agents utilisent des approches différentes pour la supervision, d'autres règles de calcul des relations de dépendance pourront être introduites. Ainsi, d'autres règles d'utilisation et de raisonnement sur ces relations peuvent être mises en place. La complémentarité des différents modèles utilisés permettrait d'obtenir une approche plus complète et plus adaptée de la tâche globale de supervision d'un système dynamique.

Annexes

Annexe A

Preuves

Dans cette annexe, nous présentons les démonstrations des différentes formules utilisées pour définir les conjonctions des relations de dépendance (chapitre 5).

A.1 Conjonction $\mathcal{CC}$

Formule 1 :

$$a_u \mathcal{C}_{t_i} a_v \ \wedge \ a_u \mathcal{C}_{t_i} a_w \Longrightarrow a_v \mathcal{C}_{t_i} a_w$$

Nous rappelons la définition de $a_u \mathcal{C}_{t_i} a_v$:

$$a_u \mathcal{C}_{t_i} a_v \Longleftrightarrow t_i \in r(a_u) \cap r(a_v)$$

Démonstration de la formule 1 :

$$\begin{aligned} a_u \mathcal{C}_{t_i} a_v \ \wedge \ a_u \mathcal{C}_{t_i} a_w &\Longrightarrow t_i \in r(a_u) \cap r(a_v) \ \wedge \ t_i \in r(a_u) \cap r(a_w) \\ &\Longrightarrow t_i \in r(a_v) \wedge t_i \in r(a_w) \\ &\Longrightarrow t_i \in r(a_v) \cap r(a_w) \\ &\Longrightarrow a_v \mathcal{C}_{t_i} a_w \ \square \end{aligned}$$

Formule 2 :

$$a_u \mathcal{C}_{t_i}(I) a_v \ \oplus \ a_u \mathcal{C}_{t_i}(I) a_w \Longrightarrow a_v \mathcal{C}_{t_i}(I) a_w$$

Nous rappelons la définition de $a_u \mathcal{C}_{t_i}(I) a_v$:

$$a_u \mathcal{C}_{t_i}(I) a_v \Longleftrightarrow a_u \mathcal{C}_{t_i} a_v \ \wedge \ (a_u : [t_i]_J \ \oplus \ a_v : [t_i]_J) \ \wedge \ I \ \{d, s, f, e\} \ J$$

Démonstration de la formule 2 :

$$a_u\mathcal{C}_{t_i}(I)a_v \oplus a_u\mathcal{C}_{t_i}(I)a_w \\ \Longrightarrow a_u\mathcal{C}_{t_i}a_v \wedge (a_u : [t_i]_J \oplus a_v : [t_i]_J) \wedge I\ \{d,s,f,e\}\ J \\ \oplus a_u\mathcal{C}_{t_i}a_w \wedge (a_u : [t_i]_J \oplus a_w : [t_i]_J) \wedge I\ \{d,s,f,e\}\ J$$

La formule 1 étant vérifiée, nous avons $a_u\mathcal{C}_{t_i}a_v$ et $a_u\mathcal{C}_{t_i}a_w$ d'où :

$$a_u\mathcal{C}_{t_i}a_v \wedge (a_u : [t_i]_J \oplus a_v : [t_i]_J) \wedge I\ \{d,s,f,e\}\ J \oplus \\ a_u\mathcal{C}_{t_i}a_w \wedge (a_u : [t_i]_J \oplus a_w : [t_i]_J) \wedge I\ \{d,s,f,e\}\ J \Longrightarrow \\ ((a_u : [t_i]_J \oplus a_v : [t_i]_J) \oplus (a_u : [t_i]_J \oplus a_w : [t_i]_J)) \wedge I\ \{d,s,f,e\}\ J$$

Il est facile de démontrer que $(a \oplus b) \oplus (a \oplus c) \Longrightarrow b \oplus c$, d'où :

$$(a_u : [t_i]_J \oplus a_w : [t_i]_J) \wedge I\ \{d,s,f,e\}\ J$$

D'après la formule 1, nous avons $a_v\mathcal{C}_{t_i}a_w$, d'où :

$$a_v\mathcal{C}_{t_i}a_w \wedge (a_u : [t_i]_J \oplus a_w : [t_i]_J) \wedge I\ \{d,s,f,e\}\ J \Longrightarrow a_v\mathcal{C}_{t_i}(I)a_w\ \square$$

A.2 Conjonction $\mathcal{CX}$

Formule 3 :

$$a_u\mathcal{C}_{t_i}a_v \wedge a_u\mathcal{X}^{t_i}_{t_j}a_w \Longrightarrow a_v\mathcal{X}^{t_i}_{t_j}a_w$$

Démonstration de la formule 3, cas $\mathcal{X}$=$\mathcal{D}$:

Nous rappelons la définition de $a_u\mathcal{D}^{t_i}_{t_j}a_v$:

$$a_u\mathcal{D}^{t_i}_{t_j}a_v \Longleftrightarrow t_i \in r(a_u) \wedge t_j \in r(a_v) \cap s(t_i) \wedge t_j \notin r(a_u)$$

$$a_u\mathcal{C}_{t_i}a_v \wedge a_u\mathcal{D}^{t_i}_{t_j}a_w \Longrightarrow t_i \in r(a_u) \cap r(a_v) \wedge \\ t_i \in r(a_u) \wedge t_j \in r(a_w) \cap s(t_i) \wedge t_j \notin r(a_u) \\ \Longrightarrow t_i \in r(a_v) \wedge t_j \in r(a_w) \cap s(t_i)$$

Nous avons également $t_j \notin r(a_v)$ car sinon, nous aurons $a_u\mathcal{D}^{t_i}_{t_j}a_v$ et nous nous trouverons dans le contexte d'une conjonction $\mathcal{XX}$.
Nous obtenons donc :

$$t_i \in r(a_v) \wedge t_j \in r(a_w) \cap s(t_i) \wedge t_j \notin r(a_v) \Longrightarrow a_v\mathcal{D}^{t_i}_{t_j}a_w\ \square$$

Démonstration de la formule 3, cas $\mathcal{X}$=$\mathcal{H}$:

Nous rappelons la définition de $a_u\mathcal{H}^{t_i}_{t_j}a_v$:

$$a_u \mathcal{H}_{t_j}^{t_i} a_v \Longleftrightarrow t_i \leftrightarrow t_j \ \wedge \ t_i \in r(a_u) - r(a_v) \ \wedge \ t_j \in r(a_v) - r(a_u)$$

$$\begin{aligned} a_u \mathcal{C}_{t_i} a_v \ \wedge \ a_u \mathcal{H}_{t_j}^{t_i} a_w &\Longrightarrow t_i \in r(a_u) \cap r(a_v) \ \wedge \\ &\quad t_i \leftrightarrow t_j \ \wedge \ t_i \in r(a_u) - r(a_w) \ \wedge \ t_j \in r(a_w) - r(a_u) \\ &\Longrightarrow t_i \leftrightarrow t_j \ \wedge \ t_i \in r(a_v) \ \wedge \ t_i \notin r(a_w) \ \wedge \ t_j \in r(a_w) \end{aligned}$$

Nous avons également $t_j \notin r(a_v)$ car sinon, nous aurons $a_u \mathcal{H}_{t_j}^{t_i} a_v$ et nous nous trouverons dans le contexte d'une conjonction $\mathcal{XX}$.
Nous obtenons donc :

$$\begin{aligned} t_i \leftrightarrow t_j \ \wedge \ t_i \in r(a_v) \ \wedge \ t_i \notin r(a_w) \ \wedge \ t_j \in r(a_w) \ \wedge \ t_j \notin r(a_v) &\\ \Longrightarrow t_i \leftrightarrow t_j \ \wedge \ t_i \in r(a_v) - r(a_w) \ \wedge \ t_j \in r(a_w) - r(a_v) &\\ \Longrightarrow a_v \mathcal{H}_{t_j}^{t_i} a_w \ \square & \end{aligned}$$

Formule 4:

$$a_u \mathcal{C}_{t_i}(I) a_v \ \wedge \ a_u \mathcal{X}_{t_j}^{t_i} a_w \ \wedge \ a_u >_{t_i} (J) a_v \ \wedge \ J\{o, m, di, e\} I \Longrightarrow a_v \mathcal{X}_{t_j}^{t_i}(I) a_w$$

Démonstration de la formule 4, cas $\mathcal{X}$=$\mathcal{D}$:

Nous rappelons la définition de $a_u \mathcal{D}_{t_j}^{t_i}(I) a_v$:

$$a_u \mathcal{D}_{t_j}^{t_i}(I) a_v \Longleftrightarrow a_u \mathcal{D}_{t_j}^{t_i} a_v \ \wedge \ a_v : [t_j]_J \ \wedge \ a_u : [t_i]_K \ \wedge \ I = J \cap K$$

Nous avons :
$a_u \mathcal{C}_{t_i}(I) a_v \Longrightarrow a_u \mathcal{C}_{t_i} a_v \ \wedge \ (a_u : [t_i]_J \ \oplus \ a_v : [t_i]_J) \ \wedge \ I \ \{d, s, f, e\} \ J$ (1)

