Contrôle du partage de l'autorité dans un système d'agents hétérogènes

Stéphane Mercier

THÈSE

présentée en vue de
l'obtention du titre de

DOCTEUR DE L'UNIVERSITE DE TOULOUSE

délivré par

l'Institut Supérieur de l'Aéronautique et de l'Espace

ÉCOLE DOCTORALE : Systèmes

SPÉCIALITÉS : Systèmes Embarqués et Robotique

par

Stéphane Mercier

Contrôle du partage de l'autorité dans un système d'agents hétérogènes

Directeurs de thèse :
Frédéric DEHAIS, enseignant-chercheur, ISAE, Toulouse
Catherine TESSIER, maître de recherche, ONERA, Toulouse

Thèse préparée au sein de l'ONERA
Département Commande des Systèmes et Dynamique du vol
Unité de Recherche Conduite et Décision (ONERA-DCSD/CD)

Table des matières

Table des figures

Liste des Algorithmes

Chapitre 1

Position du problème

> Perhaps the major human factors concern of pilots in regard to introduction of automation in the cockpit is that, in some circumstances, operations with such aids may leave the critical question, Who is in control ? unanswered.
>
> Sheldon Baron

1 Contexte

Nous nous intéressons à la conception d'un agent décisionnel, dans un contexte de réalisation de mission impliquant une entité physique (de type véhicule robotisé, drone, avion), où la conduite de mission est effectuée *à la fois* par un humain (opérateur, pilote) et par un agent décisionnel ou un pilote automatique. L'agent décisionnel est embarqué sur l'entité physique ; l'agent humain peut, suivant la mission, être embarqué (pilote d'avion), ou à distance (opérateur de drone), auquel cas ses opérations dépendent du maintien d'un lien de communication. L'existence de ce lien de communication et de sa possible défaillance requiert, de la part de l'agent décisionnel, la capacité de conduire momentanément la mission seul en exhibant une certaine « autonomie ».

Notre problématique générale est ainsi celle du contrôle mixte dans un système homme-machine. L'agent humain joue un rôle actif tout au long de la mission : il ne se limite pas à fixer les objectifs de mission, il peut réaliser lui-même des tâches et possède des savoir-faire dont ne dispose pas l'agent décisionnel. L'agent décisionnel, ou robot, doit assister l'humain dans ses fonctions, la notion d'assistance pouvant prendre des significations très variées, allant de l'exécution de tâches en délégation à la surveillance et prévention des erreurs. Ainsi, non seulement les agents humains et décisionnels contrôlent conjointement un même système physique, mais ils peuvent également interagir entre eux, comme l'illustre la figure 1.1.

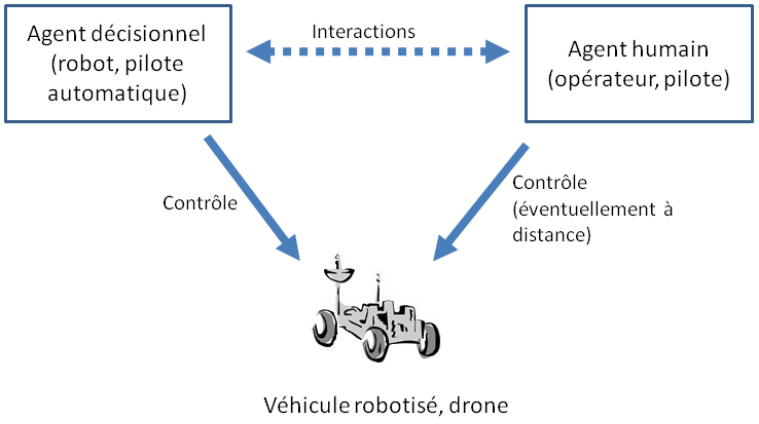

FIGURE 1.1: Contrôle mixte entre agent décisionnel et agent humain sur une entité physique

1.1 Problématique

Au cours de la mission, chacun des agents se voit attribuer ou prend l'autorité pour utiliser une ressource, réaliser une tâche, satisfaire un but : par exemple, un pilote automatique d'avion en mode *Vertical Speed* a l'autorité sur la chaîne de commande des gouvernes pour atteindre une altitude de consigne tout en garantissant certaines performances ; si l'équipage déconnecte le pilote automatique, c'est l'équipage qui a l'autorité sur la chaîne de commande pour atteindre ce but. Un changement dans l'affectation de l'autorité peut être prévu dans les procédures, le plan de mission ; ou bien il peut se produire de manière imprévue :

- l'agent humain reprend en main une tâche contrôlée par l'agent décisionnel parce qu'il constate une défaillance, un obstacle, un danger imminent, ou pour toute autre raison qui lui est propre et qui n'est pas forcément exprimée ;
- l'agent décisionnel reprend en main une tâche contrôlée par l'opérateur parce que l'action de l'opérateur viole certaines contraintes (sortie de domaine de vol, collision probable avec un obstacle, etc.) ou parce que les communications avec l'opérateur sont rompues ;
- plus aucun agent n'exerce l'autorité : par exemple, le pilote automatique s'est déconnecté et l'équipage, qui n'en est pas conscient, ne pilote pas [Mumaw *et al.*, 2001].

Le problème posé par ces changements imprévus dans l'affectation de l'autorité est le risque d'instabilité ou d'incohérence dans l'état du système, dû au fait que le plan prévu pour l'agent décisionnel n'est plus suivi, que l'opérateur a une conscience de situation erronée [Wickens, 2008], voire les deux. Ceci est illustré par l'expérience suivante, réalisée à l'ISAE Supaéro [Mercier *et al.*, 2010c], sur une mission de recherche et d'identification de cibles par un robot terrestre et un agent humain qui n'est pas en vue directe du

robot. Au cours de la mission, l'opérateur doit, *via* son écran d'interface, piloter le robot manuellement et finement afin d'identifier des cibles. Alors que l'opérateur prend en main le robot pour cette inspection, celui-ci entame une procédure autonome de retour à la base, déclenchée par le capteur de décharge de la batterie (aléa simulé par une interface de magicien d'Oz [1]). Cet événement est perceptible pour l'opérateur sur son interface *via* trois alertes : l'icône représentant la batterie passe de vert à orange, le mode de pilotage clignote deux fois de « manuel » à « supervisé » et le synoptique affiche en vert « retour base ». Toutefois, comme cet aléa se produit à un moment crucial de la mission où l'opérateur est particulièrement focalisé sur sa tâche d'identification en mode manuel, il est attendu qu'il ne perçoive pas ces changements d'état et que chacun des agents (opérateur et robot) « s'entête » à poursuivre son but (identifier la cible et retourner à la base, respectivement).

Sujet	Durée conflit (en s.)	Résolution	Alertes regardées
Dasje	6	oui	supervisé, retour base, batterie
Deffra	50	non	supervisé
Dupni	50	non	supervisé, retour base
Gatthi	50	non	supervisé, retour base
Guiju	50	non	-
Guiny	50	non	batterie
Hosal	18	oui	retour base, batterie
Jacchi	50	non	supervisé
Nival	10	oui	supervisé, batterie
Peich	29	non	retour base
Penju	50	non	supervisé
Pense	35	oui	supervisé, batterie
Rojan	50	non	-
Schpa	50	non	-

TABLE 1.1: Comportement des participants pendant le conflit d'autorité

Le tableau 1.1 résume les résultats d'oculométrie de quatorze participants à l'expérience [2] analysés à l'aide du logiciel Eye tech Lab. La première colonne est l'identifiant du participant, la deuxième indique la durée du conflit, la troisième si la raison du conflit a été comprise par l'opérateur et la quatrième si les trois alertes nécessaires à la compréhension du conflit ont été regardées au moins une fois. Par exemple, le participant Dasje a mis 6 secondes pour détecter et comprendre le conflit, il a porté au moins une fois son regard sur le mode supervisé, sur le synoptique indiquant le retour base et sur

1. Interface permettant de déclencher des événements à l'insu de l'opérateur.
2. Les sujets sont équipés d'un oculomètre de type Pertech (25Hz) afin d'analyser leur comportement oculaire (où se pose leur regard).

l'état de la batterie. Le participant Dupni n'a pas compris le conflit après 50 secondes, bien que son regard se soit posé sur l'état du mode « supervisé » et sur « retour base », en revanche il n'a jamais posé son regard sur l'état de la batterie. Les résultats de cette expérimentation confirment que le conflit d'autorité a mis en avant la mauvaise conscience de situation des opérateurs, puisque le conflit a conduit 10 sujets sur 14 à persévérer à tort dans la tâche d'identification de la cible. L'analyse du comportement oculaire des 4 participants qui ont interprété correctement la situation révèle que ceux-ci ont au moins regardé les couples d'information (« état de la batterie », mode « supervisé ») ou (« état de la batterie », « retour base ») si ce n'est le triplet d'information (« état de la batterie », mode « supervisé », « retour base »). Enfin, il convient de noter que le robot a également continué son plan de retour à la base sans prendre en compte le conflit avec l'opérateur.

Comme le montre cette expérience préliminaire, le contrôle mixte agent humain - agent décisionnel peut devenir incohérent et amener le système contrôlé dans un état instable, voire dangereux. En raison de l'hétéogénéité des agents, le contrôle mixte revêt certaines caractéristiques que nous allons étudier dans ce chapitre.

1.2 Agents et système d'agents

1.2.1 Agents

Un *agent* est une entité physique ou informatique, plongée dans un environnement, pouvant percevoir tout ou partie de cet environnement grâce à des capteurs fournissant des entrées, et le modifier grâce à des effecteurs délivrant des sorties. "La notion d'agent est destinée à être un outil pour l'analyse des systèmes et non une caractérisation absolue qui divise le monde entre agents et non-agents" [Russell et Intelligence, 1995]. En ce qui concerne la notion d'*agent autonome*, [Franklin et Graesser, 1997] propose la définition suivante : « Un agent autonome est un système au sein d'un environnement, partie intégrante de celui-ci, qui perçoit cet environnement et agit sur lui au cours du temps, en suivant son propre agenda, afin d'agir sur ce qu'il perçoit dans le futur ». L'aspect de la temporalité (continuité de la tâche) est ici primordial. [Ferber et Perrot, 1995] propose également cette définition : « Une entité réelle ou virtuelle, évoluant dans un environnement, capable de le percevoir et d'agir dessus, qui peut communiquer avec d'autres agents, qui exhibe un comportement autonome, lequel peut être vu comme la conséquence de ses connaissances, de ses interactions avec d'autres agents et des buts qu'il poursuit ». Ici s'ajoutent également la dimension sociale de l'agent et ses capacités d'interaction.

Au regard des définitions mentionnées précédemment, nous retiendrons la définition suivante :

Definition 1 (Agent)
Un agent est une entité physique ou informatique, plongée dans un environnement et partie intégrante de celui-ci, qui perçoit (partiellement) cet environnement et agit sur lui au cours du temps, qui peut interagir avec d'autres agents, et dont le comportement est dicté par la poursuite d'un ou plusieurs buts. □

Plus précisément, nous nous intéressons à deux types d'agents : l'agent décisionnel, que nous nommerons désormais *robot* et l'agent humain, que nous nommerons *opérateur humain*, qui contrôlent conjointement une ou plusieurs entités physiques devant réaliser une mission.

Definition 2 (Opérateur humain)
L'opérateur humain perçoit des informations, les interprète, prend des décisions, agit sur le robot et interagit avec lui.

Definition 3 (Robot)
Le robot constitue la partie machine du système homme-machine. Il perçoit des informations (par ses capteurs), les interprète (*via* un estimateur d'état ou suivi de situation, calcule des décisions (*via* un planificateur), agit dans l'environnement et interagit avec l'opérateur humain. □

1.2.2 Coopération multiagent

La coopération peut avoir lieu lorsqu'au moins deux agents interviennent, que ce soient deux opérateurs humains, deux robots ou un robot et un opérateur humain. Des mécanismes tels que la décomposition et l'allocation de tâches, la coordination, la négotiation, l'initiative, la gestion de conflit sont des raffinements de modèles de coopération qui sont utilisés en fonction des applications [Brezillon et Pomerol, 1997].

Il est possible de distinguer *coordination* de *coopération* dans le sens où il peut y avoir coordination sans coopération, la réciproque n'étant pas vraie : la coopération nécessite la coordination [Nwana, 1996]. Selon [Legras et Tessier, 2003], la coopération se présente comme l'association de la collaboration, cas dans lequel plusieurs agents interagissent pour atteindre des buts communs, avec la coordination, cas où des agents doivent interagir en vue de se partager une ressource commune, que leurs buts soient communs ou non.

[Bonnet et Tessier, 2007] distingue quatre types de coopération :
- la coopération pour éviter les conflits ;
- la coopération pour profiter de l'action d'un agent extérieur ;
- la coopération pour supprimer les redondances entre différents agents ;
- la coopération pour mener conjointement une action.

En regard des définitions précédentes, nous utiliserons la définition suivante :

Definition 4 (Coopération)
La coopération se définit comme l'interaction de plusieurs agents afin d'utiliser de manière efficace des ressources communes, nécessaires et limitées, pour la poursuite de buts communs ; ces agents doivent coordonner leurs actions dans un souci d'efficacité et doivent gérer les conflits dus aux imprévus ou à l'utilisation des ressources communes. □

1.2.3 Système opérateur humain - robot

Nous nous intéressons à des missions pour lesquelles le contrôle d'une ou plusieurs entités physiques est réalisé conjointement par un opérateur humain et un robot. Devant agir ensemble afin d'atteindre les objectifs de mission, le couple robot - opérateur humain peut être vu comme un système socio-technique, c'est-à-dire un ensemble de composants humains et techniques interagissant pour l'atteinte d'un but commun [Laprie *et al.*, 1995]. Par ailleurs, le couplage de deux agents pour le contrôle d'un système technique, peut, dans le cas où le robot possède un minimum de compréhension et de réactivité aux actions et intentions de l'opérateur, être dénommé sous le terme de contrôle coopératif [Flemisch *et al.*, 2008].

Dans certaines approches, le robot et l'opérateur humain sont modélisés indépendamment l'un de l'autre. L'opérateur humain est considéré comme un processeur d'information [Rouse, 1981, Wickens *et al.*, 1983], ce qui a conduit notamment à analyser ses performances et limitations du point de vue sensorimoteur (entrées - sorties). Le robot a alors pour rôle d'éviter à l'opérateur de se retrouver en dehors de son enveloppe de performance nominale. Cependant, on constate les limitations de cette approche en pratique lorsque l'opérateur humain agit avec une machine. En effet, la performance du système homme-machine n'est pas simplement la somme des performances individuelles de chaque agent, l'association des agents modifiant les rapports entre eux ainsi que la conduite du système vers les objectifs [Hollnagel, 2003].

[Hollnagel et Woods, 1983, Woods *et al.*, 1990] ont introduit le terme de Joint Cognitive System (JCS) : les composants du système, à savoir les agents artificiels et humains, sont réunis dans une relation de complémentarité : *a priori* l'un et l'autre ne peuvent réaliser la mission indépendamment l'un de l'autre [Brezillon et Pomerol, 1997]. Dans le cadre de l'approche JCS, le système homme-machine n'est pas simplement considéré comme l'association d'un robot et d'un opérateur humain modélisés séparément, mais est considéré comme un tout, et c'est cet ensemble dont les performances « externes » doivent être étudiées. De plus, dans cette approche, la modélisation ne concerne pas la composition interne de chacun des agents, mais plutôt les fonctions qu'il est capable

d'apporter au système : l'agent est considéré comme un *agent cognitif*, c'est-à-dire un agent qui n'est pas simplement réactif aux stimuli mais présente un comportement dicté par la satisfaction d'un but. Ce qui est notable ici est que ce point de vue est symétrique : le robot lui-même peut être considéré comme un agent cognitif [Hollnagel, 2003]. De plus, cela implique que les différents agents sont capables de se comprendre, ce qui est utopique dans le sens où la machine ne peut « comprendre » l'opérateur. Cependant, cela a des conséquences sur la façon de concevoir le robot, qui doit tenir compte du raisonnement de l'opérateur, être capable de lui expliquer son propre raisonnement et partager le contrôle du système.

Du point de vue de l'opérateur, le robot n'est alors plus considéré comme une simple machine, mais bel et bien comme un agent autonome [Hoc, 2000]. Enfin, puisque chacun des agents lui-même peut être considéré comme un système complexe (d'autant plus que la modélisation complète de l'opérateur n'est *a priori* pas possible), l'association des deux conduit à la création d'un système de systèmes [Luzeaux, 2004].

Definition 5 (Système)
Nous nommerons *système* l'ensemble formé par le couple robot-opérateur humain ainsi que la (ou les) entité(s) physique(s) qu'ils contrôlent conjointement (incluant également les infrastructures matérielles permettant le lien entre agents et entité physique) pour réaliser une mission. Nous considérons un système coopératif, où les agents partagent les mêmes buts de mission. □

2 Coopération homme-machine et place de l'opérateur humain

Dans ce paragraphe, nous nous intéressons à la relation entre l'homme et la machine, lorsque ceux-ci travaillent ensemble. Dans ce cadre, les termes « machine » ou « automatisation » (plus précisément au sens du terme anglais *automation*) ne font pas nécessairement référence à un robot, mais, pour reprendre la définition donnée par [Amalberti, 2002], à « toute aide qui effectue en série ou en parallèle de l'opérateur des opérations de tri, de décision, et d'action habituellement dévolues à cet opérateur (où qui furent à un moment dévolues à l'opérateur) » ; de leur côté, [Sheridan et Parasuraman, 2006, Moray, 2000] considèrent l'automatisation comme l'ensemble des processus artificiels intervenant de la perception de l'environnement jusqu'aux actions sur cet environnement.

Nous présentons ici des considérations générales sur les systèmes homme-machine, de l'intérêt de réaliser de tels systèmes aux problématiques induites par l'interaction entre ces deux types d'agents. Ces considérations constituent le contexte général du travail présenté dans ce manuscrit, avec ses difficultés et ses contraintes, qui justifient les

choix conceptuels et techniques réalisés par la suite. Les thématiques exposées dans ce paragraphe ne constituent en aucun cas le propos principal de la thèse et ne seront pas développées dans les chapitres suivants.

2.1 Pourquoi faire travailler ensemble homme et machine ?

2.1.1 Couplage des aptitudes

Il y a plusieurs raisons qui peuvent conduire à la réalisation de systèmes faisant appel au contrôle mixte : amélioration des performances de l'humain, éloignement de fait de l'opérateur humain, réduction du nombre d'opérateurs [Mitchell *et al.*, 2005].

L'intérêt du contrôle mixte repose sur l'hypothèse que les faiblesses d'un agent de contrôle peuvent être compensées par les aptitudes de l'autre, et inversement, constituant ainsi un système hybride. Plus concrètement, il est attendu de l'automatisation de permettre de réduire la charge de travail humaine, de réduire les coûts opérationnels, d'augmenter la précision et de réduire le nombre d'erreurs produites par le système [Sarter *et al.*, 1997].

Détaillons maintenant quelques caractéristiques de l'automatisation et de l'humain pris chacun de manière indépendante.

2.1.2 Comparatif des contributions de l'automatisation et de l'opérateur étudiés indépendamment

Depuis l'introduction de l'automatisation dans les systèmes, les concepteurs de systèmes ont eu besoin de critères pour identifier ce qu'il était pertinent d'automatiser ou de ne pas automatiser, et pour établir les capacités de l'opérateur et du robot afin de répartir les tâches de la meilleure manière possible. L'approche « historique », telle qu'effectuée par Fitts dès 1951, a consisté en une analyse comparative des bénéfices de l'automatisation et de l'opérateur (voir tableau 1.2).

Les humains sont meilleurs pour :	Les ordinateurs sont meilleurs pour :
Extraire des motifs	Répondre rapidement aux tâches de contrôle
Améliorer, utiliser des procédures flexibles	Éxécuter des tâches répétitives
Utiliser les faits appropriés au bon moment	Produire un raisonnement déductif
Produire un raisonnement inductif	Gérer plusieurs tâches complexes
Produire un jugement	Calculer rapidement et précisément

TABLE 1.2: Liste de Fitts pour l'affectation de rôles entre homme et machine

L'automatisation est généralement caractérisée par sa rigidité de fonctionnement et son incapacité à réagir à l'imprévu, phénomène décrit sous le nom de « fragilité » (*brittleness*, [Layton *et al.*, 1994]). Inversement, l'opérateur dispose de capacités inégalables, telles que l'imagination, la flexibilité, et surtout, le raisonnement inductif, c'est-à-dire la capacité à produire des généralisations à partir de quelques cas isolés. Ces capacités permettent à l'humain de faire face à tout type de situation et de s'y adapter, sur la base d'un raisonnement fondé sur ses connaissances, qui permet la prise de décision en situation nouvelle et incertaine [Rasmussen, 1983].

Les opérateurs humains font régulièrement des erreurs ; cependant il est à noter qu'ils sont capables de récupérer la plupart de celles-ci [Rizzo *et al.*, 1987, Visciola *et al.*, 1992]. Il existe toutefois un type particulier d'erreur, la persévération, connue encore sous le nom de fascination pour l'objectif [BEA, 2000] ou d' « erreur diabolique » [Wanner et Wanner, 1999] : elle a pour caractéristique une mauvaise gestion des ressources attentionnelles par l'opérateur [De Keyser et Woods, 1990, Amalberti, 1996]. L'information pertinente n'est plus perçue au moment opportun ou se focalise uniquement sur un sous-problème particulier : l'opérateur peut commettre des erreurs et mettre en danger le système qu'il contrôle. Ce type d'erreur peut trouver son origine dans certains mécanismes cognitifs, [Berthoz, 2003] avançant par exemple que la persévération est liée à un trouble de la fonction d'inhibition, ce qui a été observé en tant que pathologie chez des patients ayant des lésions cérébrales.

L'homme et la machine disposent de capacités propres qu'ils peuvent apporter et mutualiser lors de la réalisation d'un système hybride homme-machine. Cependant, la frontière définissant ce à quoi les humains sont meilleurs que la machine et inversement est, en dehors de tâches prises isolément, difficile à définir en pratique. C'est pourquoi certaines approches se concentrent sur les fonctions à réaliser par le système au lieu de placer les tâches comme éléments centraux, et l'approche par allocation de *fonctions* consiste à définir les répartitions des fonctions entre les différents agents. De nombreuses études ont été faites sur le sujet, conduisant à de nouvelles méthodes de conception [Coye de Brunélis et Le Blaye, 2009]. Malgré tout, les critères pour l'allocation de fonctions à proprement parler restent en eux-mêmes sujets à débats [Hancock, 1996, Sheridan, 2000].

2.2 Problématiques liées à l'automatisation

De nombreux cas d'accidents liés aux interactions homme-machine ont été rapportés, en particulier dans le domaine de l'aéronautique [Billings, 1996, Parasuraman et Riley, 1997, Parasuraman et Byrne, 2003] mais également dans les domaines du transport ferroviaire, routier, maritime. Parmi les problèmes les plus fréquents on relève un défaut

d'information de la part des automatismes sur ce qu'ils sont en train de réaliser, le manque de compréhension de l'automatisation par l'opérateur, la surconfiance en l'automatisme, une mauvaise conception de l'interface utilisateur et un entraînement inapproprié des opérateurs [Funk *et al.*, 1999].

Les problématiques liées à l'automatisation peuvent être réunies en trois grandes classes : les problématiques liées à la cognition de l'opérateur interagissant avec l'automatisation, la représentation qu'a l'opérateur du système et des actions entreprises par l'automatisation, et enfin la confiance que l'opérateur accorde au système.

2.2.1 Impact de l'automatisation sur la cognition de l'opérateur

L'introduction de l'automatisation, au lieu de décharger l'opérateur d'un certain nombre de tâches, a modifié son rôle dans le système, lui conférant celui de superviseur. Plus précisément, ce type d'interaction a été nommé *contrôle de supervision* et a fait l'objet de nombreuses études [Moray, 1986, Sheridan et Verplank, 1978]. Dans ce cadre, on peut considérer que l'humain peut avoir cinq fonctions vis-à-vis de la machine [Sheridan et Parasuraman, 2006] :

- planifier hors ligne
- enseigner à la machine
- surveiller l'exécution du plan par la machine
- intervenir si nécessaire
- apprendre par expérience

La conception des systèmes ainsi que l'entraînement des opérateurs doivent nécessairement être réalisés en tenant compte de ce transfert de rôle.

Pour un système automatisé supervisé par un opérateur humain, on peut distinguer quatre grands processus qui doivent être effectués séquentiellement par le couple homme-machine, sur la base d'un modèle humain du traitement de l'information (voir figure 1.2) [Parasuraman *et al.*, 2000] : le premier processus est la fonction de captation de l'information, suivie par la fonction de perception, qui opère un premier rapprochement de l'information brute avec l'information en mémoire de travail ; vient ensuite la fonction de prise de décision, complétée par la fonction de choix de réponse et de mise en exécution.

Chacun de ces processus peut être automatisé plus ou moins fortement en fonction de la mission, des besoins et des limites technologiques. En revanche, l'automatisation ne doit pas être directive et implémenter un seul type de stratégie, mais plutôt constituer un support flexible de la prise de décision de l'opérateur favorisant le jugement et le raisonnement [Crandall et Cummings, 2008].

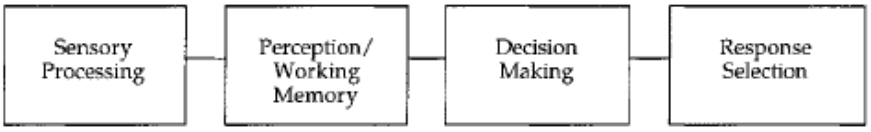

FIGURE 1.2: Les étapes séquentielles pour un système en supervision [Parasuraman *et al.*, 2000]

2.2.2 Compréhension de l'automatisme et conscience de situation de l'opérateur

Afin que le couplage entre l'homme et la machine se fasse au mieux, l'opérateur humain doit comprendre ce que réalisent les automatismes, leur fonctionnement interne, leurs états courants. Ceci peut lui permettre de se donner une représentation mentale du système suffisante pour pouvoir utiliser le système efficacement et éviter les surprises dues à l'automatisation [Sarter *et al.*, 1997]. De nombreuses études mettent en avant la nécessité de réaliser des systèmes dits « transparents » ou « human-like » afin de faciliter la transmission de l'information [Sarter et Amalberti, 2000].

La conscience de situation est un concept introduit par [Endsley, 1995] pour qualifier la compréhension de l'état actuel du système et de la mission par l'opérateur humain à un instant donné. Elle se décompose en trois niveaux successifs :

- la perception des éléments de l'environnement et de l'état du système ;
- la compréhension des paramètres perçus ;
- la projection dans le temps de ces éléments et de leur évolution future, étape permettant à l'opérateur d'anticiper les modifications à venir dans le système et l'environnement à partir des informations perçues et intégrées précédemment.

La conscience de situation est fondamentale pour l'opérateur afin qu'il puisse contrôler le système de manière pertinente, efficace et anticipée. Elle est d'autant plus importante lorsque des aléas surviennent et que l'opérateur doit réagir pertinemment et rapidement, par exemple pour produire des stratégies de réparation.

Cependant, par l'introduction de l'automatisation, l'opérateur humain est bien souvent poussé en dehors de certaines boucles décisionnelles du système, ce qui a pour effet de dégrader sa conscience de situation [Endsley, 1996]. En effet, en fonction de son niveau de conscience de situation, l'opérateur peut assurer ses fonctions de supervision du système avec plus ou moins de succès ; en ce sens, la conscience de situation est un bon indicateur de la performance du système homme-machine [Donmez *et al.*, 2009, Cummings et Mitchell, 2008].

2.2.3 Confiance et surconfiance en l'automatisme

La confiance [Lee et See, 2004, Parasuraman et Riley, 1997] définit comment l'opérateur accepte d'interagir avec la machine. La confiance de l'opérateur envers l'automatisation peut se bâtir sur deux considérations principales [Sheridan et Parasuraman, 2006] :

- la fiabilité perçue par l'opérateur de l'automatisme, en terme de répétabilité, cohérence de fonctionnement ;
- la robustesse perçue par l'opérateur du comportement de l'automatisme, c'est-à-dire sa capacité à fonctionner de manière satisfaisante sur un ensemble de situations différentes, et la sensation de comprendre son fonctionnement en règle générale (procédures utilisées, anticipation possible du comportement).

La confiance que l'opérateur a en une fonction automatisée prise isolément influe notamment sur la façon dont il va interagir avec elle [Crocquesel *et al.*, 2010].
Le phénomène de surconfiance en l'automatisme a été mis en lumière par [Layton *et al.*, 1994, Mosier *et al.*, 1998, Parasuraman et Riley, 1997] sous le terme de *biais d'automatisation* : il s'agit d'un phénomène se produisant lorsque l'opérateur se repose sur les automatismes sans adopter l'attitude critique nécessaire, ou sans chercher à recouper l'information contradictoire pertinente. On peut distinguer deux types de biais d'automatisation [Mosier *et al.*, 1998] : l'erreur par délégation, consistant pour l'opérateur à continuer à obéir aux recommendations d'un automatisme sans en vérifier la pertinence, et l'erreur par omission, consistant pour l'opérateur à ne pas intervenir alors que l'automatisme est défaillant. Toute la problématique repose dans le fait que l'opérateur doit calibrer correctement la confiance qu'il accorde dans l'automatisme, afin de ne basculer ni dans la surconfiance, ni dans l'excès inverse : si le manque de confiance en l'automatisme est avéré, il se peut que celui-ci ne soit tout simplement pas utilisé par les opérateurs [de Vries *et al.*, 2003]. [Moray et Inagaki, 2000] considèrent qu'il n'existe aucune approche fiable permettant à l'opérateur de définir le niveau approprié de confiance à accorder à l'automatisme : pour cette raison, ils proposent que l'opérateur vérifie le bon fonctionnement de l'automatisme à une fréquence proportionnelle à la fréquence de défaillance de l'automatisme. Dans tous les cas il ne s'agit pas d'une problématique résolue.

Enfin, une problématique surgit avec l'introduction des systèmes d'assistance et le contrôle mixte (par exemple les aides au suivi de trajectoire pour les pilotes ou les conducteurs automobiles), qui combinent dans des proportions variables les consignes de l'opérateur avec celles en provenance d'algorithmes : en brouillant la frontière entre le contrôle du système par l'opérateur ou les automatismes, l'opérateur peut basculer de la confiance en la surconfiance, en ayant notamment l'illusion de contrôler parfaitement un système sur lequel il n'a en réalité qu'un contrôle partiel. Cet effet peut être recherché ou au contraire repoussé par les concepteurs de système ; en ce sens, les critères conduisant au sentiment

de contrôle et d'autorité sur le système par l'opérateur (désigné également sous le terme d'« agency » en anglais) sont l'objet d'études actuelles [Berberian *et al.*, 2010]. Toutefois, des métriques pour évaluer la surconfiance et adaptées à des systèmes à contrôle mixte ont été développées [Inagaki et Itoh, 2010].

2.3 Place de l'opérateur humain

2.3.1 Rôle(s) pris en charge par l'opérateur humain dans le système

L'opérateur humain peut intervenir de différentes manières sur l'entité physique, de l'envoi direct de commandes (téléopération) jusqu'aux ordres de haut niveau qu'il transmet au robot en supervision. En inversant les positions, il est également concevable que le robot demande lui-même à l'opérateur de réaliser des tâches pour l'accomplissement de la mission. On constate que les rôles de l'opérateur peuvent être d'une grande variété et nécessitent d'être définis ; c'est l'objectif de ce paragraphe.

La figure 1.3 montre la position classique de l'opérateur vis-à-vis des lois de commande appliquées sur l'entité physique : l'opérateur donne les consignes, les lois sont élaborées afin que l'entité physique délivre les sorties désirées en respectant un certain nombre de critères de performance (stabilité, précision, temps de réponse, robustesse aux perturbations, etc.)

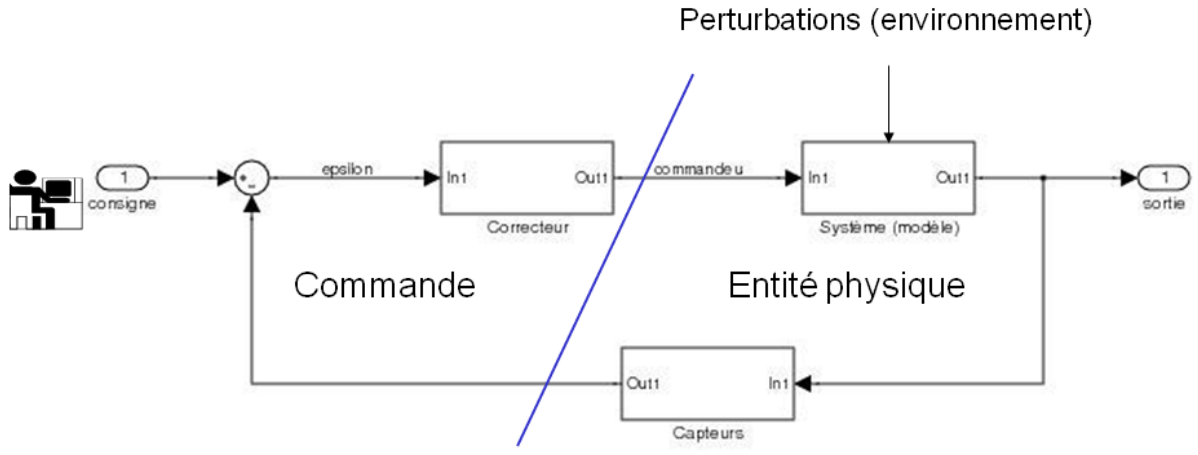

FIGURE 1.3: Asservissement et consignes de l'opérateur

La partie droite de la figure représente la partie « physique » du système, alors que la partie gauche représente la partie commande. Le système (dont les actuateurs) reçoit les commandes et les applique, conduisant le véhicule physique, plongé sans

son environnement, à évoluer. Ces évolutions sont mesurées par des capteurs, et cette information permet d'ajuster les commandes envoyées au système *via* un correcteur afin que le véhicule physique évolue comme attendu, suivant un asservissement en boucle fermée.

Lorsque le robot est doté de capacités décisionnelles (*via* les fonctions de suivi de l'état et de planification), l'opérateur peut intervenir de différentes manières. La figure 1.4 montre un système à « forte autonomie », dans lequel l'opérateur définit l'objectif mais n'intervient pas dans les décisions du robot.

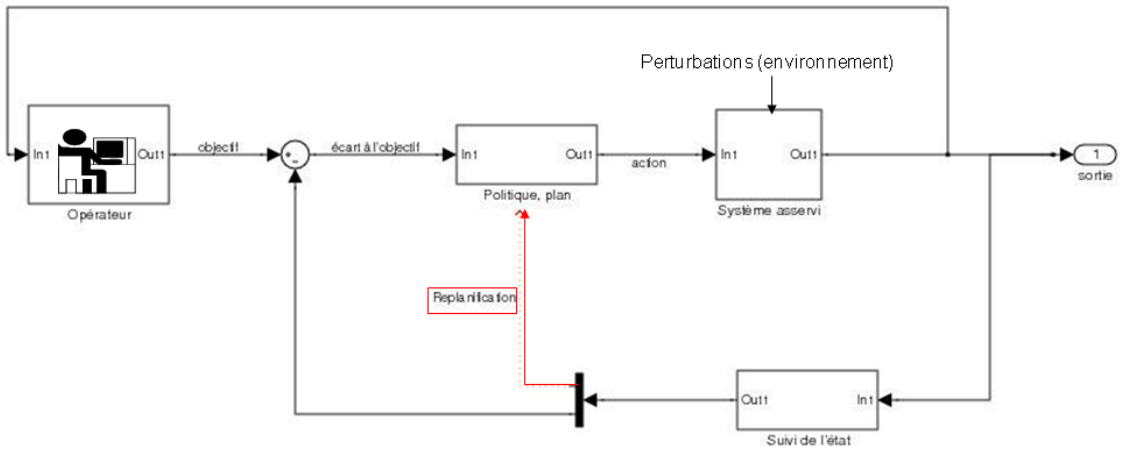

FIGURE 1.4: Planification et objectifs définis par l'opérateur

Dans le système de la figure 1.5, pour un objectif fixé, l'opérateur peut intervenir sur le processus de génération de plan (en modifiant par exemples les pondérations entre critères, les seuils, en indiquant ses préférences). Il s'agit de planification à *initiative mixte* (voir section 1.3.2).

Enfin la figure 1.6 montre le cas d'un système dans lequel le robot et l'opérateur ont les capacités d'action sur l'entité physique. Le problème que nous traitons dans cette thèse se situe dans ce cadre : en cas d'actions simultanées, ou contradictoires, des deux agents, comment réaliser leur agrégation ? Ou au contraire, faut-il en privilégier une par rapport à l'autre ? Les consignes de l'opérateur doivent-elles supplanter celles d'un algorithme, s'additionner de manière pondérée, être ignorées ?

On remarque dans ce cadre que l'opérateur prend ses décisions à l'aide d'informations en provenance du robot, mais réciproquement le robot ne perçoit pas l'opérateur. La figure 1.7 propose de rendre « symétriques » les rôles des deux agents : la fonction de suivi de l'état du robot est également alimentée par des informations relatives à l'« état » de l'opérateur (par exemple, actions sur l'interface, oculométrie, voire informations issues de capteurs physiologiques).

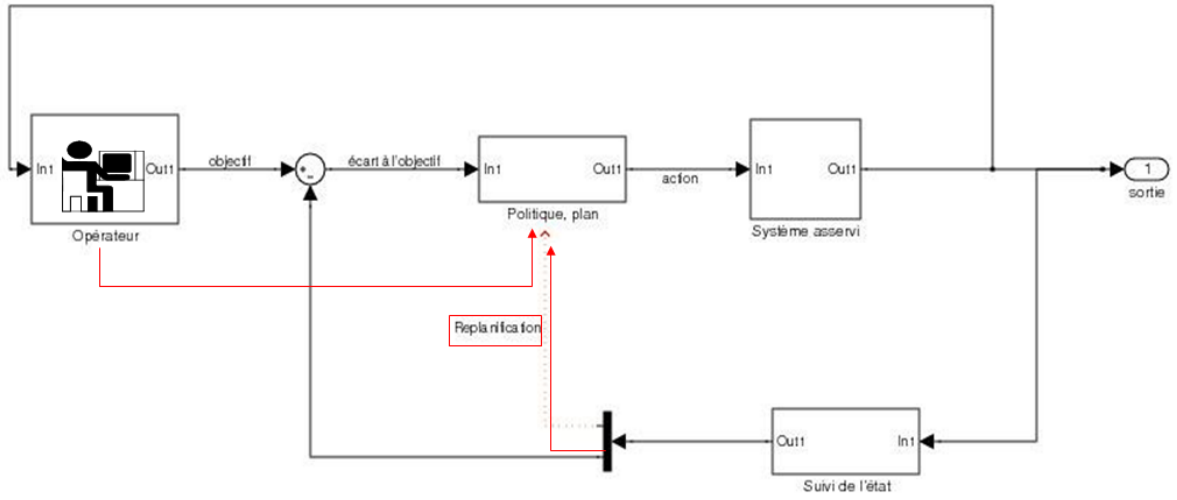

FIGURE 1.5: Initiative mixte sur la planification

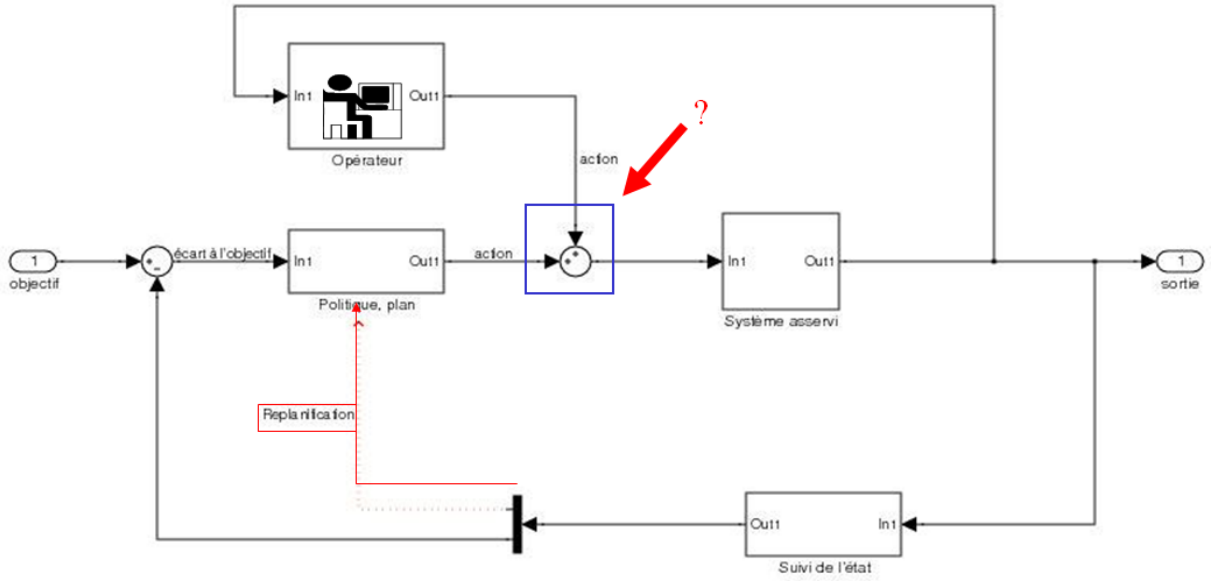

FIGURE 1.6: Comment combiner les actions du robot et de l'opérateur ?

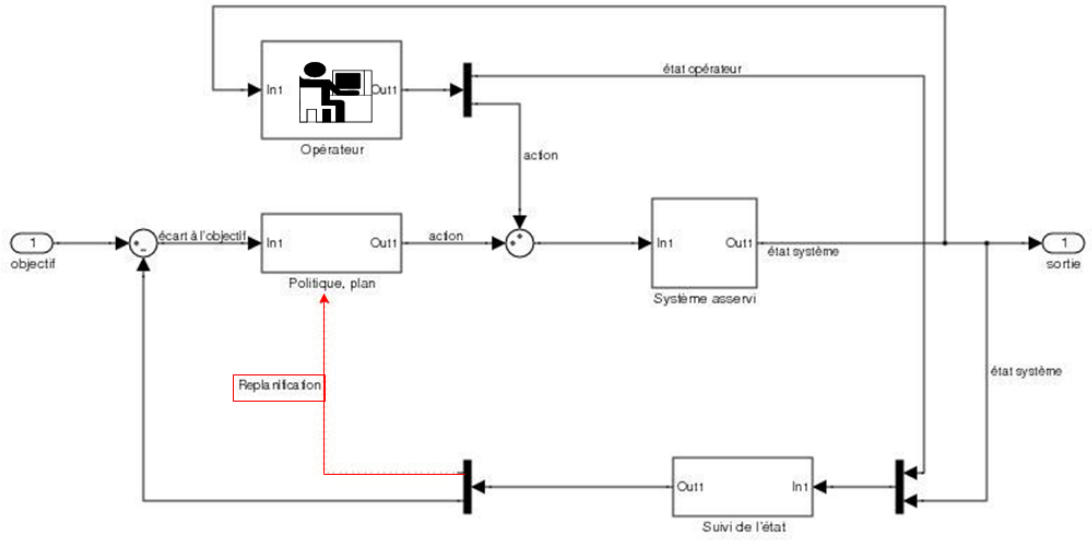

FIGURE 1.7: Prise en compte des états de l'opérateur

2.3.2 Place du robot depuis la perspective de l'opérateur

Les nouvelles formes d'automatisation modifient la perception que l'opérateur peut avoir du robot : le robot n'est plus simplement un outil à disposition de l'opérateur humain, en effet il peut corriger ses erreurs et lui déléguer des tâches.

L'opérateur est censé désormais lui allouer sa confiance (à sa juste mesure) ; il doit pouvoir communiquer ses intentions au robot afin que celui-ci le soutienne, et le robot peut venir contredire certaines des décisions de l'opérateur. [Woods, 1996] a mis en exergue le fait qu'ajouter de l'automatisation dans le but d'assister l'opérateur humain revient à ajouter un membre supplémentaire à l'équipe, qui ne parle pas nécessairement la même langue, et ne partage pas la même culture. Cela peut déboucher sur ce que [Sarter *et al.*, 1997] présentent comme les « surprises de l'automatisation », qui ramène l'opérateur à se poser les questions de [Wiener, 1985] : que fait l'automatisation ? Pourquoi ? Et que va-t-elle faire après ? Dans ce cadre, [Woods, 1996] affirme que l'augmentation de l'automatisation nécessite une augmentation de la coordination. [Christoffersen et Woods, 2002] vont même jusqu'à affirmer qu'il ne s'agit plus tant d'un problème d'autonomie et d'autorité que d'un problème de coopération et d'observabilité, ce qui suppose d'unifier les représentations, entre le modèle mental de l'opérateur et le modèle de le robot. Cependant, pour que le robot puisse vraiment être considéré comme un membre de l'équipe, [Klein *et al.*, 2004] considère qu'entre autres défis, le robot doit, au même titre que l'opérateur humain agissant avec d'autres opérateurs, être capable de modéliser les intentions de l'autre et anticiper ses actions.

3 Apports de la thèse et contributions

Les travaux réalisés dans le cadre de la thèse ont pour objectif la réalisation d'un *contrôleur de la dynamique de l'autorité*, fonction d'un robot réalisant une mission en co-opération avec un opérateur humain, et permettant à ce robot de faire varier de lui-même son comportement et son autonomie sur la base de considérations d'autorité. L'originalité de l'approche réside dans le fait de s'abstraire de la notion de niveaux d'autonomie fréquemment utilisées dans la littérature, et de contrôler finement l'autorité des agents sur des éléments génériques de la mission (*ressources*).

3.1 Contrôle de l'autonomie d'un robot sans utilisation de niveaux d'autonomie

Nous proposons une approche fondée sur la détection et la résolution de conflit, un conflit se manifestant dans l'exécution du plan lors de la survenance d'un aléa. Nous considérons qu'il s'agit d'un indicateur tout-à-fait pertinent pour déclencher l'adaptation du comportement du robot vis-à-vis de l'opérateur : nous utilisons le conflit comme signal conduisant le robot à réagir et à adapter son autonomie relativement à l'opérateur. Ainsi, en mettant à jour continuellement son modèle des actions en cours et à venir, qu'il en soit le commanditaire, l'exécutant ou qu'il en soit simplement informé, et en y incluant toute la chaîne de dépendances associées, le robot ajuste son comportement sur la base des droits qui lui sont accordés au niveau des ressources *via* les relations d'autorité. Le niveau de granularité de la modélisation n'est en rien imposé par notre outil de modélisation, le choix en est laissé au concepteur suivant ses besoins pour la réalisation d'une mission. De plus, nous ne faisons à aucun moment appel à des niveaux d'autonomie prédéfinis en ce sens où notre modélisation n'impose aucune répartition préétablie des tâches entre robot et opérateur : celle-ci s'établit d'elle-même en fonction des ressources disponibles, de l'autorité établie entre le robot et l'opérateur et des « savoir-faire » dont dispose le planificateur du robot.

3.2 Macro- et micro-modèle de l'autorité

Le modèle de l'autorité que nous proposons s'appréhende à deux niveaux distincts : le macro-modèle et le micro-modèle.

Le macro-modèle de l'autorité permet de manipuler les concepts de ressources, d'interfaces et de relations d'autorité. Les ressources sont des éléments génériques obtenus à partir du plan d'exécution des agents sur la mission lorsque l'on représente celui-ci sous la forme de dépendances. Les interfaces matérialisent les relations de dépendances entre ressources. Enfin, les relations d'autorité permettent de matérialiser les droits des différents agents du système (i.e. opérateur humain et robot), définissant l'utilisation qu'ils peuvent faire des ressources. Le macro-modèle constitue un outil de modélisation

permettant de représenter tous les éléments nécessaires à un agent pour réaliser un plan de tâches lors de sa mission, en vue d'atteindre des objectifs. Ce type de modélisation est suffisamment flexible pour permettre l'inclusion de données relatives à l'opérateur, afin d'améliorer l'interaction homme-robot ; de plus, par son approche générique et associé à une représentation graphique, cet outil permet la création rapide de nouveaux modèles.

Le micro-modèle constitue l'implémentation formelle du macro-modèle de l'autorité sous la forme d'un ensemble de réseaux de Petri synchronisés entre eux. Ces réseaux de Petri incorporent l'intégralité des différentes propriétés que nous avons définies sur les ressources et les interfaces. Le micro-modèle de l'autorité rend possible l'exécution immédiate du modèle de l'autorité, son évolution étant définie par le formalisme des réseaux de Petri, ainsi que par leurs connexions aux autres fonctions de l'architecture robotique. L'exécution du micro-modèle permet d'effectuer la détection d'aléas survenant en mission, qu'il s'agisse de défaillances ou d'interférences résultant des actions d'agents non coordonnés entre eux.

3.3 Mise en œuvre du contrôleur de la dynamique de l'autorité

Les modèles construits suivant le formalisme que nous proposons permettent la détection de conflit pendant l'exécution du plan. Pour cela, le contrôleur de la dynamique de l'autorité que nous proposons met à jour en permanence le modèle représentant le plan en cours et à venir, ainsi que les relations d'autorité entre agents sur les ressources concernées. Lorsqu'un conflit apparaît au sein du modèle, il est détecté par le contrôleur, qui, en conjonction avec le planificateur du robot, exécute des méthodes de résolution. Nous proposons des méthodes de résolution de conflit qui diffèrent suivant le type de conflit et des contraintes que l'on impose au robot en terme d'autorité. En effet, afin de réagir à l'apparition de conflit, une reconfiguration du plan du robot, voire du plan de l'opérateur, est nécessaire. Pour éviter l'apparition d'incohérences lors de la résolution de conflit, des arbitrages entre le robot et l'opérateur sont à réaliser. C'est en cela que les relations d'autorité sont pertinentes : elles permettent de réguler l'usage que les agents font des ressources sous forme de droits relatifs, rendant possible la résolution automatique des conflits, si la situation et les connaissances du robot le permettent. De plus, notre outil de modélisation ouvre la voie au changement dynamique de l'autorité des agents sur les ressources directement en mission, sur requête du robot tout comme de l'opérateur.

4 Organisation du mémoire

Ce mémoire est composé de cinq chapitres principaux.

Le premier chapitre consiste en un état de l'art relatif au concept d'autonomie dans un système opérateur humain - robot. Des approches permettant d'évaluer, puis de faire

varier l'autonomie d'un robot dans ce cadre sont présentées et critiquées, en particulier le concept des niveaux d'autonomie. Une classification des modes d'initiative entre l'opérateur et le robot est présentée, suivie d'une présentation des critères à même de déclencher un changement d'autonomie ; ceci nous permet d'introduire le concept de conflit que nous utilisons par la suite. Enfin, nous expliquons pourquoi et comment nous effectuons un glissement du concept d'autonomie aux concepts d'autorité et de partage d'autorité.

Les deuxième et troisième chapitres se concentrent sur la définition d'un modèle formel permettant la représentation d'un plan d'exécution pour le robot et l'opérateur humain sous forme d'éléments génériques liés entre eux par des relations de dépendance, et sur lesquels nous établissons le concept de relation d'autorité. Ce modèle est présenté à deux niveaux différents, d'abord le macro-modèle se présentant sous forme d'un graphe de *ressources*, puis le micro-modèle, fondé sur des réseaux de Petri génériques. Le troisième chapitre traite plus particulièrement du concept de conflit, de sa survenue au sein du modèle, et des méthodes de résolution.

Le quatrième chapitre explique la réalisation du contrôleur de la dynamique de l'autorité, fonction du robot permettant à celui-ci de se positionner en terme d'autorité sur les actions qu'il doit réaliser ou non au cours de la mission, sur la base du modèle présenté dans les chapitres 2 et 3. La mise en œuvre de ce contrôleur, aussi bien du point de vue de son intégration au sein de l'architecture du robot et du système incluant l'opérateur que dans ses mécanismes de fonctionnement interne, est présentée.

Le cinquième chapitre présente l'application qui a été faite de ce contrôleur de la dynamique de l'autorité : nous considérons un robot terrestre devant réaliser une mission sous le contrôle d'un opérateur humain, mission dans laquelle nous faisons survenir certains aléas. Le contexte applicatif et les modèles utilisés pour la mission sont présentés.

Enfin, le mémoire se conclut sur une synthèse de nos travaux. Des perspectives de travail sont présentées et discutées, sur la façon dont ce contrôleur de la dynamique de l'autorité pourrait être amélioré, mis en œuvre en conjonction avec des opérateurs humains, et évalué. De plus, nous expliquons comment les techniques utilisées dans cette thèse peuvent être adaptées à d'autres problématiques.

Chapitre 2

Autonomie relative entre agents

Ce chapitre est consacré à la littérature relative aux systèmes impliquant des agents hétérogènes (présence simultanée d'agents robotiques et d'opérateurs humains) pour la conduite d'une mission et touchant à la notion d'autonomie. Même dans ce cadre restreint, la notion d'autonomie peut recouvrir des concepts très différents et mal définis ; cependant, nous en conservons son caractère relatif instauré entre les agents. L'autonomie semble pouvoir être graduée : certaines approches évaluatives visent à mesurer l'autonomie de systèmes existants pour pouvoir les classer, alors que d'autres approches prescriptives visent à concevoir des systèmes devant exhiber une autonomie prédéfinie. Suivant cette direction, nous avons relevé une classification des systèmes hétérogènes suivant trois catégories : les systèmes à autonomie adaptative, les systèmes à autonomie ajustable et les systèmes à initiative mixte. Nous analysons ensuite la place qui est accordée à l'opérateur humain depuis la perpective du robot, comment celui-ci est modélisé ou comment ses interactions sont intégrées. Enfin, nous passons en revue certaines techniques et métriques utilisées pour déclencher un changement d'autonomie de la part du robot vis-à-vis de l'opérateur humain. En ce sens, le conflit ressort comme un indicateur pertinent du besoin d'évolution de l'autonomie. Enfin, nous expliquons en quoi le concept d'autorité se raccroche à la notion d'autonomie, dans le but d'aborder de manière égale et non biaisée les relations entre les agents humain et robotique.

1 Autonomie et contrôle de l'autonomie

L'autonomie est une propriété abstraite qui caractérise une personne, un agent, un système. Il semble que l'autonomie ne soit pas une fonction bien identifiée et localisable, mais plutôt une propriété émergente, résultant de l'expression de toutes les composantes d'un système. Immédiatement, il apparaît que l'autonomie touche à des notions aussi vastes que la prise de décision, l'initiative, la liberté d'action, et l'on prend la mesure de la

difficulté de l'expliciter formellement. Il serait donc prétentieux d'essayer d'en donner une définition absolue ; cependant, dans le cadre de notre problématique, nous pouvons dégager certaines caractéristiques de l'« autonomie » : voyons dans un premier temps quelques considérations générales sur l'autonomie dans le contexte de relations entre agents, puis comment celles-ci se projettent sur le cadre applicatif de l'interaction homme-robot.

1.1 Considérations générales sur l'autonomie des agents

L'autonomie d'un agent artificiel est difficile à appréhender, en particulier dès que l'on considère qu'elle requiert une certaine liberté décisionnelle voire une « libre volonté » [Clough, 2002], ou même une imprédictibilité de comportement [Steels, 1995]. Cependant il est possible d'en dégager certaines caractéristiques plus opérationnelles. Tout d'abord, un agent évolue normalement dans un contexte social : les autres entités en présence, tout comme les institutions dans lesquelles ils évoluent, peuvent influer sur cet agent, affectant sa liberté de décision et son comportement sous la forme d'engagements sociaux [Carabelea et Boissier, 2006]. [Castelfranchi et Falcone, 2003] proposent de considérer l'autonomie comme une relation entre plusieurs entités et portant sur un objet, par exemple un but. L'autonomie s'écrit alors sous la forme d'un triplet (X, μ, Z), X étant le sujet (correspondant à l'agent artificiel), μ une décision / action / but, Z le sujet secondaire évaluant l'autonomie, c'est-à-dire ici l'opérateur. Sur le même principe, [Brainov et Hexmoor, 2003] conservent la vision de l'autonomie comme une notion relationnelle en l'étendant, ajoutant le groupe des influençants ainsi qu'une mesure de performance. L'autonomie s'écrit alors (sujet, objet, influençants, mesure de performance). Les influençants regroupent les autres agents en présence pouvant altérer l'autonomie du sujet considéré, ce qui inclut l'opérateur. La mesure de performance est déterminée par ce dernier, définissant les objectifs à atteindre ainsi que la métrique de performances associée.

On note toutefois que ces travaux représentent dans leur ensemble des considérations de haut niveau et s'appliquent à des agents supposés idéaux, au comportement parfaitement modélisé.

1.2 L'autonomie dans un système homme-robot

Dans le cadre plus précis du contrôle d'un agent évoluant dans le monde physique et supervisé par un opérateur humain, l'autonomie se caractérise principalement par la capacité d'un agent à minimiser le besoin de supervision et à évoluer seul dans le monde réel [Schreckenghost *et al.*, 1998]. En conservant le principe d'une relation entre l'opérateur et l'agent, on constate ainsi que l'autonomie pure n'est qu'un cas extrême de cette relation, consistant précisément à la réduire à son minimum. Cependant, dans la pratique, les objectifs d'une mission ne peuvent pas toujours être atteints de façon purement autonome par un agent artificiel et l'intervention de l'opérateur est nécessaire. [Goodrich *et al.*, 2001, Goodrich *et al.*, 2007] illustrent ce point en rapprochant le temps pendant lequel un robot peut agir sans intervention de l'opérateur et ses performances

sur le terrain : celles-ci déclinent rapidement lorsque le robot est négligé par l'opérateur. Un équilibre doit donc être trouvé entre le contrôle purement manuel, qui permet en général d'avoir une grande confiance dans le système mais qui soumet l'opérateur humain à une charge de travail importante, et l'autonomie totale, qui décharge l'opérateur mais offre moins de garanties en environnement incertain [Brookshire *et al.*, 2004]. Ainsi, de façon à tirer parti des compétences complémentaires des agents humain et robotique, la variation de l'autonomie de l'agent robotique est largement considérée.

Les *approches évaluatives* proposent une mesure *a posteriori* de l'autonomie d'un agent robotique qui réalise une mission particulière, afin par exemple de pouvoir comparer des systèmes différents entre eux : on peut citer par exemple MAP [Hasslacher et Tilden, 1995], ACL [Clough, 2002] ou Alfus [Huang *et al.*, 2005]. Cette dernière (voir figure 2.1) propose d'évaluer l'autonomie suivant trois dimensions : complexité de mission, difficulté environnementale et interface avec l'opérateur. Cette méthode présentée comme objective nécessite cependant d'agréger sur chaque axe les résultats d'un grand nombre de métriques hétérogènes, ce qui pose le problème de la sémantique du résultat.

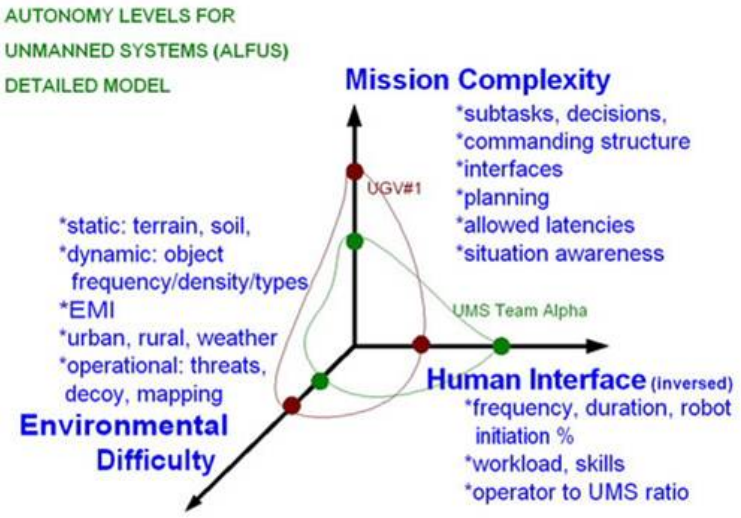

FIGURE 2.1: Évaluation de l'autonomie suivant la méthodologie Alfus [Huang *et al.*, 2005]

Les *approches prescriptives* s'intéressent à la conception de systèmes intégrant le contrôle de l'opérateur, en posant les questions de la définition des niveaux d'autonomie et du passage d'un niveau à un autre. On peut ranger ces approches selon plusieurs axes : l'*objet* sur lequel porte l'autonomie, le *mode d'initiative* adopté pour faire évoluer l'autonomie, les *critères* pour décider de cette évolution, la *perception* qu'a l'agent artificiel de l'agent humain.

1.3 Contrôler l'autonomie de l'agent artificiel

1.3.1 L'objet de l'autonomie

Deux objets principaux sont considérés : le *rôle* et la *tâche*.

Un *rôle* est défini comme un ensemble de tâches à assurer par un agent donné [Hardin et Goodrich, 2009]. Les spécifications selon des niveaux d'autonomie correspondent alors à une répartition pré-établie des rôles entre l'agent humain et l'agent artificiel : [Sheridan et Verplank, 1978] avaient proposé dès 1978 une classification de l'autonomie opérationnelle (« automation ») d'un système robotique sur dix niveaux (voir tableau 2.1). Cependant ce modèle est abstrait et ne tient pas compte de la complexité de l'environnement dans lequel évolue le robot, ni du contexte de mission.

Niveau	Description de l'automatisation
1	Le système n'apporte aucune assistance, l'opérateur prend toutes les décisions et actions.
2	Le système propose un ensemble complet de décisions / actions alternatives.
3	Le système propose une sélection de décisions / actions.
4	Le système suggère une action / décision en particulier.
5	Le système exécute une action / décision si l'opérateur approuve.
6	Le système donne un temps limité à l'opérateur pour mettre son veto sur l'exécution automatique
7	Le système exécute automatiquement, et avertit nécessairement l'opérateur
8	Le système n'informe l'opérateur que si demandé
9	Le système n'informe l'opérateur que s'il le décide
10	Le système décide et agit totalement seul, ignorant l'opérateur

TABLE 2.1: Niveaux d'automatisation de Sheridan [Sheridan et Verplank, 1978, Parasuraman, 2000]

D'autres classifications reposent sur le même principe, notamment [Dorais *et al.*, 1999], pour lesquels un niveau d'autonomie est caractérisé par la complexité des commandes traitées, [Goodrich *et al.*, 2001] pour qui un niveau représente la capacité d'un robot à fonctionner un certain temps indépendamment de l'opérateur ou encore [Bradshaw *et al.*, 2003], qui considèrent que les rôles des agents varient en fonction des actions qui leur sont obligatoires, permises, possibles, et de l'initiative qu'ils ont pour entreprendre ces actions.

Les principales limitations de ces approches sont le caractère figé du rôle des agents pour chaque niveau ainsi que le nombre restreint de niveaux qui impose un cadre rigide. En effet, l'utilisation de niveaux prédéfinis et en nombre limité ne peut rendre compte de la variété des situations rencontrées par un robot au cours de sa mission. De plus, le basculement d'un niveau à un autre de manière dynamique, en mission, est encore une interrogation quant aux conséquences sur l'opérateur en terme de performances, charge cognitive, et conscience de situation [Villaren *et al.*, 2010]. Enfin, il est difficile de trouver un juste milieu entre des définitions universelles et abstraites de niveaux, et des définitions très concrètes mais spécifiques à un système particulier dans le cadre d'une mission donnée.

L'autonomie peut également être considérée au niveau de la *tâche* (ou de l'activité) [Sellner *et al.*, 2006, Finzi et Orlandini, 2005], un agent étant autonome pour réaliser une tâche si cette tâche lui est affectée. L'autonomie considérée au niveau de chaque tâche est susceptible d'apporter la solution la plus adaptée à chaque situation de la mission, par contraste avec l'utilisation de niveaux d'autonomie rigides. En revanche, elle nécessite de pouvoir distinguer, pour chaque tâche, qui de l'agent artificiel ou humain est le plus à même de l'effectuer avec les meilleures performances. En ce sens, [Sellner *et al.*, 2006] proposent deux modes de partage des tâches, le SISA (System-Initiative Sliding Autonomy), dans lequel seul le système d'agents décide de faire appel à l'opérateur si besoin, et le MISA (Mixed-Initiative Sliding Autonomy), où l'opérateur peut également décider d'intervenir à tout moment. Le partage des tâches est réalisé sur la base suivante : toute tâche peut être assignée à l'homme ou à la machine, suivant leurs capacités. Les auteurs dotent les robots de modèles empiriques de performance du système ainsi que de l'opérateur, afin qu'ils déterminent quelle entité sera la plus à même de résoudre un problème donné. Ces modèles sont réalisés en particulier à partir de statistiques sur la réussite ou non de ces mêmes tâches par le passé, suivant l'entité considérée. Les limitations que nous voyons à cette approche résident dans le fait que la conscience de situation de l'opérateur n'est pas prise en compte, et s'il doit intervenir rapidement il risque d'en être incapable car ignorant de ce que réalise l'agent artificiel. De plus, l'attribution des tâches suivant des statistiques (comparaison du nombre de succès de l'opérateur et du robot sur une tâche donnée) ne tient pas compte du contexte de réalisation : en effet, accomplir une même tâche dans des conditions différentes ne présente pas forcément les mêmes difficultés. De plus, ce mode d'attribution statistique ne s'applique pas à des missions en environnement critique, où l'erreur n'est pas tolérée : la tâche doit être réalisée par celui qui la réalisera de manière sûre.

1.3.2 Le mode d'initiative

Le *mode d'initiative* concerne la dynamique de l'autonomie : qui de l'agent artificiel ou humain a le pouvoir de modifier le niveau d'autonomie de l'agent artificiel ? Dans la littérature, on distingue essentiellement l'*autonomie adaptative*, l'*autonomie ajustable* et l'*initiative mixte* [Hardin et Goodrich, 2009].

L'*autonomie adaptative* confère au seul robot le pouvoir de modifier son autonomie ; par exemple [Scerri *et al.*, 2003] utilisent des techniques d'optimisation pour que l'agent artificiel gère lui-même son autonomie en conservant ou transférant à un autre agent la prise de décision. Afin d'éviter les comportements dangereux, [Chopinaud *et al.*, 2006] génèrent des agents capables de se surveiller eux-mêmes et de s'autoréguler, en adoptant dynamiquement des comportements de régulation prédéfinis. Le contrôle colloboratif est une approche cherchant à créer le dialogue entre l'agent artificiel et l'opérateur, de manière à compléter la boucle décisionnelle [Fong *et al.*, 2002] : l'agent artificiel contacte l'opérateur lorsqu'il rencontre une difficulté, afin qu'il lui apporte de l'information supplémentaire. Cependant, les interactions se font *a priori* sur demande de l'agent artificiel, lorsque celui-ci le juge nécessaire, ce qui ne reflète pas nécessairement les exigences de l'opérateur qui pourrait souhaiter parfois avoir un meilleur contrôle.

Ainsi, l'autonomie adaptative, en raison de sa gestion par le robot, permet de pouvoir systématiser et fiabiliser certains comportements, notamment avec des tâches précises et répétitives. De plus, elle permet d'obtenir des réactions plus rapides que celles que l'on pourrait attendre d'un humain. En revanche, le système est privé des capacités d'analyse de l'opérateur, en particulier il ne peut intervenir si le système n'agit pas correctement : son interaction avec le robot est restreinte à ce qui est attendu de lui [Hardin et Goodrich, 2009].

À l'inverse, l'*autonomie ajustable* confère au seul opérateur l'initiative du changement d'autonomie : l'opérateur choisit son niveau d'interaction [Dorais *et al.*, 1999], ou « conseille » le robot en mission [Myers et Morley, 2001]. Pour [Myers et Morley, 2001], le contrôle de l'opérateur sur le robot est établi par l'utilisation d'une norme décrivant les permissions et comportements que l'opérateur humain veut faire respecter par l'agent. En pratique, la norme se présente sous la forme d'un langage que l'opérateur utilise pour créer des règles. Cependant, une limite de cette approche consiste en la création par l'opérateur de toutes ces règles au cas par cas, avec le risque de faire apparaître des conflits entre règles. De plus, cette méthode implique une forme d'« autonomie supervisée » de l'agent, ce qui interdit à celui-ci la possibilité de pouvoir faire varier son autonomie de lui-même.

L'autonomie ajustable permet ainsi à l'opérateur de faire usage de ses capacités d'analyse et de compréhension globale de la mission et d'anticiper les aléas. L'inconvénient est que si l'opérateur commet une erreur, s'il réagit trop lentement, ou si son expertise est insuffisante, les performances du système global s'en ressentent ; les interactions de l'opérateur sur le système ne sont d'ailleurs pas toujours bénéfiques [Schurr *et al.*, 2006].

Enfin, l'*initiative mixte* donne aussi bien au robot qu'à l'opérateur humain le pouvoir de décider de l'autonomie du robot [Hardin et Goodrich, 2009, Sellner *et al.*, 2006]. Le principe sous-jacent est de combiner les avantages des deux approches précédentes, tout en en réduisant les inconvénients. Bien qu'idéale sur le papier, cette approche reste à parfaire pour montrer ses bénéfices dans la pratique [Hardin et Goodrich, 2009].

1.4 Quelle perception de l'opérateur humain le robot a-t-il ?

L'opérateur humain a en général une connaissance des capacités de l'agent artificiel qu'il utilise, ainsi qu'une connaissance de son état courant, de ses possibles états futurs (conscience de situation [Endsley, 1995]). Dans le cadre du contrôle de l'autonomie de l'agent artificiel à l'initiative de l'un ou de l'autre agent, l'agent artificiel doit lui aussi avoir un modèle des capacités de l'agent humain, ainsi que de son « état ».

En ce qui concerne les capacités, l'opérateur humain peut intervenir à différents niveaux d'ordre, depuis des ordres « bas niveau » (commande directe des actuateurs) jusqu'à des ordres « haut niveau » (buts) [Kortenkamp *et al.*, 1997]. Bien qu'une approche fréquemment utilisée consiste à automatiser les tâches dites de « bas niveau » et à confier à l'opérateur les tâches de « haut niveau » [Hexmoor *et al.*, 2009, Goodrich *et al.*, 2007], il peut être important pour l'opérateur de pouvoir intervenir à tout niveau, du but jusqu'à la commande : par exemple, en cas de défaillance dans le système, l'opérateur a une meilleure conscience de situation s'il intervient à un niveau « bas » [Kaber *et al.*, 2000] : le robot doit pouvoir intégrer ces différents niveaux de commande.

Dans le cadre du contrôle collaboratif de [Fong *et al.*, 2002], le robot contacte l'opérateur à chaque fois qu'il rencontre une difficulté : le robot sait ce qu'il peut attendre de l'opérateur ; cependant, les interactions spécifiées ne se font *a priori* que sur demande de l'agent robotique. Dans les travaux de [Kortenkamp *et al.*, 1997], le robot dispose de modèles de tâches réalisables par l'opérateur, ce qui lui permet de planifier de manière globale (c'est-à-dire pour lui et pour l'opérateur, en requérant *explicitement* l'opérateur pour réaliser certaines tâches), et de suivre l'exécution des tâches réalisées par l'opérateur. Dans les travaux de [Finzi et Orlandini, 2005], le modèle de tâches (incluant celles de l'opérateur) qu'a le robot lui permet de diagnostiquer un problème à l'exécution et de lancer une procédure de réparation : en ce sens, l'opérateur agit sous vérification du robot.

Enfin, l'intégration de données relatives à l'« état » de l'opérateur (physiologie, expertise, charge de travail) permet à l'agent artificiel d'adapter son autonomie s'il estime que cet état est dégradé [Barber *et al.*, 2008, Coppin *et al.*, 2009]. Des études concernant la construction de modèles liés aux capacités de l'opérateur, comme par exemple pour la détection de la surcharge de travail en fonction de la tâche réalisée, ont été effectuées [Donath *et al.*, 2010].

2 Les critères pour le changement d'autonomie

2.1 Différentes métriques

Dans le cas où le choix du basculement d'un niveau d'autonomie à un autre est laissé à l'agent robotique (autonomie adaptative ou initiative mixte), plusieurs techniques permettent de définir les conditions du basculement et de rendre *opérationnelle* la variation de l'autonomie : [Scerri *et al.*, 2003] dotent les agents robotiques de capacités d'apprentissage fondées sur des processus décisionnels de Markov (MDP), qui leur permettent de transférer la prise de décision si nécessaire à l'agent humain. [Schurr *et al.*, 2009] utilisent une approche similaire pour transférer la prise de décision, mais en tenant compte des possibilités d'incohérences entre agents ; cependant, ces approches nécessitent de pouvoir estimer l'utilité de chaque stratégie possible, ce qui est de fait une approche subjective. Les critères de [Hardin et Goodrich, 2009] sont explicites et constitués par des événements présélectionnés (certaines actions de l'opérateur humain, certains événements de mission) ; cependant, ces critères, bien qu'objectifs, sont spécifiques d'une mission donnée et d'un ensemble restreint de tâches. Dans les travaux de [Myers et Morley, 2001], le comportement de l'agent robotique change en fonction de la vérification ou non d'un ensemble de règles définies par l'opérateur, qui constituent les permissions et comportements que l'opérateur humain veut faire respecter par le robot. Ce système pose toutefois le problème des conflits potentiels entre règles. Citons également les travaux de [Sellner *et al.*, 2006], où la décision de l'affectation à l'agent robotique ou à l'agent humain d'une nouvelle tâche à exécuter dépend de statistiques préétablies quant au succès de la tâche. Enfin, [Zieba *et al.*, 2009] proposent d'effectuer les changements de mode d'autonomie d'un robot suivant la métrique du taux de récupération d'erreurs (ratio des erreurs évitées sur le nombre total d'erreurs réalisables), dans le but de réaliser un système homme-robot le plus résilient possible, c'est-à-dire le moins susceptible aux perturbations éventuelles. Cependant, cette approche est pour l'instant limitée à des systèmes fondés sur le concept d'autonomie ajustable, dans laquelle seul l'opérateur a l'initiative du changement de mode d'opération du robot.

2.2 Le conflit : une alternative à l'étude de l'erreur humaine

Classiquement, la mesure et la prédiction de la performance d'un système homme-machine repose sur la détection d'écarts entre l'activité réelle des opérateurs et la tâche attendue [Leplat, 1985]. Ainsi de nombreuses approches formelles sont proposées [Heymann et Degani, 2001, Callantine, 2002] : l'activité réelle est reconstituée dynamiquement à partir des données comportementales des opérateurs (par exemple les actions sur les IHM) puis est comparée à une base de données qui représente les procédures de référence. Or ces méthodes montrent rapidement leurs limites. Cela tient principalement à deux problèmes épistémologiques auxquels se heurte l'étude de l'erreur : celui de son existence, et celui de son statut. En effet :

- la formalisation de l'erreur est hasardeuse puisque la notion de norme à laquelle elle se rattache n'est pas universelle. De plus, il est reconnu que les opérateurs redéfinissent les normes pour trouver de nouvelles procédures, plus efficaces et plus sûres [Leplat, 1997, Leplat et Hoc, 1983, Chaudron *et al.*, 1999] ;
- l'apparition de l'erreur n'est pas nécessairement synonyme de dégradation de l'activité des opérateurs. Son existence se révèle souvent *a posteriori*, une fois qu'un incident ou un accident a eu lieu ;
- les opérateurs (pilotes, conducteurs de trains), commettent immanquablement des erreurs mais en récupèrent la plupart [Rizzo *et al.*, 1987]. Bien au contraire, cette production-détection-récupération de l'erreur constitue un véritable trait d'expertise [Alwood, 1984] ;
- l'apprivoisement de l'erreur est un processus intrinsèque dans l'apprentissage de connaissances [Giroud-Fliegner, 2001] ;
- l'erreur est un moyen important dans l'auto-évaluation de sa performance : c'est au regard de ses erreurs que l'on peut estimer si une connaissance est apprise, comprise. De plus dans le domaine de la conduite (conduite automobile, pilotage d'avion...), le nombre d'erreurs commises joue un rôle de *feedback* et permet d'estimer sa performance [Wioland, 1997].

De plus, on remarque que les systèmes autonome ont fait évoluer le rôle des opérateurs vers une activité de supervision et une surveillance passive. Ils ont donc logiquement déplacé la source des erreurs et s'ils permettent d'éviter les erreurs bénignes (type erreur de routine [Reason, 1990]), ils créent en revanche des conflits avec les opérateurs dont les conséquences sont beaucoup plus graves [Sweet, 1995]. Par ailleurs, les conflits de représentation sont le résultat d'une méconnaissance des opérateurs des calculateurs embarqués [Sarter et Woods, 1994, Sarter *et al.*, 2003] : ceux-ci ont des difficultés à diagnostiquer les pannes de ces systèmes et des procédures à utiliser ou de l'attitude à adopter (débrayer ou non l'automatisme) pour recouvrer un état normal, ensuite les changements inopinés de mode de pilotage automatique ne sont pas toujours détectés ou compris par les opérateurs et peuvent avoir des conséquences catastrophiques [Mumaw *et al.*, 2001].

Une approche alternative est de considérer l'étude des conflits pour évaluer la performance des interactions opérateurs-systèmes. En effet, en psychologie sociale [Le Marc, 1999, Castelfranchi, 2000, Simmel, 2003] montrent que sa présence est révélatrice d'une dynamique de tension et d'opposition au sein d'un groupe d'individus ; en intelligence artificielle distribuée, son apparition permet de diagnostiquer un état de dysfonctionnement dans les interactions entre les agents artificiels [Dehais et Pasquier, 2000].

Par ailleurs, des travaux menés en simulateur de vol révèlent que l'apparition de conflits dans la gestion du vol [Dehais, 2004, Dehais *et al.*, 2005] est un précurseur remarquable de la dégradation des interactions équipages-systèmes. En particulier, ces expérimentations montrent que les pilotes, lorsqu'ils sont confrontés à un conflit, ont tendance à persévérer et

à s'enfermer dans la résolution du problème au détriment de la surveillance des paramètres vitaux.

Ensuite, d'un point de vue formel, ce concept est plus intéressant que l'erreur : il ne se rapporte pas systématiquement à un référentiel. Le conflit entre agents existe dans l'absolu en dehors de toute norme ou de toute vérité : son essence est la contradiction [Castelfranchi, 2000], la différence entre deux points de vue [Chaudron *et al.*, 2000]. Ainsi, il apparaît possible de le détecter en se concentrant sur l'analyse de phénomènes d'opposition, d'interférence ou d'incohérence comportementale entre plusieurs agents.

Enfin, la problématique du conflit apparaît rapidement dans les systèmes de grande échelle ou de grande complexité, en particulier lorsque plusieurs automatismes et plusieurs opérateurs humains interviennent simultanément, donnant lieu au typique « problème de l'initiative mixte » : plusieurs boucles de commande peuvent en effet coexister, ne se voyant pas l'une et l'autre, pouvant agir sur les mêmes ressources et donnant lieu à des incohérences [Sheridan et Parasuraman, 2006].

3 De l'autonomie à l'autorité

La revue de la littérature relative au contrôle de l'autonomie nous montre plusieurs façons d'envisager ce contrôle, selon la granularité, le mode d'initiative, les critères d'évolution, la perception qu'a l'agent artificiel de l'agent humain. Cependant, à notre connaissance, les changements de contrôle ne sont pas étudiés sous l'angle des conflits qu'ils peuvent provoquer entre les agents. Comme le souligne l'expérience décrite en introduction ainsi que l'analyse de rapports d'accidents aéronautiques [Dehais *et al.*, 2005], les changements d'autorité imprévus ou incompris peuvent en effet mener à des situations inefficaces, dangereuses, voire catastrophiques. De façon à pouvoir considérer de la même manière l'agent humain et l'agent artificiel, nous préférons manipuler les notions d'autorité et de contrôle d'autorité plutôt que d'autonomie, qui concerne exclusivement l'agent artificiel.

Le contrôleur de la dynamique de l'autorité que nous proposons de réaliser a donc pour objectif la détection et la résolution de conflits d'autorité, en prenant en compte les différentes facettes du contrôle :

- une mise en œuvre des différents modes d'initative (initiative donnée à l'agent artificiel, à l'agent humain ou aux deux, en fonction du contexte) ;
- des objets de l'autorité de granularité pertinente pour la détection et la résolution de conflits (objets sur lesquels les agents peuvent effectivement agir) ;
- un critère d'évolution de l'autorité objectif et indépendant de l'application ;
- l'intégration de modèles de tâches et de l'« état » de l'opérateur dans les connaissances de l'agent artificiel et la prise en compte de l'intervention de l'opérateur aux niveaux d'ordres pertinents, qu'ils soient de « haut » ou de « bas » niveau.

Chapitre 3

Modèle de l'autorité

À l'issue de l'étude bibliographique réalisée au chapitre 2, le conflit nous semble être un indicateur pertinent pour déclencher la révision du plan des agents en cours de mission. Le but de ce chapitre est de formaliser les concepts nécessaires à une définition formelle du conflit : à partir du plan d'exécution à réaliser par les agents, nous proposons d'identifier les *ressources* indispensables à la réalisation de ce plan, sous la forme d'un graphe de dépendances entre ces *ressources*. Le conflit survient lorsqu'une ressource vient à disparaître de manière non anticipée, en raison d'une défaillance ou d'une utilisation non coordonnée par les agents du système (opérateur et robot).
Nous proposons également une définition de l'autorité sur une ressource, sous la forme d'une relation pour un couple d'agents : l'autorité permet d'établir les droits de chaque agent sur une ressource, en terme d'accès et d'utilisation permise. L'autorité est un concept essentiel pour permettre la résolution de conflit, lorsque celui-ci survient.
Le modèle de l'autorité présenté dans ce chapitre comporte deux niveaux distincts :
- le macro-modèle, définissant les concepts de ressource, interface entre ressources et relation d'autorité, les liens entre ces concepts et leur usage ;
- le micro-modèle, consistant en l'implémentation des composants du macro-modèle en réseaux de Petri, qui est la couche permettant l'exécution du modèle de l'autorité.

1 Principe de l'approche

1.1 Contexte

Traditionnellement, un système mettant en œuvre un véhicule dit « autonome » ainsi qu'un opérateur humain fonctionne de la manière suivante : l'agent robotique exécute son plan généré à partir d'un planificateur, et l'opérateur humain supervise l'exécution du plan de l'agent robotique, y apportant parfois des modifications ou y ajoutant ses propres tâches. Lorsqu'une incohérence apparaît dans la suite d'actions que réalise un des agents, que cela soit dû à un aléa (perte de capteurs, échec d'une tâche), à une erreur

(violation d'une règle de sécurité, actions contradictoires) ou encore à l'action de l'autre agent (interférences), le but que poursuivait cet agent ne pourra pas être atteint s'il continue ses actions prévues. Nous nommons cette incohérence un *conflit*. Pour permettre à l'agent de revoir ses actions et prendre les décisions nécessaires, le conflit doit être évalué : quelles sont les contraintes violées, quelles actions en cours et à venir sont remises en cause, quels buts sont inatteignables. Si une reconfiguration du système est nécessaire (moyens mis en œuvre, tâches à réaliser, nouveaux buts), il est primordial de coordonner les agents afin de ne pas retourner immédiatement dans un nouvel état de conflit.

1.1.1 Modèle du plan d'exécution

Il y a ainsi nécessité d'identifier tous les éléments pouvant intervenir dans la survenue d'un conflit ainsi que dans la reconfiguration du système. Ces éléments sont de natures variées et hétérogènes : tâches des agents, décisions des agents, buts poursuivis, contraintes (physiques, de sécurité, protocoles), éléments matériels (capteurs, énergie) ou informationnels (présence de l'information de position, confiance en cette information). De plus, le rôle de ces éléments l'un par rapport à l'autre est variable : une tâche, pour être exécutée, requiert la présence d'éléments matériels et informationnels pour en produire d'autres qui deviennent alors ses buts ; elle peut nécessiter la réalisation d'autres tâches en parallèle, et elle est soumise à des contraintes à vérifier avant ou tout au long de son exécution.

Afin de pouvoir définir formellement le concept de conflit, le premier point de notre approche consiste à créer un modèle du plan exécuté par les agents, soit :

- les buts poursuivis ;
- les tâches à exécuter par les agents ;
- les moyens mis en œuvre, au sens large (objets matériels, conditions logiques devant être vérifiées), nécessaires au sein du système à la bonne exécution des tâches ;
- les relations de dépendances entre moyens et tâches, entre les tâches, et entre tâches et buts.

Nous supposons que ce modèle, qui représente le plan d'exécution, ne contient pas de branchements conditionnels.

Nous désignerons par le terme générique de *ressource* un but, une tâche, un moyen, appartenant au modèle de plan exécuté. Les ressources sont reliées entre elles *via* des relations de dépendance. Afin de représenter différents types de dépendance entre ressources, nous introduisons le concept d'*interface* entre ressources. Nous nommons *graphe de ressources* la combinaison des ressources et des liens entre elles *via* les interfaces. Le modèle du plan d'exécution sera donc un graphe de ressources.

1.1.2 Agents

Les agents sont les entités qui contrôlent les ressources, les organisent les unes par rapport aux autres : c'est par exemple l'opérateur humain qui décide de réaliser une tâche de navigation en mettant en œuvre une chaîne matérielle et fonctionnelle du système afin de « produire » l'arrivée du véhicule sur une zone d'exploration, condition nécessaire pour passer à la tâche suivante consistant en l'exploration de la zone elle-même. Dans ce travail, nous ne nous intéressons qu'à deux agents, l'opérateur humain désigné sous l'identifiant *operator* et l'agent robotique désigné sous l'identifiant *robot*.

Le modèle du plan d'exécution, pour rendre possible son traitement automatique et permettre au robot de se positionner vis-à-vis de l'opérateur humain, est embarqué sur le robot. La mise à jour des informations et l'inclusion des actions de l'opérateur dans le modèle du plan d'exécution sont réalisées depuis la perspective du robot, avec lequel l'opérateur communique et interagit *via* son interface utilisateur. Ceci permet également au robot de conserver ces informations dans le cas d'une rupture de communication avec l'opérateur. Cependant, ce n'est pas ce modèle qui prendra les décisions du robot, mais bien sa fonction de planification. Ainsi, le modèle, même s'il est embarqué sur le robot, incorpore indifféremment les actions qui lui sont communiquées par la planification du robot ou l'interface de l'opérateur humain, qui sont considérés de la même manière : le modèle est « neutre » dans sa considération des différents agents du système.