D'après le résultat de la section 5.5, nous avons :

$$\forall a_v \mid a_u \mathcal{C}_{t_i}(I) a_v \ \wedge \ J\{si, di, fi, o, m\} I \ \wedge \ a_v >_{t_i} (J) a_u \Longrightarrow a_u : [t_i]_I$$

Ainsi, nous pouvons écrire :
$a_u >_{t_i} (J) a_v \Longrightarrow \neg a_u : [t_i]_J$ (2)

(1) $\Longrightarrow (a_u : [t_i]_J \ \oplus \ a_v : [t_i]_J)$
(2) $\Longrightarrow \neg a_u : [t_i]_J$
(1)+(2) $\Longrightarrow a_v : [t_i]_J$
Or $t_j \in s(t_i) \Longrightarrow \exists K \mid a_w : [t_j]_K$
En conclusion, nous avons :
$a_v : [t_i]_J \ \wedge \ a_w : [t_j]_K$
Ayant la relation $a_v \mathcal{D}_{t_j}^{t_i} a_w$ (formule 3, cas $\mathcal{X} = \mathcal{D}$), nous pouvons écrire :
$a_v \mathcal{D}_{t_j}^{t_i} a_w \ \wedge \ a_v : [t_i]_J \ \wedge \ a_w : [t_j]_K \Longrightarrow a_v \mathcal{D}_{t_j}^{t_i}(I = J \cap K) a_w \ \square$

Démonstration de la formule 4, cas $\mathcal{X}$=$\mathcal{H}$:

Nous rappelons la définition de $a_u \mathcal{H}_{t_j}^{t_i}(I) a_v$:

$$a_u \mathcal{H}^{t_i}_{t_j}(I) a_v \Longleftrightarrow a_u \mathcal{H}^{t_i}_{t_j} a_v \ \wedge \ (a_u : [t_i]_J \ \vee \ a_v : [t_j]_J) \ \wedge \ I\{s,d,f,e\}J$$

Nous avons :
(1)+(2) $\Longrightarrow a_v : [t_i]_J \Longrightarrow (a_v : [t_i]_J \ \vee \ a_w : [t_j]_J)$
Ayant la relation $a_u \mathcal{H}^{t_i}_{t_j} a_v$ (formule 3, cas $\mathcal{X} = \mathcal{H}$), nous pouvons écrire :
$a_u \mathcal{H}^{t_i}_{t_j} a_v \ \wedge \ (a_v : [t_i]_J \ \vee \ a_w : [t_j]_J) \Longrightarrow a_v \mathcal{H}^{t_i}_{t_j}(I\{s,d,f,e\}J) a_w$ □

A.3 Conjonction $\mathcal{X}\mathcal{X}$

Formule 5 :

$$a_u \mathcal{X}^{t_i}_{t_j} a_v \ \wedge \ a_u \mathcal{X}^{t_i}_{t_j} a_w \Longrightarrow a_v \mathcal{C}_{t_j} a_w$$

Démonstration de la formule 5, cas $\mathcal{X}$=$\mathcal{D}$:

$$\begin{aligned}
&a_u \mathcal{D}^{t_i}_{t_j} a_v \Longrightarrow t_i \in r(a_u) \ \wedge \ t_j \in r(a_v) \cap s(t_i) \ \wedge \ t_j \notin r(a_u)\\
&a_u \mathcal{D}^{t_i}_{t_j} a_w \Longrightarrow t_i \in r(a_u) \ \wedge \ t_j \in r(a_w) \cap s(t_i) \ \wedge \ t_j \notin r(a_u)\\
&a_u \mathcal{D}^{t_i}_{t_j} a_v \ \wedge \ a_u \mathcal{D}^{t_i}_{t_j} a_w \Longrightarrow t_i \in r(a_u) \ \wedge \ t_j \in r(a_v) \cap s(t_i) \ \wedge\\
&\qquad\qquad t_j \notin r(a_u) \ \wedge \ t_j \in r(a_w) \cap s(t_i)\\
&\qquad\qquad \Longrightarrow t_j \in r(a_v) \ \wedge \ t_j \in r(a_w)\\
&\qquad\qquad \Longrightarrow t_j \in r(a_v) \cap r(a_w)\\
&\qquad\qquad \Longrightarrow a_v \mathcal{C}_{t_j} a_w \ \square
\end{aligned}$$

Démonstration de la formule 5, cas $\mathcal{X}$=$\mathcal{H}$:

$$\begin{aligned}
&a_u \mathcal{H}^{t_i}_{t_j} a_v \Longrightarrow t_i \leftrightarrow t_j \ \wedge \ t_i \in r(a_u) - r(a_v) \ \wedge \ t_j \in r(a_v) - r(a_u)\\
&a_u \mathcal{H}^{t_i}_{t_j} a_w \Longrightarrow t_i \leftrightarrow t_j \ \wedge \ t_i \in r(a_u) - r(a_w) \ \wedge \ t_j \in r(a_w) - r(a_u)\\
&a_u \mathcal{H}^{t_i}_{t_j} a_v \ \wedge \ a_u \mathcal{H}^{t_i}_{t_j} a_w \Longrightarrow t_i \leftrightarrow t_j \ \wedge \ t_i \in r(a_u) - r(a_v) \ \wedge \ t_j \in r(a_v) - r(a_u)\\
&\qquad\qquad \wedge \ t_i \in r(a_u) - r(a_w) \ \wedge \ t_j \in r(a_w) - r(a_u)\\
&\qquad\qquad \Longrightarrow t_j \in r(a_v) \ \wedge \ t_j \in r(a_w)\\
&\qquad\qquad \Longrightarrow t_j \in r(a_v) \cap r(a_w)\\
&\qquad\qquad \Longrightarrow a_v \mathcal{C}_{t_j} a_w \ \square
\end{aligned}$$

Formule 6 :

$$a_u \mathcal{X}^{t_i}_{t_j} a_v \ \wedge \ a_u \mathcal{X}^{t_i}_{t_j}(I) a_w \Longrightarrow a_v \mathcal{C}_{t_j}(I) a_w$$

Démonstration de la formule 6, cas $\mathcal{X}$=$\mathcal{D}$:

$$\begin{aligned}
a_u \mathcal{D}^{t_i}_{t_j}(I) a_w &\Longrightarrow a_u \mathcal{D}^{t_i}_{t_j} a_w \ \wedge \ a_w : [t_j]_J \ \wedge \ a_u : [t_i]_K \ \wedge \ I = J \cap K\\
&\Longrightarrow a_w : [t_j]_J \ \wedge \ I \ \{d,s,f,e\} \ J
\end{aligned}$$

Nous pouvons également écrire :
$(a_w : [t_j]_J \ \oplus \ a_v : [t_j]_J) \ \wedge \ I \ \{d,s,f,e\} \ J$ (3)

D'après la formule 5, nous avons : $a_v \mathcal{C}_{t_j} a_w$ (4)
(3)+(4) $\Longrightarrow a_v \mathcal{C}_{t_j} a_w \wedge (a_w : [t_j]_J \oplus a_v : [t_j]_J) \wedge I \{d, s, f, e\} J$
$\Longrightarrow a_v \mathcal{C}_{t_j}(I) a_w$ □

Démonstration de la formule 6, cas $\mathcal{X}$=$\mathcal{H}$:

$a_u \mathcal{H}^{t_i}_{t_j}(I) a_w \Longrightarrow a_u \mathcal{H}^{t_i}_{t_j} a_w \wedge (a_u : [t_i]_J \vee a_w : [t_j]_J) \wedge I\{s, d, f, e\}J$
$\Longrightarrow a_w : [t_j]_J \wedge I\{s, d, f, e\}J$ car $a_u \mathcal{H}^{t_i}_{t_j} a_v$ n'est pas active
$\Longrightarrow (a_w : [t_j]_J \oplus a_v : [t_j]_J) \wedge I \{d, s, f, e\} J$ (5)
(4)+(5) $\Longrightarrow a_v \mathcal{C}_{t_j} a_w \wedge (a_w : [t_j]_J \oplus a_v : [t_j]_J) \wedge I \{d, s, f, e\} J$
$\Longrightarrow a_v \mathcal{C}_{t_j}(I) a_w$ □

Annexe B

Algorithmes de reconnaissance

Dans cette annexe, nous présentons les algorithmes de reconnaissance de scénarios temporels. Ces algorithmes sont détaillés dans [Fon93]. Un graphe de scénario est représenté par la couple (U, C) où U est une liste de variables d'instant : $\{u_1, ..., u_n\}$ et C est une liste de contraintes temporelles : $c(i, j)$, $i, j = 1, ..., n$ où $c(i, j) = [a_{ij}, b_{ij}]$.