1.1.3 Relation d'autorité

Si un conflit survient entre deux agents pour le contrôle d'une ressource particulière (par exemple, la commande de cap du véhicule requis par le robot et par l'opérateur humain en même temps), il se pose le problème de déterminer quel agent devrait avoir le contrôle de la ressource au détriment de l'autre agent afin de résoudre ce conflit. Cependant, même dans le cadre nominal, par conception, on peut choisir de laisser la priorité à l'opérateur sur le robot concernant le contrôle de certaines ressources, et inversement pour d'autres. Par extension, on peut imaginer qu'il soit temporairement interdit à un agent d'utiliser une ressource, si la situation l'exige (par exemple, pour le suivi d'une trajectoire très précise, le robot peut interdire l'accès à la commande de cap à l'opérateur ; inversement, l'opérateur peut interdire au robot d'utiliser un algorithme de traitement d'images qu'il juge inefficace sur la situation rencontrée). Le contrôle d'une ressource se présente alors comme un *droit* pouvant évoluer en cours de mission, mais également comme une relation entre agents afin de réguler leur accès commun à une même ressource.

Afin de représenter et pouvoir réguler ce droit de contrôle sur une ressource par les agents, nous introduisons le concept de *relation d'autorité*. Cette relation est posée sur chaque ressource et pour chaque couple d'agents susceptibles de contrôler la ressource.

Ainsi, pour toute ressource du modèle, il y a autant de relations d'autorité que de couples d'agents susceptibles de la contrôler.

Le contrôleur de la dynamique de l'autorité sera donc fondé sur les ressources, les interfaces entre ces ressources, et les relations d'autorité exercées par les agents sur les ressources.

1.2 Du plan au graphe de ressources

Ce paragraphe donne l'idée de l'extraction d'un graphe de ressources à partir d'un plan. Les détails formels et algorithmiques de l'extraction n'ont pas été traités dans le cadre de ce travail.

1.2.1 Préconditions, tâches et effets

Nous supposons que le robot, dont nous réalisons le contrôleur de la dynamique de l'autorité, dispose de capacités de planification. Un premier graphe de dépendances s'obtient à partir d'un plan calculé par le planificateur. Par exemple, considérons un planificateur hiérarchique de type HTN - Hierarchical Task Network - tel que JSHOP2 [Nau *et al.*, 2001]. L'opérateur de planification *NavigationRobot*, qui décrit la tâche de navigation automatique pour atteindre un point d'arrivée donné, est le suivant :

```
1/      ( :operator (!NavigationRobot ?z1, ?z2, ?a)
2/          ((is_zone ?z1)
(is_zone ?z2)(is_robot ?a)(position ?a ?z1)(enough_energy ?z2 ?a)
/                  (SteeringWheel_available ?a)(SafetyDistance_respected ?a) ))
3/          ((forall ( ?zone) ((is_zone ?zone)) ((position ?a ?zone))))
4/          ((position ?a ?z2))
5/      )
```

L'opérateur *NavigationRobot* prend en paramètre la zone de départ $z1$, la zone d'arrivée $z2$ et l'agent a devant effectuer la tâche (ligne 1). En ligne 2 sont donnés les prédicats/préconditions devant être validés avant l'exécution de la tâche, à vérifier en fonction des paramètres fournis ($z1$, $z2$ et a). Ensuite viennent les effets de l'opérateur : les effets négatifs en ligne 3, c'est-à-dire ce qui est « retiré » de l'état du système et les effets positifs en ligne 4 (ce qui est « ajouté »).

Un planificateur cherche un chemin pour passer d'un état de départ, décrit par un ensemble de valeurs sur les variables décrivant le système, à un état d'arrivée-but, dans lequel les variables décrivant le système ont pris d'autres valeurs. Un chemin

consiste en l'application des opérateurs de planification en respectant la vérification des préconditions d'un opérateur pour pouvoir l'appliquer, de manière à ce que ses effets permettent l'application des opérateurs suivants.

La figure 3.1 illustre le principe utilisé par un planificateur classique pour réaliser un plan, ici une suite de deux tâches.

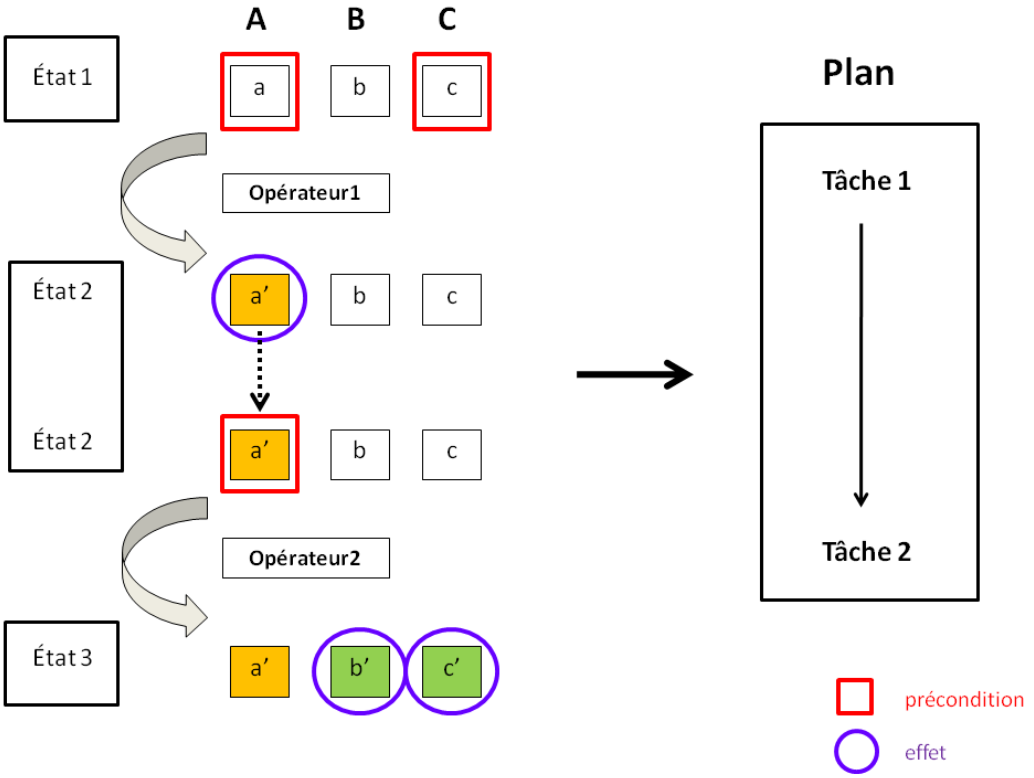

FIGURE 3.1: Opérations du planificateur pour créer un plan

L'ensemble du « monde » considéré est décrit par trois variables A, B, C qui peuvent chacune prendre différentes valeurs. L'état initial est décrit par le triplet des valeurs de départ sur ces trois variables $Etat1 = \{a, b, c\}$. L'état final recherché est l'état $\{a', b', c'\}$. Partant de l'état initial, le planificateur applique l'opérateur 1, qui nécessite les préconditions a et c pour pouvoir être appliqué ; ces préconditions sont bien vérifiées par l'état initial. L'effet de l'opérateur 1 est de faire passer la variable A de la valeur a à la valeur a', ce qui fait qu'après application de l'opérateur 1, le monde est décrit par le triplet $Etat2 = \{a', b, c\}$. a' est justement la précondition nécessaire pour pouvoir appliquer l'opérateur 2, qui fait alors passer les variables b à b' et c à c'. L'état final recherché est atteint : $Etat3 = \{a', b', c'\}$. Le planificateur renvoie ainsi le plan final

{tâche1, tâche2}, qui consiste à d'abord réaliser la tâche 1 (instanciation de l'opérateur 1 sur les préconditions a et b), puis ensuite la tâche 2 (instanciation de l'opérateur 2 sur la précondition a').

Ainsi, lorsque le planificateur a calculé un plan, celui-ci se présente sous la forme d'une liste ordonnée de tâches à effectuer du type $\{tâche_1, tâche_2, ..., tâche_n\}$, qui correspond à la liste des opérateurs instanciés à appliquer, dans l'ordre. Dans cette liste, les liens entre tâches n'apparaissent pas : ce qui fait qu'un opérateur a été appliqué avant un autre a disparu. Pour obtenir un graphe de dépendances à partir du plan, il faut récupérer cette information, ou plutôt la collecter directement *lors* du processus de planification : si une précondition p d'une tâche $tâche_j$ correspond à l'un des effets d'une tâche précédente $tâche_i$, alors $tâche_j$ est dépendante de $tâche_i$ *via* p. Cette relation construit un premier graphe de dépendances ; toutefois, cela suppose la réalisation d'algorithmes de planification adaptés à cette transformation.

La figure 3.2 représente le graphe de ressources tel que l'on souhaite l'obtenir. Dans ce graphe ont été conservés tous (et uniquement) les éléments justifiant l'ordre des opérateurs.

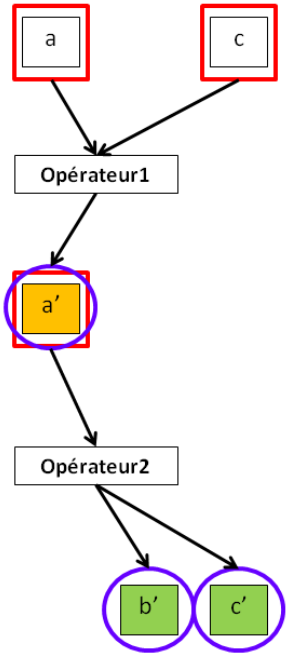

FIGURE 3.2: Graphe de ressources obtenu à partir du plan

Cependant, l'extraction du graphe de ressources à partir de la planification classique est restreinte par l'usage même des concepts de préconditions, opérateurs et effets : en effet, les préconditions ne sont vérifiées que de manière discrète, juste *avant* l'application de l'opérateur. Ainsi, la relation de dépendance entre une précondition et l'opérateur de planification est une *dépendance en initialisation*. À l'inverse, l'effet produit par l'opérateur de planification n'apparaît qu'*après* son application : la dépendance entre l'effet et l'opérateur est une *dépendance en terminaison*. Ainsi, dans ce type de formalisme, la durée de l'application de l'opérateur de planification importe peu, il n'est attendu de lui que la production des effets désirés.

Il est pourtant légitime de s'intéresser aux dépendances s'appliquant *pendant* l'application d'un opérateur de planification : nous nommons ces dépendances des *contraintes*.

1.2.2 Contraintes

La relation de dépendance entre une précondition et l'opérateur de planification qui la nécessite pour pouvoir être appliqué est fugitive : peu importe ensuite qu'elle soit détruite ou modifiée, seule importe sa vérification à l'initialisation de l'opérateur.

Par opposition, une contrainte, au sens où nous la considérons, s'inscrit dans une durée : il s'agit d'une condition logique qui doit rester vraie pendant un intervalle de temps, pour permettre par exemple la réalisation d'une tâche : présence permanente d'énergie en quantité suffisante, exécution d'une autre tâche en parallèle, en support. Si la contrainte n'est plus vérifiée pendant l'exécution de la tâche, alors la tâche en sera affectée : la relation de dépendance est elle-même inscrite dans la durée. C'est pourquoi nous associons à la notion de contrainte la relation de dépendance en *exécution* (voir paragraphe 2.2.3.b).

1.3 Exemple

Afin de produire le but « Atteindre point d'arrivée WP-Goal », l'opérateur de planification *Navigation Robot* est instancié avec comme paramètres la destination *WP-Goal* et comme agent exécutant le robot. On obtient ainsi le graphe de ressources présenté figure 3.3, avec *WP-Goal* comme effet (et sur ce graphe, but), la tâche instanciée *Navigation Robot*, la précondition *Start Position*, les préconditions mais également contraintes *Energy* et *Steering Wheel* (ces ressources doivent rester présentes tout au long de l'exécution de la tâche), ainsi que la contrainte de sécurité *Safety Distance*.

Les ressources $\{StartPosition, Energy, SteeringWheel, SafetyDistance, NavigationRobot, WP{-}Goal\}$ sont reliées entre elles *via* des interfaces représentant les relations de dépendance. Le rôle associé à chaque ressource (précondition, tâche, contrainte, etc.), induit par les relations de dépendance, est inscrit à côté de chaque ressource. Enfin, à chaque ressource est associée une relation d'autorité pour le couple (robot, opérateur humain) qui définit les modalités de contrôle sur la ressource considérée.

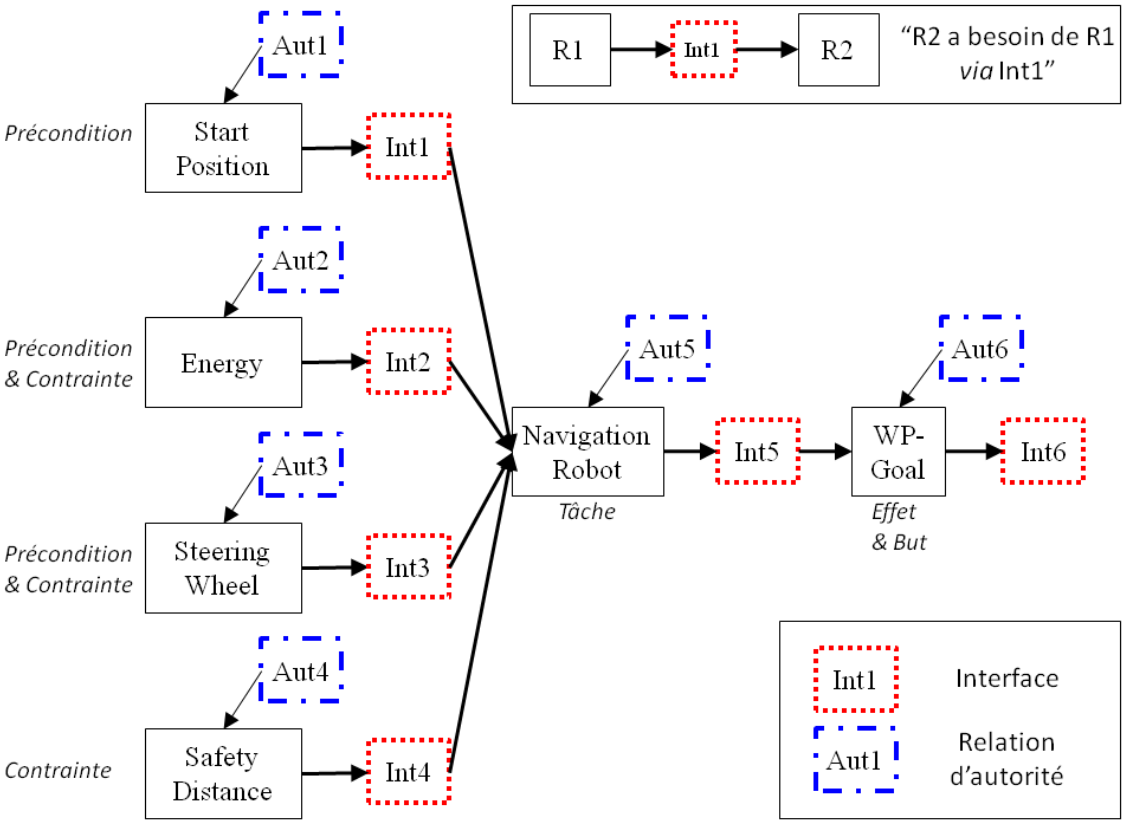

FIGURE 3.3: Graphe de ressources pour le plan de but *WP-Goal*

On remarque que si l'une des ressources du graphe disparaît en cours d'exécution (la ressource subit un aléa ou est détruite par l'action d'un agent), les ressources qui en dépendent vont elles-mêmes disparaître.

2 Macro-modèle d'autorité

2.1 Macro-modèle

Nous considérons l'ensemble des ressources et arcs composant un graphe de ressources, ainsi que les notions sur l'autorité mentionnées ci-dessus. Nous nous plaçons ici d'un point de vue global, à savoir comment s'articulent les relations entre ces composants, ce qui nous conduit à définir un macro-modèle. Dans la section suivante, le micro-modèle viendra détailler chacun des composants utilisés par le macro-modèle, avec comme objectif la présentation de leur constitution interne.

Le macro-modèle d'autorité que nous proposons a pour objectif de définir formellement les liens entre ressources dans le graphe de ressources, tout en introduisant la notion de relations d'autorité entre agents. Comme l'approche proposée est une représentation de type graphe, nous introduisons ici plusieurs notations nécessaires pour la suite.

Definition 1 (Macro-modèle d'autorité)
On appelle macro-modèle d'autorité le quadruplet $\mathcal{V} = \{G, \mathcal{C}, \phi, a\}$, avec :
- $G = \{\mathcal{R}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$ le graphe de ressources du modèle ;
- $\mathcal{R} = \{r_i\}$ l'ensemble des ressources ;
- $\mathcal{I} = \{i_j\}$ l'ensemble des interfaces ;
- $\mathcal{C} = \{c_l\}$ l'ensemble des agents ;
- $\phi : \mathcal{I} \to \mathcal{C}$ la fonction d'assignation ;
- a la fonction d'autorité. □

2.2 Graphe de ressources

2.2.1 Définition

Definition 2 (Graphe de ressources)

Le graphe de ressources $G = \{\mathcal{R}, \mathcal{I}, \mathcal{F}, \mathcal{B}\}$ du macro-modèle d'autorité est un graphe biparti sans circuit sur $\mathcal{R}$ l'ensemble des ressources et $\mathcal{I}$ l'ensemble des interfaces :
- $\mathcal{F}$ est la fonction d'incidence avant $\mathcal{F} : \mathcal{R} \times \mathcal{I} \to \{0, 1\}$, et représente les arcs orientés des ressources vers les interfaces ;
- $\mathcal{B}$ est la fonction d'incidence arrière $\mathcal{B} : \mathcal{I} \times \mathcal{R} \to \{0, 1\}$ et représente les arcs orientés des interfaces vers les ressources. □

Les arcs parallèles ne sont ainsi pas autorisés ; de plus, il ne peut y avoir deux interfaces (créée par le même agent) dans $\mathcal{I}$ reliant les mêmes ressources dans la même direction (voir figure 3.4).

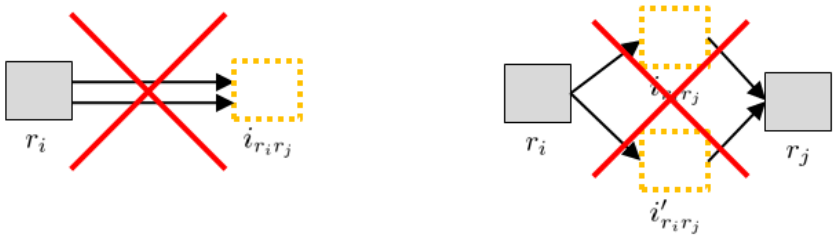

FIGURE 3.4: Configurations interdites dans le graphe de ressources

2.2.2 Ressource

2.2.2.a Définition

Definition 3 (Ressource)
Une ressource est un nœud du graphe de ressources. Une ressource représente un élément du plan d'exécution (objet physique, information, permission, tâche, contrainte, but) dont on peut vérifier :

- la présence ou la satisfaction (ressource produite) ;
- l'absence ou la violation (ressource détruite) ;
- la disponibilité ou l'indisponibilité. □

$\mathcal{R}$ est l'ensemble des ressources présentes dans le macro-modèle de l'autorité.

En fonction de l'élément représenté, une ressource peut avoir les propriétés suivantes (voir également la figure 3.5) :

- une ressource r peut être *partageable* : plusieurs ressources peuvent dépendre simultanément de r ;
- une ressource peut être *non-partageable* : une seule ressource peut dépendre de r à un instant donné ;
- une ressource peut être *préemptable* : bien qu'une ressource r' dépende actuellement de la ressource r, ce lien peut être mis en concurrence avec une autre ressource r'' qui va forcer l'activation de la relation de dépendance à son profit. Une ressource ne peut être préemptable que si elle est non-partageable ;
- une ressource peut être *non-préemptable* : une fois que la relation de dépendance entre la ressource r et une ressource r' est établie, elle est exclusive et ne peut être remise en cause par une autre ressource r''.

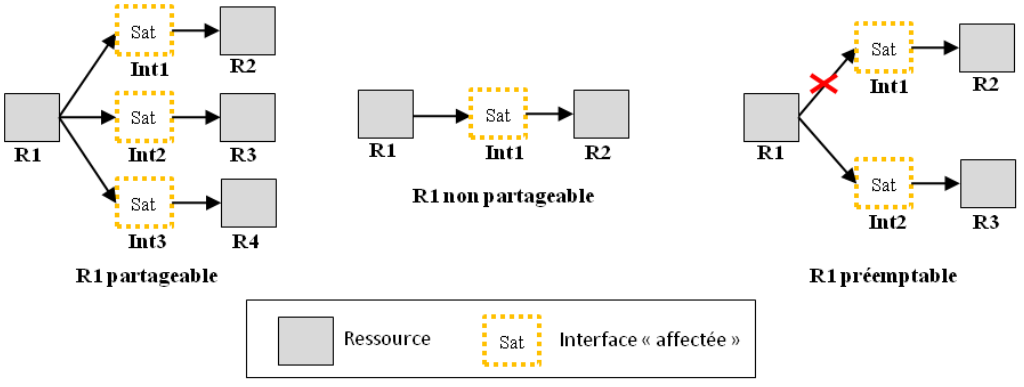

FIGURE 3.5: Propriétés d'une ressource.

2.2.2.b Exemple

La ressource *Steering Wheel* est définie comme non-partageable et préemptable (voir figure 3.3).

2.2.3 Interface

2.2.3.a Définition

Definition 4 (Interface)
Une interface $i_{r_i r_j}$ est un nœud du graphe de ressources qui connecte une ressource requise r_i avec une ressource dépendante r_j (voir figure 3.6) : l'interface matérialise la relation de dépendance entre ces deux ressources.
$\mathcal{I}$ est l'ensemble des interfaces présentes dans le macro-modèle de l'autorité. □

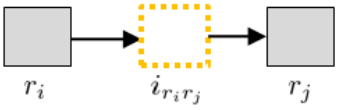

FIGURE 3.6: Interface $i_{r_i r_j}$ reliant une ressource requise r_i et une ressource dépendante r_j.

2.2.3.b Types de dépendance

Une interface $i_{r_i r_j}$ permet de modéliser différents types de relation de dépendance entre la ressource r_i et la ressource r_j :

- la dépendance en initialisation *Init-Dep* : la ressource r_i est requise (i.e. doit être présente) pour permettre à la ressource r_j d'être produite ; dès que le lien de dépendance est effectif, il disparaît aussitôt, cette relation de dépendance a une durée nulle. Exemple : le robot réalise sa tâche de navigation uniquement si le véhicule se trouve sur sa position de départ, qui est une précondition. Cette ressource *StartPosition* est détruite dès que le véhicule se déplace, mais cela n'affecte pas l'exécution ultérieure de la tâche.
- la dépendance en exécution *Exec-Dep* ; la ressource r_i est requise pour permettre à la ressource r_j d'être produite et r_i doit rester présente pour que r_j reste présente. La relation de dépendance est maintenue. Exemple : la tâche de navigation du robot a besoin d'énergie pour commencer, et l'énergie doit rester suffisante pendant toute la durée de l'exécution de la tâche.

– la dépendance en terminaison *End-Dep* ; la ressource r_i est nécessaire pour la production de r_j, le lien de dépendance est effectif alors que r_j n'est pas encore produite. La relation de dépendance s'arrête *lorsque* la ressource r_j est produite (cas d'une tâche, qui est exécutée afin de produire un but, mais dont l'exécution n'a plus de sens lorsque le but est atteint). Exemple : le but *WP-Goal* est poursuivi dès que la tâche de navigation du robot est lancée, activant le lien de dépendance entre ces deux ressources. Le lien de dépendance prend fin dès que la ressource *WP-Goal* est produite, c'est-à-dire dès que le but a été atteint.

La figure 3.7 permet d'illustrer sur un chronogramme comment, pour deux ressources r_i et r_j liées entre elles par une relation de dépendance, les modifications de l'état de r_i (événements de production et destruction) affectent la ressource dépendante r_j.

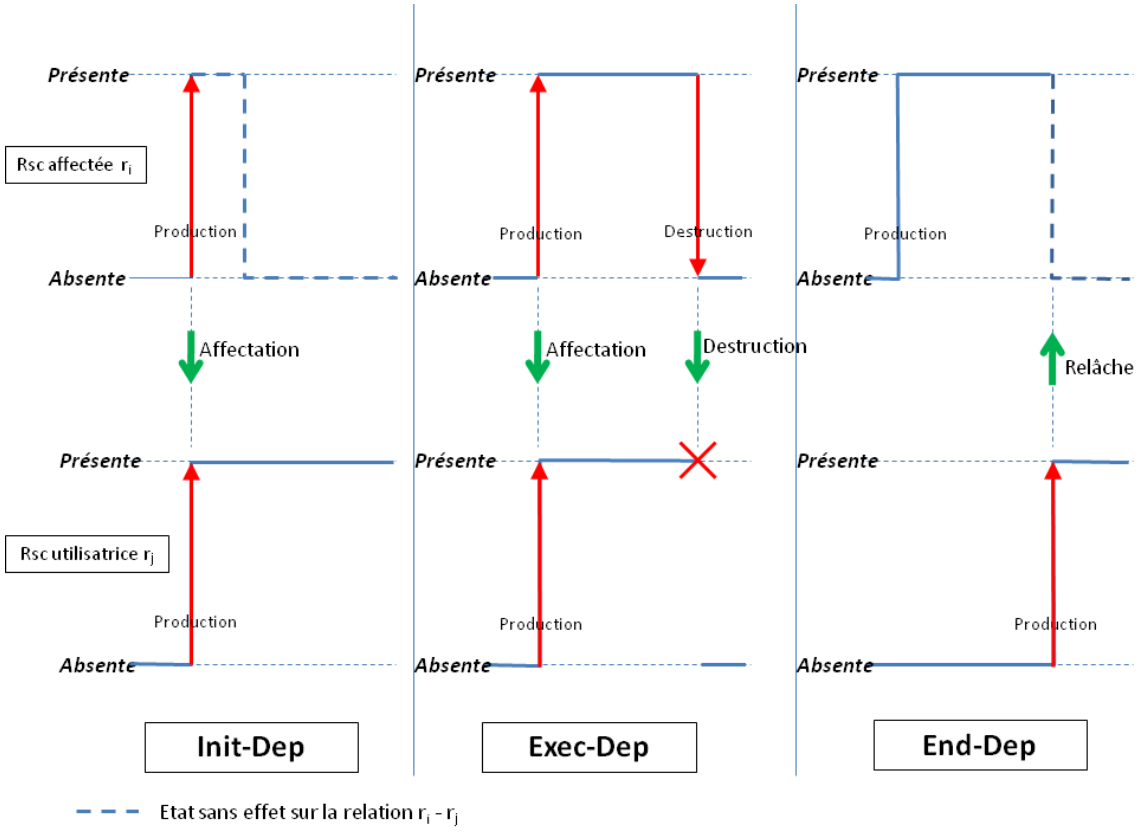

FIGURE 3.7: Dépendances entre r_i et r_j

Le processus d'affectation signifie que la relation de dépendance devient effective entre les deux ressources ; l'étape d'affectation peut être fugitive ou avoir une durée, en fonction de la relation de dépendance. La relâche correspond à la fin de la relation de dépendance.

2.2.3.c Affectation

Definition 5 (Affectation) L'affectation d'une ressource requise r_i à une ressource dépendante r_j *via* une interface $i_{r_i r_j}$ est l'établissement de la relation de dépendance entre r_i et r_j : la ressource r_j « utilise » la ressource r_j. On distingue :

- l'événement d'affectation, qui correspond à l'*activation* du lien de dépendance ; ceci correspond à un changement d'état simultané sur l'interface $i_{r_i r_j}$ et sur la ressource r_i ;
- l'état d'affectation, maintenu aussi longtemps que le lien de dépendance est effectif.□

2.2.3.d Exemple

Sur cet exemple, le plan consiste pour le robot à aller de manière autonome sur le point d'arrivée *WP-Goal*. La figure 3.8 présente le graphe de ressources correspondant. Le but est la production de la ressource $r_{WP-Goal}$, qui dépend de la ressource $r_{NavigationRobot}$; celle-ci requiert les ressources $r_{StartPosition}$, r_{Energy} et $r_{SteeringWheel}$ (commande de cap).

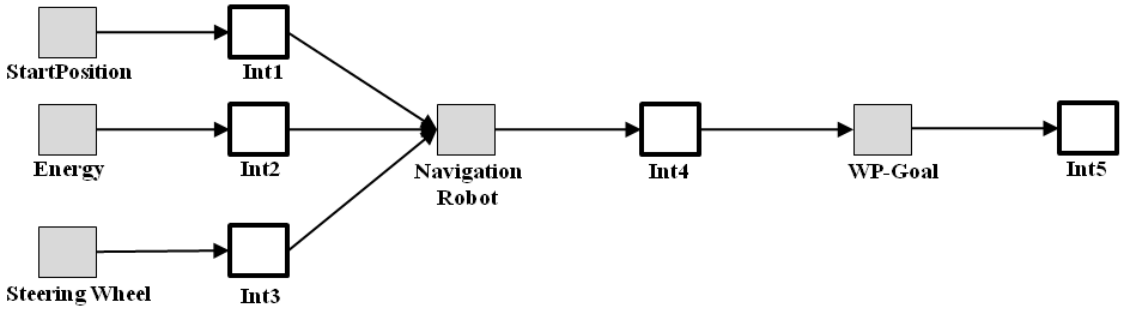

FIGURE 3.8: Graphe de ressources initial

Les ressources sont en gris alors que les interfaces sont en blanc. Pour alléger les notations, les interfaces ont été étiquetées de manière courte ($Int1$, $Int2$, etc.)

La ressource $r_{WP-Goal}$ est le but de ce graphe de ressources : elle dépend pour sa production de la ressource $r_{NavigationRobot}$ *via* l'interface $Int4$, qui représente une relation de dépendance de type $End-Dep$: la production de la ressource $r_{WP-Goal}$ signifie la relâche de la ressource $r_{NavigationRobot}$.

La ressource $r_{NavigationRobot}$ dépend des trois ressources suivantes :

- $r_{StartPosition}$ *via* $Int1$ suivant une relation de dépendance de type $Init-Dep$, la position de départ étant une précondition nécessaire pour lancer la tâche de navigation ;

- r_{Energy} *via* $Int2$ suivant une relation de dépendance de type $Exec{-}Dep$ car $r_{NavigationRobot}$ a besoin de r_{Energy} jusqu'à achèvement de la tâche ;
- $r_{SteeringWheel}$ *via* $Int3$ suivant une relation de dépendance de type $Exec{-}Dep$ car $r_{NavigationRobot}$ a besoin de $r_{SteeringWheel}$ jusqu'à achèvement de la tâche.

Pour que l'affectation ait lieu, il faut que toutes les ressources dont dépend la ressource utilisatrice soient prêtes à être affectées, c'est-à-dire présentes et disponibles ; ainsi en pratique $r_{StartPosition}$, r_{Energy}, $r_{SteeringWheel}$ ne sont affectées simultanément à $r_{NavigationRobot}$ que si toutes individuellement peuvent l'être.

Quant à l'interface $Int5$, elle ne crée pas de relation de dépendance, n'étant pas connectée à une ressource en aval. Son rôle consiste à matérialiser la demande qui est faite sur le but $r_{WP-Goal}$.

2.3 Agents et assignation

2.3.1 Définitions

Definition 6 (Agent)
On appelle agent du macro-modèle d'autorité une entité du système capable de :
- créer un graphe de ressources, c'est-à-dire connecter entre elles des ressources par des relations de dépendance *via* des interfaces ;
- modifier un graphe de ressources, en modifiant les relations de dépendance entre ressources.

L'organisation des ressources d'un agent suivant des relations de dépendance reflète le plan d'exécution de l'agent. □

Soit $\mathcal{C}$ l'ensemble des agents représenté dans le macro-modèle de l'autorité [1].

Une interface entre ressources est toujours créée par un agent, qu'il soit humain ou artificiel : il est le résultat d'une décision, un agent donné ayant choisi de connecter entre elles deux ressources particulières, étant donné les buts poursuivis et les contraintes. Une interface est ainsi marquée par le « sceau » de l'agent qui l'a créée.

Definition 7 (Fonction d'assignation)
On appelle fonction d'assignation $\phi : \mathcal{I} \rightarrow \mathcal{C}$ la fonction associant à chaque interface $i_{r_i r_j} \in \mathcal{I}$ un et un seul agent $c_k \in \mathcal{C} : \phi(i_{r_i r_j}) = c_k$, c_k est l'agent ayant associé la ressource r_i à la ressource r_j *via* l'interface $i_{r_i r_j}$. □

1. Dans le micro-modèle fondé sur les réseaux de Petri et présenté par la suite, les agents sont associés à une couleur de jeton qui leur est propre. À l'ensemble d'agents correspond ainsi un ensemble de *couleurs*, ce qui justifie l'emploi de la lettre $\mathcal{C}$; nous ne ferons ainsi pas de distinction entre *agent* et *couleur d'agent*.

Lorsqu'un agent c crée une interface $i_{r_i r_j}$ entre une ressource r_i et une ressource dépendante r_j, on dit que la ressource r_i a été *assignée* par l'agent c. La figure 3.9-gauche représente une telle assignation.

Remarque : un but étant une ressource comme une autre est lui aussi assigné par un agent. Cependant, aucune ressource ne dépendant du but, l'interface correspondante n'est connectée à aucune ressource dépendante en aval. La figure 3.9-droite représente une telle assignation.

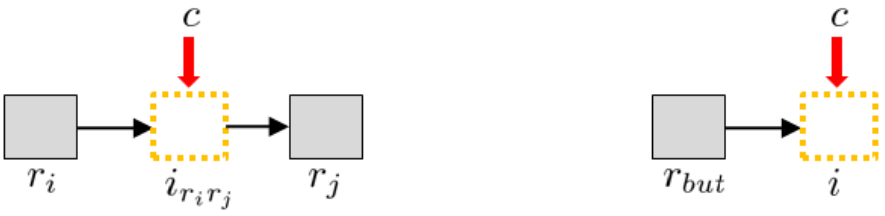

FIGURE 3.9: Assignations par l'agent c

2.3.2 Exemple

Nous examinons ici la sémantique de la création d'une interface entre deux ressources par l'agent *robot*, avec une ressource $r_{NavigationRobot}$ et une ressource $r_{SteeringWheel}$, la ressource $r_{SteeringWheel}$ étant l'unique précondition pour la tâche représentée par la ressource $r_{NavigationRobot}$. Cela signifie que l'agent *robot* prévoit que la ressource $r_{SteeringWheel}$ soit assignée à la ressource $r_{NavigationRobot}$ pour que celle-ci puisse être produite. Ceci constitue une décision de l'agent *robot*.

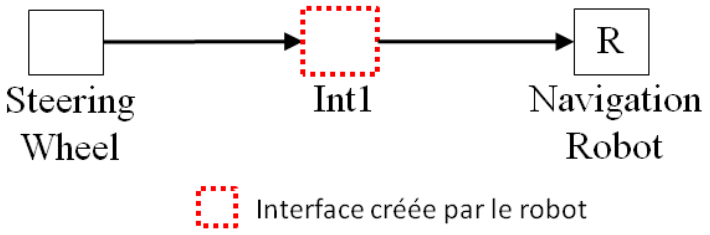

FIGURE 3.10: Graphe de ressources entre ressource physique et tâche

Chaque boîte rectangulaire sur la figure 3.10 représente une ressource, avec son étiquette en-dessous. Si la boîte contient la mention « R », cela signifie que la ressource représente une tâche réalisée par le robot uniquement. La mention « Op » indique une tâche réalisée par l'opérateur.

Lorsque l'opérateur prend en main le joystick contrôlant les mouvements du véhicule, l'opérateur prend la main sur la ressource $r_{SteeringWheel}$ et on peut inférer que c'est dans le but de réaliser la tâche de navigation en vue de rejoindre un point d'arrivée $r_{WP-Goal}$, ce qui donne le graphe de ressources de la figure 3.11 :

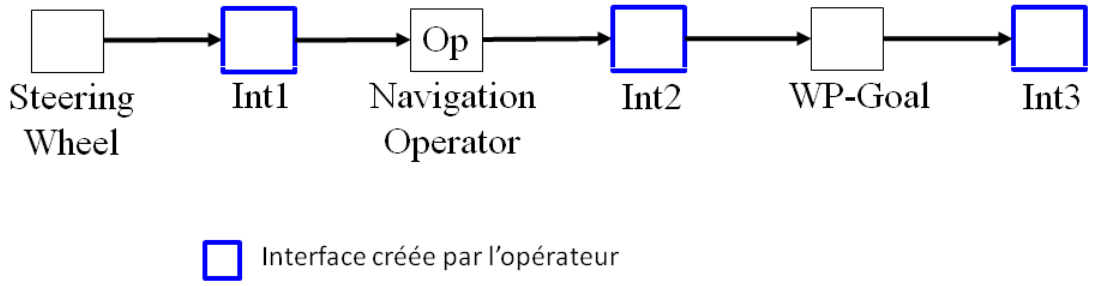

FIGURE 3.11: Graphe de ressources créé par opérateur

Dans ce cas, c'est l'opérateur qui a contrôlé l'assignation de ces ressources entre elles : en notant $Int1 = i_{r_{SteeringWheel} r_{NavigationOperator}}$ et $Int2 = i_{r_{NavigationOperator} r_{WP-Goal}}$, on a bien $\phi(Int1) = operator$ et $\phi(Int2) = operator$. On remarque également l'introduction de l'interface $Int3$, avec $\phi(Int3) = operator$, qui traduit le fait que l'opérateur est à l'origine de ce but.

Prenons maintenant le cas suivant : l'opérateur donne au robot l'ordre d'effectuer une navigation pour rejoindre le point d'arrivée. Il s'agit d'une *délégation*, puique l'opérateur donne un ordre et le robot se charge de mettre en œuvre les moyens nécessaires à la réalisation de cet ordre en complétant le graphe de ressources. Sur ce graphe (voir figure 3.12), c'est l'opérateur qui est à l'origine de la poursuite de ce but.

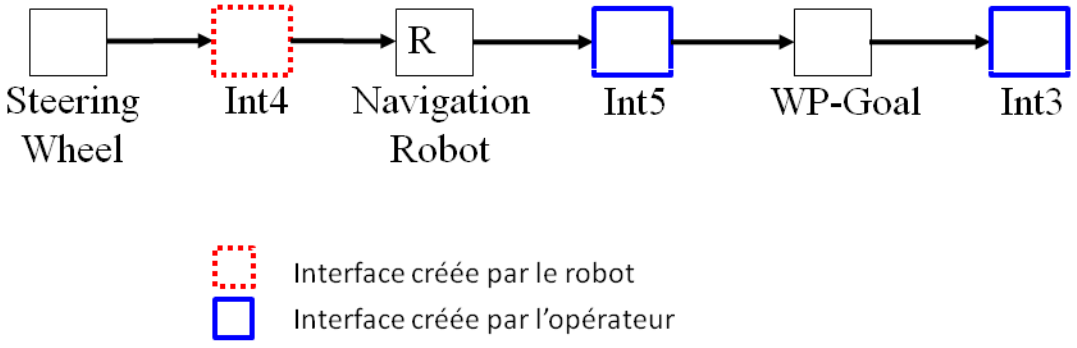

FIGURE 3.12: Graphe de ressources correspondant à une délégation au robot

Soit $Int4 = i_{r_{SteeringWheel} r_{NavigationRobot}}$ et $Int5 = i_{r_{NavigationRobot} r_{WP-Goal}}$, alors $\phi(Int4) = robot$ et $\phi(Int5) = operator$, et on a également $\phi(Int3) = operator$.

Le cas inverse est possible (voir figure 3.13) : dans un plan qu'il a lui-même créé, le robot fait explicitement appel à l'opérateur pour la réalisation de la tâche de navigation.

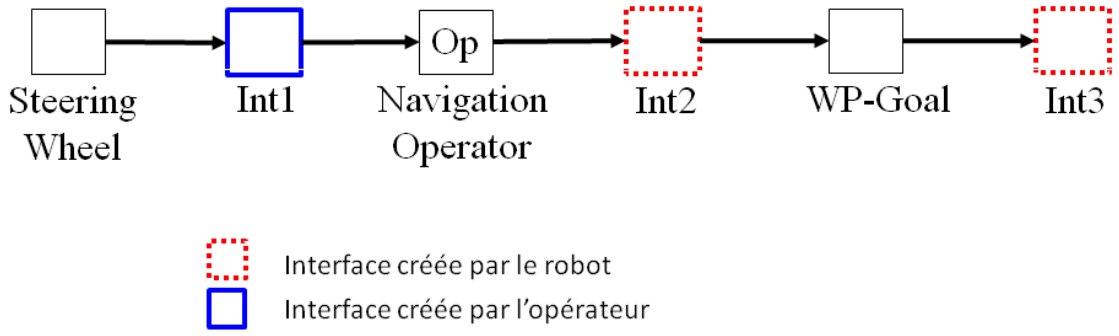

FIGURE 3.13: Graphe de ressources correspondant à une délégation à l'opérateur

Ici, $\phi(Int1) = operator$, $\phi(Int2) = robot$ et $\phi(Int3) = robot$.

2.4 Autorité

2.4.1 Droits d'accès

Definition 8 (Droits d'accès à une ressource) Soit c_x un agent de $\mathcal{C}$ et r une ressource de $\mathcal{R}$.
Soit une interface Int, représentant l'assignation de r par l'agent c_x dans un graphe de ressources en vue de l'accomplissement d'un but. L'ensemble des droits d'accès de c_x à r est $Rights = \{Access, Preemptability, NoAccess\}$, avec :

- *Access* (accès) : capacité de l'agent c_x à ce que la ressource r soit affectée *via* Int, qui correspond à la capacité de l'agent à « contrôler » r.
- *Preemptability* (préemption) : si r est une ressource non-partageable, bien que c_x ait le droit d'accès à r, elle peut être contrôlée par c_y, c'est-à-dire qu'elle peut être déjà affectée *via* une interface $Int2$ créée par l'agent c_y. Le droit de préemption confère à c_x le droit de voir r affectée *via* Int dès qu'il le souhaite, aux dépens de c_y le cas échéant ; inversement, si l'autre agent c_y possède le droit de préemption, c_x peut être menacé de perdre son affectation au profit de c_y. La *garantie de contrôle* assure à c_x que c_y ne pourra pas lui prendre r (en d'autres termes, garantie de contrôle pour c_x = pas de préemption pour c_y).
- *No Access* (pas d'accès) : même si c_x a assigné r *via* l'interface Int, l'événement d'affectation ne peut pas avoir lieu. c_x ne peut pas « contrôler » la ressource r. □

2.4.2 Autorité

Definition 9 (Relation d'autorité)
On appelle relation d'autorité la fonction
$a : \mathcal{R} \times \mathcal{C}^2 \rightarrow Rights \times Rights$
$(r, c_x, c_y) \rightarrow a_{<c_x,c_y>}(r)$
associant à une ressource, pour un couple donné d'agent, le droit d'accès à la ressource pour chaque agent, relativement l'un à l'autre. □

$a_{<c_x,c_y>}(r)$ est définie dans le tableau 3.1.

Agent	Droit	Accès	Préemption	Garantie
c_x	No Access	Non	-	-
c_y	Preemptability	Oui	Oui	Oui
c_x	Access	Oui	Non	Non
c_y	Preemptability	Oui	Oui	Oui
c_x	Access	Oui	Non	Oui
c_y	Access	Oui	Non	Oui
c_x	Preemptability	Oui	Oui	Non
c_y	Preemptability	Oui	Oui	Non

TABLE 3.1: $a_{<c_x,c_y>}(r)$

Remarque : on observe deux configurations pour lesquelles l'autorité des agents est équivalente : $a_{<c_x,c_y>}(r) = (Access, Access)$ et $a_{<c_x,c_y>}(r) = (Preemptability, Preemptability)$. Dans la première, les agents ne peuvent se prendre mutuellement le contrôle de la ressource r, chacun doit attendre si nécessaire que l'autre agent ait terminé. Il s'agit d'une configuration de *coopération*. La deuxième relation peut conduire à une oscillation du contrôle : les agents peuvent se prendre mutuellement et indéfiniment le contrôle de la ressource r, ce qui provoque un comportement du système inefficace voire dangereux. Il s'agit d'une configuration de *concurrence*.

Suivant ce tableau, on observe que le droit de garantie de contrôle pour un agent est en fait implicite, correspondant au fait que l'autre agent de la relation ne possède *pas* le droit de préemption.

Propriétés :
- l'autorité est *graduelle* : le contrôle de l'agent c_x sur la ressource r est de plus en plus fort au fur et à mesure qu'il possède, dans cet ordre, les droits : accès, préemption, garantie de contrôle.

– l'autorité, comme l'autonomie [Castelfranchi et Falcone, 2003] est une notion *relative* entre deux agents : par exemple, pour une ressource r donnée, un agent c_x peut avoir le droit de préemption sur un agent c_y, mais pas sur un agent c_z. Ainsi il y a autant de relations d'autorité qu'il y a de couples d'agents susceptibles d'utiliser r.
– l'autorité est *partagée* entre les agents : pour un couple d'agents $<c_x, c_y>$ pouvant contrôler une ressource r, le gain d'autorité de c_x sur r se fait au détriment de c_y. Par exemple, si c_x obtient la garantie de contrôle sur r, c_y n'a pas la préemption. À l'extrême, c_x peut avoir l'exclusivité de la ressource au détriment de c_y si celui-ci n'y a pas accès du tout : c_x *interdit* l'accès de c_y à r, même si c_x ne contrôle pas r en permanence.

2.4.3 Dynamique de la relation d'autorité

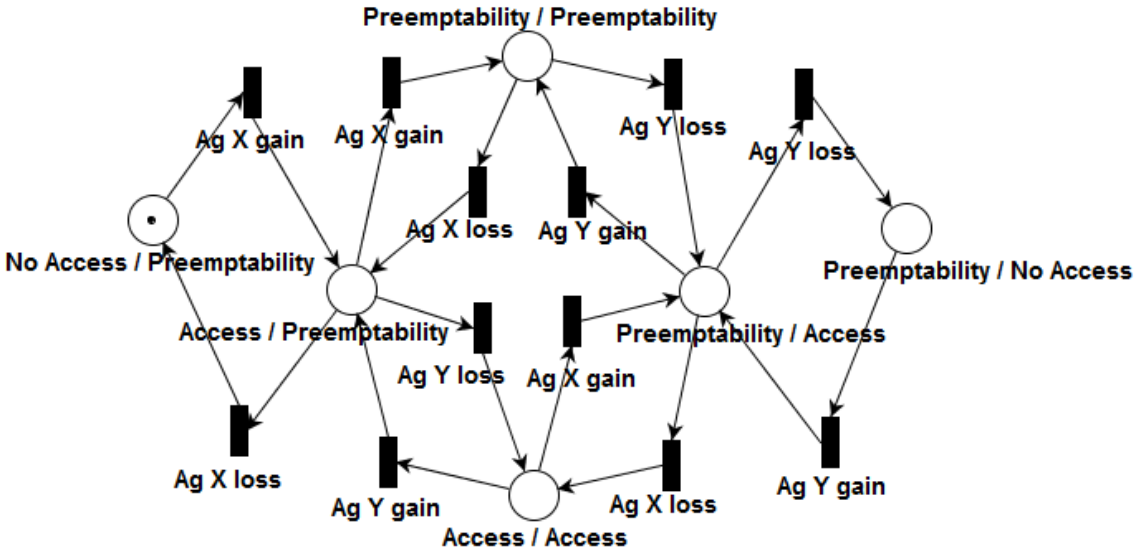

FIGURE 3.14: Dynamique de la relation d'autorité $a_{<c_x,c_y>}(r)$ entre les agents c_x et c_y sur la ressource r.

Le réseau de Petri de la figure 3.14 représente les valeurs de la relation d'autorité $a_{<c_x,c_y>}(r)$ entre deux agents c_x et c_y sur une ressource r. Ce réseau est sauf : une et une seule place peut être marquée à la fois. Chacune des places de ce réseau représente une des six combinaisons des droits d'accès pour les deux agents : chaque place du réseau est étiquetée par (Droit de l'agent c_x / Droit de l'agent c_y), par exemple : $(NoAccess, Preemptability)$. Une transition indique un gain ou une perte d'autorité pour un seul des agents à la fois. Pour passer d'un extrême à l'autre, un agent doit gagner de l'autorité (de « No Access » à « Preemptability ») et l'autre agent en perdre (de « Preemptability » à « No Access »).

2.4.4 Exemple

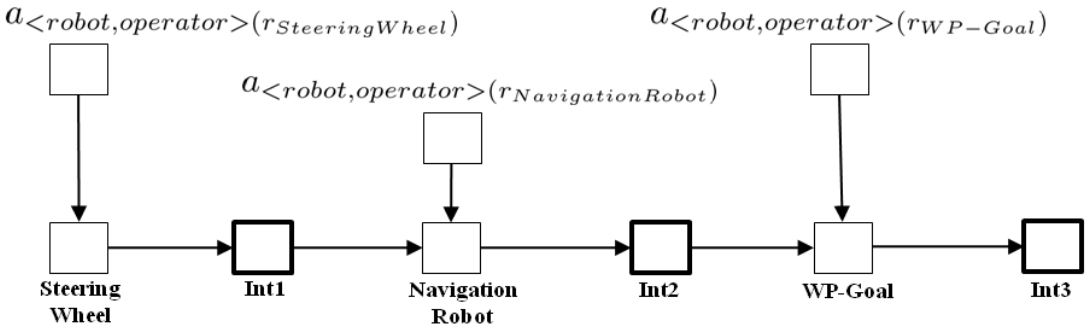

FIGURE 3.15: Exemple de graphe de ressources incluant les relations d'autorité

La figure 3.15 représente une partie de l'exemple précédent (cf section 2.2.3.d), partant de la ressource $r_{SteeringWheel}$. Les relations d'autorité $a_{<robot,operator>}(r_i)$ ont ici été figurées : il y a une relation d'autorité par couple d'agents (ici *robot* et *operator*) sur chaque ressource.

3 Micro-modèle d'autorité

Nous présentons ici le modèle formel des constituants fondamentaux du macro-modèle de l'autorité et du graphe de ressources associé : la *ressource*, l'*interface* et la *relation d'autorité*. Chacun de ces constituants est modélisé par un ensemble fini de propriétés et d'états, les changements d'états étant déclenchés par des événements. Nous adoptons donc le formalisme des réseaux de Petri, qui permettent :

- de représenter la dynamique de systèmes à événements discrets ;
- d'exécuter les modèles obtenus.

De plus, deux des extensions classiques des réseaux de Petri conviennent particulièrement aux constituants que nous modélisons :

- les réseaux de Petri synchronisés : cette extension associe au formalisme standard des réseaux de Petri un ensemble d'événements, qui conditionnent le tir des transitions. Cette extension, associé à la fusion de transitions, nous permet de synchroniser des réseaux de Petri élémentaires entre eux et avec des programmes ou des actions extérieurs.
- les réseaux de Petri colorés : dans leur forme simple, on associe une couleur (un « type ») aux jetons des réseaux de Petri. Les jetons de différentes couleurs peuvent cohabiter dans un même réseau sans interférer entre eux. Nous utilisons dans notre modèle la couleur des jetons comme un sceau propre à chaque agent, afin de distinguer ses actions.

Pour plus de détails sur le formalisme des réseaux de Petri et des extensions que nous utilisons, nous renvoyons vers l'annexe A.

3.1 Notations

Dans les réseaux de Petri utilisés par la suite :
- une transition étiquetée « transition1 » est notée $t_{\text{transition1}}$; une place étiquetée « place1 » est notée p_{place1}.
- le marquage $M_r = \{Present, Allocated, \#Users^3\}$ d'un réseau de Petri r correspond au fait que les places p_{Present} et $p_{\text{Allocated}}$ contiennent un jeton, la place $p_{\#\text{Users}}$ trois jetons et que toutes les autres places de r sont vides.
- la relation $\{Present\} \subseteq M_r$ signifie que la place p_{Present} du réseau r est marquée avec *au moins* un jeton. Elle ne précise rien sur l'état des autres places du réseau.
- la relation $\{Present\} \nsubseteq M_r$ signifie que la place p_{Present} n'est pas marquée (aucun jeton dans la place).

3.2 Ressource

3.2.1 Définition : réseau de Petri de ressource

Toute ressource r du macro-modèle de l'autorité est représentée par le réseau de Petri de la figure 3.16. Il s'agit d'un réseau générique, une ressource donnée sera une instance de ce réseau. Les propriétés de r, ainsi que ses états, seront représentés par le marquage M_r de ce réseau de Petri.

3.2.2 Les propriétés de la ressource

Les propriétés intrinsèques, invariables de la ressource r, conditionnent son comportement :
- r est *partageable* si et seulement si $\{Shareable\} \subseteq M_r$;
- r est *non-partageable* si et seulement si $\{NonShareable\} \subseteq M_r$;
- r est *préemptable* si et seulement si $\{NonShareable, Preemptable\} \subseteq M_r$;
- r est *non-préemptable* si et seulement si $\{Preemptable\} \nsubseteq M_r$.

Exemple :

Soit $r_{SteeringWheel}$ la ressource modélisant l'objet « Commande de cap » du robot. Par conception, nous considérons que l'algorithme de navigation du robot et l'opérateur humain réalisant la tâche de navigation manuellement ne doivent pas accéder à cette ressource simultanément (sinon la conduite du véhicule risquerait d'être incohérente) : $r_{SteeringWheel}$ est ainsi *non-partageable*. Toutefois, nous n'excluons pas que l'opérateur puisse reprendre la main sur le robot dès qu'il le souhaite : la ressource est ainsi *préemptable*. On a donc $\{NonShareable, Preemptable\} \subseteq M_{r_{\text{SteeringWheel}}}$.

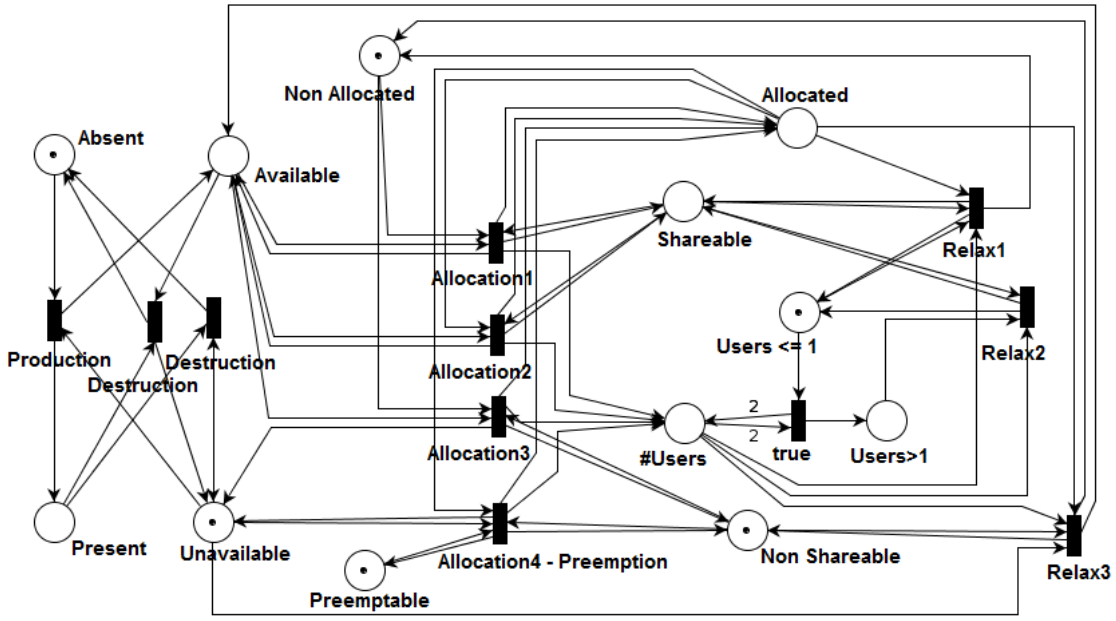

FIGURE 3.16: Réseau générique de ressource (ici non-partageable, préemptable)

3.2.3 Les états de la ressource

L'état de la ressource varie en fonction de l'exécution du plan de mission, des actions des agents : il change en fonction des événements reçus par la ressource et de ses propriétés.

La ressource r peut être dans les états suivants :

- r existe si et seulement si $\{Present\} \subseteq \mathrm{M_r}$;
- r n'existe pas si et seulement si $\{Absent\} \subseteq \mathrm{M_r}$;
- r est disponible pour être affectée à une autre ressource si et seulement si $\{Available\} \subseteq \mathrm{M_r}$;
- r est indisponible si et seulement si $\{Unavailable\} \subseteq \mathrm{M_r}$, cependant r peut toujours être affectée $via\mathrm{t}_{\mathrm{Allocation4}}$ par préemption ;
- r est affectée à une ressource dépendante si et seulement si $\{Allocated\} \subseteq \mathrm{M_r}$;
- r n'est pas affectée si et seulement si $\{NonAllocated\} \subseteq \mathrm{M_r}$.

La place $\mathrm{p}_{\#\mathrm{Users}}$ contient autant de jetons qu'il y a de ressources dépendantes de r et à qui r est affectée. L'une ou l'autre des places $\mathrm{p}_{\mathrm{Users}<=1}, \mathrm{p}_{\mathrm{Users}>1}$ est marquée en fonction du nombre d'affectations simultanées de la ressource r.

3.3 Interface

3.3.1 Définition : réseau de Petri d'interface

Toute interface $i_{r_i r_j}$ du macro-modèle de l'autorité est représentée par le réseau de Petri de la figure 3.17. Ce réseau modélise la relation de dépendance entre les ressources r_i (ressource requise) et r_j (ressource dépendante).

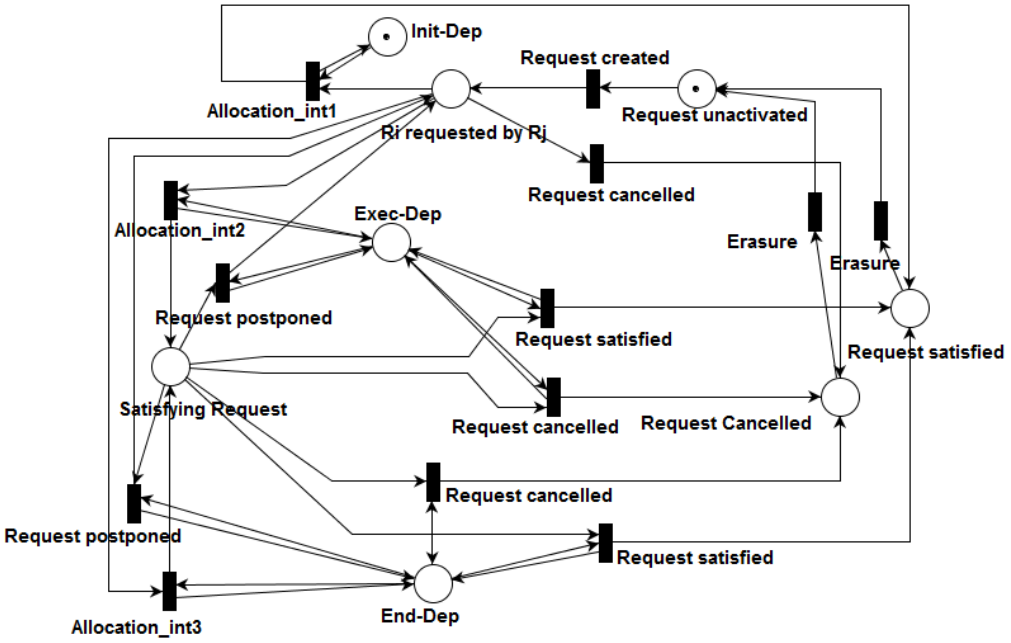

FIGURE 3.17: Réseau générique d'interface (ici interface dépendante en initialisation)

3.3.2 Les états de l'interface

Un état de l'interface $i_{r_i r_j}$ est l'état courant du lien de dépendance entre les ressources r_i et r_j, qui change en cours de mission, en fonction des événements reçus par l'interface $i_{r_i r_j}$ et des propriétés de celle-ci. Le lien de dépendance entre r_i et r_j est une requête d'affectation de la ressource r_i à la ressource r_j, exprimée par l'un des agents du système. Les états de la requête d'affectation sont les suivants :

- la requête est *inactive* si et seulement si $\{RequestUnactivated\} \subseteq \mathrm{M}_{\mathrm{i_{r_i r_j}}}$: la requête n'existe pas encore ;
- la requête est *active* si et seulement si $\{RirequestedbyRj\} \subseteq \mathrm{M}_{\mathrm{i_{r_i r_j}}}$: le lien entre r_i et r_j est effectif, il existe un besoin de r_j envers r_i ;
- la requête est *en cours de satisfaction* si et seulement si $\{SatisfyingRequest\} \subseteq \mathrm{M}_{\mathrm{i_{r_i r_j}}}$: r_i est affectée à r_j, la partie de plan impliquant r_i et r_j est en cours d'exécution ;

- la requête est *satisfaite* si et seulement si $\{RequestSatisfied\} \subseteq \mathrm{M}_{\mathrm{i}_{\mathrm{r_i r_j}}}$: le besoin de r_j envers r_i a été satisfait. Cet état est conservé jusqu'à réinitialisation du réseau d'interface par tir de la transition $\mathrm{t}_{\mathrm{Erasure}}$;
- la requête est *annulée* si et seulement si $\{RequestCancelled\} \subseteq \mathrm{M}_{\mathrm{i}_{\mathrm{r_i r_j}}}$: le besoin de r_j envers r_i a été annulé par l'un des agents du système. Cet état est conservé jusqu'à réinitialisation du réseau d'interface par tir de la transition $\mathrm{t}_{\mathrm{Erasure}}$.

3.3.3 Les propriétés de l'interface

Les propriétés intrinsèques et invariables de l'interface $i_{r_i r_j}$ sont les suivantes (voir figure 3.7) :

- r_j dépend de r_i à l'*initialisation* si et seulement si $\{Init{-}Dep\} \subseteq \mathrm{M}_{\mathrm{i}_{\mathrm{r_i r_j}}}$: r_j a besoin de r_i uniquement pour devenir disponible ; lorsque la transition $\mathrm{t}_{\mathrm{Allocation\_int1}}$ sur $i_{r_i r_j}$ est franchie, alors $\mathrm{t}_{\mathrm{Production}}$ de r_j est tirée. Il n'y a alors plus de lien entre r_i et r_j : $\{Requestsatisfied\} \subseteq i_{r_i r_j}$, la requête est close. Si r_i disparaît par la suite, cela n'affecte pas r_j ;
- r_j dépend de r_i à l'*exécution* si et seulement si $\{Exec{-}Dep\} \subseteq \mathrm{M}_{\mathrm{i}_{\mathrm{r_i r_j}}}$: r_j a besoin de r_i pour devenir disponible et pour le rester. Le tir de $\mathrm{t}_{\mathrm{Allocation\_int2}}$ sur $i_{r_i r_j}$ entraîne le tir de $\mathrm{t}_{\mathrm{Production}}$ de r_j, mais r_i reste affectée à r_j jusqu'à ce que celle-ci la libère : $\{SatisfyingRequest\} \subseteq i_{r_i r_j}$;
- r_j dépend de r_i en *terminaison* si et seulement si $\{End{-}Dep\} \subseteq \mathrm{M}_{\mathrm{i}_{\mathrm{r_i r_j}}}$: r_i ne peut être libérée que lorsque r_j est produite. L'affectation se fait *via* $\mathrm{t}_{\mathrm{Allocation\_int3}}$ sur $i_{r_i r_j}$. C'est uniquement lorsque $\mathrm{t}_{\mathrm{Production}}$ de r_j est tirée que $\{Requestsatisfied\} \subseteq i_{r_i r_j}$. Ceci provoque alors le tir d'une transition $\mathrm{t}_{\mathrm{Relax(1,2ou3)}}$ de r_i.

3.3.4 Exemple

Reprenons plus en détail l'exemple développé en section 2.2.3.d : voici à nouveau le graphe de ressources correspondant sur la figure 3.18.

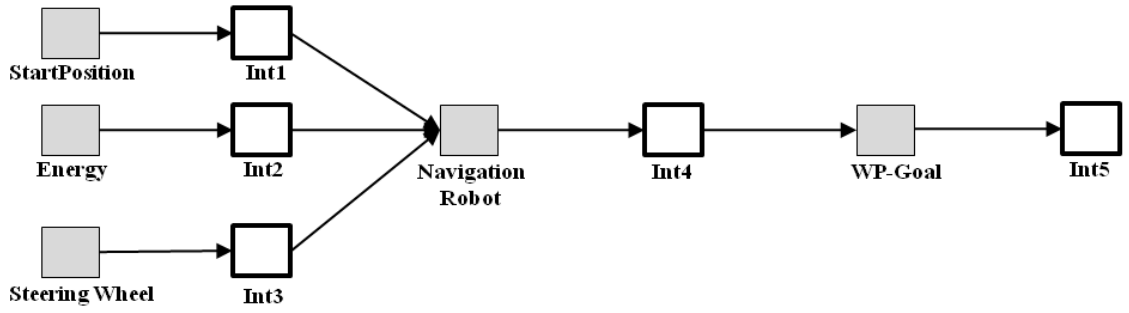

FIGURE 3.18: Graphe de ressources initial

La ressource $r_{StartPosition}$ est assignée à la ressource $r_{NavigationRobot}$ *via* l'interface $Int1$. La relation de dépendance est de type $Init{-}Dep$ car $r_{NavigationRobot}$ a besoin de $r_{StartPosition}$ uniquement pour lancer la tâche. Les ressources r_{Energy} et $r_{SteeringWheel}$ sont également assignées à $r_{NavigationRobot}$ suivant une relation de dépendance de type $Exec{-}Dep$, car elles sont requises du début à l'achèvement de la tâche. Pour que l'affectation ait lieu, il faut que toutes les ressources dont dépend la ressource utilisatrice soient prêtes à être affectées, c'est-à-dire présentes et disponibles. Prenons le cas de la ressource $r_{SteeringWheel}$: il s'agit d'une ressource *non-partageable*, et *préemptable*. Elle est affectée à $r_{NavigationRobot}$ *via* $Int3$, qui représente une dépendance en exécution. Pour que $r_{SteeringWheel}$ soit affectable, il faut que la transition $t_{Allocation3}$ sur $r_{SteeringWheel}$ soit validée par son marquage (c'est-à-dire que $\{Available, NonAllocated, NonShareable\} \subseteq \mathrm{M}_{\mathrm{r_{SteeringWheel}}}$) et que $t_{Allocation\_int2}$ sur $Int3$ le soit également ($\{RirequestedbyRj, Exec{-}Dep\} \subseteq \mathrm{M}_{\mathrm{Int3}}$).

3.3.5 Interfaces et fonction d'assignation ϕ

Ainsi qu'il a été mentionné dans la partie 2.3.2, la fonction d'assignation associe à chaque interface l'agent à l'origine de la création de l'interface, c'est-à-dire de la relation de dépendance entre deux ressources r_i et r_j. Afin d'inclure cette information dans le réseau générique d'interface, nous utilisons un jeton coloré sur ce réseau de Petri : lorsqu'un réseau d'interface est ajouté dans le macro-modèle de l'autorité, le jeton indiquant l'état de la requête entre les deux ressources porte la couleur de l'agent assignateur, c'est-à-dire *robot* si c'est le robot qui a créé cette interface, *operator* si c'est l'opérateur.

En reprenant l'exemple précédent, il est désormais possible de faire figurer la fonction ϕ sur le graphe de ressources (3.19).

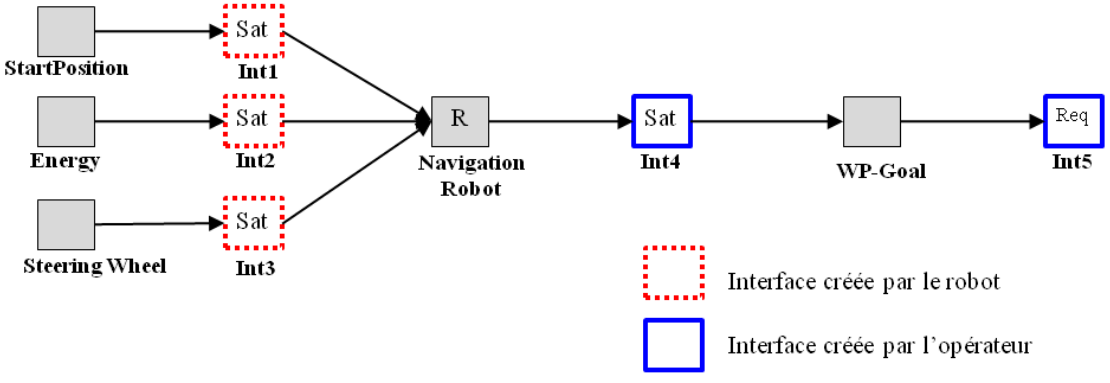

FIGURE 3.19: Graphe de ressources nominal

Ici $\phi(Int1) = \phi(Int2) = \phi(Int3) = robot$ (rouge pointillé) ; $\phi(Int4) = \phi(Int5) = operator$ (bleu continu).

3.4 Autorité sur une ressource

3.4.1 Définition : réseau de Petri de relation d'autorité

Le réseau de la figure 3.14 présentait uniquement les valeurs de la relation d'autorité $a_{<c_x,c_y>}(r)$ entre deux agents c_x et c_y sur une ressource r. Ce réseau doit être complété afin de pouvoir être connecté au graphe de ressources et effectuer sa fonction dans le macro-modèle de l'autorité.

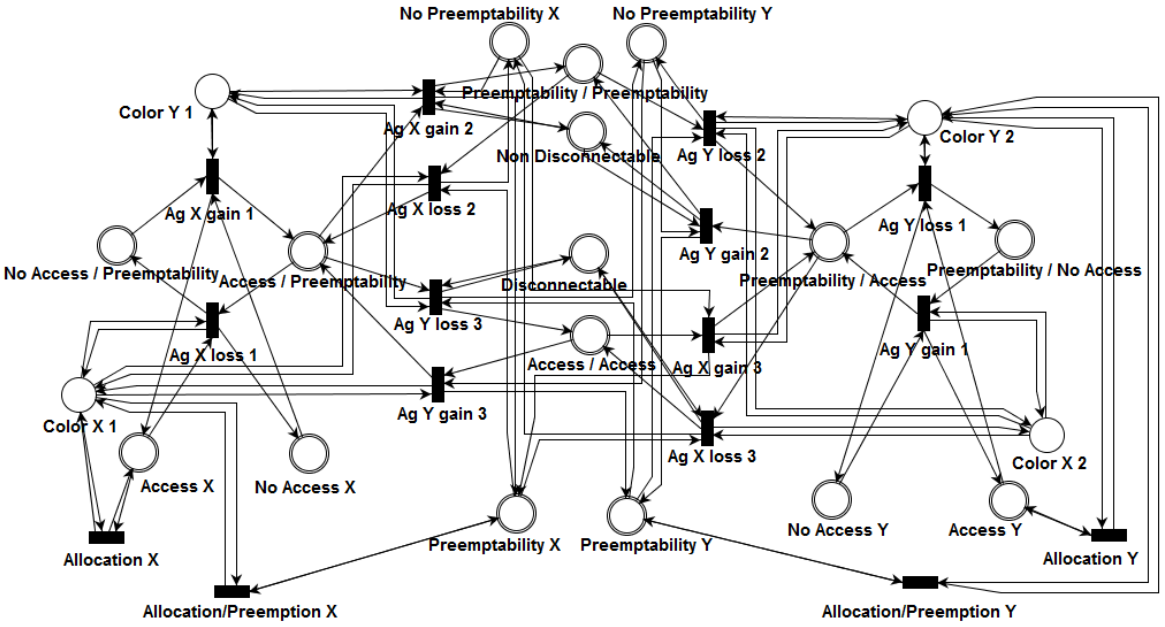

FIGURE 3.20: Relation d'autorité entre les agents c_x et c_y sur la ressource r : $a_{<c_x,c_y>}(r)$.

Sur le réseau de la figure 3.20, on retrouve les droits respectifs des agents c_x et c_y sur la ressource r :

- *No Access* pour c_x et *Preemptability* pour c_y si et seulement si $\{NoAccess/Preemptability\} \subseteq \mathrm{M}_{\mathrm{a}_{<\mathrm{c_x},\mathrm{c_y}>}(\mathrm{r})}$;
- *Access* pour c_x et *Preemptability* pour c_y si et seulement si $\{Access/Preemptability\} \subseteq \mathrm{M}_{\mathrm{a}_{<\mathrm{c_x},\mathrm{c_y}>}(\mathrm{r})}$;
- *Access* pour c_x et *Access* pour c_y si et seulement si $\{Access/Access\} \subseteq \mathrm{M}_{\mathrm{a}_{<\mathrm{c_x},\mathrm{c_y}>}(\mathrm{r})}$;
- *Preemptability* pour c_x et *Preemptability* pour c_y si et seulement si $\{Preemptability/Preemptability\} \subseteq \mathrm{M}_{\mathrm{a}_{<\mathrm{c_x},\mathrm{c_y}>}(\mathrm{r})}$;

- *Preemptability* pour c_x et *Access* pour c_y si et seulement si $\{Preemptability/Access\} \subseteq \mathrm{M}_{\mathrm{a}_{<\mathrm{c_x},\mathrm{c_y}>}(\mathrm{r})}$;
- *Preemptability* pour c_x et *No Access* pour c_y si et seulement si $\{Preemptability/NoAccess\} \subseteq \mathrm{M}_{\mathrm{a}_{<\mathrm{c_x},\mathrm{c_y}>}(\mathrm{r})}$.

Par rapport au réseau précédent de la figure 3.14 ont été ajoutés dans le réseau de la figure 3.20 :

- les places destinées uniquement à porter la couleur des agents de la relation, c'est-à-dire c_x et c_y. Il s'agit des places $\{p_{ColorX1}, p_{ColorY1}, p_{ColorX2}, p_{ColorY2}\}$ (il y a deux places porteuses de couleur pour chaque agent afin de simplifier la représentation du modèle, limitant la longueur des arcs à tracer) ;
- les places représentant les droits effectifs des agents dans la relation d'autorité : $\{p_{AccessX}, p_{AccessY}, p_{PreemptabilityX}, p_{PreemptabilityY}, p_{NoPreemptabilityX}, p_{NoPreemptabilityY}\}$;
- les places $p_{Disconnectable}$ et $p_{NonDisconnectable}$: cette propriété additionnelle permet de distinguer, en cas de gain d'autorité d'un des agents si, de par la conception du système, la relation d'autorité se place soit sur l'état $p_{Access/Access}$ (état stable), soit sur l'état $p_{Preemptability/Preemptability}$ (état instable) ;
- les transitions $\{t_{AllocationX}, t_{AllocationY}, t_{Allocation/PreemptionX}, t_{Allocation/PreemptionY}\}$: ces transitions permettent à la connexion du réseau de la relation d'autorité avec les réseaux de ressource associée r et d'interfaces assignant r ; ces transitions sont destinées à rendre effectifs les droits possédés par chaque agent de la relation d'autorité lors de l'affectation à proprement parler (voir section 2.2 pour plus d'explications concernant l'affectation).

De plus, les places du réseau de la figure 3.14 ont ici été transformées en places *monochromatiques*, utilisant uniquement des jetons de couleur neutre (nous renvoyons à l'annexe A pour plus de détails sur l'usage des places monochromatiques). Toutes les places de la figure 3.20 sont monochromatiques à l'exception des places destinées à porter la couleur des agents $\{p_{ColorX1}, p_{ColorY1}, p_{ColorX2}, p_{ColorY2}\}$: l'évolution interne du réseau de relation d'autorité ne dépend pas de la couleur des jetons, cependant les couleurs sont prises en compte pour la connexion des transitions $\{t_{AllocationX}, t_{AllocationY}, t_{Allocation/PreemptionX}, t_{Allocation/PreemptionY}\}$ avec les réseaux de ressource associée r et d'interfaces assignant r, pour l'affectation.

3.4.2 Exemple

Soit la ressource $r_{SteeringWheel}$: cette ressource est *non-partageable* et *préemptable*. Considérons maitenant la relation d'autorité pour le couple d'agent (*robot*, *operator*) sur la ressource : soit $a_{<robot,operator>(r_{SteeringWheel})} = \{Access, Preemptability\}$. Ceci est refleté dans le marquage du réseau de la relation d'autorité, avec $\{Access/Preemptability\} \subseteq \mathrm{M}_{\mathrm{a}_{<\mathrm{robot},\mathrm{operator}>}(\mathrm{r}_{\mathrm{SteeringWheel}})}$.

Cela signifie que le robot peut créer des assignations de cette ressource avec d'autres ressources dépendantes : ces relations de dépendance pourront être activées seulement si la ressource $r_{SteeringWheel}$ n'est pas utilisée dans une autre relation de dépendance. Á l'inverse, l'opérateur humain peut créer des relations de dépendance qui seront activées dès que possible, c'est-à-dire dès que la ressource $r_{SteeringWheel}$ est présente, quel que soit son état d'affectation.

Chapitre 4

Conflits et résolution : contrôleur de la dynamique de l'autorité

Sur la base du formalisme défini au chapitre précédent, nous proposons une définition du concept de conflit, qui apparaît au sein des réseaux de Petri de ressources du micro-modèle. Nous avons identifié deux types de conflit, le conflit de destruction, apparaissant à la suite d'une défaillance au sein du système, et le conflit de préemption, faisant suite à une mauvaise coordination entre agents. La suite du chapitre est consacrée à la présentation des méthodes de résolution de conflit que nous proposons, en fonction du type de conflit ainsi que des contraintes qui sont imposées au système lors de la recherche de solutions. Nous présentons comment l'autorité peut être redistribuée entre agents de manière dynamique sur les ressources afin de permettre ou faciliter la résolution de conflit. Les transformations à réaliser pour passer du modèle de l'autorité en conflit à un modèle de l'autorité solution, c'est-à-dire sans conflit et respectant les contraintes de résolution imposées (telle que la conservation des buts du plan d'exécution), sont également détaillées.

1 Conflits sur les ressources

1.1 Définition

Lors du déroulement de la mission, des différences peuvent apparaître entre l'exécution prévue du plan et son exécution observée. Ces différences proviennent d'aléas liés à l'environnement, de défaillances, ou d'actions imprévues des agents, et vont se manifester sur une ou plusieurs des ressources du graphe de ressources qui vont se retrouver dans un état incohérent, que nous appelons conflit. Pour déterminer les conflits possibles, il faut construire l'ensemble des marquages atteignables des réseaux de Petri représentant les ressources.

Les ressources étant toutes représentées par le même réseau de Petri (figure 3.16), il suffit d'étudier les marquages atteignables de ce réseau en fonction de son marquage initial, c'est-à-dire en fonction des propriétés intrinsèques de la ressource. Nous avons classé ces marquages atteignables selon qu'ils résultent d'une séquence de tirs de transitions correspondant à l'exécution d'un plan nominal ou au contraire à une séquence correspondant à un aléa, ces derniers étant des conflits. Ainsi, en considérant le réseau de ressource, on constate que le tir des transitions $t_{Production}$ et $t_{Destruction}$ ne dépend que du marquage des places $p_{Present}, p_{Absent}, p_{Available}$ et $p_{Unavailable}$, ainsi que de l'arrivée d'événements externes de production et de destruction ; le reste des états de la ressource n'est pas pris en considération. Pourtant, sémantiquement, une ressource ne peut être à la fois absente et allouée, par exemple. Bien qu'un tel marquage soit atteignable, il indique qu'une séquence de tirs de transitions que nous considérons comme « non nominale » s'est produite. Il s'agit donc de recenser l'ensemble de ces marquages et d'en vérifier l'apparition.

Un conflit sur une ressource r est un marquage du réseau de Petri représentant r résultant d'une séquence non nominale de tirs de transitions. Il existe deux types de conflits : le conflit de destruction $\text{Conflict}_{\text{Destruction}}$ et le conflit de préemption $\text{Conflict}_{\text{Preemption}}$.

Definition 10 (Conflit de destruction)
On appelle conflit de destruction sur la ressource r un marquage M_r de r tel que $\{Absent, Allocated\} \subseteq \text{M}_\text{r}$. $\text{Conflict}_{\text{Destruction}} = \{Absent, Allocated\}$ se produit lorsqu'une transition $\text{t}_{\text{Destruction(1ou2)}}$ de la ressource r est tirée alors que $\{Allocated\} \subseteq \text{M}_\text{r}$ (voir figure 4.1). Selon ce que représente r, ce conflit est la manifestation d'une panne, de l'échec d'une tâche, ou d'une violation de contrainte : r est détruite alors qu'elle était affectée. □

Definition 11 (Conflit de préemption)
On appelle conflit de préemption sur la ressource r un marquage M_r de r tel que $\{NonShareable, \#Users^2\} \subseteq \text{M}_\text{r}$. $\text{Conflict}_{\text{Preemption}} = \{NonShareable, \#Users^2\}$ se produit lorsque la transition d'affectation $\text{t}_{\text{Allocation4-Preemption}}$ est tirée alors que $\{Allocated, NonShareable\} \subseteq \text{M}_\text{r}$. C'est le cas d'une ressource dépendante d'une ressource non-partageable, forçant l'activation de ce lien de dépendance (par préemption) alors que la ressource non-partageable est déjà affectée à une autre ressource (voir figure 4.2), deux interfaces sont alors connectées à r dans l'état *Satisfying Request*. □

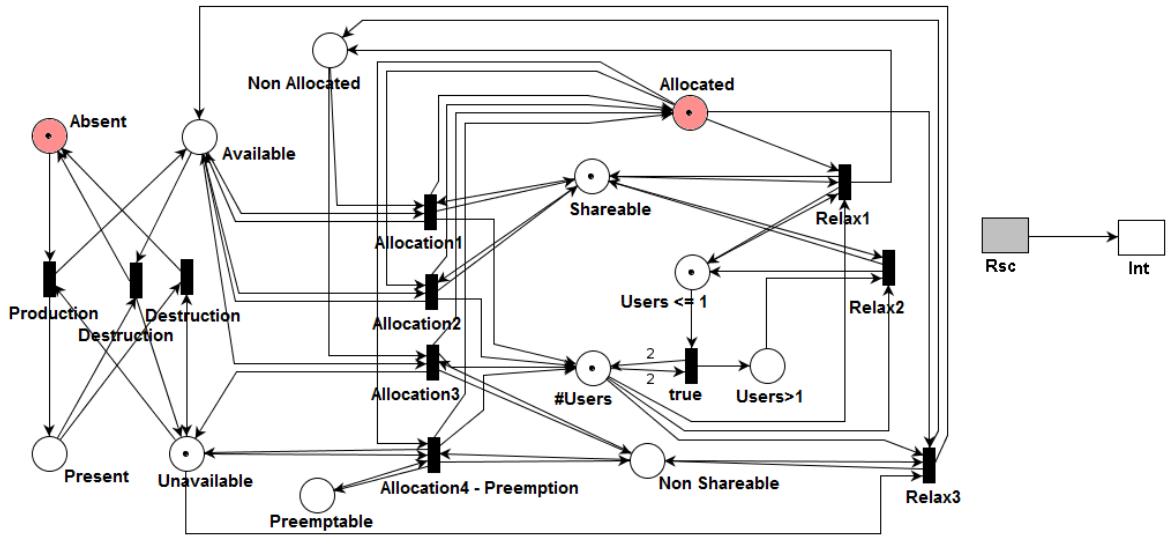

FIGURE 4.1: Conflit de destruction sur r

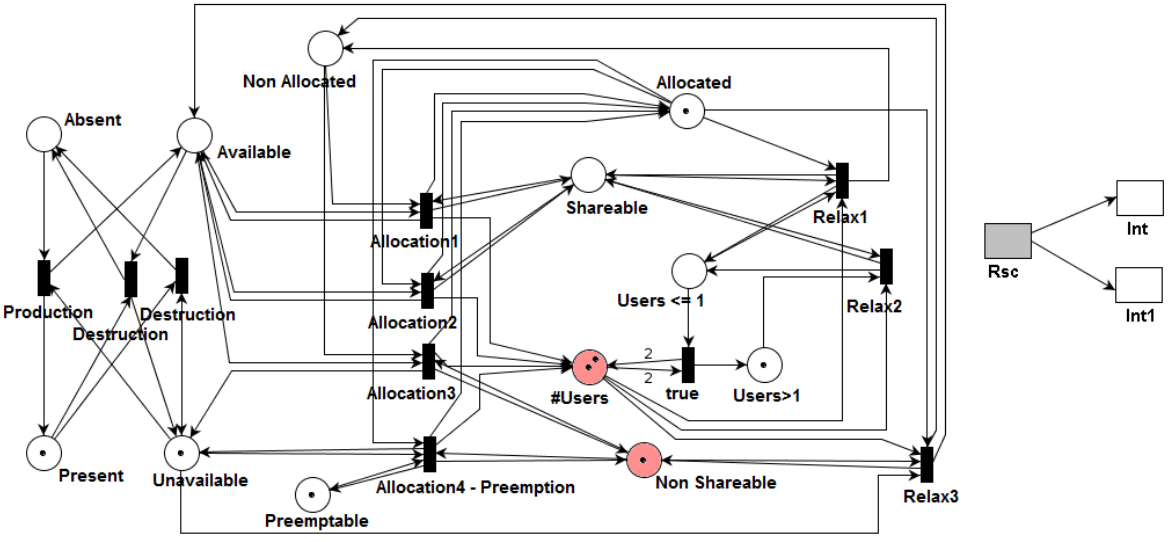

FIGURE 4.2: Conflit de préemption sur r

1.2 Exemple de conflit de destruction

En reprenant l'exemple précédent (voir figure 3.19), alors que le robot réalise sa navigation autonome vers *WP-Goal*, l'indicateur de batterie indique que celle-ci présente un niveau d'énergie anormalement bas, indiquant une avarie. La situation est détectée

(voir la section sur le suivi de situation 1.4) et la ressource d'énergie du robot est alors détruite : la transition $t_{Destruction}$ de r_{Energy} est tirée. Il en résulte la situation illustrée figure 4.3, qui est une situation de conflit de destruction : $Conflict_{Destruction} \subseteq M_{r_{Energy}}$.

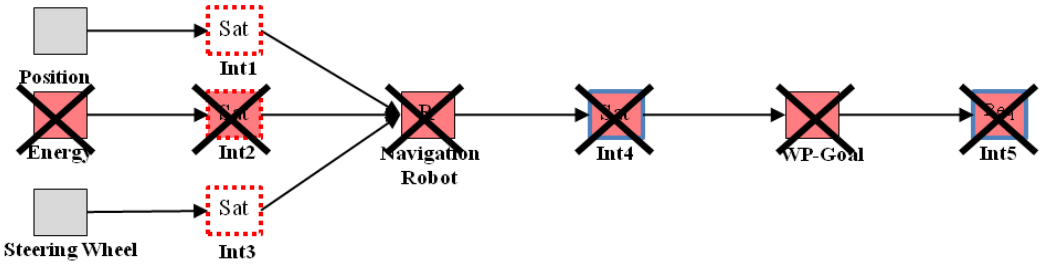

FIGURE 4.3: Propagation de la défaillance de r_{Energy} dans le graphe de ressources

Le conflit apparu sur r_{Energy} est propagé suivant les relations de dépendance, invalidant une partie du plan représenté par le graphe de ressources et la poursuite du but associé $r_{WP-Goal}$.