B.1 Minimalisation

L'algorithme de minimalisation d'un graphe de contraintes est le suivant. Il permet à partir d'un graphe g d'obtenir le graphe minimal $g_{min} = min(g)$:

---

pour i = 1 à n **faire**
    $c(i, i) <- [0, 0]$
pour i,j = 1 à n et i *neq* j et $c(i, j)$ est un arc manquant **faire**
    $c(i, j) <- [-\infty, +\infty]$
pour k = 1 à n **faire**
    pour i = 1 à n **faire**
        pour j = 1 à n **faire**
            $c(i, j) <- c(i, j) \cap [c(i, k) + c(k, j)]$

L'opérateur + est défini comme suit :
$c(i, j) + c(k, l) = [a_{ij} + a_{kl}, b_{ij} + b_{kl}]$.

L'intersection entre deux intervalles $c(i, j) = [a_{ij}, b_{ij}]$ et $c(k, l) = [c_{kl}, d_{kl}]$ est définie comme suit :

$c(i, j) \cap c(k, l) =$ (contrainte vide) **si** $b_{ij} < c_{kl}$ ou $d_{kl} < a_{ij}$
$[\max(a_{ij}, c_{kl}), \min(b_{ij}, d_{kl})]$ **sinon.**

Si le graphe g_{min} contient une contrainte vide, le graphe g est dit *incohérent.*

B.2 Définitions préliminaires

Les définitions suivantes vont servir à définir les différents types de reconnaissance décrits dans le chapitre 8.

B.2.1 Graphe complet

Un graphe complet g^c est obtenu à partir d'un graphe g, en ajoutant tous les arcs manquants (u_i, u_j) et en les étiquetant par $[-\infty, +\infty]$.

B.2.2 Intersection

Soient deux graphes $g_1 = (U, C_1)$ et $g_2 = (U, C_2)$ où $U = \{u_1, ..., u_n\}$. $g_1^c = (U, C_1^c)$ et $g_2^c = (U, C_2^c)$ sont leurs graphes complets respectifs où $C_1^c = \{c_{11}^1, ..., c_{nn}^1\}$ et $C_2^c = \{c_{11}^2, ..., c_{nn}^2\}$

L'intersection entre les deux graphes g_1 et g_2 est définie par:
$g_1 \cap g_2 = (U, C^\cap)$ où $C^\cap = \{c_i^\cap j \mid c_i^\cap j = c_{ij}^1 \cap c_{ij}^2\}$. $c_{ij}^1 \cap c_{ij}^2$ est définie dans la section B.1.

B.2.3 Augmentation

L'augmentation du graphe g sur $U_a \supset U$ est définie par $g^{U_a} = (U_a, C_a)$ où
$C_a = \{ca_{ij} \mid ca_{ij} = [-\infty, +\infty]$ si $u_i \notin U$ ou $u_j \notin U$
c_{ij} si $u_i, u_j \in U\}$

B.2.4 Fusion

La fusion de deux graphes $g_1 = (U_1, C_1)$ et $g_2 = (U_2, C_2)$ est définie par $g_1 \circ g_2 = (g_1^{U_1 \cup U_2})^c \cap (g_2^{U_1 \cup U_2})^c$.

B.2.5 Inclusion

L'inclusion du graphe g_1 dans le graphe g_2 est définie comme suit:
$g_1 \subset g_2$ si et seulement si $U_1 \subset U_2$ et si chaque contrainte c_{ij} dans C_1 est incluse dans une contrainte de C_2 de mêmes extrémités.

L'inclusion entre deux contraintes $c_{ij} = [a_{ij}, b_{ij}]$ et $c_{kl} = [a_{kl}, b_{kl}]$ est définie comme suit:
$c_{ij} \subset c_{kl} \Longleftrightarrow [a_{ij}, b_{ij}]\{s, f, d, e\}[a_{kl}, b_{kl}]$
où les symboles s, f, d, e sont définis dans le chapitre 3.

B.3 Compatibilité

Soient un scénario s ayant le graphe $G = (U, C)$ et une session S ayant le graphe $G' = (U', C')$.

S est *compatible* avec s, si $min(G \circ G')$ est un graphe cohérent. Cette compatibilité est *totale*, si de plus $U = U'$.

B.4 Satisfaction

La session S *satisfait* le scénario s, si $G' \subset G$. La satisfaction est *totale*, si de plus $U = U'$.

Annexe C

Listes des principaux symboles

s(T)	ensemble des sous-tâches de T
a(T)	ensemble des tâches dont T fait partie
d(T)	date de début d'exécution de la tâche T
f(T)	date de fin d'exécution de la tâche T
$\leftrightarrow$	relation d'opposition entre tâches
r(a_u)	ensemble de tâche sous la responsabilité de l'agent a_u
$a_u : [T]_I$	l'agent a_u exécute la tâche T pendant l'intervalle I
$a_u :< T >_t$	l'agent a_u a fini d'exécuter la tâche T à l'instant t
I^-	borne inférieure de l'intervalle I
I^+	borne supérieure de l'intervalle I
$\neg$	opérateur de négation logique
$\vee$	opérateur de disjonction logique
$\oplus$	opérateur de disjonction exclusive logique
$\wedge$	opérateur de conjonction logique
$\exists$	quantificateur existentiel
$\forall$	quantificateur universel
$\cap$	opérateur d'intersection ensembliste
$-$	opérateur de soustraction ensembliste
$\mathcal{C}$	relation de concurrence
$\mathcal{D}$	relation de besoin
$\mathcal{H}$	relation d'aide
$>$	relation de pouvoir
;	opérateur de choix entre deux protocoles
.	opérateur de concaténation de deux protocoles

Annexe D

Système *STARS*

Cette annexe présente les différentes copies d'écran, résultats du déroulement de l'exemple décrit dans la section 10.8. Par la suite, nous avons choisi de suivre l'évolution de l'agent a_3 en cours de reconnaissance du scénario *Aller*.

D.1 Fenêtre principale

Le domaine de responsabilité (*domain*) de l'agent a_3 est constitué des scénarios *Marche*, *Aller* et *Retour* (cf. figure D.1). L'agent a_3 n'admet pas de buts pour l'instant (*Goals:0*), ni de plans correspondants à ces buts (*Plans:0*). Il admet une représentation des agents a_1, a_2, a_5 et a_6 (*Others descriptions:4*). Aucun événement n'est encore observé (*Perceived events:*) et aucun message n'a été envoyé ou reçu par un autre agent (*Message exchange:*). L'agent a_3 est capable de traiter les événements relatifs aux scénarios *Marche*, *Aller* et*Retour* (*Sensor:*). Un panel de session (*Session Control:*) permet d'enregistrer une session de communication avec l'agent et de redérouler cette session. A noter, la représentation de la date courante (*Local Time:*) et la date relative aux notifications envoyées par l'agent ou à la session en cours de déroulement. Enfin, la fenêtre principale offre un service de connexion à un agent (*Connect*) et de deconnection d'un agent (*Disconnect*). Elle permet également de supprimer un agent (*KILL*).

D.2 Réseaux de dépendances

L'attribution des domaines de responsabilité permet aux agents de calculer leurs réseaux de dépendances statiques $\mathcal{C}$ (figure D.2) et $\mathcal{D}$ (figure D.3), ainsi que les différentes conjonctions statiques $\mathcal{CC}$ (figure D.4), $\mathcal{CX}$ (figure D.5) et $\mathcal{XX}$ (figure D.6). Le réseau de dépendances $\mathcal{H}$ est vide.

Dans un réseau, un nœud contient l'identificateur d'un agent. Un arc contient les identificateurs des deux scénarios concernés par la dépendance

entre les deux agents. Par exemple, dans la figure D.3, l'arc étiqueté par "Retour/Terminus-A" représente la dépendance $a_3 \mathcal{D}^{Retour}_{TerminusA} a_1$.

D.3 Scénario *Aller*

La figure D.7 montre l'observation des événements $t27$, $t19$, et $t24$ en cours de reconnaissance du scénario *[Aller.* L'événement $t27$ correspond à un message d'information de l'agent a_1 concernant la reconnaissance du scénario *Terminus A*. L'événement $t18$ a été observé avant même l'activation du scénario *Aller* (retard du message d'information provenant de l'agent a_1 correspondant à l'observation de l'événement $t27$ et donc à l'activation du scénario *Aller*). L'événement $t27$ est attendu.

D.4 Activation des relations de dépendance

Les figures D.8, D.9 montrent l'activation des relations de dépendance dans les différents réseaux après l'observation de l'événement $t27$.

D.5 Raisonnement hypothétique

L'événement $t28$ correspond à l'activation du scénario *Terminus D*. Cet événement étant attendu (figure D.10), l'agent a_3 consulte son réseau de dépendances $\mathcal{D}$, active la relation $a_3 \mathcal{D}^{Aller}_{TerminusD} a_2$ (figures D.11 et D.12).

Au delà d'une seconde d'attente, l'agent a_3 utilise un raisonnement hypothétique concernant le but associé au scénario *Aller* (figure D.13).