1.3 Exemple de conflit de préemption

En reprenant l'exemple précédent (voir figure 3.19), alors que le robot réalise sa navigation autonome vers *WP-Goal*, l'opérateur humain reprend le contrôle de la trajectoire *via* son joystick.

Lorsque l'opérateur veut prendre le contrôle du véhicule, il agit sur son interface homme-machine (IHM) en saisissant son joystick de commande. La détection de cette action provoque l'insertion dans la modélisation de l'autorité d'un nouveau sous-graphe de ressources suivant un modèle de tâches prédéfini, en fonction du point d'entrée de l'IHM activé par l'opérateur. Ceci donne le graphe de ressources présenté sur la figure 4.4. On notera que le but de l'opérateur $r_{WP-Operator}$ est inclu dans le graphe de ressources, tout en étant inconnu. Le graphe de ressources de la figure 4.5 constitue ainsi l'état explicite de connaissance de l'agent robotique consécutif aux actions de l'agent humain. On constate que la partie du graphe en état d'affectation est toujours celle du robot réalisant une tâche de navigation vers *WP-Goal* car les interfaces $Int1, Int2, Int3, Int4$ sont marquées *Satisfying Request*. $Int5$ est marquée comme *Ri requested by Rj* car, matérialisant la requête de l'opérateur sur le but $r_{WP-Goal}$, elle est de type *End-Dep*, ne pouvant être satisfaite que lorsque la ressource $r_{WP-Goal}$ est produite. Les interfaces du sous-graphe introduit suite à l'intervention de l'opérateur $Int6, Int7, Int8, Int9, Int10$ sont initialement marquées *Ri requested by Rj*.

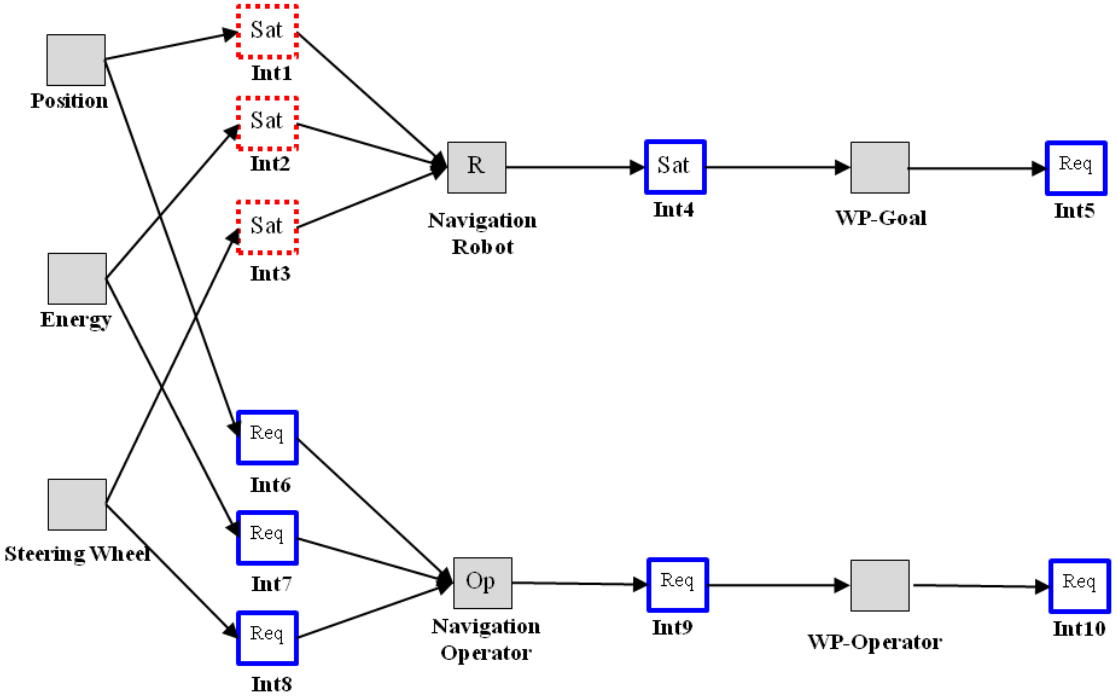

FIGURE 4.4: Graphe de ressources initial avec insertion du graphe induit par l'action de l'opérateur

Une fois ce sous-graphe de ressources introduit, le joueur de Petri procède à la mise à jour du modèle. Considérons que nous avons la relation d'autorité suivante : $a_{robot,operator}(r_{SteeringWheel}) = (Access, Preemptability)$ c'est-à-dire que l'opérateur dispose du droit de préemption sur le robot pour la ressource $r_{SteeringWheel}$. La ressource $r_{SteeringWheel}$, déjà affectée à $r_{NavigationRobot}$ *via* l'interface $Int3$, est préemptable et requise par $r_{NavigationOperator}$ *via* $Int8$. La requête de l'opérateur peut toutefois être satisfaite par la réaffectation de $r_{SteeringWheel}$: les transitions $t_{Allocation4}$ de $r_{SteeringWheel}$ et $t_{Allocation\_int2}$ sur $Int8$ sont tirées. On obtient alors le graphe de ressources de la figure 4.5. $r_{SteeringWheel}$ est alors affectée simultanément à $r_{NavigationRobot}$ et à $r_{NavigationOperator}$ *via* $Int3$ et $Int8$, ce qui est incohérent car il s'agit d'une ressource non-partageable.

On vérifie alors $Conflict_{Preemption} \subseteq M_{r_{SteeringWheel}}$. Le conflit apparu sur $r_{SteeringWheel}$ est propagé suivant les relations de dépendance, invalidant une partie du plan représenté sous forme de graphe de ressources et la poursuite des buts associés.

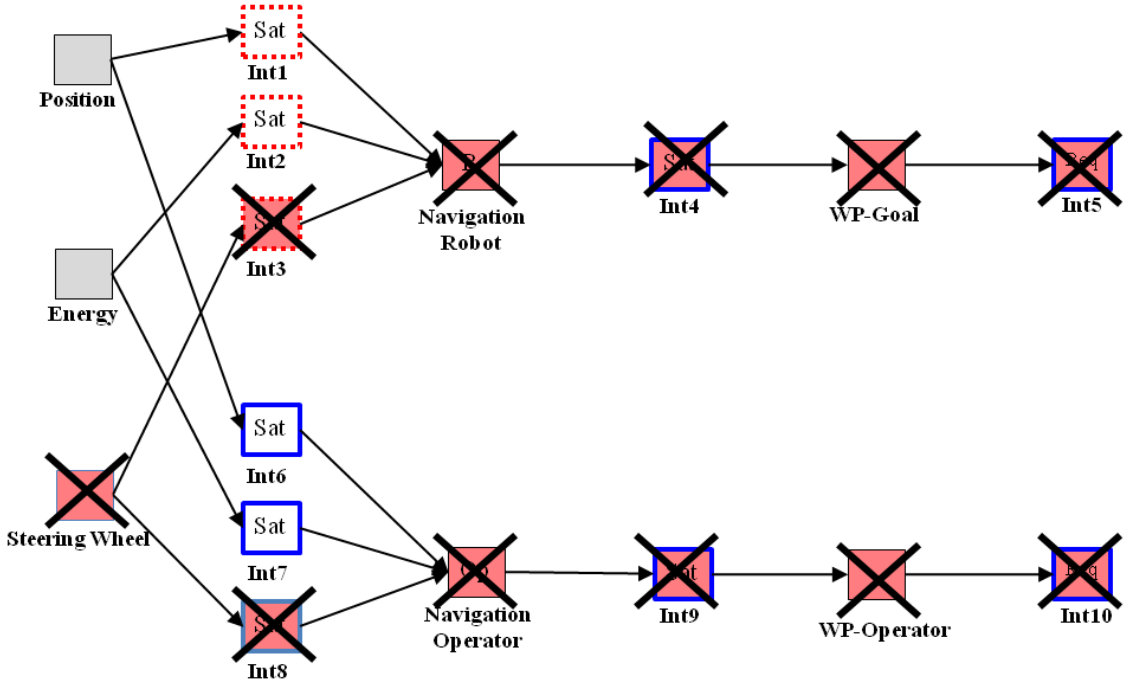

FIGURE 4.5: Graphe de ressources avec conflit de préemption

1.4 Définitions et notations

Conflit : soit la fonction $conflict : \mathcal{R} \to \mathcal{R}$ telle que
$conflict(\mathcal{R}) = \{r_i \in \mathcal{R}, \text{Conflict}_{\text{Preemption}} \subseteq \text{M}_{\text{ri}}\} \cup \{r_j \in \mathcal{R}, \text{Conflict}_{\text{Destruction}} \subseteq \text{M}_{\text{rj}}\}$,
c'est-à-dire l'ensemble des ressources de $\mathcal{R}$ ayant un marquage de conflit.

Definition 12 (Relation de dépendance)
Soit $i_{r_i r_j} \in \mathcal{I}$ l'interface de G intercalée entre les ressources r_i et r_j.
$i_{r_i r_j}$ modélise la relation de dépendance « r_j dépend de r_i » notée $(r_i \rhd_G r_j)$. On dit alors que r_i est une ressource *requise* par r_j et que r_j est une ressource *dépendante* de r_i.
La relation de dépendance est transitive : si dans le graphe G, $(r_i \rhd_G r_j)$ et $(r_j \rhd_G r_k)$, alors $(r_i \rhd_G r_k)$: il existe dans G un chemin entre r_i et r_k, r_k dépend de r_i.
On écrit encore $(r_i \rhd_G r_k) \in \mathcal{P}(\mathcal{I})$, avec $\mathcal{P}(\mathcal{I})$ l'ensemble des parties de $\mathcal{I}$.
La notation $r_i \trianglerighteq_G r_l$ exprime l'absence de chemin dans G entre r_i et r_l, r_l ne dépend pas de r_i.

Connexion entre interface et ressource :
Soit $r \in \mathcal{R}$ et $Int \in \mathcal{I}$,
- on écrit $r \to_G Int$ si et seulement si $\mathcal{F}(r, Int) = 1$: r est *connectée* à Int dans G ;
- on écrit $Int \to_G r$ si et seulement si $\mathcal{B}(Int, r) = 1$: Int est *connectée* à r dans G.

Definition 13 (Ressources requises sur G)
Soit $r_i \in \mathcal{R}$.
On appelle $rdep_G^-$ l'ensemble des ressources requises par r_i dans G
$rdep_G^-(r_i) = \{r_k,\, r_k \in \mathcal{R} \text{ et } (r_k \rhd_G r_i) \in \mathcal{P}(\mathcal{I})\}$.

Definition 14 (Ressources dépendantes sur G)
Soit $r_i \in \mathcal{R}$.
On appelle $rdep_G^+$ l'ensemble des ressources dépendantes de r_i dans G
$rdep_G^+(r_i) = \{r_k,\, r_k \in \mathcal{R} \text{ et } (r_i \rhd_G r_k) \in \mathcal{P}(\mathcal{I})\}$.

Sur le même principe, on écrit :

Definition 15 (Interfaces requises sur G)
Soit $r_i \in \mathcal{R}$.
On appelle $idep_G^-(r_i)$ l'ensemble des interfaces connectées en amont de la ressource r_i
$idep_G^-(r_i) = \{Int \in \mathcal{I} \text{ telle que } \exists r_j \in (rdep_G^-(r_i) \cup \{r_i\}), Int \rightarrow_G r_j\}$.

et :

Definition 16 (Interfaces dépendantes sur G)
Soit $r_i \in \mathcal{R}$.
On appelle $idep_G^+(r_i)$ l'ensemble des interfaces connectées en aval de la ressource r_i
$idep_G^+(r_i) = \{Int \in \mathcal{I} \text{ telle que } \exists r_j \in (rdep_G^+(r_i) \cup \{r_i\}), r_j \rightarrow_G Int\}$.

On écrit alors :

Definition 17 (Graphe de ressources requis)
On appelle graphe de ressources requis de r_i le sous-graphe de G
$d_G^-(r_i) = \{rdep_G^-(r_i), idep_G^-(r_i), \mathcal{F}, \mathcal{B}\}$. □

et :

Definition 18 (Graphe de ressources dépendant)
On appelle graphe de ressources dépendant de r_i le sous-graphe de G
$d_G^+(r_i) = \{rdep_G^+(r_i), idep_G^+(r_i), \mathcal{F}, \mathcal{B}\}$.

Par extension :

Definition 19 (Graphe de ressources requis étendu)
On appelle graphe de ressources requis étendu de r_i le sous-graphe de G requis par la ressource r_i $d_G^-(r_i)$ auquel on ajoute la ressource r_i elle-même :
$D_G^-(r_i) = \{rdep_G^-(r_i) \cup \{r_i\}, idep_G^-(r_i), \mathcal{F}, \mathcal{B}\}$.

Dans le même esprit :

Definition 20 (Graphe de ressources dépendant étendu)
On appelle graphe de ressources dépendant étendu de r_i le sous-graphe de G dépendant de la ressource r_i auquel on ajoute la ressource r_i elle-même :
$D_G^+(r_i) = \{rdep_G^+(r_i) \cup \{r_i\}, idep_G^+(r_i), \mathcal{F}, \mathcal{B}\}$.

Voici quelques figures illustrant ces définitions. Considérons la ressource $R0$ dans un graphe de ressource G présenté figure 4.6. Sur cette figure, les interfaces ont été colorées, pour permettre de mieux distinguer les interfaces situées en amont de la ressource $R0$ (couleur orange et trait pointillé) des interfaces situées en aval de $R0$ (couleur verte, trait continu).

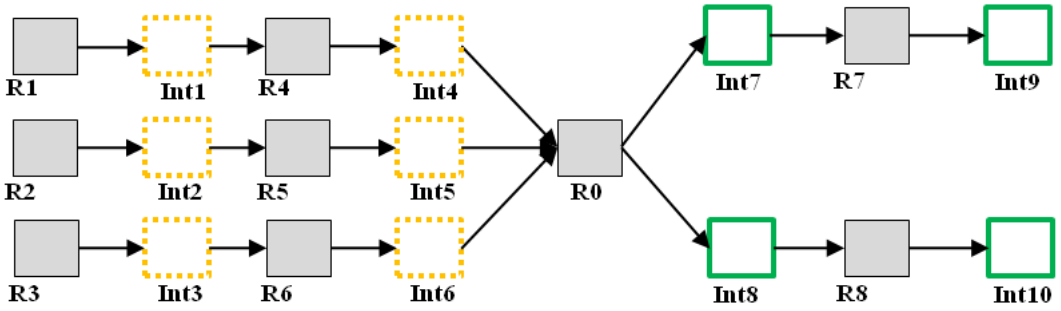

FIGURE 4.6: Graphe de ressources comprenant la ressource $R0$

Voici alors successivement $d_G^-(R0)$, puis $d_G^+(R0)$, $D_G^-(R0)$ et $D_G^+(R0)$ (figure 4.7).

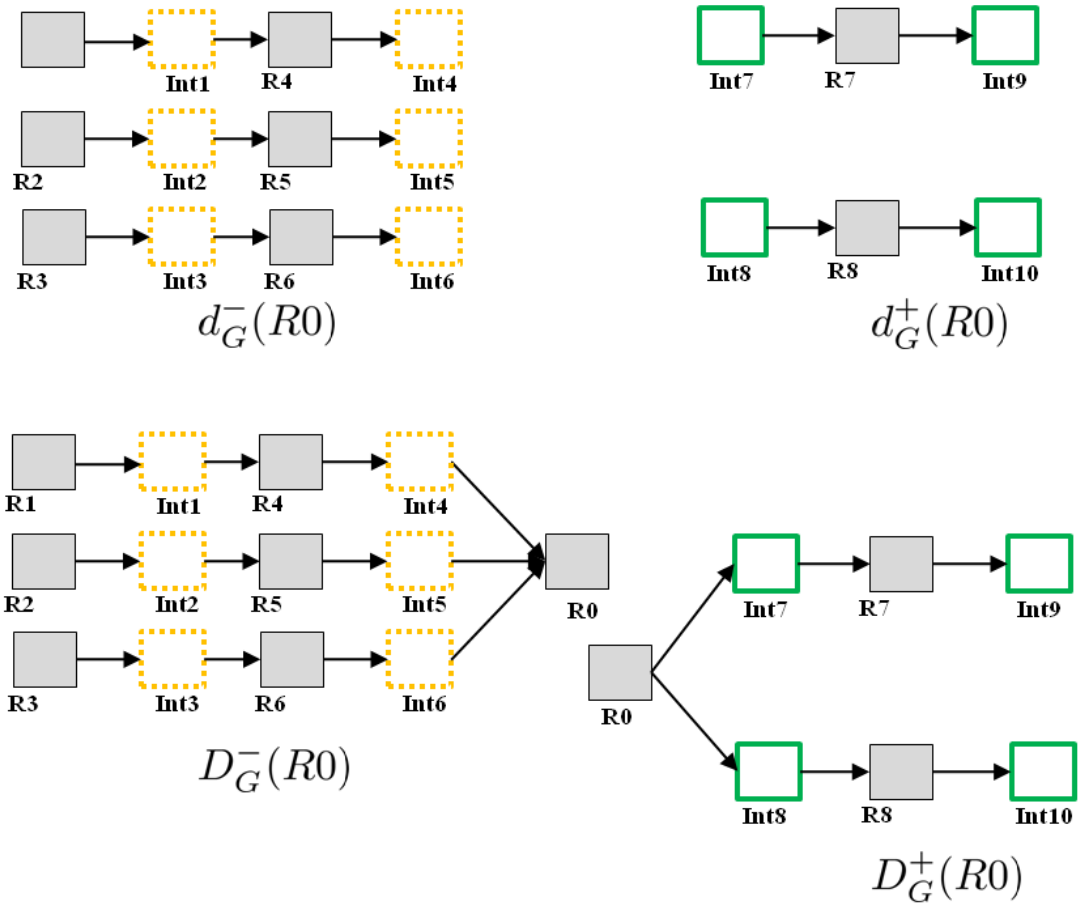

FIGURE 4.7: Graphes obtenus : $d_G^-(R0), d_G^+(R0), D_G^-(R0), D_G^+(R0)$

Definition 21 (But)
Une ressource $r \in \mathcal{R}$ est un but de G, si et seulement si :
- $rdep_G^+(r) = \emptyset$;
- par convention $\exists Int \in \mathcal{I}$ telle que $r \rightarrow_G Int$;

On note $goals(G)$ l'ensemble des buts de G. □

Definition 22 (Ressources amont)
Soit S un sous-ensemble de ressources de $\mathcal{R}$. L'ensemble $upstream_G(S)$ des ressources amont de S dans G est le sous-ensemble des ressources de S ne dépendant d'aucune autre ressource de S, ou encore les ressources de S les plus en amont dans le graphe de dépendances G :
$upstream_G(S) = \{r_i \in S \text{ telle que } \nexists r_k \in S, r_k \rhd_G r_i\}$. □

Exemple : sur la figure 4.6, avec $S = \{R0, R1, R2, R3, R4, R5, R6, R7, R8\}$,
$upstream_G(S) = \{R1, R2, R3\}$.

2 Résolution de conflits d'autorité

2.1 Principe de la détection et de la résolution de conflit

Nous avons vu au paragraphe 1 qu'il existe deux types de conflits : $\text{Conflict}_{\text{Destruction}}$ et $\text{Conflict}_{\text{Preemption}}$, qui peuvent se produire au sein d'une ou plusieurs ressources du graphe de ressources.

Le rôle du contrôleur de la dynamique de l'autorité, fonction de l'agent *robot*, est de détecter les conflits puis de restaurer la cohérence au sein du macro-modèle de l'autorité $\mathcal{V} = \{G, \mathcal{C}, \phi, a\}$, avec $G = \{\mathcal{R}, \mathcal{I}\}$, en réalisant les opérations suivantes :

- Surveillance : observation du marquage de chacune des ressources du graphe de ressources $r_i \in \mathcal{R}$, pour détecter l'apparition d'un des deux types de conflit au sein de l'une ou plusieurs d'entre elles.
- Détection du conflit : $\text{Conflict} \subseteq \text{M}_{\text{r}_{\text{conflict}}}$ pour au moins une ressource $r_{conflict} \in \mathcal{R}$.
- Analyse du conflit : identification du sous-graphe de ressources affecté par le conflit, $\mathcal{D}_G^+(r_{conflict})$.
- Requête à la planification : sur demande du contrôleur de la dynamique de l'autorité, un plan est recherché pour restaurer la cohérence dans le graphe de ressources, tout en conservant les buts poursuivis jusqu'alors.
- Identification des transformations nécessaires pour passer du macro-modèle de l'autorité $\mathcal{V}$ comprenant un conflit à un nouveau macro-modèle sans conflit. En effet, le contrôleur de la dynamique de l'autorité ne crée pas un nouveau macro-modèle de l'autorité $\mathcal{V}'$ *ex nihilo*, mais transforme le macro-modèle initial $\mathcal{V}$, en en conservant un certain nombre d'éléments.

On cherche à obtenir $\mathcal{V}' = \{G', \mathcal{C}', \phi', a'\}$ un nouveau macro-modèle de l'autorité, avec $G' = \{\mathcal{R}', \mathcal{I}'\}$, tel que :

- $goals(G') = goals(G)$: les buts de G sont conservés par G' ;
- $\mathcal{C}' = \mathcal{C}$: les agents intervenant dans la modélisation sont inchangés ;
- $conflict(R') = \emptyset$: aucune des ressources de $\mathcal{R}'$ n'est en état de conflit.

Il est du ressort du planificateur de fournir les transformations nécessaires pour passer de $\mathcal{V}$ à $\mathcal{V}'$, en fonction des considérations sur l'autorité dont il faut tenir compte. Cela implique que le planificateur fasse des modifications parmi les suivantes :

- enlever ou rajouter des ressources, c'est-à-dire modifier $\mathcal{R}$;
- enlever, ajouter ou modifier des interfaces, c'est-à-dire modifier $\mathcal{I}$ ainsi que ϕ ;
- optionnellement, modifier les relations d'autorité sur les ressources présentes dans G, c'est-à-dire modifier a.

Cependant, une considération est à prendre en compte : le planificateur est une fonction de l'agent *robot*. En ce sens, toutes ses actions portent le sceau de l'agent *robot* (ou en termes de réseaux de Petri, sa couleur, *robot*) et sont de ce fait conditionnées par les relations d'autorité entre l'agent *robot* et l'opérateur (*operator*) pour chacune des ressources de $\mathcal{V}$. Ainsi, s'il crée un nouveau graphe de ressources $G' = \{\mathcal{R}', \mathcal{I}'\}$, toutes les interfaces en faisant partie porteront la couleur de l'agent *robot*, c'est-à-dire que $\forall i_{r_j r_k} \in \mathcal{I}', \phi'(i_{r_j r_k}) = robot$. Dans la même idée, le planificateur ne devrait pas pouvoir modifier l'état d'une requête sur une interface si celle-ci n'a pas été créée par l'agent *robot*.

Cependant, dans le cas où une situation de conflit s'est produite, le planificateur doit pouvoir restaurer la cohérence du modèle de l'autorité, y compris dans le cas où un agent externe (tel que l'opérateur humain) *n'est lui-même pas cohérent* dans ses ordres et actions sur le système. C'est pour cela que nous ajoutons les droits suivants au planificateur avec leur justification :

- dans le cas où l'autorité du robot est supérieure à celle de l'opérateur sur une ressource r, c'est-à-dire si
  $a_{<robot,operator>}(r) = (Preemptability, NoAccess)$ ou
  $a_{<robot,operator>}(r) = (Preemptability, Access)$,
  le planificateur peut, dans le but unique de restaurer la cohérence du système, sur une interface Int créée par l'opérateur (soit $\phi(Int) = operator$) et requérant la ressource r, faire repasser une requête de l'état *Satisfying Request* à l'état *Ri requested by Rj via* le tir de la transition $t_{Requestpostponed}$ sur Int. Ceci est nécessaire dans le cas où une requête de l'opérateur en cours de satisfaction doit être décalée dans le temps à la suite d'un conflit de préemption ou de défaillance. Toutefois, la requête de l'opérateur est conservée et pourra être satisfaite dès que r sera à nouveau disponible.
- dans le cas d'une autorité forte du robot relativement à l'opérateur sur la ressource r, c'est-à-dire si
  $a_{<robot,operator>}(r) = (Preemptability, NoAccess)$,
  le planificateur peut « forcer » l'opérateur à abandonner sa requête sur une interface Int requérant la ressource r par le tir de la transition $t_{Requestcancelled}$; si la requête était en cours de satisfaction, cela signifie en d'autres termes que l'interface doit « relâcher » cette ressource. Ce cas est légitime dans le sens où, d'après la relation d'autorité, l'opérateur ne doit plus avoir accès à r : ainsi, toutes ses requêtes sur cette ressource n'ont plus de raison d'exister.

Definition 23 (Interface décalable et annulable)
Si pour une ressource $r \in \mathcal{R}$, associée à la relation d'autorité $a_{<robot,operator>}(r)$, et assignée par l'interface Int (c'est-à-dire $r \rightarrow_G Int$) :

- $a_{<robot,operator>}(r) = (Preemptability, NoAccess)$ et $\phi(Int) = operator$,
  ou $a_{<robot,operator>}(r) = (Preemptability, Access)$ et $\phi(Int) = operator$,
  alors, Int est dite *décalable* par *robot* dans G.
- $a_{<robot,operator>}(r) = (Preemptability, NoAccess)$ et $\phi(Int) = operator$, alors, Int est dite *annulable* par *robot* dans G. □

Nous nous intéressons dans un premier temps au cas où le planificateur n'est pas autorisé à modifier les relations d'autorité existantes.

2.2 Procédure de résolution de conflit sans changement d'autorité

Nous nous plaçons dans le cas où les contraintes $goals(G') = goals(G)$ et $a' = a$ doivent être respectées. Nous allons dérouler la procédure de résolution sur les deux types de conflits existants, à savoir

- le conflit de destruction ;
- le conflit de préemption.

2.2.1 Résolution du conflit de destruction

Nous développerons la procédure de résolution de conflit de destruction en nous basant sur l'exemple développé précédemment en section 1.2, en repartant de la situation telle que présentée figure 4.3.

La résolution de conflit consiste en la réalisation successive des étapes suivantes :

- identification du sous-graphe de ressources affecté par le conflit ;
- identification des buts affectés ;
- identification des sous-buts à replanifier ;
- recherche de graphes de ressources solutions et sélection ;
- détermination des transformations à effectuer pour passer de G à G'.

Voici maintenant le détail de chacune de ces opérations.

Identification du sous-graphe de ressources affecté par le conflit

Nous supposons qu'un conflit de destruction concerne une seule ressource à la fois : la ressource est brutalement détruite alors qu'elle est affectée à une ressource dépendante. Nous nommons $r_{conflict}$ cette ressource détruite.
Soit $G_{conflict}$ le sous-graphe de G affecté par le conflit sur la ressource $r_{conflict}$. On a $G_{conflict} = D_G^+(r_{conflict})$.

Exemple (cf figure 4.8) :
$G_{conflict} = D_G^+(r_{Energy}) = (\{r_{Energy}, r_{NavigationRobot}, r_{WP-Goal}\}, \{Int2, Int4, Int5\})$.

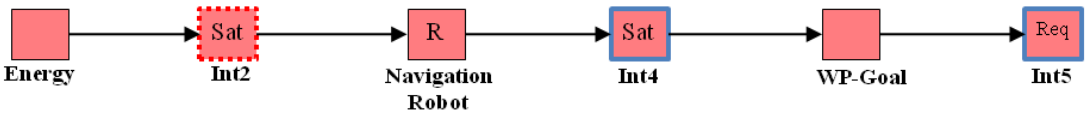

FIGURE 4.8: Graphe de ressources affecté par le conflit $G_{conflict}$

Identification des buts affectés

Soit $goals(G_{conflict})$ l'ensemble des buts de G affectés par le conflit et qui ne pourront donc pas être atteints. La restauration de la cohérence au sein du macro-modèle de l'autorité est contrainte par le fait que ces buts soient toujours présents dans le nouveau macro-modèle de l'autorité, puisque nous avons supposé $goals(G') = goals(G)$.

Exemple (cf figure 4.8) : $goals(G_{conflict}) = \{r_{WP-Goal}\}$.

Identification des sous-buts à replanifier

Il se peut que les ressources à conserver pour la replanification soient des sous-buts intermédiaires. En effet, si une ressource r_i de G a été *assignée* par l'agent *operator* à une autre ressource r_j, le planificateur (agissant sous le sceau de *robot*) doit, dans la mesure du possible et en fonction des relations d'autorité concernant la ressource r_i, conserver cette assignation décidée par l'opérateur.

S'il existe une de ces ressources r_i en amont d'un but r_g dans le graphe de ressources, c'est-à-dire si r_i est *requise* pour atteindre le but, alors c'est ce sous-but qui sera transmis au planificateur en lieu et place du but. La ressource r_i devient alors un sous-but à atteindre et se substitue pour le planificateur au but initial. La ressource r_i et la partie de graphe en aval de r_i (c'est-à-dire $D_G^+(r_i)$) doivent être conservées dans G'.

Pour une ressource $r_g \in goals(G_{conflict})$ un but affecté par le conflit, avec $G_{conflict} = \{\mathcal{R}_{conflict}, \mathcal{I}_{conflict}\}$,
soit $operatorAssigned(G_{conflict})$ l'ensemble des ressources assignées par l'opérateur dans $G_{conflict}$ telles que :
$operatorAssigned(G_{conflict}) = \{r_i \in \mathcal{R}_{conflict}$ telle que $\exists Int \in \mathcal{I}_{conflict}$ avec $r_i \rightarrow_{G_{conflict}} Int$ et $\phi(Int) = operator\}$.

Exemple (cf figure 4.8) : $operatorAssigned(G_{conflict}) = \{r_{NavigationRobot}, r_{WP-Goal}\}$.

On distingue :
- $cancelableRsc(G_{conflict}) = \{r_i \in operatorAssigned(G_{conflict})$ telle que $\exists Int \in \mathcal{I}_{conflict}$ avec $r_i \rightarrow_{G_{conflict}} Int$ et Int annulable par $robot\}$, c'est-à-dire l'ensemble des ressources assignées par l'opérateur dans $G_{conflict}$, dont l'assignation est annulable et décalable par le planificateur du robot ;
- $postponableRsc(G_{conflict}) = \{r_i \in operatorAssigned(G_{conflict})$ telle que $\exists Int \in \mathcal{I}_{conflict}$ avec $r_i \rightarrow_{G_{conflict}} Int$ et Int décalable par $robot\}$, c'est-à-dire l'ensemble des ressources assignées par l'opérateur dans $G_{conflict}$, dont l'assignation est non-annulable mais décalable par le planificateur du robot ;

- $unchangeableRsc(G_{conflict}) = \{r_i \in operatorAssigned(G_{conflict}), \nexists Int \in \mathcal{I}_{conflict}$ telle que $(r_i \rightarrow_{G_{conflict}} Int$ et Int décalable par $robot)\}$, c'est-à-dire l'ensemble des ressources assignées par l'opérateur dans $G_{conflict}$, dont l'assignation est non annulable, non-décalable par le planificateur du robot.

Ainsi,

$$operatorAssigned(G_{conflict}) = cancelableRsc(G_{conflict}) \cup postponableRsc(G_{conflict}) \cup unchangeableRsc(G_{conflict})$$

Exemple (cf figure 4.8) : Considérons que les relations d'autorité sur les ressources de la figure 4.8 sont les suivantes :

- $a_{<robot,operator>}(r_{Energy}) = (Access, Access)$;
- $a_{<robot,operator>}(r_{NavigationRobot}) = (Preemptability, Access)$;
- $a_{<robot,operator>}(r_{WP-Goal}) = (Access, Preemptability)$.

On a alors : $cancelableRsc(G_{conflict}) = \emptyset$, $postponableRsc(G_{conflict}) = \{r_{NavigationRobot}\}$, et $unchangeableRsc(G_{conflict}) = \{r_{WP-Goal}\}$.

En conséquence, on pose :

Definition 24 (Sous-buts)
Pour un but $r_g \in goals(G_{conflict})$, on définit l'ensemble des ressources qui sont des sous-buts assignés par l'opérateur pour atteindre r_g :
$subgoal(r_g) = upstream\left(\{r_g\} \cup operatorAssigned(D^-_{G_{conflict}}(r_g))\right)$
Remarque : s'il n'y a aucune ressource assignée par l'opérateur dans $(D^-_{G_{conflict}}(r_g))$, alors $subgoal(r_g) = r_g$. □

De toutes les ressources du graphe $D^-_{G_{conflict}}(r_g)$, on identifie les ressources assignées par l'opérateur les plus en amont du graphe de dépendance, c'est-à-dire ne dépendant d'aucune autre ressource de $operatorAssigned(D^-_{G_{conflict}}(r_g))$; s'il n'y en a pas, on conserve alors r_g.

La figure 4.9 illustre le principe d'identification du sous-but pour le but r_g sur un graphe de ressource $G_{conflict}$, à partir d'une ressource en conflit r_1.

Le sous-graphe de ressources $D^-_{G_{conflict}}(r_g)$ est encadré en pointillé, on ne s'intéresse qu'au but r_g. L'interface $Int1$ a été créée par $robot$ (trait rouge pointillé), les interfaces $Int2$, $Int3$ et $Int4$ ont été créées par $operator$. Ainsi, r_2, r_3 et r_g ont été assignées par l'opérateur : $operatorAssigned(D^-_{G_{conflict}}(r_g)) = \{r_2, r_3, r_g\}$. Le sous-but retenu est la ressource la plus en amont de ce graphe de ressources, qui est ici r_2.

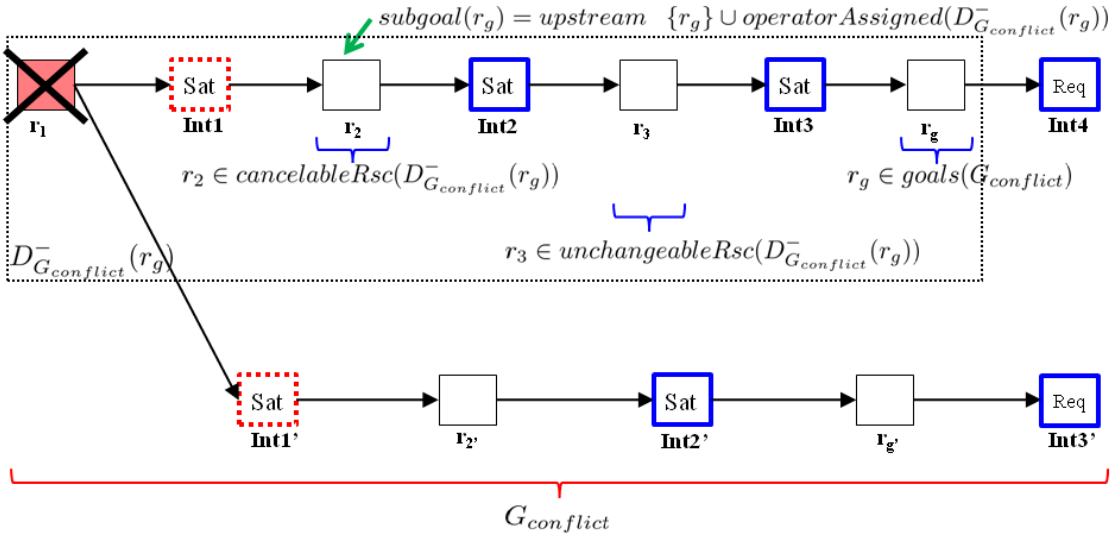

FIGURE 4.9: Identification de $subgoal(r_g)$.

Finalement, on crée $plannerGoals(G_{conflict})$, l'ensemble des ressources identifiées comme sous-buts constituant la consigne d'entrée du planificateur :
$plannerGoals(G_{conflict}) = \bigcup_{r_g \in goals(G_{conflict})} subgoals(r_g)$.

Toutefois, on remarque qu'ici sont incluses les ressources totalement modifiables par le planificateur $cancelableRsc(G_{conflict})$, ceci afin de respecter au plus près les assignations (et donc les requêtes) de l'opérateur. Ceci constitue cependant une contrainte qui peut être relaxée au vu des relations d'autorité. C'est pourquoi, si le planificateur ne retourne pas de solution, on retire (si applicable) de $subgoal(r_g)$ les sous-buts annulables afin d'élargir son espace de recherche. On pose alors :

$$subgoal(r_g) = upstream\left(\{r_g\} \cup postponableRsc(D^-_{G_{conflict}}(r_g)) \cup unchangeableRsc(D^-_{G_{conflict}}(r_g))\right).$$

La figure 4.10 illustre cette deuxième approche sur le même exemple que précédemment. On peut constater que sur ce graphe de ressources, ce n'est plus la ressource r_2 qui est retenue comme sous-but mais r_3.

Si le planificateur ne retourne toujours pas de solution, rien d'autre n'est possible en l'état sans modification des relations d'autorité existantes.

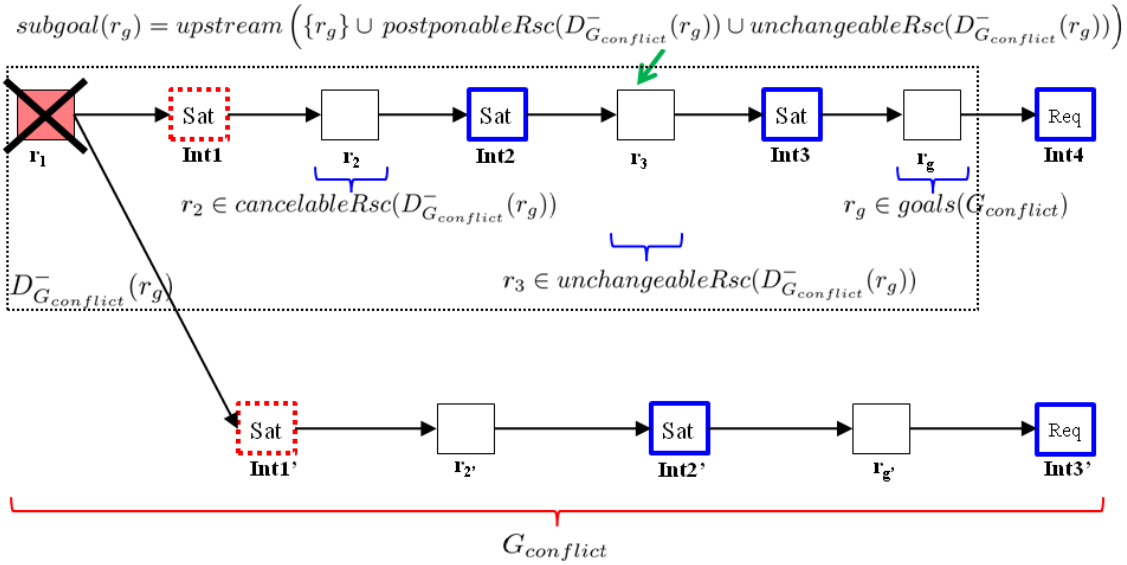

FIGURE 4.10: Identification de $subgoal(r_g)$ en relâchant la contrainte de respecter au plus près les assignations de l'opérateur.

Exemple (cf figure 4.8) : $plannerGoals(G_{conflict}) = \{r_{NavigationRobot}\}$.

Remarque : si $plannerGoals(G_{conflict}) = r_{conflict}$, aucune solution ne pourra être renvoyée par le planificateur : cela signifie que $r_{conflict}$ a été assignée directement par l'opérateur et que, en l'état actuel des relations d'autorité, le robot n'a pas l'autorité nécessaire pour modifier ou contourner cette assignation sur la ressource défaillante.

Les parties du graphe de ressource $G_{conflict}$ en aval des sous-buts fournis au planificateur ne doivent pas modifiées dans le nouveau macro-modèle de l'autorité :

On définit ainsi
$$G_{conflictingPartToKeep} = \bigcup_{r_k \in plannerGoals(G_{conflict})} \{D^+_{G_{conflict}}(r_k)\ \}$$

Ce qui précède r_k peut être modifié par le planificateur, r_k et ce qui suit r_k doit être gardé en l'état (voir figure 4.13).

Exemple (cf figure 4.8) :
$G_{conflictingPartToKeep} = (\{r_{NavigationRobot}, r_{WP-Goal}\}, \{Int4, Int5\})$.

Recherche de graphes de ressources solutions et sélection

Les paramètres fournis en entrée du planificateur sont les suivants : $\{\mathcal{V}, plannerGoals(G_{conflict})\}$, c'est-à-dire l'ensemble des paramètres de la situation courante (état des ressources, état des interfaces, relations d'autorité) et les sous-buts que le planificateur doit considérer afin de trouver un ensemble cohérent d'actions pour les satisfaire.

Le graphe de ressource $G_{solving}$ issu de la solution du planificateur (s'il en existe) est tel que : $\forall r_{subgoal} \in plannerGoals(G_{conflict}), rdep^{-}_{G_{solving}}(r_{subgoal}) \neq \emptyset$, c'est-à-dire que tout sous-but passé au planificateur doit avoir des ressources mobilisées pour sa production lors de l'exécution nominale du nouveau plan. Un sous-but qui ne satisferait pas cette condition serait un but *non poursuivi*.

Lorsque plusieurs solutions sont disponibles, le planificateur les évalue et en sélectionne une en fonction de critères prédéterminés endogènes (par exemple minimiser le nombre de transformations nécessaires pour passer du macro-modèle de l'autorité initiale avec conflit au nouveau macro-modèle sans conflit) ou exogènes (par exemple, utilisation de métriques de performance sur les plans solutions).

Si aucun plan n'est trouvé, comme expliqué auparavant le planificateur peut être éventuellement relancé sur un espace de recherche élargi, en fonction des relations d'autorité (voir précédemment).

Exemple (voir figure 4.11) :
On suppose qu'il existe une source auxiliaire d'énergie disponible. On a alors :
$G_{solving} = (\{r_{Position}, r_{AuxiliaryEnergy}, r_{SteeringWheel}, r_{NavigationRobot}\}, \{Int1, Int6, Int3\})$,

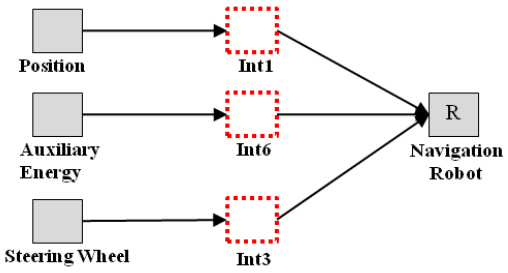

FIGURE 4.11: Graphe de ressources solution $G_{solving}$ pour le sous-but $r_{NavigationRobot}$.

Détermination des transformations à effectuer pour passer de G à G'

Dans le cas du conflit de destruction, pour passer du graphe de ressources en état de conflit à un graphe cohérent, il faut identifier les ressources et interfaces à ajouter, ainsi que les ressources et interfaces à supprimer :

Sous-graphe à ajouter
$G_{toAdd} = G_{solving} \setminus (G_{solving} \cap G)$,
soit tout simplement le graphe solution retourné par le planificateur auquel on retire les ressources et interfaces existant déjà dans le graphe de ressources courant.