Par la suite, une réponse de l'agent a_2 permet l'observation des événements $t28$ et $t29$ et donc la reconnaissance du scénario *Aller*. D'où la satisfaction des buts *Terminus D* et *Aller* (figure D.14).

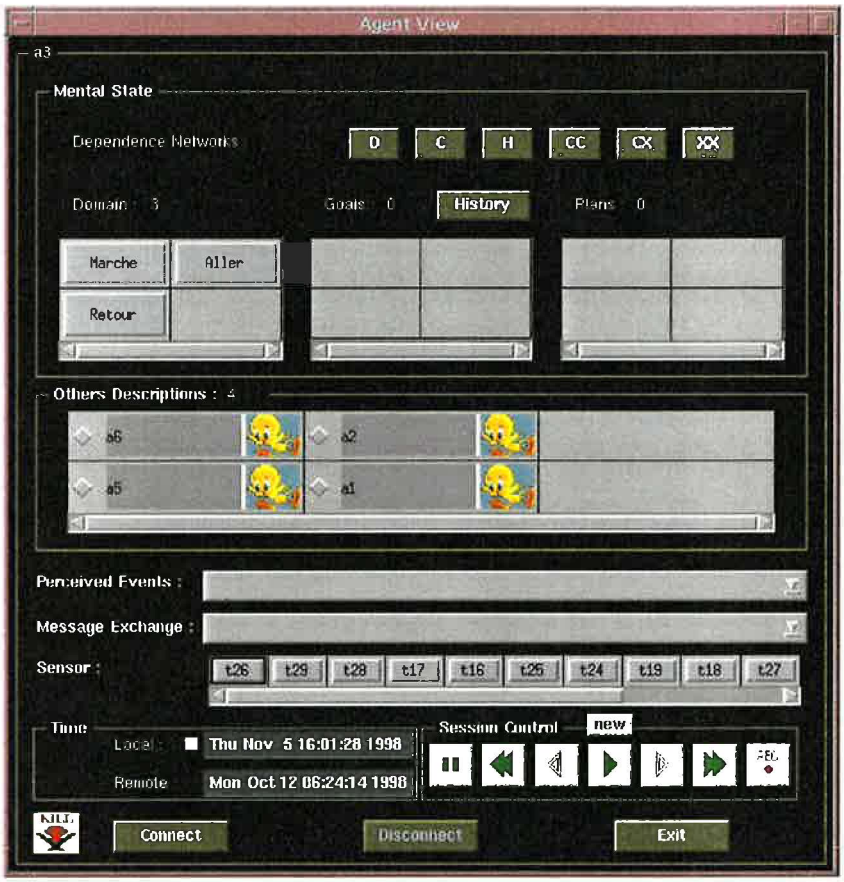


Figure D.1 : L'état de l'agent a_3 après attribution des domaines de responsabilité aux différents agents.

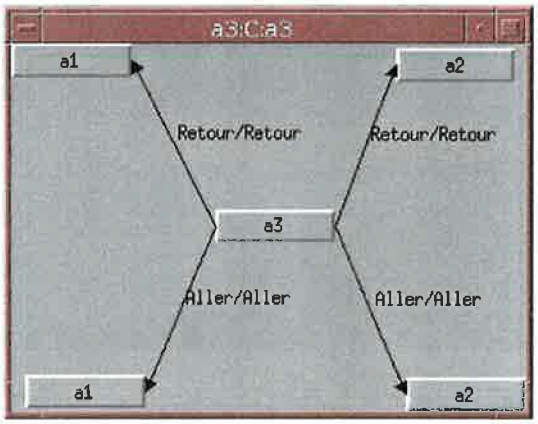


Figure D.2 : Réseau de dépendances $\mathcal{C}$ de l'agent a_3.

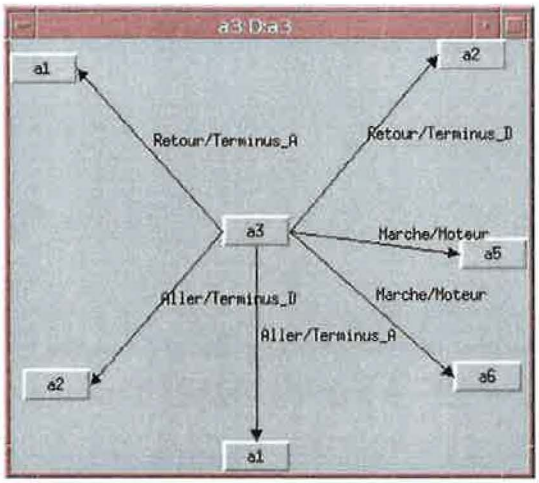


Figure D.3 : Réseau de dépendances $\mathcal{D}$ de l'agent a_3.

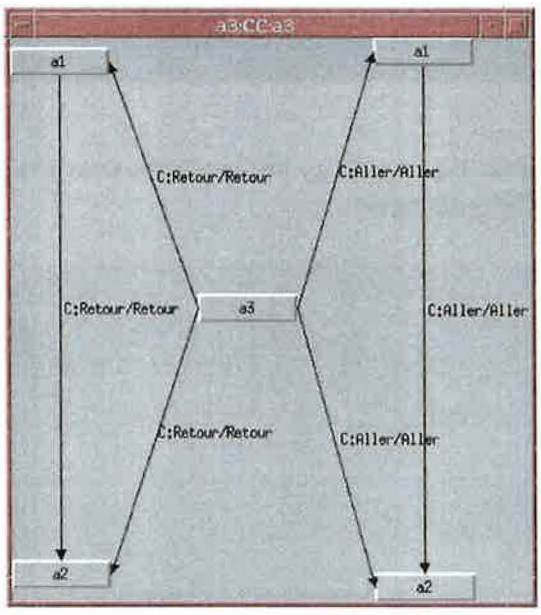


Figure D.4 : Conjonction $\mathcal{CC}$ de l'agent a_3.

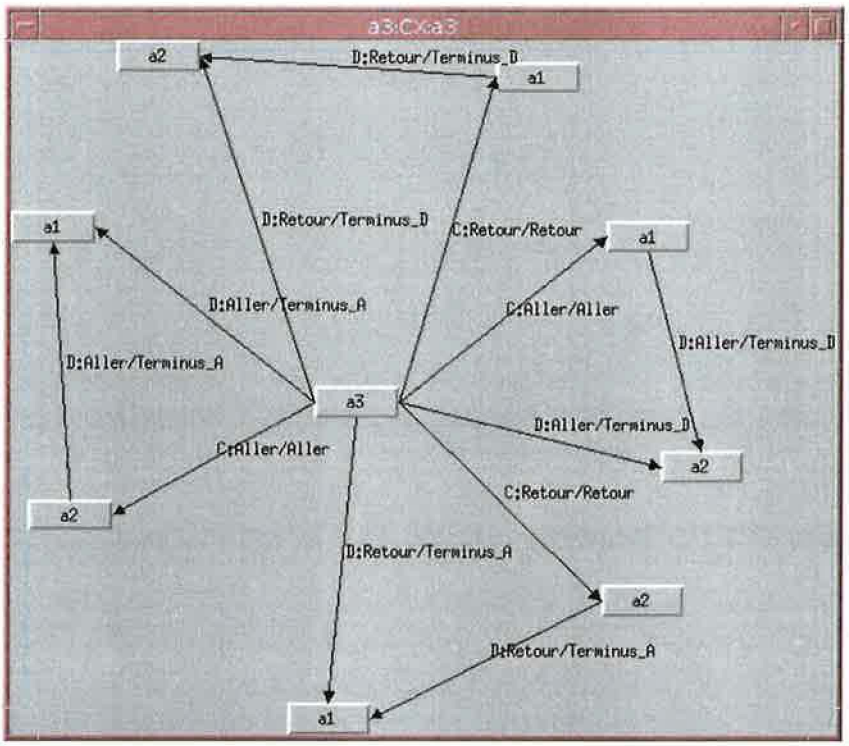


Figure D.5 : Conjonction $\mathcal{CX}$ de l'agent a_3.

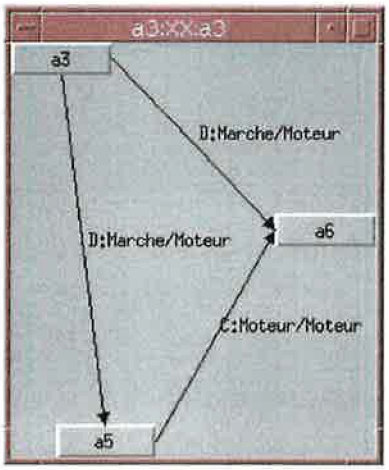


Figure D.6 : Conjonction $\mathcal{XX}$ de l'agent a_3.