Exemple : $G_{toAdd} = (\{r_{AuxiliaryEnergy}\}, \{Int6\})$.

Sous-graphe à retirer (voir figure 4.13) :
Afin de supprimer toutes les ressources inutiles dans le nouveau macro-modèle de l'autorité, on réalise l'inventaire de toutes les ressources *strictement nécessaires* à la réalisation des buts conservés. Toute ressource ne faisant pas partie de cet inventaire doit être retirée.
Soit $H = goals(G) - goals(G_{conflict})$ l'ensemble des buts poursuivis dans le graphe de ressource initial privé des buts concernés par le conflit.

Exemple : $H = \emptyset$; dans ce graphe de ressources, un seul but était poursuivi : $r_{WP-Goal}$. C'est ce but qui est affecté par le conflit.

Pour un but $r_g \in goals(G)$, on désigne par $Int_{request}(r_g)$ l'interface représentant la requête, formulée par l'un des agents, de la poursuite de ce but telle que $r_g \rightarrow_G Int_{request}(r_g)$. On note alors $goalRequest_G(r_g) = (\emptyset, \{Int_{request}(r_g)\})$ le graphe ne comprenant que cette interface.

Soit $G_{necessary} = \bigcup_{r_g \in H} \{D_G^-(r_g) \cup goalRequest_G(r_g)\}$ le graphe de ressource comprenant uniquement les ressources (et interfaces) requises pour réaliser l'ensemble des buts de H.

Exemple : $G_{necessary} = (\emptyset, \emptyset)$.

Ainsi, $\mathcal{G}_{toRemove} = G \setminus (G_{necessary} \cup G_{conflictingPartToKeep} \cup G_{solving})$,
ou autrement dit, tout ce qui n'est pas strictement nécessaire ou à conserver est à retirer du modèle initial. Ceci permet d'enlever les ressources résiduelles devenues inutiles suite à la restauration de la cohérence.

Exemple : $G_{toRemove} = (\{r_{Energy}\}, \{Int2\})$.

Enfin, on peut écrire : $G' = G \cup G_{toAdd} \setminus G_{toRemove}$,
ou encore $G' = G_{necessary} \cup G_{solving} \cup G_{conflictingPartToKeep}$.

Le résultat de ces opérations est illustré sur la figure 4.12 :

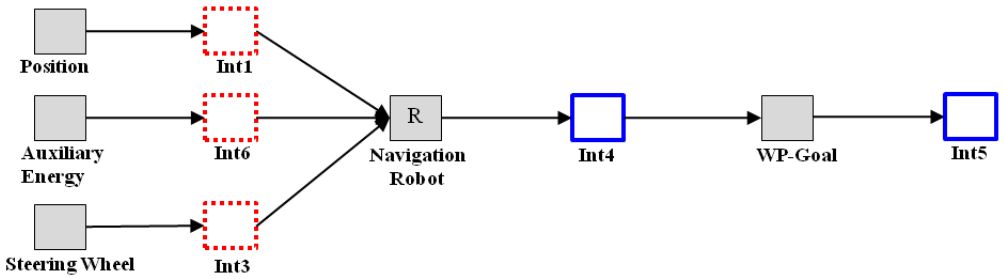

FIGURE 4.12: Graphe de ressources G' solution au conflit de destruction.

La figure 4.13 illustre, sur un exemple « abstrait », l'ensemble des différents sous-graphes utilisés lors de la résolution, à l'exception de $G_{solving}$ qui se substitue à $G_{toRemove}$. Les ensembles $goals(G_{conflict})$, $subgoalr_g$, $subgoalr'_g$ et H ont également été représentés.

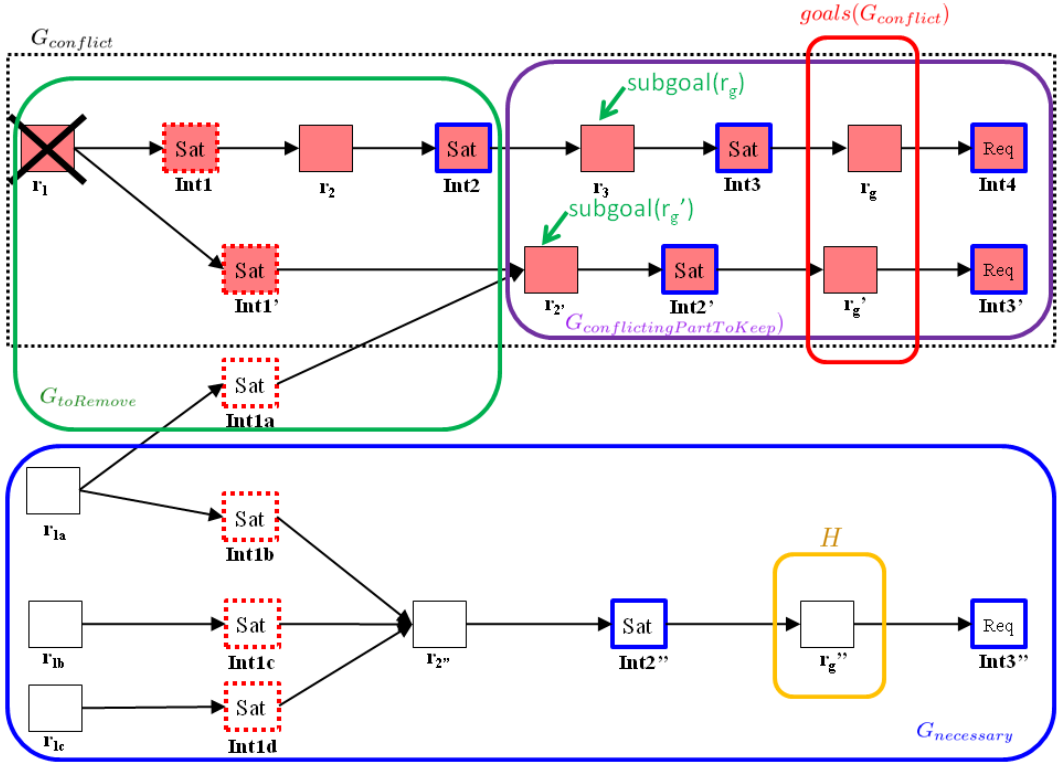

FIGURE 4.13: Ensemble des sous-graphes utilisés pour la résolution de conflit.

Le sous-graphe $G_{conflict}$ est encadré en trait pointillé, alors que les autres sous-graphes sont en trait continu : $G_{conflictingPartToKeep}$, $G_{necessary}$, $G_{toRemove}$.

2.2.2 Résolution du conflit de préemption

Nous développerons la procédure de résolution de conflit de préemption en nous basant sur l'exemple précédemment développé en section 1.3, en repartant de la situation telle que présentée figure 4.14. Le cadre posé pour la résolution du conflit de préemption est défini par les contraintes $goals(G') = goals(G)$ et $a' = a$.

La résolution de conflit consiste en la réalisation successive des étapes suivantes :
- identification du sous-graphe de ressources affecté par le conflit ;
- identification des buts affectés ;
- identification des sous-buts à replanifier ;
- recherche de graphes de ressources solutions et sélection ;
- détermination des transformations à effectuer pour passer de G à G'.

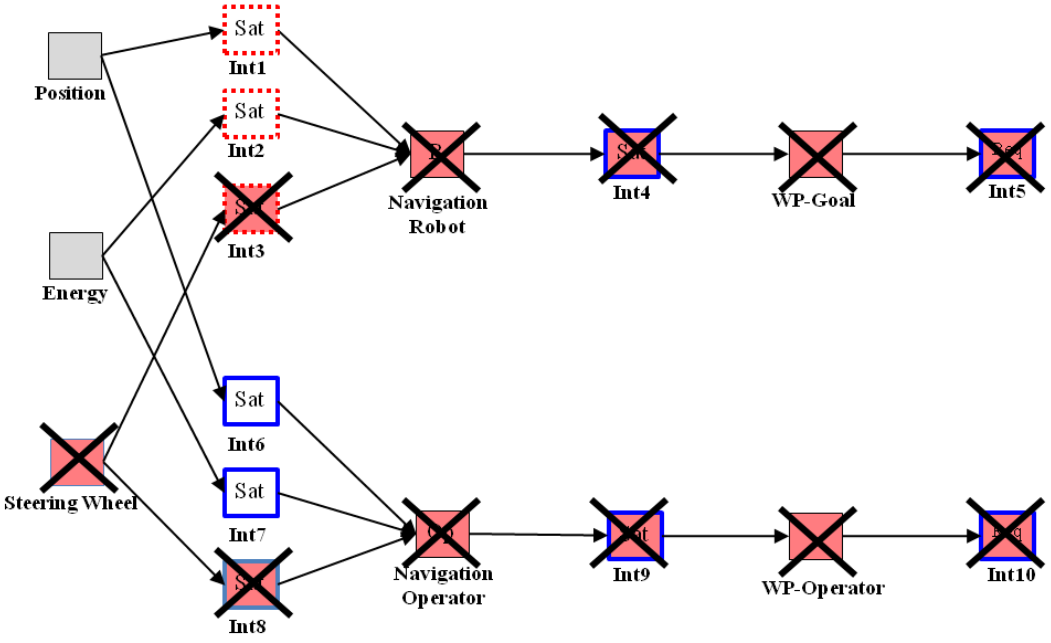

FIGURE 4.14: Graphe de ressources avec conflit de préemption

Ces étapes sont exactement les mêmes que lors de la résolution d'un conflit de destruction. Cependant, la différence se situe dans l'identification des sous-buts pour lesquels il s'agit de trouver un plan alternatif. Développons à nouveau les points précédents.

Identification du graphe de ressources affecté par l'apparition du conflit

Dans le cas d'un conflit de préemption, une ou plusieurs ressources peuvent être dans un état incohérent à la fois, les ressources non-partageables déjà affectées une première fois et étant réaffectées une deuxième fois par préemption.

Soit $G_{conflict} = D_G^+(r_{conflict})$ le sous-graphe de G affecté par le conflit sur la ressource $r_{conflict}$.

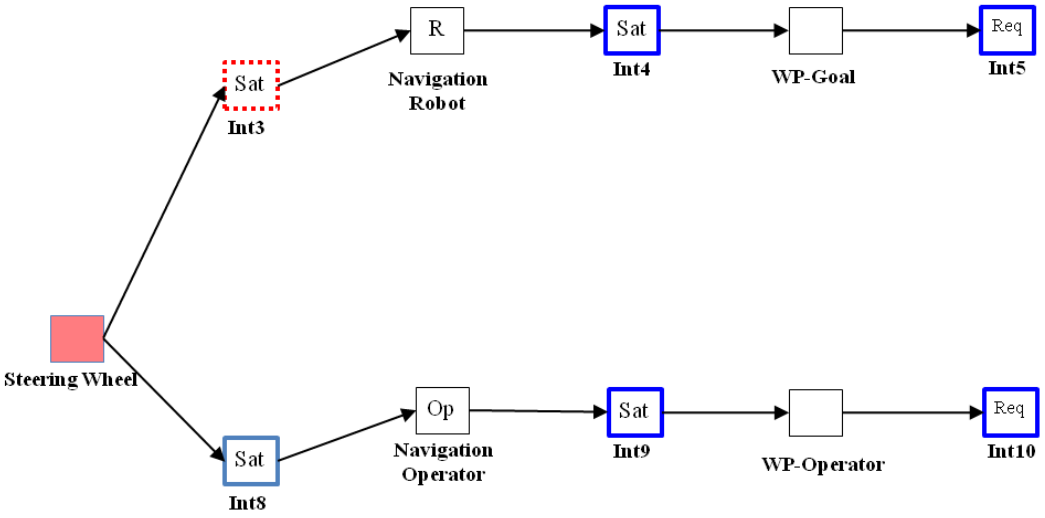

FIGURE 4.15: Graphe de ressources $G_{conflict}$ affecté par le conflit

Exemple (cf figure 4.15) :

$$\begin{aligned} G_{conflict} &= D_G^+(r_{SteeringWheel}) \\ &= (\{r_{SteeringWheel}, r_{NavigationRobot}, r_{WP-Goal}, r_{NavigationOperator}, r_{WP-Operator}\}, \\ &\qquad \{Int3, Int4, Int5, Int8, Int9, Int10\}). \end{aligned}$$

Identification des buts affectés

Soit $goals(G_{conflict})$ l'ensemble des buts de G affectés par le conflit et qui ne pourront donc être atteints en l'état du graphe de ressources. Ces buts doivent être toujours présents dans le nouveau macro-modèle de l'autorité, puisque nous avons supposé $goals(G') = goals(G)$.

Exemple : $goals(G_{conflict}) = \{r_{WP-Goal}, r_{WP-Operator}\}$.

Identification des sous-buts à replanifier

Il se peut que les ressources à conserver pour la replanification soient des sous-buts intermédiaires. En effet, si une ressource r_i de G a été *assignée* par l'agent *operator* à une autre ressource r_j, le planificateur (agissant sous le sceau de *robot*) doit, dans la mesure du possible et en fonction des relations d'autorité concernant la ressource r_i, conserver cette assignation décidée par l'opérateur.

Soit $operatorAssigned(G_{conflict})$ l'ensemble des ressources de $G_{conflict}$ assignées par l'opérateur dans $G_{conflict}$.

Exemple (cf figure 4.15) :

$$operatorAssigned(G_{conflict}) = \{r_{SteeringWheel}, r_{NavigationOperator}, r_{WP-Operator}, r_{NavigationRobot}, r_{WP-Goal}\}.$$

Il convient de définir les sous-buts à fournir en entrée au planificateur pour la recherche de plans solutions au conflit.

À nouveau, dans un premier temps, on définit ainsi $subgoal(r_g)$ pour un but $r_g \in goals(G_{conflict})$:
$subgoal(r_g) = upstream\left(\{r_g\} \cup operatorAssigned(D^-_{G_{conflict}}(r_g))\right)$.

Exemple :

Considérons que les relations d'autorité sont les suivantes :
- $a_{<robot,operator>}(r_{SteeringWheel}) = (Access, Access)$;
- $a_{<robot,operator>}(r_{NavigationRobot}) = (Preemptability, Access)$;
- $a_{<robot,operator>}(r_{NavigationOperator}) = (NoAccess, Preemptability)$.
- $a_{<robot,operator>}(r_{WP-Goal}) = (Access, Preemptability)$.
- $a_{<robot,operator>}(r_{WP-Operator}) = (Access, Preemptability)$.

On a : $goals(G_{conflict}) = \{r_{WP-Goal}, r_{WP-Operator}\}$.

Prenons le premier but, $r_{WP-Goal}$; le graphe dont dépend ce but est (voir figure 4.16) : $D^-_{G_{conflict}}(r_{WP-Goal}) = (\{r_{SteeringWheel}, r_{NavigationRobot}, r_{WP-Goal}\}, \{Int3, Int4\})$.

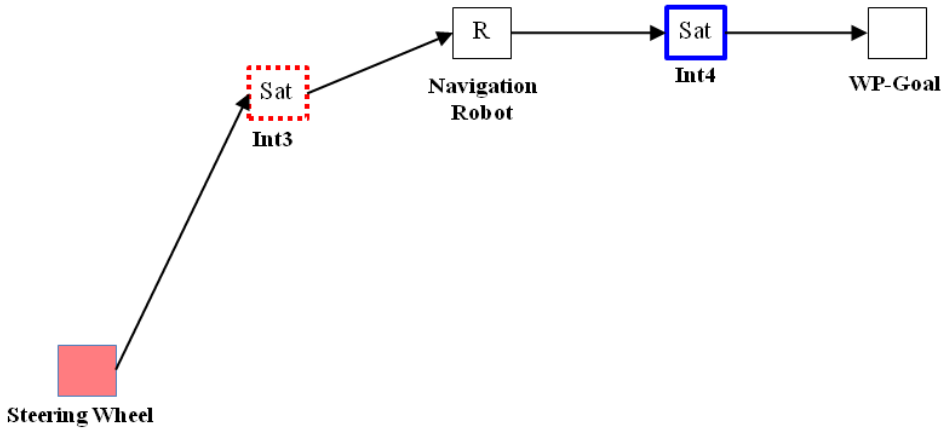

FIGURE 4.16: Graphe de ressources $D^-_{G_{conflict}}(r_{WP-Goal})$.

Identifions alors :
- $cancelableRsc(D^-_{G_{conflict}}(r_{WP-Goal})) = \emptyset$;
- $postponableRsc(D^-_{G_{conflict}}(r_{WP-Goal})) = \{r_{NavigationRobot}\}$;
- $unchangeableRsc(D^-_{G_{conflict}}(r_{WP-Goal})) = \emptyset$.

On a alors :

$$subgoal(r_{WP-Goal}) = upstream\,(\{r_{WP-Goal}, r_{NavigationRobot}\})$$
$$= \{r_{NavigationRobot}\}$$

Considérons maintenant le second but, $r_{WP-Operator}$; le graphe dont dépend ce but est (voir figure 4.17) :

$$D^-_{G_{conflict}}(r_{WP-Operator}) = (\{r_{SteeringWheel}, r_{NavigationOperator}, r_{WP-Operator}\}, \{Int8, Int9\}).$$

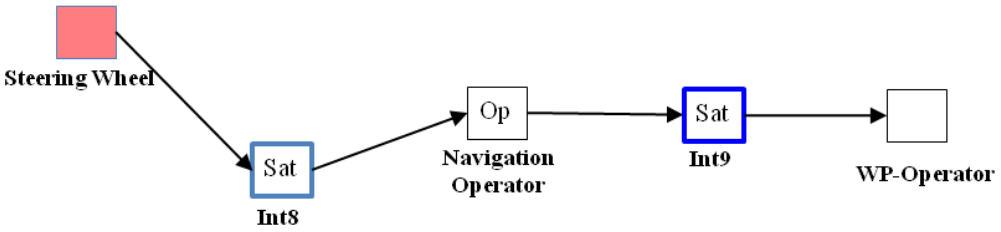

FIGURE 4.17: Graphe de ressources $D^-_{G_{conflict}}(r_{WP-Operator})$.

Identifions alors :
- $cancelableRsc(D^-_{G_{conflict}}(r_{WP-Operator})) = \emptyset$;
- $postponableRsc(D^-_{G_{conflict}}(r_{WP-Operator})) = \emptyset$;
- $unchangeableRsc(D^-_{G_{conflict}}(r_{WP-Operator})) = \{r_{SteeringWheel}, r_{NavigationOperator}\}$.

On a alors :

$$subgoal(r_{WP-Operator}) = upstream\left(\{r_{WP-Operator}, r_{SteeringWheel}, r_{NavigationOperator}\}\right)$$
$$= \{r_{SteeringWheel}\}$$

Ainsi, finalement, $plannerGoals(G_{conflict}) = \{r_{NavigationRobot}, r_{SteeringWheel}\}$.

Remarques :
- $r_{SteeringWheel}$ et $r_{NavigationOperator}$ sont les deux sous-buts fournis au planificateur pour sa recherche ; or, $\phi(r_{SteeringWheel}) = operator$ et $\phi(r_{NavigationOperator}) = operator$: ces deux sous-buts ont été assignés par l'opérateur humain, ce qui signifie une contradiction de la part de celui-ci : en prenant le contrôle manuel de la navigation, il contredit lui-même son ordre précédent demandant au robot de réaliser la tâche de navigation automatique jusqu'au waypoint « WP-Goal ». Le planificateur doit ici démêler une situation incohérente engendrée par l'opérateur humain uniquement.
- la ressource en conflit $r_{SteeringWheel}$ fait elle-même partie des sous-buts à conserver fournis au planificateur : en effet, l'opérateur a spécifiquement indiqué qu'il souhaite utiliser cette ressource à ses fins et le robot n'a pas l'autorité nécessaire pour modifier cette assignation : le planificateur doit donc retourner un plan permettant d'obtenir un graphe solution $G_{solving}$ conservant cette ressource. Cependant, dans le cas du conflit de préemption, contrairement au cas du conflit de destruction, devoir conserver la ressource en conflit n'équivaut pas à une impossibilité de trouver un plan solution.

Concernant le dernier point, $r_{SteeringWheel}$ n'est requise que pour la satisfaction d'un seul des buts affectés par le conflit $goals(G_{conflict})$, puisque $subgoal(r_{WP-Operator}) = r_{SteeringWheel}$, c'est-à-dire que $r_{SteeringWheel}$ est imposée pour le but $r_{WP-Operator}$; ce n'est pas le cas pour le deuxième but $r_{WP-Goal}$. Le planificateur dispose donc potentiellement de la marge de manœuvre nécessaire pour parvenir à l'obtention d'un graphe de ressources solution : le principe consisterait à conserver en l'état le graphe correspondant au second but $D^-_{G_{conflict}}(r_{WP-Operator})$ (voir figure 4.17), et à conserver également le graphe du premier but $D^-_{G_{conflict}}(r_{WP-Goal})$ (voir figure 4.16) mais en « décalant » sa réalisation, de manière à ce que ces deux morceaux de plan s'exécutent *l'un après l'autre* et non simultanément (ce qui induit le conflit de préemption).

Le planificateur doit permettre d'établir les transformations à réaliser sur le macro-modèle de l'autorité original, en conflit, pour parvenir à un macro-modèle sans conflit. Cependant, avant même de lancer le planificateur, ce qui peut être coûteux en termes de calcul, il est possible de faire une vérification préalable pour s'assurer s'il est tout simplement réalisable pour le planificateur de trouver un plan solution, en considérant les relations d'autorité.

Vérification de la possibilité de modifier un graphe de ressources existant

On cherche sur la base de l'exemple, à déterminer si le graphe $D^-_G(r_{WP-Goal})$ peut être modifié. On remarque qu'on recherche les ressources et interfaces requises pour le but $r_{WP-Goal}$ sur le graphe G dans son ensemble, et non pas uniquement sur la partie affectée par le conflit $G_{conflict}$. Le graphe $D^-_G(r_{WP-Goal})$ est représenté sur la figure 4.18.

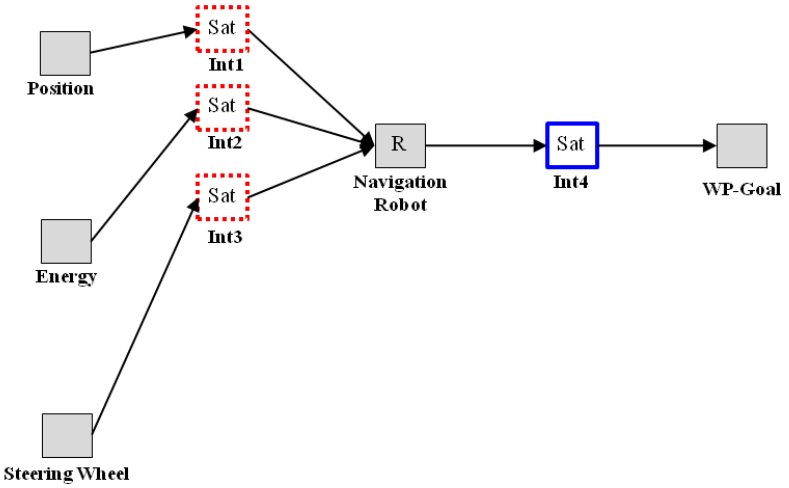

FIGURE 4.18: Graphe de ressources $D^-_G(r_{WP-Goal})$.

Lors de la préemption de la ressource $r_{SteeringWheel}$ induite par l'activation du lien de dépendance sur l'interface $Int8$, le robot réalisait une tâche de navigation (i.e dans l'état d'exécution du plan, les interfaces $Int1$, $Int2$, $Int3$ et $Int4$ sont en état *Satisfying Request*, indiqué par l'indicateur « Sat », voir figure 4.14).

On cherche à savoir si l'état de ces interfaces, dans leur ensemble, peut être modifié. Il faut établir si le plan concernant la poursuite du but $r_{WP-Goal}$, contenant partiellement des assignations de l'opérateur, peut être changé de manière à ce que la ressource $r_{SteeringWheel}$ soit relâchée, faisant ainsi disparaître le conflit. Cela signifie par exemple que la requête sur l'interface $Int3$ doit être décalée, mais pas seulement : la requête sur l'interface $Int4$ doit également être décalée. Or, $Int4$ a été assignée par l'opérateur.

Pour un but r_g, le principe de vérification est le suivant :
soit $satisfyingRequestInterfaces(D_G^-(r_g)) = \{Int \in \mathcal{I}_{D_G^-(r_g)}$ telle que $\{SatisfyingRequest\} \subseteq \mathrm{M_{Int}}\}$,
c'est-à-dire l'ensemble des interfaces sur le graphe $D_G^-(r_g)$ concernant le but r_g ; et $satisfyingRequestResources(D_G^-(r_g)) = \{r_i \in D_G^-(r_g)$ telle que $\exists Int \in satisfyingRequestInterfaces(D_G^-(r_g)), r_i \rightarrow_G Int\}$,
c'est-à-dire l'ensemble des ressources sur le graphe $D_G^-(r_g)$ et assignées par une interface de $satisfyingRequestInterfaces(D_G^-(r_{WP-Goal}))$.

La vérification consiste alors à s'assurer que :

$$satisfyingRequestResources(D_G^-(r_g)) \subseteq \left(robotAssigned(D_G^-(r_g)) \cup cancelableRsc(D_G^-(r_g)) \cup postponableRsc(D_G^-(r_g))\right),$$

soit que l'ensemble des ressources actuellement assignées et affectées dans le graphe initial G concernant le but r_g ne contient que des ressources, soit assignées par le robot, soit assignées par l'opérateur mais dont l'assignation peut être annulée ou décalée.

Cela revient encore à vérifier que :
$unchangeableRsc(D_G^-(r_g)) \cap satisfyingRequestResources(D_G^-(r_g)) = \emptyset$:
soit que sur l'ensemble des ressources actuellement assignées et affectées dans le sous-graphe G concernant le but r_g, aucune de ces ressources n'a une affectation qui n'est pas modifiable par le planificateur du robot.

Si cette condition est vérifiée, alors le planificateur a la possibilité de fournir un plan débouchant sur un graphe solution.

Exemple :
$satisfyingRequestInterfaces(D_G^-(r_{WP-Goal})) = \{Int1, Int2, Int3, Int4\}$;
$satisfyingRequestResources(D_G^-(r_{WP-Goal})) = \{r_{Position}, r_{Energy}, r_{SteeringWheel}, r_{NavigationRobot}\}$;
$robotAssigned(D_G^-(r_{WP-Goal})) = \{r_{Position}, r_{Energy}, r_{SteeringWheel}\}$;
$cancelableRsc(D_G^-(r_{WP-Goal})) = \emptyset$;
$postponableRsc(D_G^-(r_{WP-Goal})) = \{r_{NavigationRobot}\}$;
$unchangeableRsc(D_G^-(r_{WP-Goal})) = \emptyset$;

et on a bien :
$satisfyingRequestResources(D_G^-(r_{WP-Goal})) \subseteq$
$\big(robotAssigned(D_G^-(r_{WP-Goal})) \cup cancelableRsc(D_G^-(r_{WP-Goal}))$
$\cup postponableRsc(D_G^-(r_{WP-Goal}))\big)$;

soit
$\{r_{Position}, r_{Energy}, r_{SteeringWheel}, r_{NavigationRobot}\} \subseteq$
$(\{r_{Position}, r_{Energy}, r_{SteeringWheel}\} \cup \emptyset \cup \{r_{NavigationRobot}\})$;

ou encore
$unchangeableRsc(D_G^-(r_{WP-Goal})) \cap satisfyingRequestResources(D_G^-(r_{WP-Goal})) = \emptyset$.

On pose alors $\mathcal{I}_{postponed}$ l'ensemble des interfaces dont la requête doit être décalée, c'est-à-dire les interfaces dont il faudra tirer la transition $t_{Requestpostponed}$: $\mathcal{I}_{postponed} = satisfyingRequestInterfaces(D_G^-(r_{postponedGoal}))$,
soit l'ensemble des interfaces dont la requête est active sur le sous-graphe en conflit correspondant à la poursuite du but dont la poursuite est décalée.

Exemple :
$\mathcal{I}_{postponed} = satisfyingRequestInterfaces(D_G^-(r_{WP-Goal}))$

$= \{Int1, Int2, Int3, Int4\}$.

Recherche de solutions alternatives et identification du sous-graphe à conserver

Enfin, comme précédemment dans le cas du conflit de destruction, nous avons pris en considération les ressources totalement modifiables par le planificateur $cancelableRsc(G_{conflict})$, ceci afin de respecter au plus près les assignations (et donc les requêtes) de l'opérateur. Ceci constitue cependant une contrainte qui peut être relaxée au vu des relations d'autorité. C'est pourquoi, si le planificateur ne retourne pas de solution, on retire de $subgoal(r_g)$, comme précédemment (et si applicable) les sous-buts annulables afin d'élargir son espace de recherche.

Si le planificateur ne retourne toujours pas de solution, rien d'autre n'est possible en l'état sans modification des relations d'autorité existantes.

À nouveau, on définit :
$G_{conflictingPartToKeep} = \bigcup_{r_k \in plannerGoals(G_{conflict})} \{D^+_{G_{conflict}}(r_k)\ \}$
le graphe dépendant de r_k, sous-but donné au planificateur. Ce qui précède r_k est susceptible d'être modifié par le planificateur, en revanche le graphe en aval de r_k doit être gardé en l'état.

Exemple :
$G_{conflictingPartToKeep} = (\{r_{NavigationRobot}, r_{WP-Goal}, r_{SteeringWheel}, r_{NavigationOperator}, r_{WP-Operator}\},$
$\{Int4, Int5, Int8, Int9, Int10\}).$

Recherche de graphes de ressources solutions et sélection

Les paramètres fournis en entrée du planificateur sont les suivants : $\{\mathcal{V}, plannerGoals(G_{conflict})\}$, c'est-à-dire l'ensemble des paramètres de la situation courante (état des ressources, état des interfaces, relations d'autorité) et les buts que le planificateur doit considérer, afin de trouver un ensemble cohérent d'actions à réaliser pour les satisfaire.

Afin de restaurer la cohérence, le planificateur doit permettre d'obtenir un graphe solution $G_{solving}$ tel que chacun des sous-buts passés au planificateur soit produit si l'exécution du plan reflété par le nouveau modèle de l'autorité $\mathcal{V}'$ se déroule de manière nominale.

Exemple : Pour rappel,
$plannerGoals(G_{conflict}) = \{r_{NavigationRobot}, r_{SteeringWheel}\}$.
On obtient alors :
$G_{solving} = (\{r_{Position}, r_{Energy}, r_{SteeringWheel}, r_{NavigationRobot}\}, \{Int1, Int2, Int3\})$, voir figure 4.19.

La solution apportée par le planificateur est illustrée sur la figure 4.19.

Comme mentionné précédemment la solution consiste à « décaler » dans le temps la réalisation du graphe $D^-_{G_{conflict}}(r_{WP-Goal})$, en faisant revenir les requêtes de toutes les interfaces de ce graphe à l'état *Ri requested by Rj*, en tirant sur ces interfaces la transition $t_{Requestpostponed}$. Cependant, ceci n'apparaît pas dans le graphe solution en tant que tel, il s'agit d'une opération supplémentaire que nous verrons ultérieurement. Le graphe solution trouvé correspond donc exactement au graphe précédent.

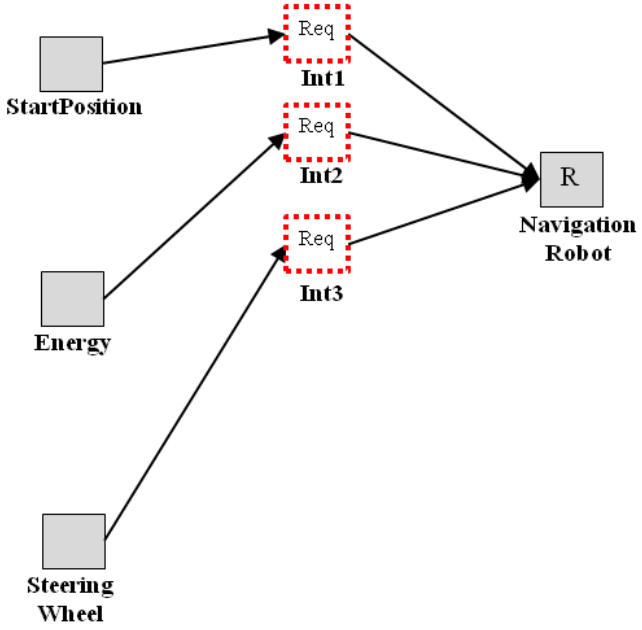

FIGURE 4.19: Graphe de ressources solution $G_{solving}$ pour le conflit de préemption.

Détermination des transformations à effectuer pour passer de G à G'

Comme dans le cas du conflit de destruction, pour passer du graphe de ressources en état de conflit à un graphe cohérent, il faut identifier les ressources et interfaces à ajouter, tout comme les ressources et interfaces à supprimer :

Sous-graphe à ajouter
Soit $G_{toAdd} = G_{solving} \setminus (G_{solving} \cap G)$,
soit tout simplement le graphe solution retourné par le planificateur moins les ressources et interfaces existant déjà dans le graphe de ressources courant.

Exemple : $G_{toAdd} = (\emptyset, \emptyset)$: en effet, le graphe solution renvoyé par le planificateur correspond exactement à ce qui existait précédemment.

Sous-graphe à retirer
Afin de supprimer toutes les ressources inutiles, on réalise l'inventaire de toutes les ressources *strictement nécessaires* à la satisfaction des buts conservés. Toute ressource ne faisant pas partie de cet inventaire doit être retirée.
Soit $H = goals(G) - goals(G_{conflict})$ l'ensemble des buts poursuivis dans le graphe de ressource initial privé des buts concernés par le conflit.

Exemple : $H = \{\emptyset\}$: tous les buts étaient affectés par le conflit.

Pour un but $r_g \in goals(G)$, on désigne par $Int_{request}(r_g)$ l'interface représentant la requête, formulée par l'un des agents, de la poursuite de ce but telle que $r_g \rightarrow_G Int_{request}(r_g)$. On note alors $goalRequest_G(r_g) = (\emptyset, \{Int_{request}(r_g)\})$ le graphe ne comprenant que cette interface.
Soit $G_{necessary} = \bigcup_{r_g \in H} \{D_G^-(r_g) \cup goalRequest_G(r_g)\}$ le graphe de ressource comprenant uniquement les ressources (et interfaces) requises pour réaliser l'ensemble des buts de H.

Exemple : $G_{necessary} = (\{\emptyset\}, \{\emptyset\})$.

Ainsi, $\mathcal{G}_{toRemove} = G \setminus (G_{necessary} \cup G_{conflictingPartToKeep} \cup G_{solving})$,
ou autrement dit, tout ce qui n'est pas strictement nécessaire ou à conserver est à retirer du modèle initial.

Exemple : $G_{toRemove} = (\emptyset, \emptyset)$.

Enfin, on peut écrire :
$G' = G \cup G_{toAdd} \setminus G_{toRemove}$,
ou encore $G' = G_{necessary} \cup G_{solving} \cup G_{conflictingPartToKeep}$.

Cependant, il reste encore à modifier l'état des requêtes sur certaines interfaces.

Modification de l'état de requêtes sur les interfaces :
dans le cas d'un conflit de préemption où deux buts r_1 et r_2 sont en compétition sur une même ressource $r_{conflict}$, et s'il est possible de garder les deux buts, le planificateur doit repousser la réalisation de l'un (r_2) après la réalisation du premier (r_1). Alors que toutes les requêtes sur les interfaces dans le graphe $G_{conflict} = \mathcal{D}_G^+(r_{conflict})$ sont en l'état *Satisfying Request*, il faut faire repasser les requêtes associées au but r_2 en l'état *Ri requested by Rj*.

Cette transformation implique donc le tir de la transition $t_{Requestpostponed}$ sur toutes les interfaces de $\mathcal{I}_{postponed}$ défini précédemment.

Exemple : $\mathcal{I}_{postponed} = \{Int1, Int2, Int3, Int4\}$.

Le résultat de l'ensemble de ces opérations est illustré sur la figure 4.20.

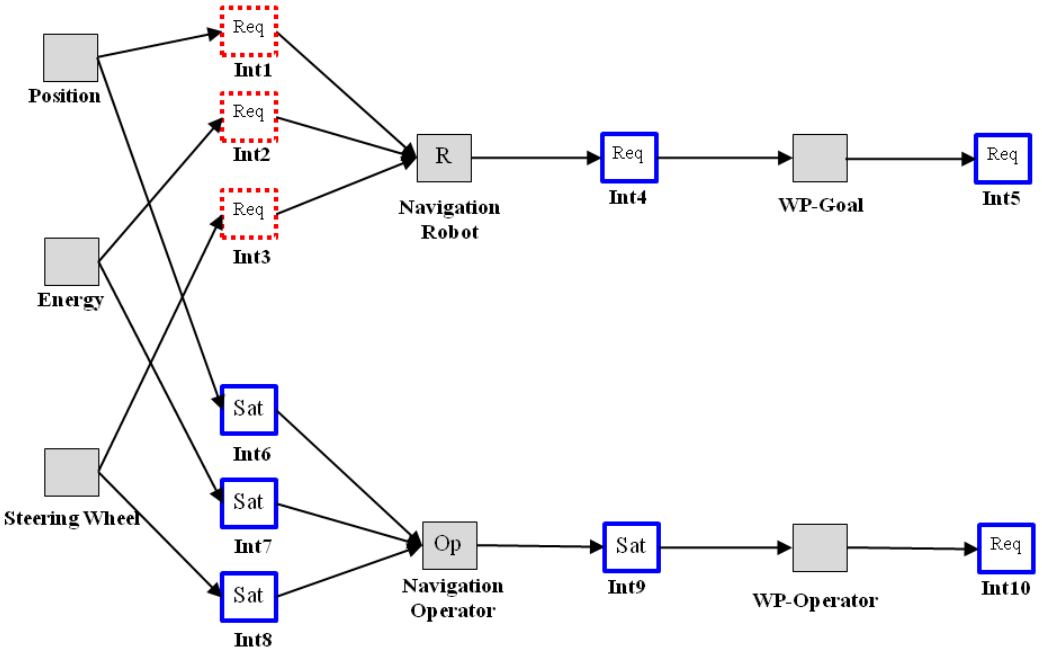

FIGURE 4.20: Graphe de ressources solution $G_{solving}$ pour le conflit de préemption.

2.3 Dynamique du partage de l'autorité

2.3.1 Principe

Si, pour un conflit donné, le planificateur ne parvient pas à trouver de solution selon les étapes précédentes, une solution consiste à élargir son espace de recherche, en relâchant les contraintes associées aux relations d'autorité. En effet, il se peut que sous ces contraintes, le planificateur (agissant sous le sceau de *robot*) ne puisse pas inclure certaines ressources dans son plan, ou ne puisse pas modifier certaines requêtes de l'opérateur, même si elles sont contradictoires entre elles ou ne permettent pas de trouver une solution.

La modification des relations d'autorité se fait par le tir des transitions $t_{Agrgain}$, $t_{Agrloss}$, $t_{Aghgain}$ et $t_{Aghloss}$ sur les réseaux $a_{<robot,operator>}(r)$, r étant une des ressources en conflit ou toute autre ressource disponible permettant de créer un plan solution et sur laquelle l'agent robotique n'a initialement pas l'autorité suffisante pour la contrôler.

Les relations d'autorité a du macro-modèle $\mathcal{V} = \{G, \mathcal{C}, \phi, a\}$ sont susceptibles de changer en cours de mission, cédant la place à un nouvel ensemble de relations d'autorité a'. Nous appelons ce principe *la dynamique de l'autorité*. Illustrons maintenant les implications de ce principe sur un exemple, reprenant le conflit de préemption décrit précédemment mais légèrement étendu.

2.3.2 Exemple

Reprenons l'exemple précédent tel que décrit sur la figure 4.20, après que l'opérateur a pris la main sur la ressource $r_{SteeringWheel}$. Ajoutons à cet exemple une simple ressource supplémentaire, une contrainte de sécurité nommée $r_{SafetyDistance}$, qui est une ressource *partageable*, et dont le lien de dépendance avec les tâches de navigation par le robot ou l'opérateur *via* les interfaces $Int11$ et $Int12$ est de type *Exec-dep* : cette ressource matérialise l'obligation pour le véhicule de rester à une distance minimale de sécurité des obstacles, ce quel que soit l'agent réalisant la tâche de navigation en cours : cette distance doit être vérifiée *avant* le démarrage de la tâche et doit le rester *pendant* toute l'exécution.

Les relations d'autorité sont les suivantes :
- $a_{<robot,operator>}(r_{Position}) = (Access, Access)$;
- $a_{<robot,operator>}(r_{Energy}) = (Access, Access)$;
- $a_{<robot,operator>}(r_{SteeringWheel}) = (Access, Access)$;
- $a_{<robot,operator>}(r_{SafetyDistance}) = (Access, Access)$.
- $a_{<robot,operator>}(r_{NavigationRobot}) = (Preemptability, Access)$;
- $a_{<robot,operator>}(r_{NavigationOperator}) = (NoAccess, Preemptability)$.
- $a_{<robot,operator>}(r_{WP-Goal}) = (Access, Preemptability)$.
- $a_{<robot,operator>}(r_{WP-Operator}) = (Access, Preemptability)$.

La main étant laissée à l'opérateur pour la tâche de navigation manuelle (ressource $r_{NavigationOperator}$), il apparaît que l'opérateur viole la contrainte de distance de sécurité, détruisant la ressource $r_{SafetyDistance}$, ce qui donne alors le graphe de la figure 4.21.

On a alors $\text{Conflict}_{\text{Destruction}} \subseteq \text{M}_{\text{r}_{\text{SafetyDistance}}}$. Un plan solution à ce conflit consiste à redonner la navigation en urgence au robot : en faisant appel à une tâche exécutée par le robot consacrée uniquement à la restauration de cette distance de sécurité, le reste du plan peut être conservé. Le graphe solution est représenté sur la figure 4.22.