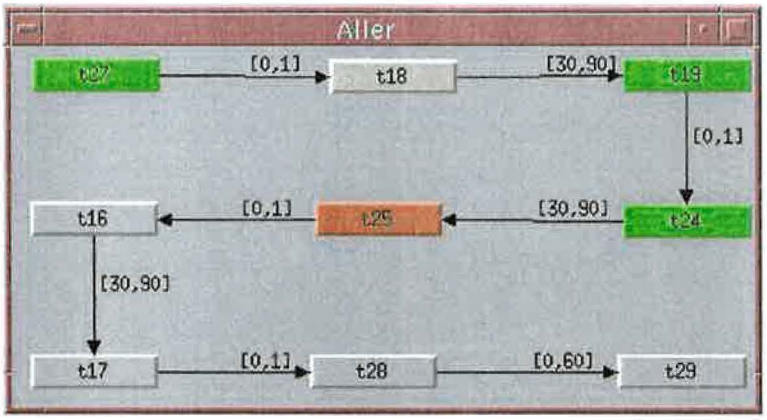


Figure D.7 : Le graphe du scénario *Aller* en cours de reconnaissance.

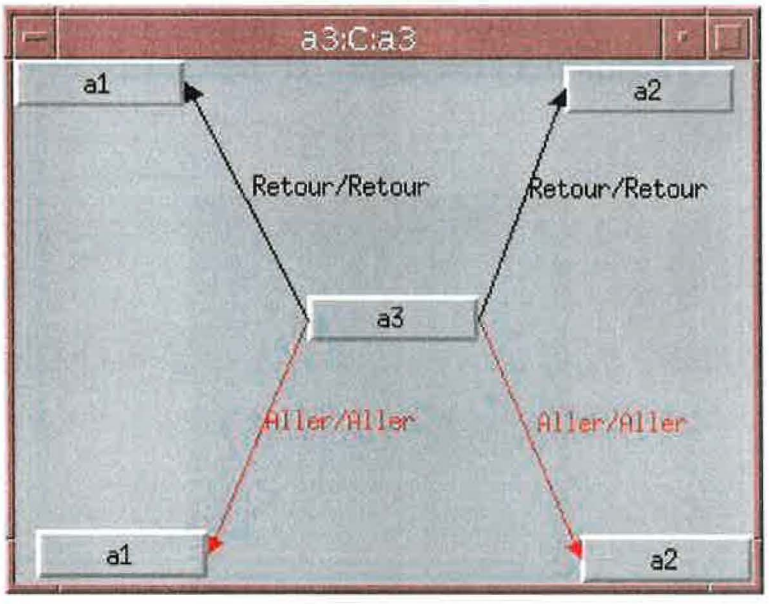


Figure D.8 : Activation des relations $\mathcal{C}$.

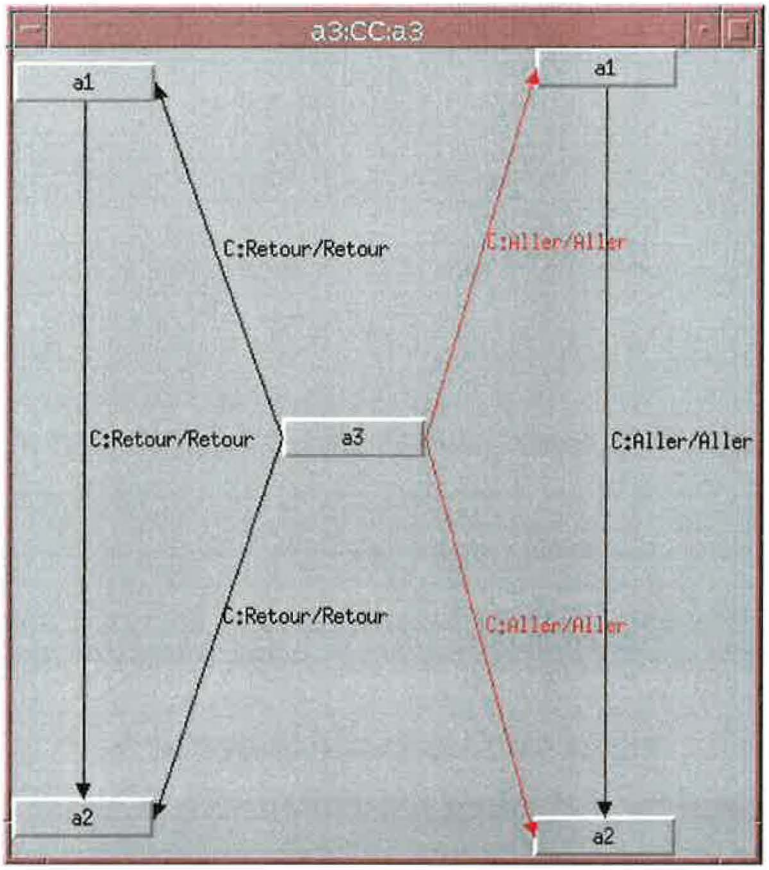


Figure D.9 : Règle d'activation pour une conjonction *CC*.

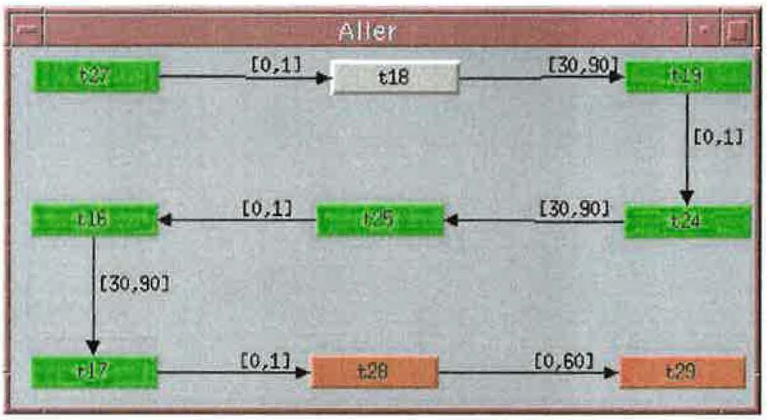


Figure D.10 : Le graphe du scénario *Aller*. Attente des événements $t28$ et $t29$.

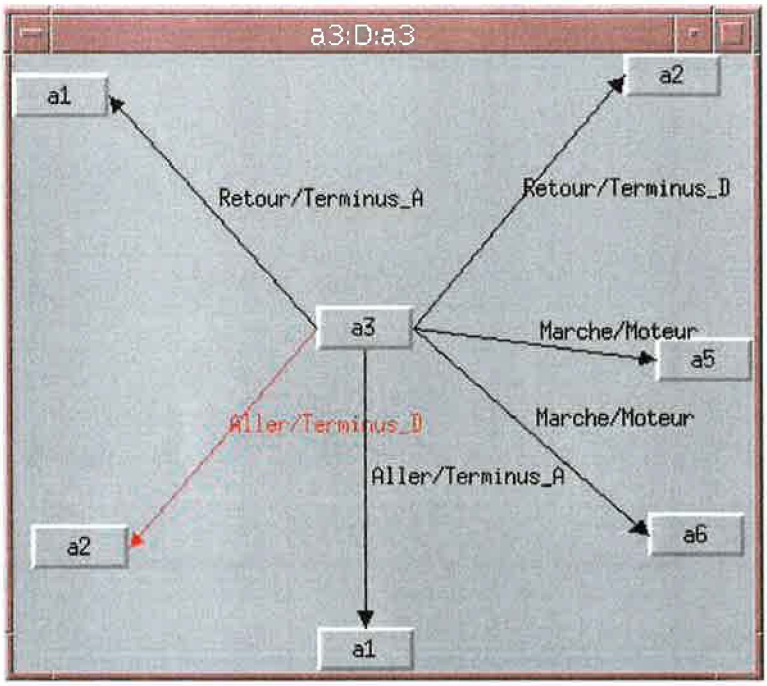


Figure D.11 : Activation des relations $\mathcal{D}$.

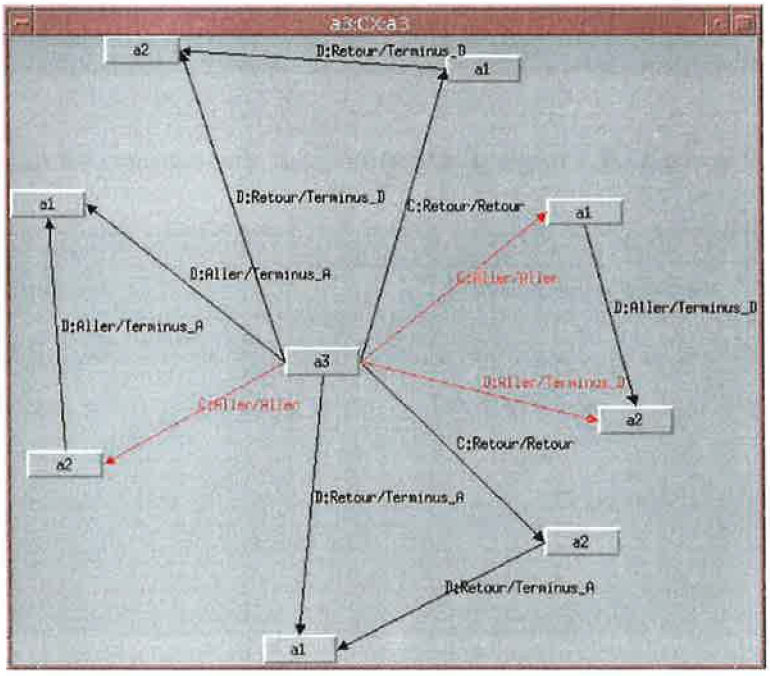


Figure D.12 : Règle d'activation pour une conjonction $\mathcal{CX}$.