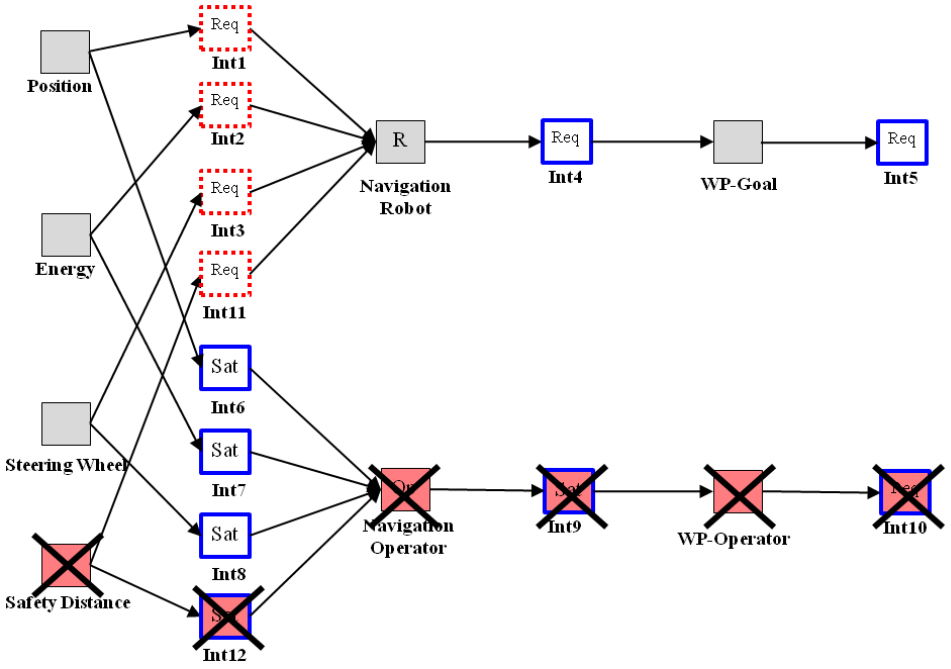

FIGURE 4.21: Graphe de ressources après destruction de $r_{SafetyDistance}$ par l'opérateur humain

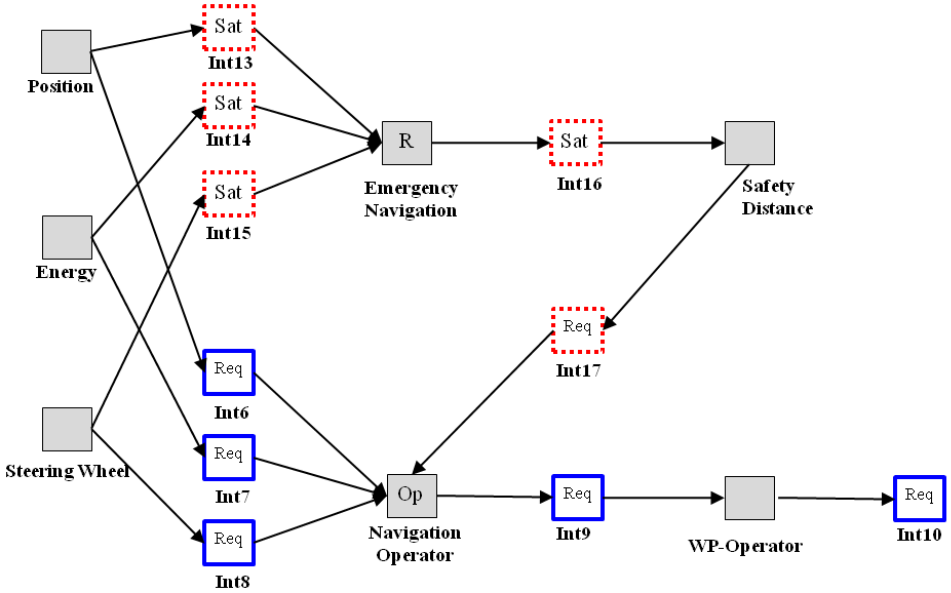

FIGURE 4.22: Graphe de ressources après prise d'autorité par l'agent robotique

On note que la ressource $r_{EmergencyNavigation}$ ne requiert pas $r_{SafetyDistance}$ puisqu'elle représente une tâche appelée précisément dans le cas particulier où la distance de sécurité n'est pas respectée et doit être rétablie. Une fois celle-ci rétablie, l'agent robotique relâche la ressource $r_{SteeringWheel}$, la main est rendue à l'opérateur (ressource $r_{NavigationOperator}$) qui doit toujours disposer de l'accès à cette ressource, comme dans le cadre de la relation d'autorité initiale (avec $a_{<robot,operator>}(r_{SteeringWheel}) = (Access, Access)$).

Cependant, pour que le basculement entre ces deux graphes puissent être opéré, il faut impérativement que les requêtes de l'opérateur sur les interfaces $Int6, Int7, Int8, Int9$ et $Int12$ soient décalées, c'est-à-dire que la transition $t_{Requestpostponed}$ soit tirée sur ces interfaces. Mais pour ce faire, il faut que l'agent *robot* ait l'autorité suffisante, ce qui n'est le cas pour aucune d'entre elles, même de manière temporaire simplement pour pouvoir modifier la graphe de ressources avant son exécution. Pour que le graphe solution puisse être appliqué, il faut que le robot « gagne » de l'autorité sur les ressources $r_{Position}, r_{Energy}, r_{SteeringWheel}, r_{SafetyDistance}, r_{NavigationOperator}$ et atteigne l'autorité $a_{<robot,operator>}(r) = (Preemptability, Access)$ pour chacune d'entre elles. De même, il faut que les requêtes de l'opérateur sur les interfaces $Int4$ et $Int5$ soient simplement annulées, par le tir de la transition $t_{Requestcancelled}$ sur chacune d'entre elles : encore une fois, le robot doit acquérir l'autorité nécessaire pour pouvoir le faire.

Ceci ne peut être réalisé qu'en faisant appel à une entité externe, autre que le robot ou l'opérateur, établissant les règles de basculement de l'autorité, que nous nommons *méta-autorité*. Dans le cas de cet exemple, la violation d'une règle de sécurité justifie la prise d'autorité par le robot vis-à-vis de l'opérateur sur un ensemble identifié de ressources afin de rétablir cette exigence le plus rapidement possible : il s'agit d'un cas défini par les concepteurs. La méta-autorité se pose alors comme une entité neutre dans l'architecture du système, ne portant le sceau d'aucun agent ; la méta-autorité agit ainsi comme un « arbitre », faisant respecter en cours des missions des règles du jeu prédéfinies et immuables que tous les agents du système sont censés connaître.

Appelée par le planificateur, la méta-autorité provoque le tir des transitions $t_{Agrgain}$ ou $t_{Aghloss}$ sur les relations d'autorité correspondantes ; par exemple, ces transitions sont tirées sur $a_{<robot,operator>(SteeringWheel)}$ afin d'obtenir $a_{<robot,operator>}(r_{SteeringWheel}) = (Preemptability, Access)$: l'agent robotique a gagné de l'autorité sur l'opérateur humain pour la ressource *SteeringWheel*. Cela rend possible l'affectation de la ressource $r_{SteeringWheel}$ à la ressource $r_{EmergencyNavigation}$ *via* l'interface $Int15$ créée par *robot* suite à la replanification pour produire la ressource $r_{SafetyDistance}$, devenue sous-but du point de vue du planificateur.

2.4 Résolution alternative : perte de buts

Nous nous plaçons dans le cas où, suite à un conflit, aucune solution n'a été trouvée par le planificateur pour faire revenir le graphe de ressources à un état cohérent : il n'y a pas de modèle de l'autorité $\mathcal{V}' = \{G', \mathcal{C}', \phi', a'\}$ cohérent (c'est-à-dire sans conflit) avec $goals(G') = goals(G)$.

Si aucune autre approche n'a fonctionné jusqu'alors, la solution restante consiste à lever la dernière contrainte existante, c'est-à-dire à supprimer l'égalité $goals(G') = goals(G)$. Ceci impose alors de sélectionner un ou plusieurs buts qui devront être annulés. Cette approche constitue donc une résolution *dégradée* du conflit.

Le choix des buts à annuler n'est pas nécessairement trivial et dépend fortement des exigences de la mission. Dans tous les cas, annuler un but impose que l'agent *robot* dispose d'une autorité totale sur celui-ci, ce qui dans le cas d'un but r dont la satisfaction a été demandée par l'opérateur, implique que : $a_{<robot,operator>}(r) = (Preemptability, NoAccess)$.

La relation d'autorité initiale $a_{<robot,operator>}(r)$ pour une ressource-but r, dont la satisfaction est demandée par l'opérateur, est définie lors de la conception du système, et est créée lorsque l'opérateur passe un ordre *via* son interface utilisateur. L'opérateur assignant cette ressource, son autorité ne peut pas être telle que $a_{<robot,operator>}(r) = (Preemptability, NoAccess)$. Pour que ce but soit annulé par le robot, la relation d'autorité initiale sur ce but r doit être modifiée. Ce problème nous renvoie vers le concept de méta-autorité, que nous avons déjà évoqué plus haut.

Chapitre 5

Mise en œuvre

Nous présentons dans ce chapitre la mise en œuvre des macro- et micro-modèles présentés auparavant afin de réaliser le contrôleur de la dynamique de l'autorité. Dans un premier temps, nous situons le contrôleur par rapport aux autres fonctions de l'architecture robotique et comment celui-ci interagit avec ces fonctions. Par la suite, les liens entre le micro-modèle, le macro-modèle et les fonctions de l'architecture seront détaillés, en expliquant toutes les interconnexions réalisées entre les réseaux de Petri, et entre les réseaux de Petri et le reste de l'architecture. Enfin, la mise en œuvre informatique de ces concepts sera présentée, en abordant certains problèmes posés par l'interconnexion des réseaux et les solutions techniques correspondantes. Entre autres, l'algorithme du joueur de Petri sera expliqué.

1 Architecture fonctionnelle

Afin de pouvoir être mis en œuvre, le macro-modèle de l'autorité doit être connecté à un certain nombre de fonctions, ces fonctions devant permettre de :

- jouer les réseaux de Petri du micro-modèle ;
- recevoir et envoyer des événements pour que le macro-modèle puisse être mis à jour, pour que les ordres sur les actuateurs puissent être envoyés suivant l'exécution du plan, pour que les actions et ordres de l'opérateur soient pris en compte, et pour permettre la résolution des conflits.

Ce paragraphe présente les différentes fonctions nécessaires pour rendre opérationnel le modèle de l'autorité ainsi que leur organisation.

1.1 Vue d'ensemble

L'architecture décisionnelle embarquée de l'agent *robot* comprend :
- le *Planificateur* : le planificateur génère le plan du robot (incluant les tâches réalisables par l'opérateur) afin d'atteindre les buts de mission tout en respectant les contraintes. Un graphe de ressources est extrait à partir du plan et est transmis au *Contrôleur de la dynamique de l'Autorité*.
- le *Suivi de Situation* : il estime en permanence l'état courant du système robotique dans son ensemble (y compris l'« état » de l'opérateur humain [1]) et prédit ses états futurs, sur la base des informations en provenance des capteurs et des communications.
- le *Contrôleur de la dynamique de l'autorité* : il met à jour le macro-modèle d'autorité, supervise l'exécution du plan du robot, détecte les conflits et restaure la cohérence du plan d'exécution.

Les autres fonctions du système sont les suivantes :
- la collecte et le prétraitement éventuel des données : il s'agit de la fonction « perception » du robot, et de la mise en forme des données pour leur traitement par le reste des fonctions du système ;
- la partie exécutive, un ensemble d'algorithmes qui constituent les « savoir-faire » du robot, les actuateurs du système et les asservissements de base. Cette fonction embarquée s'occupe de la réalisation effective des tâches à exécuter par le robot, sur la base des ordres provenant de l'architecture décisionnelle.
- l'interface opérateur : reliée au robot par des moyens de communication, l'interface permet à l'opérateur de connaître l'état du robot et de lui envoyer des ordres ou des consignes.

Les paragraphes suivants détaillent le contrôleur de la dynamique de l'autorité ainsi que ses liens avec les autres fonctions.

1.2 Contrôleur de la dynamique de l'autorité

Le contrôleur de la dynamique de l'autorité, dont le schéma est présenté sur la figure 5.1, est composé des trois éléments suivants :
- le macro-modèle de l'autorité lui-même $\mathcal{V} = \{G, \mathcal{C}, \phi, a\}$;
- le joueur de Petri qui tire toute transition des réseaux de Petri du micro-modèle dès que cela est possible en fonction de l'état des réseaux, des relations définies sur les transitions par fusion ou passage d'événements, et des événements. Par le tir de certaines transitions identifiées du modèle, le joueur de Petri transmet également des événements au superviseur ;

1. par exemple, *via* un oculomètre et des capteurs physiologiques

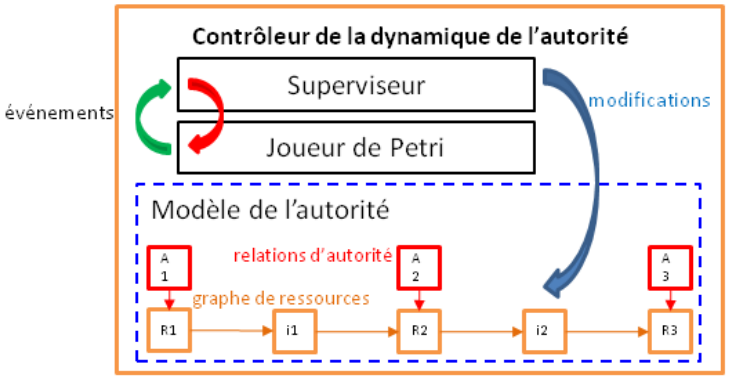

FIGURE 5.1: Schéma du contrôleur de la dynamique de l'autorité

– le superviseur qui détecte les conflits et orchestre les relations entre le modèle à proprement parler et les fonctions externes. Le superviseur convertit les événements transmis en interne par le joueur de Petri en ordres pour la fonction exécutive du robot, transmet au joueur de Petri les événements en provenance du suivi de situation, interagit avec l'interface opérateur, insère et supprime des ressources et interfaces, modifie les relations d'autorité en communiquant avec le planificateur.

1.3 Planificateur

La figure 5.2 illustre la place du planificateur par rapport au contrôleur de la dynamique de l'autorité :

– le planificateur reçoit des requêtes en provenance du contrôleur de la dynamique de l'autorité (*via* le superviseur) lors du processus de résolution de conflit. Le planificateur reçoit alors l'état complet du macro-modèle, en plus des buts à satisfaire ;
– le planificateur recherche un plan satisfaisant les requêtes en fonction de la situation courante ;
– lorsque le planificateur a terminé la recherche de plan, si un plan est trouvé, un graphe de ressources en est extrait ; éventuellement, certaines modifications des relations d'autorité sont requises. Le planificateur renvoie alors au contrôleur de la dynamique de l'autorité l'ensemble des transformations à réaliser pour passer du modèle sans conflit $\mathcal{V}$ au modèle corrigé $\mathcal{V}'$: ces transformations se présentent sous la forme de graphes de ressources à ajouter ou enlever sur le graphe de ressources courant G, ainsi que de modifications à réaliser sur les interfaces et relations d'autorité existantes.

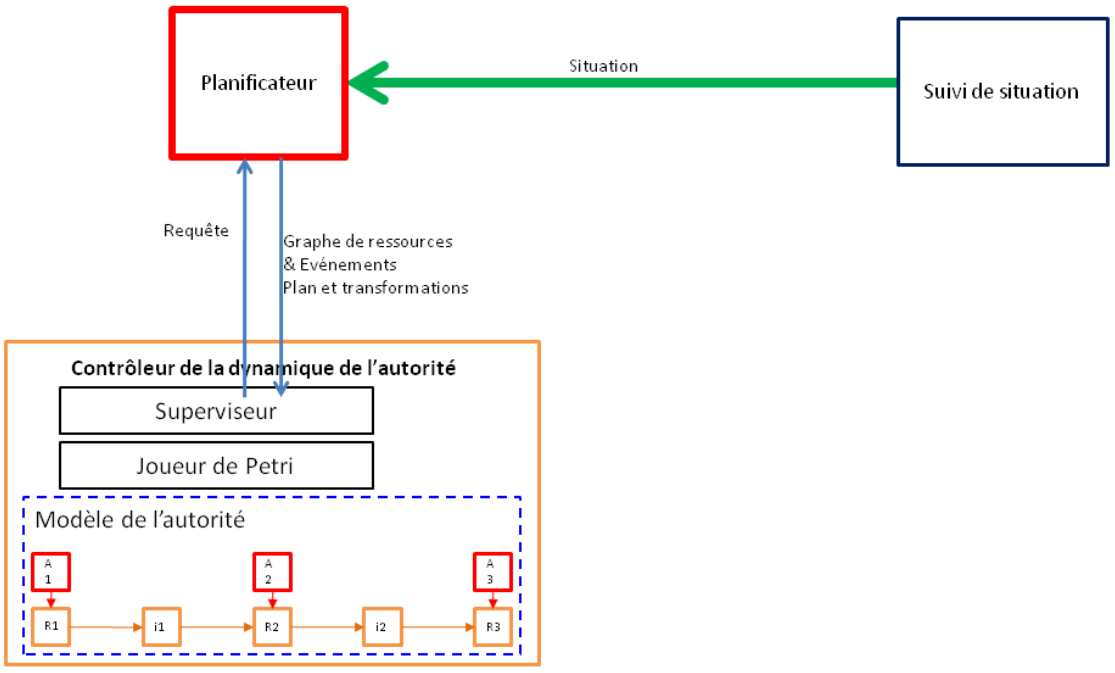

FIGURE 5.2: Place du planificateur dans l'architecture

1.4 Suivi de situation

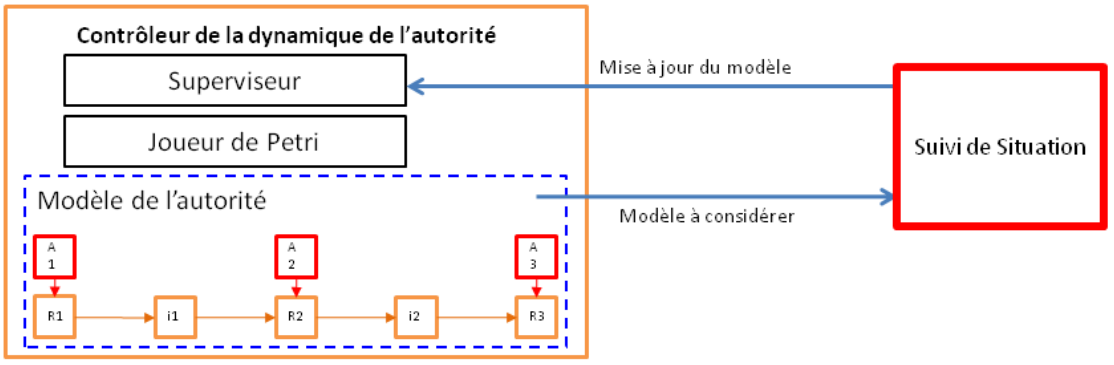

FIGURE 5.3: Place du suivi de situation dans l'architecture

La figure 5.3 illustre la place du suivi de situation par rapport au contrôleur de la dynamique de l'autorité :

- le contrôleur de la dynamique de l'autorité fournit au suivi de situation le graphe de ressources courant G afin que le suivi de situation sache quelles sont les ressources à surveiller. Si G est modifié, le suivi de situation en est informé ;

– le produit de l'estimation d'état des ressources par le suivi de situation est renvoyé au contrôleur de la dynamique de l'autorité, afin que celui-ci mette le modèle à jour. Lorsque le superviseur du contrôleur de la dynamique de l'autorité reçoit les informations de mise à jour d'une ressource en provenance du suivi de situation, il génère un événement à destination d'une des transitions de changement d'état sur le réseau de Petri de la ressource considérée. Le joueur de Petri procède alors à la mise à jour du micro-modèle en conséquence.

1.5 Fonction exécutive du robot

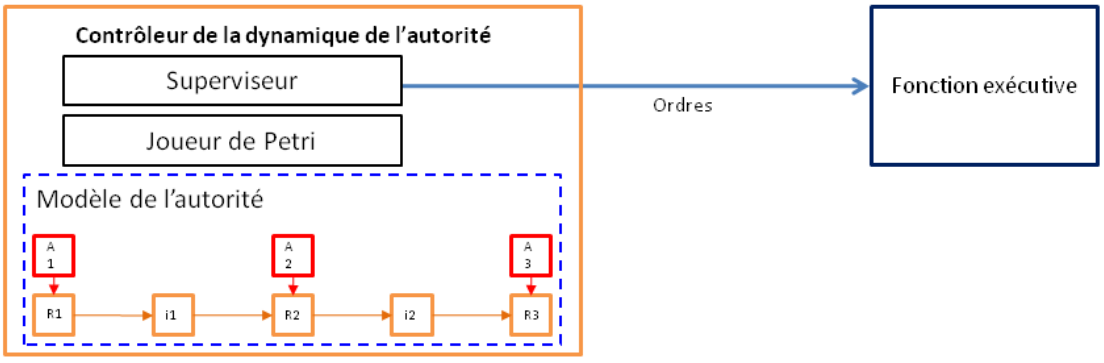

FIGURE 5.4: Place de la commande et des actuateurs dans l'architecture

Toutes les tâches à exécuter par le robot sont gérées depuis le contrôleur de la dynamique de l'autorité : celui-ci a la charge de tenir une représentation à jour du plan d'exécution et d'en vérifier la cohérence. Cela permet au superviseur opérant sur ce modèle de mettre en œuvre le plan d'exécution sur la base de l'évolution du modèle, qui contient toutes les informations nécessaires relatives à la gestion des tâches : quelle tâche doit être exécutée et à quel moment, quand elle doit être arrêtée.

Prenons l'exemple d'une tâche de navigation réalisée par le robot, nécessitant pour être réalisée la présence d'énergie. La figure 5.5 représente le graphe de ressources correspondant. Le lien entre les deux ressources est du type *Exec-Dep*. Lorsque la ressource *Energy* est produite (par observation provenant de la fonction de suivi de situation), l'affectation peut avoir lieu : la ressource *Energy* est affectée à *Navigation Robot via* l'interface *Int1*. Ce faisant, étant donné le type de dépendance, et suivant un mécanisme décrit en 2.3, le passage de la requête sur l'interface *Int1* à l'état $SatisfyingRequest$ conduit à la production de la ressource *Navigation Robot*. C'est cet événement, le passage de la ressource *Navigation Robot* de l'état *Absent* à l'état *Present*, qui est détecté par le superviseur du contrôleur de la dynamique de l'autorité ; en conséquence, il envoie l'ordre à la fonc-

tion exécutive de commencer la tâche de navigation du robot correspondante. Suivant ce principe, c'est l'évolution du modèle de l'autorité, mis à jour par le suivi de situation et suivant son évolution interne, qui déclenche *via* le superviseur le lancement et l'arrêt des algorithmes de la fonction exécutive.

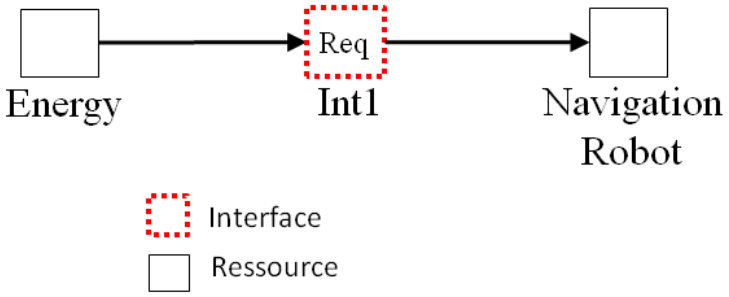

FIGURE 5.5: Exemple d'une tâche de navigation pour la fonction exécutive

1.6 Interface opérateur

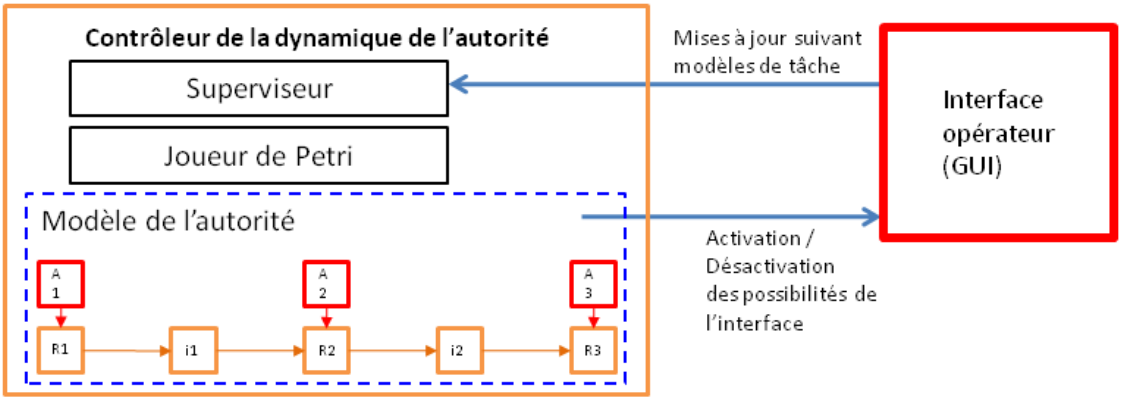

FIGURE 5.6: Interface opérateur et contrôleur de la dynamique de l'autorité

La figure 5.6 illustre la place de l'interface par rapport au contrôleur de la dynamique de l'autorité :

- les relations d'autorité du modèle sont transmises à l'interface : certaines commandes de l'interface peuvent se griser, indiquant qu'en l'état actuel des relations d'autorité leur usage n'est pas possible, ou au contraire s'activer, indiquant de nouvelles possibilités pour l'opérateur. Ainsi, l'interface de l'opérateur, et ce qu'il peut réaliser sur le système, est mis à jour à partir du modèle de l'autorité ;
- chaque commande de l'interface graphique est associée à un modèle de tâche élémentaire défini à la conception, représentant l'usage auquel la commande est destinée.

Lorsque l'opérateur active une commande particulière, le graphe de ressources correspondant à son modèle de tâche est transmis au contrôleur de la dynamique de l'autorité afin d'y être inséré ; inversement, ce graphe de ressource est retiré lorsque l'opérateur arrête sa commande. La figure 5.7 donne un exemple d'un tel graphe de ressources.

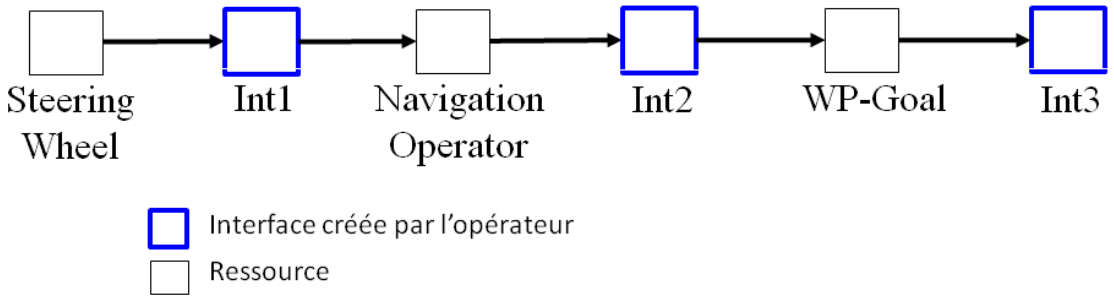

FIGURE 5.7: Graphe de ressources correspondant au modèle de tâche associé à la prise du joystick par l'opérateur

La figure 5.7 représente le graphe de ressources correspondant au modèle de tâche qui a été associé, par conception, avec le joystick de commande de cap de l'interface opérateur (représenté par la ressource *Steering Wheel*). Lorsque l'opérateur se saisit du joystick, l'interface fait insérer ce graphe de ressources dans le macro-modèle de l'autorité, en communiquant avec le superviseur du contrôleur de l'autorité. En effet, sur l'exemple, il a été décidé que la prise du joystick par l'opérateur correspond au fait qu'il veuille réaliser une tâche de navigation manuelle vers un point d'arrivée inconnu du robot. Même si tous les paramètres de l'action en cours réalisée par l'opérateur ne sont pas connus, insérer un tel graphe de ressources permet de réaliser la détection de conflit avec le reste des actions en cours ou à venir déjà représentées dans le modèle de l'autorité. De plus, en cas de conflit, cela peut modifier les plans solutions que le robot pourrait trouver. Ce graphe de ressources est retiré du macro-modèle dès que l'opérateur relâche son joystick.

2 Réalisation de l'architecture et du macro-modèle par les composants du micro-modèle

Au niveau de son implémentation concrète, le micro-modèle d'autorité est constitué de réseaux de Petri génériques, connectés entre eux ainsi qu'aux fonctions de l'architecture externes au modèle. Ce paragraphe a pour objectif de présenter comment les considérations de niveau « macro » sont transposées au niveau « micro ».

2.1 Interconnexions des fonctions de l'architecture avec le micro-modèle

2.1.1 Interconnexions avec le suivi de situation et la fonction exécutive

Le suivi de situation met à jour l'état de chaque ressource du macro-modèle de l'autorité à partir de la vérification de la condition logique associée à la sémantique de la ressource : toute ressource, quelle que soit sa nature, doit être définie de telle manière qu'il soit possible d'indiquer si elle est présente ou absente ; cela revient à associer à la ressource une condition logique portant sur les variables du système. Si la condition logique devient vraie, un événement $e1$ est généré et transmis au joueur de Petri, et si le marquage le permet la transition $t_{production}$ est tirée. Inversement, si la condition logique devient fausse, un événement $e2$ est généré, transmis au joueur de Petri, et une des transitions $t_{destruction}$ de la ressource est tirée en fonction du marquage de la ressource. Ainsi, la ressource est marquée comme *présente* ou *absente* en fonction de la vérification de la condition logique associée à la ressource, ce qui constitue une modification de l'état d'une ressource par *observation*. Par la suite, le suivi de situation se charge de mettre à jour toute évolution de l'état de la ressource en cours de mission. La figure 5.8 illustre cette approche, avec les événements reçus représentés en couleur rouge.

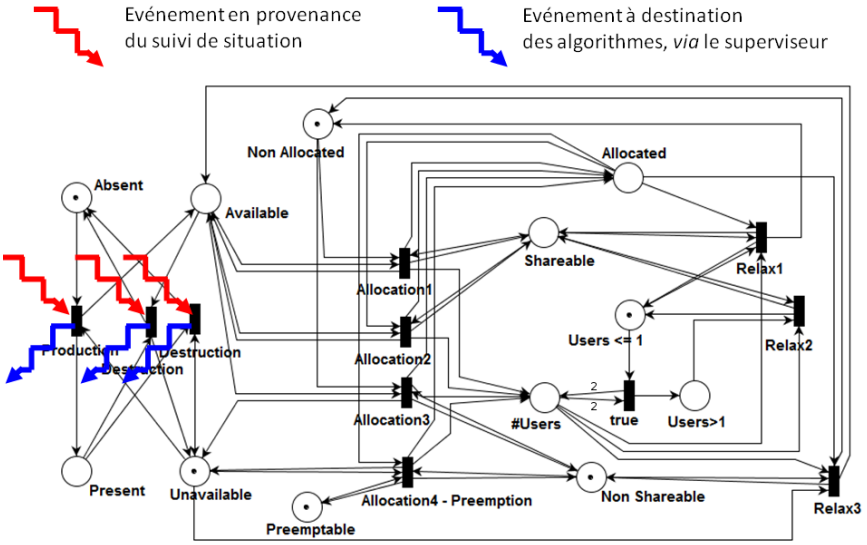

FIGURE 5.8: Connexions d'une ressource avec les fonctions externes

Exemple : sur le cas utilisé précédemment dans le paragraphe sur la fonction exécutive (voir figure 5.5), l'état de la ressource *Energy* doit être déterminé par le suivi de situation. Lorsque celui-ci observe un changement d'état de cette ressource, il le communique au superviseur du contrôleur de la dynamique de l'autorité, qui en conséquence génère l'événement destiné aux transitions $t_{production}$ ou $t_{destruction}$ sur le réseau de la ressource *Energy*.

La fonction exécutive reçoit les ordres de lancement et d'arrêt des algorithmes, en fonction de l'état courant du plan d'exécution représenté dans le modèle de l'autorité. Comme expliqué au paragraphe 1.5, pour une ressource représentant une tâche, c'est le passage de l'état *Absent* à l'état *Present* qui est utilisé comme ordre de lancement de l'algorithme correspondant. Pour ce faire, les événements générés par le tir des transitions $t_{production}$ et $t_{desctruction}$ sont récupérés par le superviseur du contrôleur de la dynamique de l'autorité, qui déclenche en conséquence le démarrage ou l'arrêt des tâches correspondantes par la fonction exécutive, ce qui est illustré sur la figure 5.8, avec les événements émis représentés en couleur bleue.

Exemple : sur le cas utilisé précédemment dans le paragraphe sur la fonction exécutive (voir figure 5.5), après production de la ressource *Energy* par le suivi de situation, celle-ci est affectée *via* l'interface *Int1*, ce qui par le jeu du lien de dépendance amène à la production de la ressource *Navigation Robot*. Le tir de la transition $t_{production}$ sur la ressource *Navigation Robot* génère un événement qui est capté par le superviseur du contrôleur de la dynamique de l'autorité, qui à son tour envoie l'ordre de lancement de l'algorithme de navigation automatique du robot à la fonction exécutive.

2.1.2 Interconnexions avec la planification

Le planificateur, lorsqu'il a trouvé un plan solution pour satisfaire les buts recherchés, génère un graphe de ressources solution à partir duquel sont déduites les transformations à réaliser pour passer du graphe de ressources en état de conflit G à un graphe de ressources cohérent G'. Certaines transformations consistent en l'ajout ou le retrait de réseaux de ressource, interface ou autorité au sein du micro-modèle, ce qui est effectué par le superviseur. Une fois tous les réseaux mis en place, le reste des transformations demandées par le planificateur consiste en un changement d'état sur les réseaux existants dans le modèle : ces transformations comprennent en particulier l'activation de requêtes sur les réseaux d'interface, l'annulation de ces requêtes, et enfin le décalage dans le temps des requêtes en cours de satisfaction (voir figure 5.9).

Voici les transformations réalisées sur les réseaux d'interfaces :

- pour une transformation consistant en l'activation d'une requête sur un réseau d'interface, le superviseur du contrôleur de la dynamique de l'autorité génère un événement à destination de la transition $t_{Requestcreated}$ de l'interface considérée ;

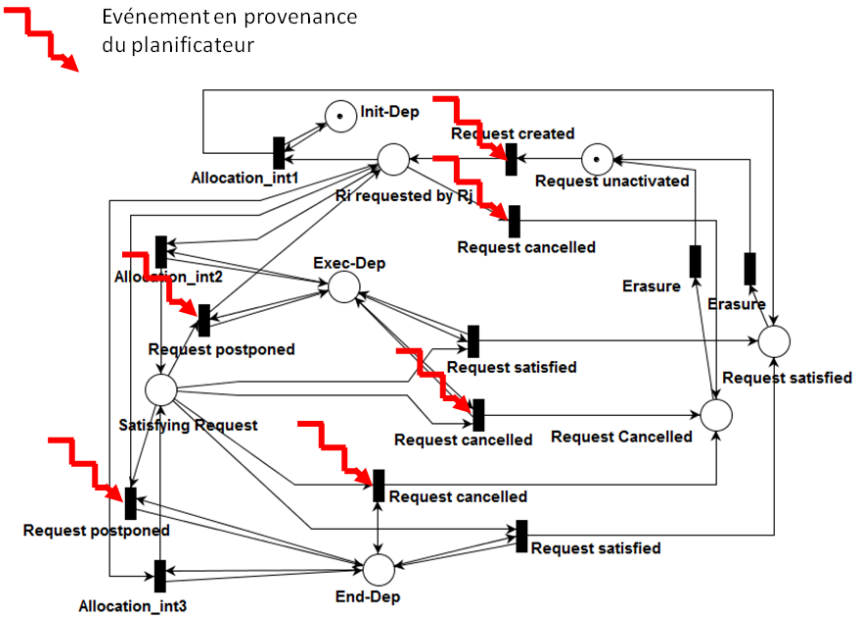

FIGURE 5.9: Connexions d'une interface avec le planificateur

- pour une transformation consistant en l'annulation d'une requête sur un réseau d'interface, le superviseur du contrôleur de la dynamique de l'autorité génère un événement à destination de la transition $t_{Requestcancelled}$ de l'interface considérée ;
- pour une transformation consistant à décaler une requête sur un réseau d'interface, faisant revenir l'interface de l'état *Satisfying Request* à l'état *Ri requested by Rj*, le superviseur du contrôleur de la dynamique de l'autorité génère un événement à destination de la transition $t_{Requestpostponed}$ de l'interface considérée.

Certaines transormations consistent à modifier l'état des réseaux d'autorité : en effet, lors de la génération du plan, le planificateur a eu besoin d'élargir son espace de recherche et a déterminé, avec une fonction telle que la méta-autorité, que certaines relations d'autorité pouvaient être modifiées. Ces transformations sont elles aussi transmises au superviseur, qui génère alors les événements destinés aux transitions $t_{AgXgain}$, $t_{AgYgain}$, $t_{AgXloss}$ et $t_{AgYloss}$. Toutes sont réceptrices du même événement, mais en raison du marquage du réseau de la relation d'autorité, une seule de ces transitions peut être validée : en effet, l'événement généré par le superviseur est également porteur de la couleur de l'agent gagnant de l'autorité [2], et toutes les transitions du réseau de relation d'autorité permettant à un agent de gagner de l'autorité sont connectées à une des places $p_{ColorX1}$, $p_{ColorX2}$, $p_{Colory1}$ ou $p_{ColorY2}$.

2. Une couleur peut être associée à un événement. Une transition réceptrice de cet événement ne sera tirée que si elle est validée par un marquage de la même couleur, sinon l'événement sera ignoré.

La figure 5.10 représente les transformations réalisées sur les réseaux d'autorité.

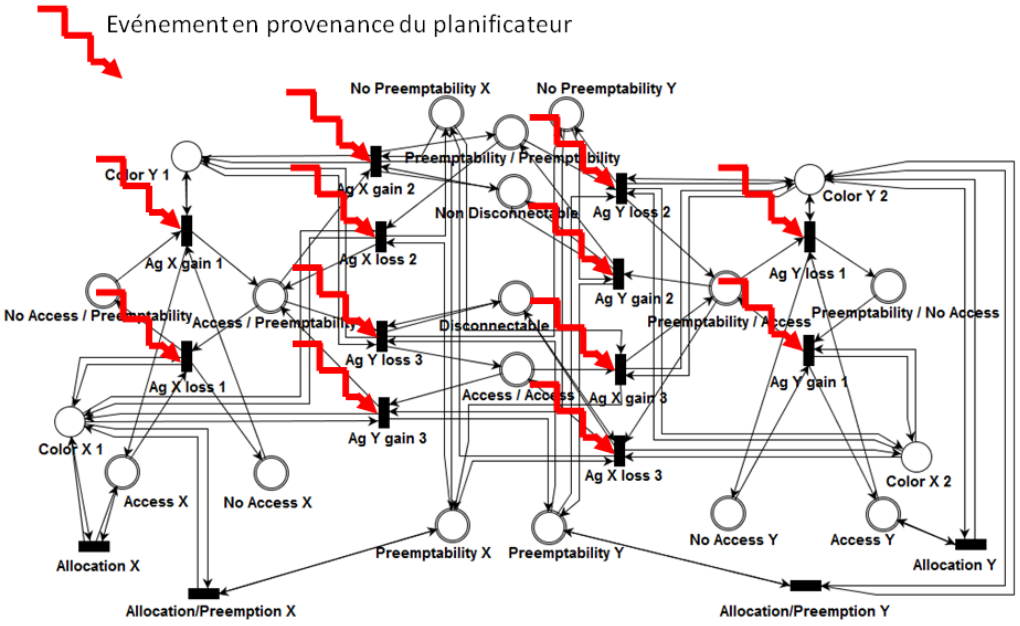

FIGURE 5.10: Connexions d'une relation d'autorité avec le planificateur

2.2 Affectation d'une ressource

Considérons une ressource r, une interface Int connectée à r telle que $r \rightarrow Int$, et la relation d'autorité $a_{<robot,operator>}(r)$ associée à r. Pour que l'affectation entre r et Int ait lieu, il faut que les trois conditions suivantes soient simultanément vérifiées :

- une des transitions $t_{Allocation1}$, $t_{Allocation2}$, $t_{Allocation3}$ ou $t_{Allocation4-Preemption}$ sur r doit être validée ;
- une des transitions $t_{Allocation\_int1}$, $t_{Allocation\_int2}$, $t_{Allocation\_int3}$ sur Int doit être validée ;
- une des transitions $t_{AllocationX}$, $t_{AllocationY}$, $t_{Allocation/PreemptionX}$ ou $t_{Allocation/PreemptionY}$ sur $a_{<robot,operator>}(r)$ doit être validée.
- toutes ces transitions doivent être validées par un marquage de *même couleur* (ou comprenant la couleur neutre) : en effet, ceci permet de garantir que l'agent ayant assigné la ressource r *via* l'interface Int dispose bien des droits suffisants dans la relation d'autorité $a_{<robot,operator>}(r)$. La requête sur Int est matérialisée par un jeton de la couleur *robot* ou *operator* ; de même, les droits des agents sur $a_{<robot,operator>}(r)$ sont aussi représentés en utilisant des jetons des couleurs *robot* et *operator*.

Ces quatre conditions correspondent à la simultanéité de la disponibilité de la ressource r pour être affectée, d'une requête activée sur l'interface assignante Int, et de la validation du droit d'affectation à la ressource pour un des agents du couple ($robot$, $operator$) sur $a_{<robot,operator>}(r)$.

Pour s'assurer de la simultanéïté de la vérification de ces conditions sur les trois réseaux de ressource, interface et relation d'autorité, et pour coordonner le changement d'état des réseaux lors du processus d'affectation, la fusion de transitions est utilisée : toute transition d'affectation $t_{Allocation1}$, $t_{Allocation2}$, $t_{Allocation3}$, $t_{Allocation4-Preemption}$ sur r doit être fusionnée avec une transition d'affectation $t_{Allocation_int1}$, $t_{Allocation_int2}$, $t_{Allocation_int3}$ sur Int et une transition d'affectation $t_{AllocationX}$, $t_{AllocationY}$, $t_{Allocation/PreemptionX}$, $t_{Allocation/PreemptionY}$ sur $a_{<robot,operator>}(r)$.

Dans le cas de la connexion entre r et Int, toutes les transitions d'affectation de r doivent être fusionnées avec toutes les transitions de Int, ce qui donne $4 * 3 = 12$ combinaisons. Dans le cas de la connexion entre r et $a_{<robot,operator>}(r)$, toutes les transitions d'affectation hors préemption de r, c'est-à-dire $t_{Allocation1}$, $t_{Allocation2}$, $t_{Allocation3}$, doivent être fusionnées avec toutes les transitions d'affectation de $a_{<robot,operator>}(r)$ hors préemption, c'est-à-dire $t_{AllocationX}$ et $t_{AllocationY}$. De plus, il faut connecter entre elles les transitions d'affectation concernant la préemption : $t_{Allocation4-Preemption}$ sur r ainsi que $t_{Allocation/PreemptionX}$ et $t_{Allocation/PreemptionY}$ sur $a_{<robot,operator>}(r)$. Cela fait ainsi $3 * 2 + 1 * 2 = 8$ combinaisons de transitions à fusionner entre r et $a_{<robot,operator>}(r)$. Lorsque l'on combine ces ensembles simultanément sur r, Int et $a_{<robot,operator>}(r)$, on obtient ainsi $12 * 8 = 96$ ensembles de transitions fusionnées pour la relation d'affectation.

La figure 5.11 illustre ce principe : r est une ressource partageable, Int représente une dépendance en exécution, et la relation d'autorité est telle que $a_{<robot,operator>}(r) = (Access, Access)$. Les transitions encadrées en vert sont les transitions d'affectation pour chaque réseau ; les transitions en rouge sont les transitions validées. Sur l'exemple, Int porte une requête de l'agent $robot$, matérialisée par un jeton de couleur rouge sur la figure. La transition $t_{Allocation1}$ sur r est validée par un marquage de couleur neutre, la transition $t_{Allocation\_int2}$ sur Int est validée par un marquage de couleur $robot$, et la transition $t_{AllocationX}$ sur $a_{<robot,operator>}(r)$ est validée par un marquage de couleur $robot$: l'affectation peut avoir lieu, par le tir simultané des transitions mentionnées. Le marquage résultant est le suivant : sur l'interface on a le marquage$\{SatisfyingRequest, Exec-Dep\} \subseteq \mathrm{M_{Interface}}$; sur la ressource, on obtient $\{Present, Available, Allocated, Shareable, Users<=1, \#Users\} \subseteq \mathrm{M_{Ressource}}$; sur le réseau d'autorité on obtient le marquage $\{ColorX1, ColorX2, ColorY1, ColorY2, AccessX, AccessY, NoPreemptabilityX, NoPreemptabilityY, \} \subseteq \mathrm{M_{Autorite}}$.