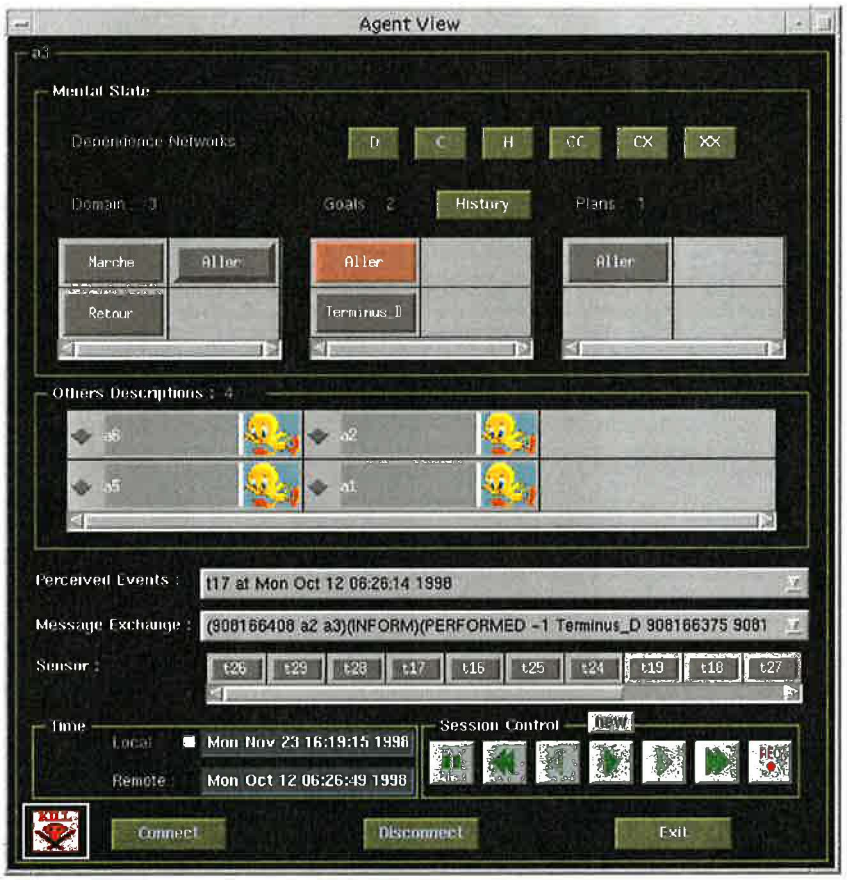


Figure D.13 : Raisonnement hypothétique. Le but *Terminus D* est considéré comme étant satisfait, mais il n'est pas supprimé de la liste des buts. Le but *Aller* ne pourra être satisfait qu'après une vérification effective de la satisfaction du but *Terminus D*.

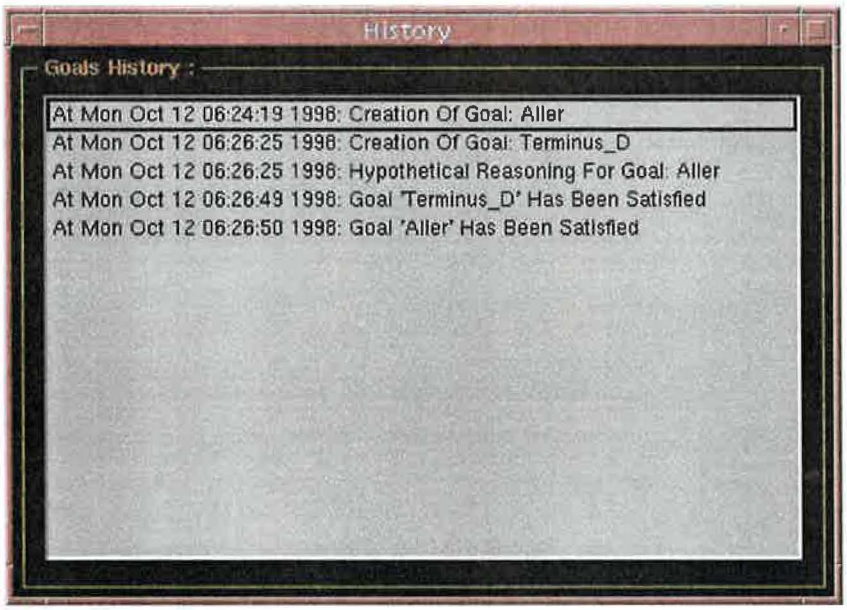


Figure D.14 : L'historique des buts de l'agent a_3 depuis l'activation et jusqu'à la reconnaissance du scénario *Aller*.

Bibliographie

[ABS97] M. Allouche, O. Boissier, and C. Sayettat. Supervision de systèmes èvolutifs par une société d'agents. *Technique et science informatique*, 16(8):1063–1084, October 1997.

[AF90] A. Alj and R. Faure. *Guide de la recherche opérationnelle*, volume 2. MASSON, 1990.

[ALA98] Groupe ALARM. Monitoring and alarm interpretation in industrial environments. *AI Communications*, 1998.

[All83] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26, 1983. Temps, Coop ration,Loo.

[All84] J. F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[All91] J. F. Allen. Time and time again : the many ways to represent time. *International Journal of Intelligent Systems*, 6, 1991.

[ASB97] M.-K. Allouche, C. Sayettat, and O. Boissier. Towards a multi-agent system for the supervision of dynamic systems. In *Proceedings of the Third IEEE International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 9–16, Berlin, Germany, 1997. IEEE Computer Society Press.

[Aus62] J. L. Austin. *How to Do Things With Words.* Oxford University Press: Oxford, England, 1962.

[BAPM83] M. Ben-Ari, A. Pnueli, and Z. Manna. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[BC96] C. Brassac and V. Chevrier. Vers un réexamen du statut de l'interaction dans les systémes multi-agents. *Interaction et cognitions*, 1, 1996.

[BCD+96] S. Bibas, M. O. Cordier, P. Dague, C. Dousson, F. Lévy, and Rozé L. Alarm driven supervision for telecommunication: I- off-line scenario generation. In *Annales des Télécommunication*, volume 9/10, pages 493–500. October 1996.

[BG88] A. H. Bond and L. Gasser, editors. *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA, 1988.

[BTM94] K. Bousson and Travé-Massuyès. Putting more numbers in the qualitative simulator ca-en. *Intelligent Systems Engineering*, pages 62–69, 1994.

[Cas90] C. Castelfranchi. Social power. A point missed in Multi Agent, DAI and HCI. In Y. Demazeau and J.P. Müller, editors, *Distributed Artificial Intelligence*, pages 49–62. Elsevier Science Publishers B.V. (North Holland), 1990.

[Cas95] C. Castelfranchi. Commitments: from individual intentions to groups and organizations. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 41–48, San Francisco, CA, June 1995.

[CB95] S. Cauvin and L. Bes. Model based alarm filtering on pilot plants. In *Preprints IFAC On-Line Fault Detection and Supervision in the Chemical Process Industries*, pages 224–236, Newcastle, June 1995.

[CD90] J.A. Campbell and M.P. D'Inverno. Knowledge interchange protocols. In Y. Demazeau & J.P. Müller, editor, *Distributed Artificial Intelligence*, pages 63–80. Elsevier Science Publishers B.V. (North Holland), 1990.

[CDT89] L. Console, D. T. Dupré, and P. Torasso. A theory of diagnosis for incomplete causal models. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, Detroit, 1989.

[Cho57] C. K. Chow. An optimum character recognition system using decision functions. *R. E. Trans. on Electronic Computers*, pages 247–254, 1957.

[CL90a] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[CL90b] P. R. Cohen and H. J. Levesque. Rational interaction as the basis for communication. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*. MIT Press, 1990.

[CL91] P. R. Cohen and H. J. Levesque. Teamwork. Technote 504, SRI International, Menlo Park, CA, March 1991.

[CL95] P. R. Cohen and H. J. Levesque. Communicative actions for artificial agents. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 65–72, San Francisco, CA, June 1995.

[CMC91] R. Conte, M. Miceli, and C. Castelfranchi. Limits and levels of cooperation : disentangling various types of prosocial interaction. In Y. Demazeau and J.P. Müller, editors, *Distributed Artificial Intelligence Volume II*, pages 147–160. Elsevier Science Publishers B.V. (North Holland), 1991.

[CMC92] . Castelfranchi C, M. Micelli, and A. Cesta. Dependence relations among autonomous agents. In éd. E. Werner et Y. Demazeau, editor, *Decentralized A. I.*, volume 3, pages pp. 215–227. Elseiver Science Publishers B. V., Amsterdam, NL, 1992.

[CT94] M. O. Cordier and S. Thiébaux. Event-based diagnosis for evolutive systems. In *Proceedings of workshop on diagnosis from first principles (DX-94)*, New-Paltz, 1994.

[CW92] M. K. Chang and C. C. Woo. SANP: A communication level protocol for negotiations. In E. Werner and Y. Demazeau, editors, *Decentralized AI 3 — Proceedings of the Third European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-91)*, pages 31–56. Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1992.

[Dav83] R. Davis. Diagnosis via causal reasoning: paths of interaction and the locality principle. In *AAAI Conference*, pages 88–94, 1983.

[DCL95] A. Drogoul, B. Corbara, and S. Lalande. MANTA: New experimental results on the emergence of (artificial) ant societies. In N. Gilbert and R. Conte, editors, *Artificial Societies: The Computer Simulation of Social Life*, pages 190–211. UCL Press: London, 1995.

[DGG93] C. Dousson, P. Gaborit, and M. Ghallab. Situation recognition. In *Proceedings of the 13th international joint conference on AI*, volume 1, pages 166–172, Chambéry-France, 1993.

[Die91] R. Dieng. Relations linking cooperating agents. In Y. Demazeau and J.P. Müller, editors, *Distributed Artificial Intelligence Volume II*, pages 51–70. Elsevier Science Publishers B.V. (North Holland), 1991.

[DLC87] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Coherent cooperation among communicating problem solvers. *IEEE Transactions on Computers*, 36, 1987.

[DLPT82] E. Diday, J. Lemaire, J. Pouget, and F. Testu. *El ments d'analyse de donn es.* Paris, dunod edition, 1982.

[DM91] Y. Demazeau and J.P. Müller. From reactive to intentional agents. In Y. Demazeau and J.P. Müller, editors, *Distributed Artificial Intelligence Volume II*, pages 3–13. Elsevier Science Publishers B.V. (North Holland), 1991.

[DMP91] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1–3):61–95, 1991.

[Dou94] C. Dousson. *Suivi d'évolutions et reconnaissance de chroniques.* PhD thesis, Université Paul Sabatier, Toulouse, France, September 1994.

[Dou96] C. Dousson. Alarm driven supervision for telecommunication networks: 2- on-line chronicle recognition. In *Annales des Télécommunications*, volume 9/10, pages 501–508. October 1996.

[Doy91] J. Doyle. Rational belief revision (preliminary report). In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 163–174, Cambridge, Massachusetts, April 1991.

[DS76] E. Diday and J. C. Simon. *Cluster Analysis.* Digital Patern recognition. Springer-Verlag, 1976.

[Dub90] Bernard Dubuisson. *Diagnostic et reconnaissance des formes.* Hermès edition, 1990.

[EG92] P. Estraillier and C. Girault. Applying petri net theory to the modelling analysis and prototyping of distributed systems. In *Proceedings of the IEEE International Worshop on Emerging Technologies and Factory Automation*, Cairns, Australia, 1992.

[FDV96] H. Weigand F. Dignum and E. Verharen. Meeting the deadline : on the formal specification of temporal deontic constraints. In *ISMIS'96 conference*, 1996. (ismis96.ps) : paper about the specification of deadlines using dynamic deontic logic.

[Fer94] Jacques Ferber. Coopération réactive et émergence. *Intellectica*, 19:19–52, 1994.

[Fer95] J. Ferber. *Les systèmes multi-agents.* InterEditions, 1995.

[FFMM94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as agent communication langage. In *CIKM'94*. ACM Press, 1994.

[Fon93] D. Fontaine. Reconnaissance de scénarii temporels. Technical Report 93/106, Université de Technologie de Compiègne, June 1993.

[Fon96] D. Fontaine. Une approche par graphes pour la reconnaissance de sc énarios temporels. *Revue d'intelligence artificielle*, 10(4):439–468, 1996.

[Gal90] A. Galton. A critical examination of allen's theory of action and time. *Artificial Intelligence*, 42:159–188, 1990.

[Geo84] Michael Georgeff. A theory of action for multiagent planning. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 121–125, Austin, TX, August 1984. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 205–209, Morgan Kaufmann, 1988.).

[Geo86] Michael Georgeff. A representation of events in multi-agent domains. In *Proceedings of the National Conference on Artificial Intelligence*, pages 70–75, Philadelphia, PA, August 1986. (Also published in *Readings in Distributed Artificial Intelligence*, Alan H. Bond and Les Gasser, editors, pages 210–215, Morgan Kaufmann, 1988.).

[GH89] L. Gasser and M. Huhns, editors. *Distributed Artificial Intelligence*, volume 2 of *Research Notes in Artificial Intelligence*. Pitman/Morgan Kaufman, 1989. Chapter 6, p. 119–137.

[Gin91] M. L. Ginsberg. Knowledge interchange format: The KIF of death. Technical report, Stanford University, 1991.

[HC68] G. E. Hughes and M. J. Cresswell. *Introduction to Modal Logic*. Methuen and Co., Ltd., 1968.

[Hin62] J. Hintikka. *Knowledge and Belief*. Cornell University Press: Ithaca, NY, 1962.

[HS98] Michael N. Huhns and Munindar P. Singh, editors. *Readings in Agents*. Morgan Kauffman Publishers, Inc., 1998.

[HSBS98] M. Hannoun, J. S. Sichman, O. Boissier, and C. Sayattat. Dependence realtions between roles in a multi-agent system. In *Multi-agent systems and Agent-based Simulation, MABS'98*, 1998.

[ILO95a] ILOG Inc., ILOG, BP 85, 9 rue Verdun, 94253 Gentilly cedex - France. *ILOG SERVER - Reference Manual - Version 2.0*, 1995.

[ILO95b] ILOG Inc., ILOG BP 85, 9 rue Verdun, 94253 Gentilly cedex - France. *ILOG VIEWS - Reference Manual - Version 2.2*, 1995.

[IS92] Hideki Isozaki and Yaov Shoham. A mechanism for reasoning about time and belief. In ICOT, editor, *Proceedings of the International Conference on Fifth Generation Computer Systems*, pages 694–701, 1992.

[Jen92a] N. R. Jennings. *Joint Intentions as a Model of Multi-Agent Cooperation.* PhD thesis, Department of Electronic Engineering, Queen Mary & Westfield College, 1992.

[Jen92b] N. R. Jennings. Towards a cooperation knowledge level for collaborative problem solving. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)*, pages 224–228, Vienna, Austria, 1992.

[Jen93a] N. R. Jennings. Commitments and conventions: The foundation of coordination in multi-agent systems. *The Knowledge Engineering Review*, 8(3):223–250, 1993.

[Jen93b] N. R. Jennings. Specification and implementation of a belief desire joint-intention architecture for collaborative problem solving. *Journal of Intelligent and Cooperative Information Systems*, 2(3):289–318, 1993.

[Jen95] Nick Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 1995.

[Jen96] Nick R. Jennings. Coordination techniques for DAI. In Greg O'Hare and Nick Jennings, editors, *Foundations of Distributed Artificial Intelligence*, chapter 6. John Wiley and Sons, 1996.

[KEL91] Sarit Kraus, Eithan Ephrati, and Daniel Lehmann. Negotiation in a non-cooperative environment. *Journal of Experimental and Theoretical Artificial Intelligence*, 3(4):255–282, 1991.

[KMP77] D. E. Knuth, J. H. Morris, and V. R. Pratt. Fast pattern matching in strings. *SIAM J. on Computing*, 6(2):323–350, 1977.

[KMSK91] T. Kohonen, K. Makisara, O. Simula, and J. Kangas. *Artificial Neural Networks*, volume 1–2. Elseiver Science Publishers, 1991.

[Koh84] T. Kohonen. *Self-organization and associative memory.* Springer-Verlag, 1984.

[Kri63] S. Kripke. Semantical analysis of modal logic. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9:67–96, 1963.

[Kro95] C. Krogh. Obligations in multiagent systems. In *Scandinavian Conference on Artificial Intelligence (SCAI'95)*, May 1995.

[Lé94] F. Lévy. Recognising scenarios. In *Proceedings of workshop on diagnosis from first principles (DX-94)*, New Paltz, 1994.

[LCd95] S. Lizotte and B. Chaib-draa. Coordination en situations non familières. In *Troisièmes Journèes Francophones Intelligence Artificielle Distribuée et Systèmes Multi-agents*, Avril 1995.

[LCd96] K. Lechilli and B. Chaib-draa. Structures relationnelles pour les interactions entre agents. In *Quatrièmes Journées Francophones Intelligence Artificielle Distribuée et Systèmes Multi-agents*, Avril 1996.

[LK97] P. Laborie and J. P. Krivine. Automatic generation of chronicles and its application to alarm processing in power distribution systems. In *International Workshop on Principles of Diagnosis (DX' 97)*, pages 61–68, Mont St-Michel, France, September 1997.

[LS92] Fangzen Lin and Yoav Shoham. Concurrent actions in the situation calculus. In William Swartout, editor, *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 590–595, San Jose, CA, jul 1992. MIT Press.

[McD82] D. McDermott. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6, 1982.

[MH69] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*. Edinburgh University Press, 1969.

[MT93] K. Matsubayashi and M. Tokoro. A collaboration mechanism on positive interactions in multi-agent environments. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 346–351, Chambéry, France, 1993.

[OJ96] G. M. P. O'Hare and N. R. Jennings, editors. *Foundations of Distributed Artificial Intelligence.* WILEY-INTERSCIENCE, John Wiley and Sons, Inc., 1996.

[Pau81] L. F. Pau. *Failure diagnosis and performance monitoring.* Marcel Dekker Inc., 1981.

[PJ96] Simon Parsons and N. R. Jennings. Negotiation through argumentation—a preliminary report. In *Proceedings of the Second International Conference on Multi–Agent Systems, Kyoto, Japan*, 1996.

[PLE92] PLEIAD. Vers une taxinomie du vocabulaire en IAD. In *Journ es syst mes Multi-Agents du PRC-IA*, Nancy, December 1992.

[PPS93] O. Boissier P. Populaire, Y. Demazeau and J.S. Sichman. Description et implèmentation de protocoles de communication en univers multi-agents. In *Premières Journèes Francophones Intelligence Artificielle Distribuée et systèmes Multi-Agents*, Toulouse, Avril 1993.

[PR90] Y. Peng and J. A. Reggia. *Abductive inference models for diagnostic problem solving.* Springer-Verlag, 1990.

[PR97] M. Porcheron and B. Ricard. An application of abductive diagnostic methods to a real world problem. In *International Workshop on Principles of Diagnosis (DX'97)*, pages 87–94, Mont St-Michel - France, 1997.

[Rao64] C. R. Rao. The use and interpretation of principal component analysis in applied research. *Sankhya, S rie A*, 26:329–357, 1964.

[Rei87] Han Reichgelt. Semantics for reified temporal logic. In *Artificial Intelligence*, Edimbourg, 1987.

[RG91a] A. S. Rao and M. P. Georgeff. Asymmetry thesis and side-effect problems in linear time and branching time intention logics. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 498–504, Sydney, Australia, 1991.

[RG91b] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484. Morgan Kaufmann Publishers: San Mateo, CA, April 1991.

[RG93] A. S. Rao and M. P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93)*, pages 318–324, Chambéry, France, 1993.

[RJM91] C. Roda, N.R. Jennings, and E.H. Mamdani. The impact of heterogeneity on cooperating agents. *Proc. AAAI Workshop on Cooperation among Heterogeneous Intelligent Systems, Anaheim, USA*, 1991.

[Roz97] L. Rozé. Supervision of telecommunication network: A diagnoser approach. In *International Workshop on Principles of Diagnosis (DX'97)*, pages 103–11, Mont St-Michel - France, 1997.

[Rum91] J. Rumbaugh. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[SA93] M. P Singh and N. M Asher. A logic of intentions and beliefs. *Journal of Philosophical Logic*, 1993.

[SCCD94] J. S. Sichman, R. Conte, C. Castelfranchi, and Y. Demazeau. A social reasoning mechanism based on dependence networks. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pages 188–192, Amsterdam, The Netherlands, 1994.

[SD95] J. Sichman and Y. Demazeau. Exploiting social reasoning to deal with agency level inconsistency. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 352–359, San Francisco, CA, June 1995.

[SD96] J. Sichman and Y. Demazeau. A model for the decision phase of autonomous belief revision in open multi-agent systems. *Journal of the Brazilian Computer Society*, 3(1):40–50, 1996.

[SDB92] J.S. Sichman, Y. Demazeau, and O. Boissier. When can knowledge-based systems be called agents? In SBC Anais, editor, *9nth Simpósio brasileiro em inteligencia artificial (sbia)*, pages 172–185, Rio de Janeiro, 1992.

[Sea69] J. R. Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press: Cambridge, England, 1969.

[Sea79] J. R. Searle. *Expression and Meaning.* Cambridge University Press: Cambridge, England, 1979.

[Sea90] J. R. Searle. Collective intentions and actions. In P. R. Cohen, J. Morgan, and M. E. Pollack, editors, *Intentions in Communication*, pages 401–416. The MIT Press: Cambridge, MA, 1990.

[SG94] B. Séroussi and J. L. Golmard. An for finding the k most probable configurations in baysian networks. *International journal of approximative reasoning*, 1:202–233, 1994.

[Sho90] Y. Shoham. Agent-oriented programming. Technical Report STAN–CS–1335–90, Computer Science Department, Stanford University, Stanford, CA 94305, 1990.

[Sic95] J. Sichman. *Du Raisonnement Social Chez les Agents: Une approche Fondée sur la Théorie de la Dépendance.* PhD thesis, Institut National Polytechnique de Grenoble - France, 1995.

[Sic96] J. Sichman. On achievable goals and feasible plans in open multi-agent systems. In *First Ibero-American Workshop on DAI/MAS*, Xalapa, Mexico, 1996.

[Sin94] M. P. Singh. *Multiagent Systems: A Theoretical Framework for Intentions, Know-How, and Communications (LNAI Volume 799).* Springer-Verlag: Heidelberg, Germany, 1994.

[TB88] A. Trognon and C. Brassac. Actes de langages et conversation. *Intellectica*, 6(2):pp. 211–232, 1988.

[Tom76] I. Tomek. A generalisation of the kNN rule. *I.E.E.E Trans. on Information theory*, 6, 2:121–126, 1976.

[Van88] D. Vanderveken. *Les actes de discours.* Pierre Mardaga, 1988.

[Wer89] E. Werner. Cooperating agents: A unified theory of communication and social structure. In L. Gasser and M. Huhns, editors, *Distributed Artificial Intelligence Volume II*, pages 3–36. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.

[WF86] T. Winograd and F. Flores. *Understanding Computers and Cognition: a New Foundation for Design.* Ablex Publishing Corporation, 1986.

[WF94] M. Wooldridge and M. Fisher. A decision procedure for a temporal belief logic. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic — Proceedings of the First International Conference (LNAI Volume 827)*, pages 317–331. Springer-Verlag: Heidelberg, Germany, July 1994.

[ZR91] Gilad Zlotkin and Jeffrey S. Rosenschein. Cooperation and conflict resolution via negotiation among autonomous agents in non-cooperative domains. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6), December 1991. (Special Issue on Distributed AI).

Résumé

La complexité croissante des systèmes industriels en termes d'expansion, d'hétérogénéité et de décentralisation entraine de plus en plus de contraintes dans leur fonctionnement. Ces contraintes ont souvent une composante temporelle. Des techniques, telles que les systèmes multi-agents, sont bien adaptées au caractère hétérogène et décentralisé de ces systèmes, elles sont de plus en plus souvent utilisées dans les applications telles que : Contrôle, Supervision, Simulation, Pilotage ... Des travaux existent sur la représentions du temps dans la modélisation et le raisonnement sur les systèmes industriels. Dans le contexte multi-agents, peu de travaux prennent en compte ce facteur temporel explicitement et il est quasiment absent de tous les aspects sociaux de l'agent. Dans cette thèse, nous proposons et définissons une société d'agents temporels qui permet la prise en compte explicite du temps à la fois dans le raisonnement individuel et le raisonnement social au sein de l'agent. L'accent est mis d'une part sur l'interaction et d'autres part sur les différents types de dépendances qui existent entre agents et leur évolution au cours du temps. Ce modèle générique, a été appliqué à la supervision par reconnaissance de scénarios temporels, domaine dans lequel l'exécution des différentes taches est extrémement contrainte par le temps. Le système *STARS* est une implémentation du modèle. Il permet une reconnaissance distribuée des scénarios par un ensemble d'agents utilisant un raisonnement temporel.

Abstract

Industrial systems are getting increasingly complex, heterogeneous and decentralized. Consequently, their functioning undergoes more and more constraints, which very often have temporal characteristics. Multi-agent techniques are well-adapted to this kind of environment. They are widely used in many applications such as Control, Supervision, Simulation, Piloting ... Temporal representations have been studied for modelling and reasoning purposes but they still are at an early stage within a multi-agent context, especially on the social level. The aim of this study is to define a multi-agent society where temporal aspects are explicitly taken into account within the individual as well as the social reasoning of an agent. The model we proposed focuses essentially on agent dependencies and interactions. It also permits to study their evolution in time. This generic model is applied to the supervision of industrial systems through scenario recognition. In this domain, tasks are always executed under temporal constraints. The *STARS* system is an implementation of this model. It enables a group of agents to recognize scenarios in a distributed way using a temporal reasoning.

Mots clés : Systèmes Multi-Agents, Intelligence Artificielle Distribuée, Réseaux de dépendances, Raisonnement Temporel, Supervision

Keywords : Multi-Agent Systems, Distributed Artificial Intelligence, Dependence Networks, Temporal Reasoning, Supervision