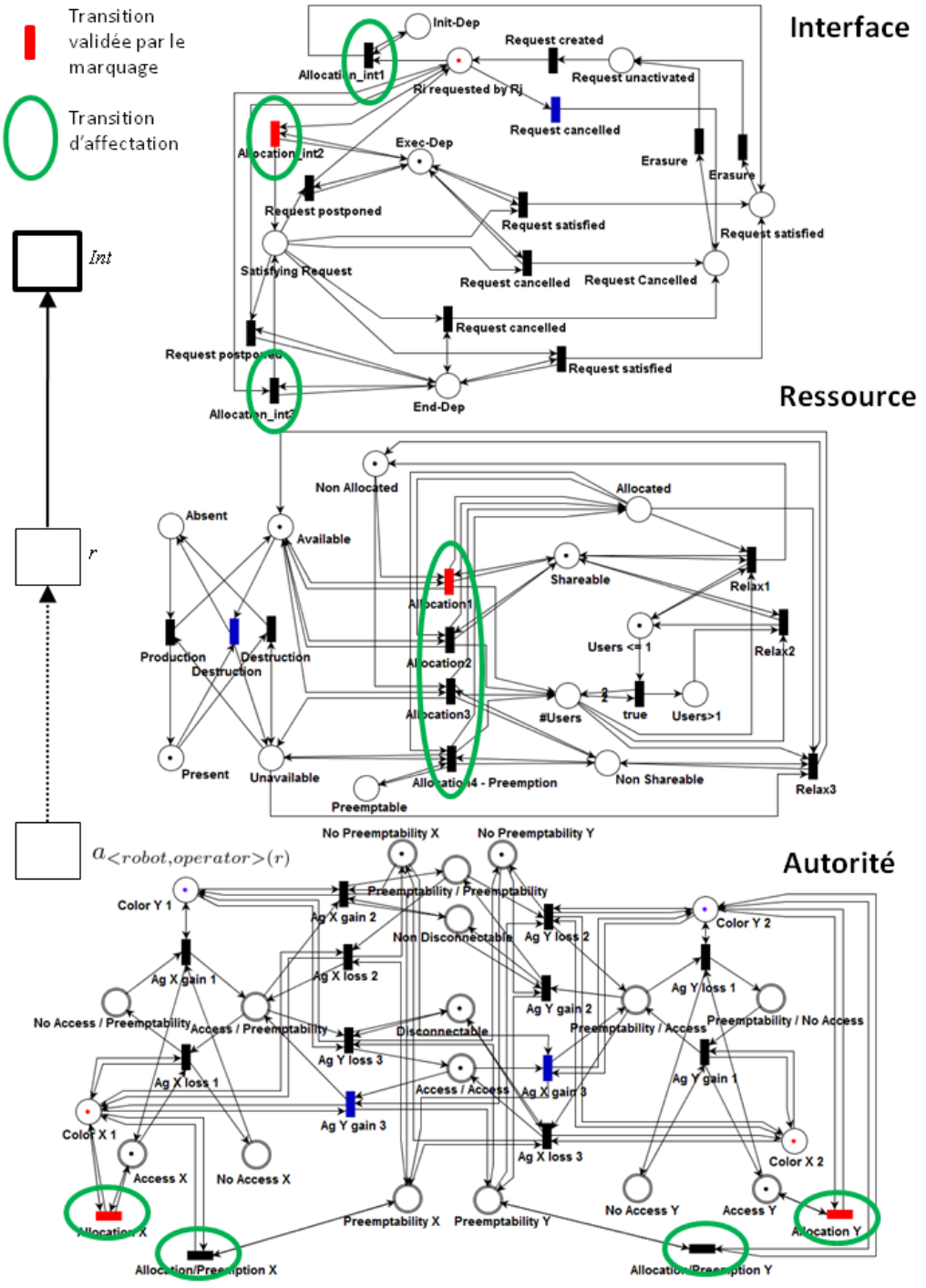

FIGURE 5.11: Fonctionnement de l'affectation entre une ressource, une interface, une relation d'autorité

2.3 Évolution conjointe des réseaux de ressource et d'interface

La modification de l'état d'une ressource peut avoir des répercussions sur l'état d'autres ressources ou interfaces, en fonction des relations de dépendance matérialisées par les connexions des ressources et interfaces dans le graphe de ressources. Par exemple, lorsqu'une ressource représentant un but est produite (signifiant que le but a été atteint), l'ensemble des ressources requises peut être relâché, l'ensemble des requêtes sur les interfaces représentant les liens de dépendance peut être marqué comme $Requestsatisfied$. Dans le cas des liaisons « Ressource vers Interface » et « Interface vers Ressource », la modification de l'état d'un réseau se propage à l'autre réseau par le passage d'événements, conduisant à l'évolution conjointe de ces réseaux. Dans le cas où plusieurs de ces réseaux sont connectés à la suite dans le graphe de ressources, l'évolution du modèle se propage en cascade au fur et à mesure que les événements sont générés et traités par le joueur de Petri : c'est d'ailleurs la raison pour laquelle les événements *internes*, générés par le tir de transitions des réseaux du modèle, sont prioritaires sur les événements *externes*, générés par les fonctions de l'architecture externes au modèle : l'état du modèle de l'autorité doit d'abord se stabiliser avant d'être capable de traiter un événement extérieur.

2.3.1 Connexion d'une ressource à une interface

Nous nous intéressons à la liaison « Ressource vers Interface » qui correspond à la notation $r \rightarrow Int, r \in \mathcal{R}, Int \in \mathcal{I}$. Les liens entre ces deux réseaux pour le sens considéré sont les suivants :

- (en jaune sur la figure 5.12) le tir d'une des transitions $t_{Destruction}$ sur r envoie un événement $e_{DestroyDependantResource}$ à destination de la transition $t_{RequestCancelled}$ sur Int présentant la propriété *Exec-Dep* qui représente la dépendance en exécution de r' envers r *via* Int : si r disparaît alors qu'affectée à r', r' doit être détruite ;
- (en bleu sur la figure 5.12) le tir d'une des transitions $t_{Requestsatisfied}$ ou $t_{Requestcancelled}$ sur Int présentant la propriété *Exec-Dep* ou *End-Dep* envoie un événement $e_{ReleaseRequestedResource}$ à destination d'une des transitions t_{Relax1} ou t_{Relax2} sur r ; ceci correspond à la relâche de la ressource requise r, dans le cas où le lien de dépendance avec r' est satisfait ou annulé.

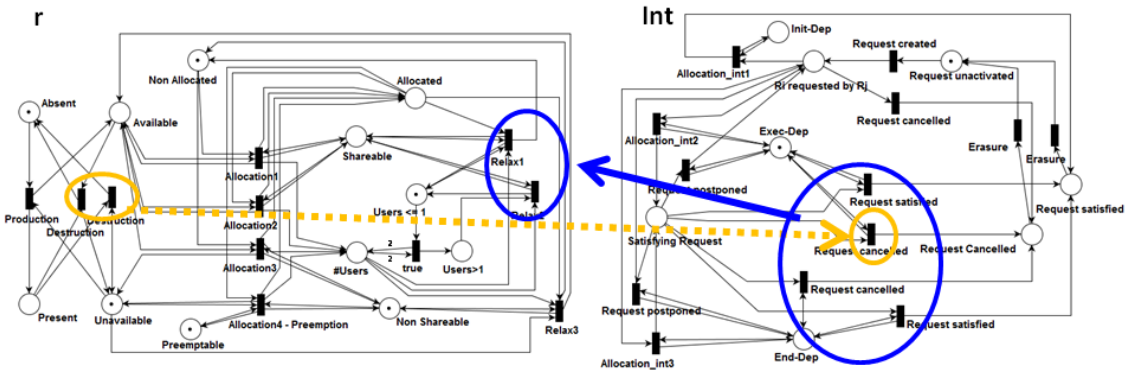

FIGURE 5.12: Lien entre le réseau de ressource requis r et l'interface assignante Int

2.3.2 Connexion d'une interface à une ressource

Nous nous intéressons à la liaison « Interface vers Ressource » qui correspond à la notation $Int \rightarrow r', r' \in \mathcal{R}, Int \in \mathcal{I}$. Les liens entre ces deux réseaux pour le sens considéré sont les suivants :

- (en vert sur la figure 5.13) le tir de la transition $t_{RequestCancelled}$ sur Int présentant la propriété *Exec-Dep* envoie un événement $e_{DestroyDependantResource}$ à destination de la transition $t_{Destruction}$ sur r' ; ceci représente le fait que la ressource r' requise par r a disparu alors qu'affectée à r'. Du fait de la dépendance en exécution, r' doit par conséquent être détruite ;
- (en jaune sur la figure 5.13) le tir des transitions t_{Relax1} ou t_{Relax3} sur r' envoie un événement $e_{PropagatingRelease}$ à destination d'une transition $t_{RequestSatisfied}$ sur Int présentant la propriété *Exec-Dep* ou *End-Dep* ; ceci permet la relâche de ressources en cascade dans le cas où le lien de dépendance est resté toujours activé : r' a été relâchée et ne nécessite plus r, la requête sur Int en est donc modifiée (elle est considérée comme satisfaite) ;
- (en rouge sur la figure 5.13) le tir de la transition $t_{Allocation\_Int1}$ ou de la transition $t_{Allocation\_Int2}$ sur Int envoie un événement $e_{productionDependentResource}$ à destination de la transition $t_{produtionn}$ sur r' ; ceci représente le fait que l'activation du lien de dépendance *via* l'affectation entraîne la production de la ressource dépendante r' pour les propriétés de dépendance *Init-Dep* et *Exec-Dep* (par exemple la tâche r' est lancée dès que la précondition r est vérifiée) ;
- (en bleu sur la figure 5.13) le tir de la transition $t_{production}$ sur r' envoie un événement $e_{requestSatisfied}$ à destination de la transition $t_{RequestSatisfied}$ sur Int présentant la propriété *End-Dep* ; ceci représente la satisfaction et la fin du lien de dépendance lorsque la ressource dépendante r est produite (par exemple, le but r' est atteint, indiquant la fin de la tâche r).

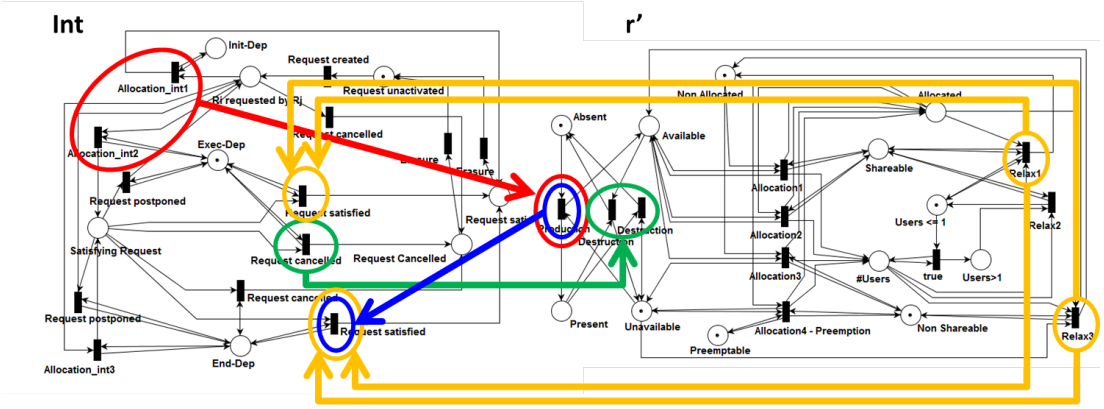

FIGURE 5.13: Lien entre le réseau d'interface requise Int et la ressource dépendante r'

3 Mise en œuvre informatique

Les caractéristiques des micro- et macro-modèles étant établies, il faut pouvoir mettre en œuvre les réseaux de Petri correspondants avec toutes leurs interconnexions.

Le logiciel MNMS[3] a été réalisé afin de pouvoir manipuler un grand nombre de réseaux de Petri et créer des graphes de ressources en s'affranchissant de la création et de la connexion des réseaux de Petri sous-jacents. Ce programme automatise ainsi toute la construction et la gestion du micro-modèle, permettant au concepteur de se concentrer sur la réalisation de modèles uniquement au niveau macro.

Voici le détail de certaines problématiques liées à la mise en œuvre du modèle de l'autorité et de son exécution, ainsi que les solutions techniques adoptées pour les résoudre.

3.1 Fusion d'ensembles de transitions

3.1.1 Problématique

Tel que cela a été présenté dans la section 2.2, l'affectation nécessite de constituer des ensembles de transitions fusionnées englobant toutes les transitions liées à l'affectation sur les réseaux de ressource, d'interface en aval, et de relation d'autorité associée : chacune des transitions d'affectation sur la ressource doit être fusionnée avec une transition d'affectation sur le réseau d'interface et une transition d'affectation sur le réseau d'autorité, constituant ainsi un ensemble de trois transitions fusionnées. Dans le cas où il n'y a qu'une ressource requise, une interface et une relation d'autorité et en faisant

3. Multiple (Petri) Nets Management System

toutes les associations possibles le nombre d'ensembles de transitions fusionnées $n_{ensembles}$ est de 96, comme l'illustre la figure 5.14. Ceci permet de s'assurer que, simultanément, la ressource r peut être affectée, la requête sur Int est active et l'agent à l'origine de la requête a les droits d'accès à r définis par $a_{<robot,operator>}(r)$.

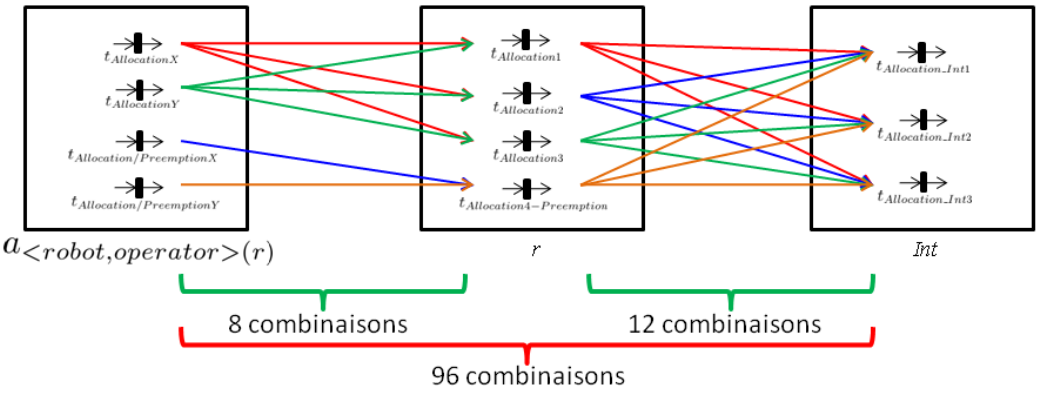

FIGURE 5.14: Ensemble des transitions à fusionner pour réaliser l'affectation entre une ressource, une interface, une relation d'autorité.

Cependant, le nombre d'ensembles de transitions à fusionner grandit fortement dès qu'il y a plus d'une ressource requise. La figure 5.15 illustre le cas suivant : une ressource r_{dep} dépend de trois ressources $r1$, $r2$ et $r3$ à qui elle est connectée *via* les interfaces $Int1$, $Int2$ et $Int3$. Le type des relations de dépendance importe peu pour la problématique. Afin que l'affectation de $r1$, $r2$ et $r3$ ait lieu, toutes ces ressources doivent être simultanément présentes, toutes les requêtes actives sur $Int1$, $Int2$ et $Int3$ et créées par le même agent, celui-ci devant avoir les droits nécessaires sur les relations d'autorité associées.

Le nombre inscrit dans chaque réseau indique le nombre de transitions concernées par l'affectation. Pour chaque triplet de réseaux ressource, interface, relation d'autorité encadré en pointillé, le nombre d'ensembles de transitions fusionnées est de 96. Cependant, chacun de ces ensembles devant être fusionné avec chacun des ensembles similaires des autres groupes, le nombre total d'ensembles de transitions à fusionner est de $96^3 = 884736$.

Ainsi, pour une ressource dépendante r_{dep}, en considérant qu'il n'y a qu'une seule relation d'autorité par ressource requise (c'est-à-dire qu'il n'y a qu'un seul couple d'agents intervant sur la ressource), le nombre d'ensembles de transitions fusionnées à créer est égal à $96^{n_{RscRequises}}$, (avec $n_{RscRequises}$ le nombre de ressources requises par r_{dep}). Le nombre d'ensembles augmente ainsi en exponentielle de $n_{RscRequises}$: ce nombre est trop élevé pour que tous ces groupes de fusion soient créés en pratique, ce qui pourrait poser des problèmes d'utilisation mémoire du programme ou de performances.

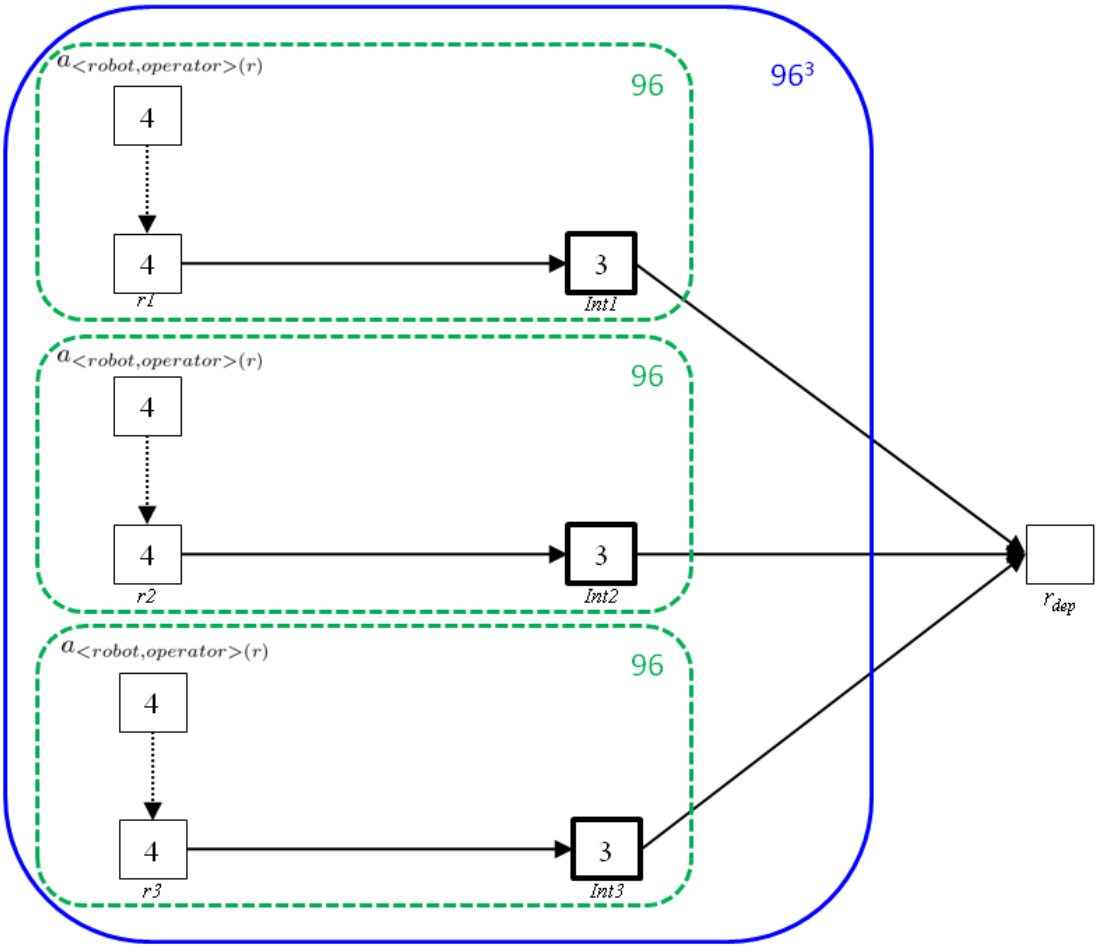

FIGURE 5.15: Ensemble des transitions à fusionner pour réaliser l'affectation avec plusieurs ressources requises, leur interface et relation d'autorité.

3.1.2 Solution adoptée

La solution consiste à ne pas générer tous les ensembles de transitions fusionnées, mais à les « simuler », en utilisant des structures de niveau supérieur à celui des transitions, gérant des *ensembles* de transitions. Ainsi, en pratique, nous avons utilisé trois types d'ensembles :

- le *groupe de fusion* est un ensemble de transitions (de 2 à $k, k \in \mathbb{N}$) fusionnées ;
- l'*agrégat* est un ensemble de groupes de fusion (de 2 à $l, l \in \mathbb{N}$) ;
- le *groupe d'agrégats* est un ensemble d'agrégats (de 2 à $m, m \in \mathbb{N}$).

Chacune de ces structures, à la manière d'une transition validée ou non par son marquage, est soit dans un état *validé*, soit dans un état *non-validé*, cet état étant associé à une couleur :

- le groupe de fusion est validé si *toutes* les transitions fusionnées dans le groupe sont validées par un marquage de la même couleur ;
- l'agrégat est validé si *au moins un* des groupes de fusion qu'il contient est validé ;
- le groupe d'agrégats est validé si *tous* les agrégats sont validés par la même couleur.

La figure 5.16 illustre comment ces structures sont utilisées.

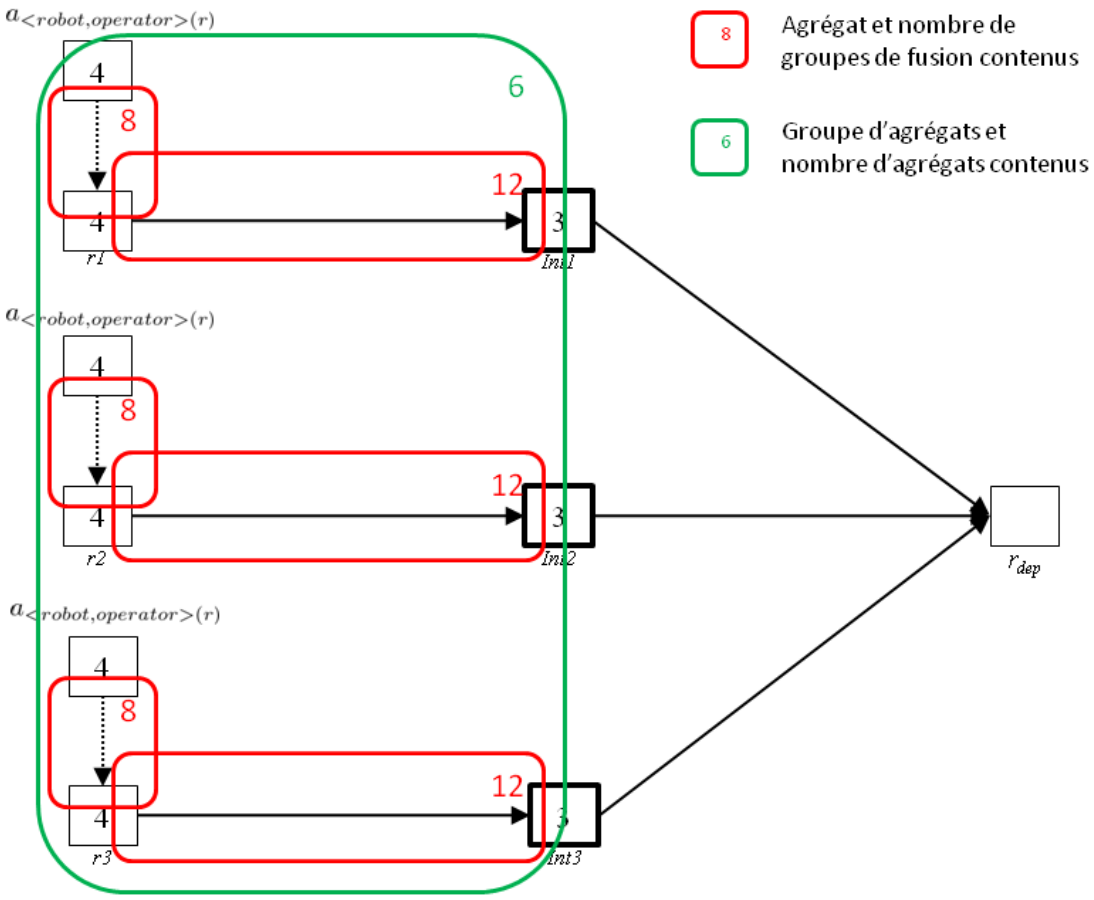

FIGURE 5.16: Ensemble des transitions à fusionner pour réaliser l'affectation avec plusieurs ressources requises, leur interface et relation d'autorité.

Les transitions sont fusionnées deux à deux dans des groupes de fusion pour chaque connexion ressource-interface (12 groupes de fusion) et autorité-ressource (8 groupes de fusion). Chacun des ensembles de groupes de fusion obtenus est placé dans un agrégat. Sur l'exemple, pour trois ressources requises on obtient un total de six agrégats. Finalement, ces agrégats sont placés dans un seul et même groupe d'agrégats.

Ainsi, pour que l'affectation ait lieu, il faut que :
- le groupe d'agrégats soit validé,
- ce qui signifie que tous les agrégats sont validés par la même couleur,
- ce qui implique que pour chaque connexion ressource-interface et autorité-ressource, au moins un groupe de fusion, c'est-à-dire une paire de transitions d'affectations, est validé par la couleur considérée.

Si le groupe d'agrégats est validé, alors l'affectation peut avoir lieu : toutes les transitions validées (une et une seule paire de transitions validées par agrégat) sont simultanément tirées. Dans le cas de l'exemple, avec trois ressources requises, le nombre total de structures créées est de $(8 + 12) * 3 + 6 + 1 = 67$.

3.2 Algorithme du joueur de Petri

Le joueur de Petri a un rôle central dans l'architecture : c'est lui qui exécute les réseaux de Petri et fait que le modèle d'autorité évolue, sur la base des événements générés par les fonctions externes au modèle. En dehors des règles de base de franchissement des transitions, le joueur de Petri du contrôleur de la dynamique de l'autorité doit tenir compte de la réception et de l'émission d'événements, des couleurs de jeton, mais également des structures d'ensembles de transitions fusionnées. L'algorithme du joueur de Petri (cf algorithme en pseudo-code 1 en page suivante) tel qu'il a été implémenté dans le logiciel MNMS concerne le traitement d'un événement e pris au sommet de la pile d'événements de l'algorithme ; en effet, pour l'implémentation, les événements transmis au joueur de Petri sont placés dans une pile et sont traités les uns après les autres dans leur ordre d'arrivée.

Le principe du joueur de Petri est le suivant : lorsqu'un événement e arrive, les transitions réceptrices de l'événement sont répertoriées, parmi celles-ci on identifie celles qui sont validées par un marquage d'une couleur similaire à celle(s) portée(s) par l'événement (plusieurs sont possibles) : on obtient la liste $listeTransitionsReceptrices$. Pour chacune d'entre elles, on regarde de quelle structure d'ensembles de fusion la transition fait partie (groupe d'agrégats, groupe de fusion, ou aucun) pour, si cela est possible, tirer la structure de plus haut niveau : ainsi on vérifie d'abord si la transition fait partie d'un groupe d'agrégats qu'il serait possible de tirer, puis d'un groupe de fusion ;

sinon la transition est tirée individuellement. Toute structure ou transition peut être tirée autant de fois qu'il y a de couleurs portées par l'événement, tant que la ou les transitions sous-jacentes sont validées par un marquage de la couleur correspondante. Toutefois, avant chaque tir pour une couleur donnée c, on revérifie si la structure est bien validée par cette couleur ; en effet, en raison de la présence de la couleur *neutre*, cette couleur s'associant à toute autre couleur pour valider un marquage, l'ensemble des couleurs validant une transition ou structure évolue au fur et à mesure que les jetons de la couleur *neutre* sur les marquages sont utilisés par les tirs successifs suivant les différentes couleurs.

Une fois que cette opération est faite, suite au traitement de l'événement e et des modifications que cela a pu engendrer sur les marquages du modèle, on recherche toutes les transitions du modèle *non réceptrices d'événements* qui sont validées par ces nouveaux marquages ; celles-ci sont alors tirées *via* la fonction $tirerToutesTransitionsValideesNonReceptrices$. Enfin, après ces opérations, l'événement e est retiré de la pile, son traitement est terminé.

Algorithme 1: Algorithme du joueur de Petri du logiciel MNMS

Données : événement e
Résultat : évolution du modèle consécutive au traitement de e

```
1  pour chaque transition t réceptrice (validée ou non) de e faire
2      pour chaque couleur c portée par l'événement e faire
3          si t est validée par son marquage de couleur c alors
4              ajouter t à la liste listeTransitionsReceptrices;
5          fin
6      fin
7  fin
8  pour chaque transition t de la liste listeTransitionsReceptrices faire
9      si t fait partie de groupes d'agrégats alors
10         pour chaque groupe d'agrégat g faire
11             Liste de couleurs listeCouleursATirer = obtenirCouleursInter(g, e);
12             pour chaque couleur c de listeCouleursATirer faire
13                 si g est validée par la couleur c alors  tirer g pour la couleur c;
14             fin
15             si taille(listeCouleursATirer) > 0 alors Sortir de la boucle
16         fin
17     fin
18     sinon si  t fait partie d'un groupe de fusion alors
19         pour chaque groupe de fusion f faire
20             Liste de couleurs listeCouleursATirer = obtenirCouleursInter(f, e);
21             pour chaque couleur c de listeCouleursATirer faire
22                 si f est validée par la couleur c alors  tirer f pour la couleur c;
23             fin
24             si taille(listeCouleursATirer) > 0 alors Sortir de la boucle
25         fin
26     fin
27     sinon
28         Liste de couleurs listeCouleursATirer = obtenirCouleursInter(t, e);
29         pour chaque couleur c de listeCouleursATirer faire
30             si t est validée par la couleur c alors  tirer t pour la couleur c;
31         fin
32     fin
33 fin
34 tirerToutesTransitionsValideesNonReceptrices();
35 Enlever l'événement traité de la pile;
```

La fonction suivante *obtenirCouleursInter*, utilisée par l'algorithme du joueur de Petri (cf algorithme en pseudo-code 2), prend en paramètre une structure de fusion (groupe d'agrégats, groupe de fusion, ou une transition) s et un événement e. La fonction renvoie l'intersection des couleurs portées par l'événement e et des couleurs qui valident la structure s, c'est-à-dire les couleurs qui peuvent tirer la structure s.

Algorithme 2: Algorithme de la fonction *obtenirCouleursInter*

Données : structure de fusion(transition, groupe de fusion, groupe d'agrégats) s, événement e
Résultat : liste de couleurs *listeCouleursATirer*

```
1 listeCouleursATirer = obtenirCouleursATirer(s);
2 listeCouleursEvenement = e.getCouleursAssociees();
3 listeCouleursATirer =
  Intersection(listeCouleursATirer, listeCouleursEvenement);
4 retourner listeCouleursATirer
```

Enfin, la fonction suivante *tirerToutesTransitionsValideesNonReceptrices* est également utilisée par l'algorithme du joueur de Petri (cf algorithme en pseudo-code 3). Son rôle consiste à identifier toutes les transitions du modèle *non réceptrices d'événements* qui sont validées après évolution du modèle et à les tirer. Cependant, ce faisant, cette fonction contribue également à faire évoluer les marquages du modèle : il faut donc vérifier à nouveau qu'il ne subsiste pas de transition non réceptrice validée et prête à être tirée. De plus, cette fonction tient également compte, comme l'algorithme du joueur de Petri, des structures de fusion dont une transition à tirer peut faire partie. Respectivement, on vérifie si la transition fait partie d'un groupe d'agrégats qui serait validé, puis d'un groupe de fusion ; sinon la transition est tirée individuellement. La fonction prend fin lorsque le modèle se « stabilise » et n'évolue plus, c'est-à-dire lorsqu'il ne reste plus aucune transition à tirer dans le modèle dans l'état courant.

Algorithme 3: Algorithme de la fonction *tirerToutesTransitionsValideesNonReceptrices*

Données : rien
Résultat : tir de toutes les transitions du modèle validées

```
1  listeTransitionsATirer = obtenirToutesTransitionsValideesNonReceptrices();
2  tant que taille(listeTransitionsATirer) > 0 faire
3      pour chaque transition transitionATirer de listeTransitionsATirer faire
4          listeCouleursATirer = obtenirCouleursATirer(transitionATirer) si
           transitionATirer fait partie d'un groupe d'agrégats g alors
5              pour chaque couleur c de listeCouleursATirer faire
6                  si g est validé par la couleur c alors  tirer g pour la couleur c;
7              fin
8          fin
9          sinon
10             si  transitionATirer fait partie d'un groupe de fusion f alors
11                 pour chaque couleur c de listeCouleursATirer faire
12                     si f est validé par la couleur c alors  tirer f pour la couleur c;
13                 fin
14             fin
15             sinon
16                 pour chaque couleur c de listeCouleursATirer faire
17                     si transitionATirer est validée par la couleur c alors  tirer
                       transitionATirer pour la couleur c;
18                 fin
19             fin
20         fin
21     fin
22     listeTransitionsATirer =
       obtenirToutesTransitionsValideesNonReceptrices();
23 fin
```

Chapitre 6

Application

Ce chapitre présente la mise en application des concepts développés dans les chapitres 3 et 4 sur la base des moyens techniques présentés dans le chapitre 5. Un scénario de mission faisant intervenir des conflits de défaillance et de préemption est proposé. Les différents macro- (et micro-) modèles correspondant aux différentes phases sont construits, afin de permettre l'exécution du scénario dans sa totalité. L'environnement technique permettant le jeu de ce scénario, incluant la simulation de différents éléments de l'architecture du système, est présenté. Enfin, la connexion de ce système à une véritable plate-forme robotique est discutée.

1 Scénario

Un scénario, destiné à mettre en pratique l'ensemble des concepts proposés dans le cadre de cette thèse, a été écrit sur la base d'une mission effectuée par un robot terrestre supervisé par un opérateur humain situé à distance. L'objectif de la mission est de réaliser l'identification de cibles placées sur le sol, ces cibles étant situées dans une zone accessible uniquement par le robot. Les différentes phases constituant ce scénario sont inspirées des observations réalisées sur l'expérimentation préliminaire présentée au chapitre 1, en particulier en ce qui concerne le comportement attendu de l'opérateur.

Le scénario nominal est composé des phases suivantes :
- partant de son point de base, le robot réalise en navigation automatique son trajet jusqu'à la zone d'identification des cibles ;
- une fois sur place, le robot rend la main à l'opérateur afin que celui-ci procède manuellement à l'identification des cibles. La téléopération permet de guider finement les déplacements du robot pour la tâche d'identification ;
- lorsque l'identification est réalisée, l'opérateur ramène le robot à la base.

Le scénario avec conflits est le suivant :

- lors de la phase d'identification des cibles, un premier aléa survient : le niveau de la batterie du robot est considéré comme insuffisant pour mener la mission dans son intégralité, et le robot doit en priorité retourner à sa base, ce qui revient à supprimer certaines phases de mission pour accomplir uniquement la dernière, le retour à la base. Du point de vue du modèle de l'autorité, il s'agit de l'apparition d'un conflit de défaillance sur l'une des ressources du modèle ;
- après le processus de résolution du conflit de défaillance par le robot, le robot retourne à la base de manière automatique, l'énergie disponible de la batterie étant supposée suffisante pour cette tâche. Nous présenterons deux résolutions possibles au conflit, avec ou sans modification des relations d'autorité ;
- cependant, alors que le robot a commencé son retour vers la base, l'opérateur humain cherche à reprendre le contrôle manuel du véhicule ; l'évolution de la situation va alors dépendre de la solution apportée au conflit à l'étape précédente.

2 Le logiciel MNMS

Le logiciel MNMS [1] que nous avons développé permet de concevoir et d'exécuter le modèle de l'autorité pour une mission donnée, et de simuler les fonctions de l'architecture décisionnelle du robot pour la résolution des conflits.

2.1 Édition et jeu des modèles

Le logiciel MNMS permet de concevoir le modèle de l'autorité et de l'exécuter, grâce à son joueur de Petri intégré. De plus, il réalise la fonction de superviseur du contrôleur de l'autorité, en effectuant les tâches de détection de conflit, communication avec le planificateur, suivi de situation, commande, interface utilisateur. Par ailleurs il procède à l'ajout, retrait et changement d'état des ressources, interfaces et relations d'autorité du modèle. La figure 6.1 présente une capture d'écran de MNMS en mode d'exécution.

Le modèle représenté sur cette capture d'écran de MNMS est le graphe de ressources matérialisant les dépendances issues du plan d'exécution prévu dans notre scénario d'application. Celui-ci consiste en l'atteinte par le robot de deux points de navigation WP1 puis WP2 de manière automatique avant un passage de relais à l'opérateur pour une tâche d'identification de cibles au sol et le retour à la base.

1. Multiple (Petri) Nets Management System

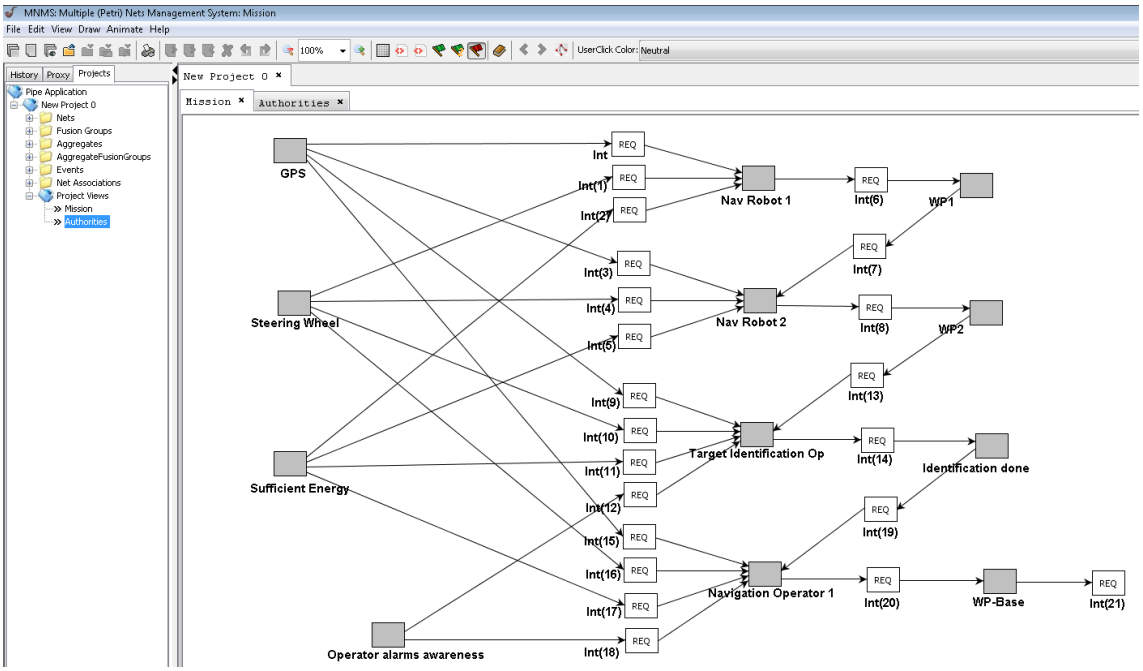

FIGURE 6.1: Interface MNMS à l'initialisation du scénario de l'application

2.2 Simulation des fonctions décisionnelles

Le développement des fonctions de l'architecture décisionnelle du robot (voir chapitre 5 ne rentre pas dans le cadre de la thèse. Nous simulons donc ces fonctions dans MNMS afin de gérer les entrées / sorties propres à chacune du point de vue du contrôleur de la dynamique de l'autorité, pour que celui-ci puisse fonctionner (voir figures 6.2 à 6.5).

Les fonctions simulées sont les suivantes :

- le suivi de situation (voir figure 6.2) ; le programme permet l'envoi des événements de production et de destruction pour toute ressource présente dans le modèle de l'autorité ;
- l'interface utilisateur (voir figure 6.3) : ce programme permet l'envoi des graphes de ressources correspondant aux actions de l'opérateur sur son interface (sur la base d'un modèle de tâches associé à chaque action sur l'interface). L'interface utilisateur permet de déclencher les événements de prise et relâche du joystick de contrôle manuel du déplacement du robot ;

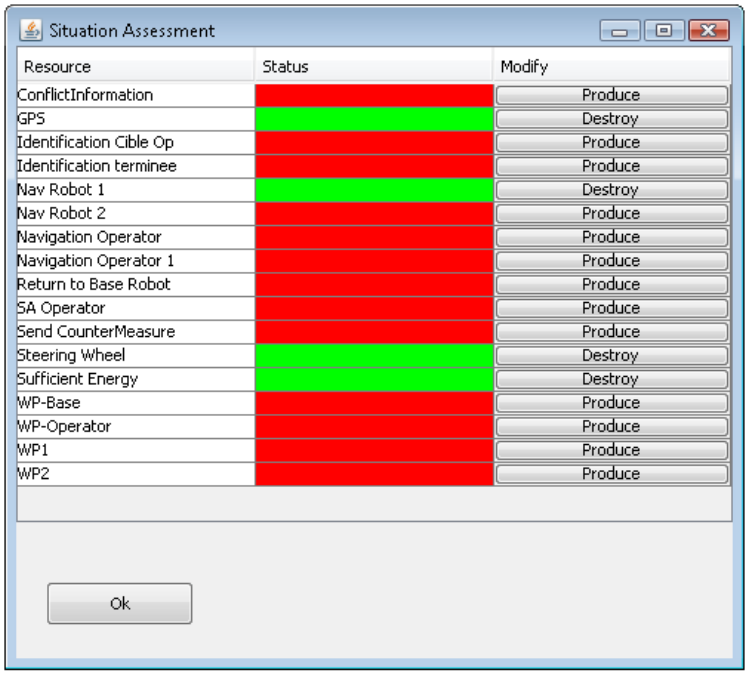

FIGURE 6.2: Interface de la fonction suivi de situation

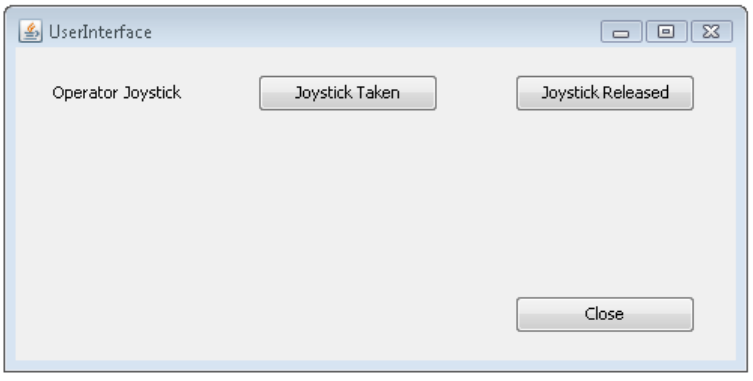

FIGURE 6.3: Interface de la fonction interface utilisateur

– le planificateur (voir figure 6.4) ; en lieu et place d'un véritable planificateur, le programme contient une banque de procédures à instancier pour réaliser un but donné, sous forme de graphes de ressources pré-écrits. Ceci s'apparente à une liste de « recettes » dont dispose le robot, de manière similaire aux recettes utilisées par la planification de type hiérarchique [Nau *et al.*, 2001]. Ainsi, en fonction des ressources disponibles, et de la ressource identifiée comme but à poursuivre, ce programme renvoie un graphe de ressources solution, s'il en existe un ;

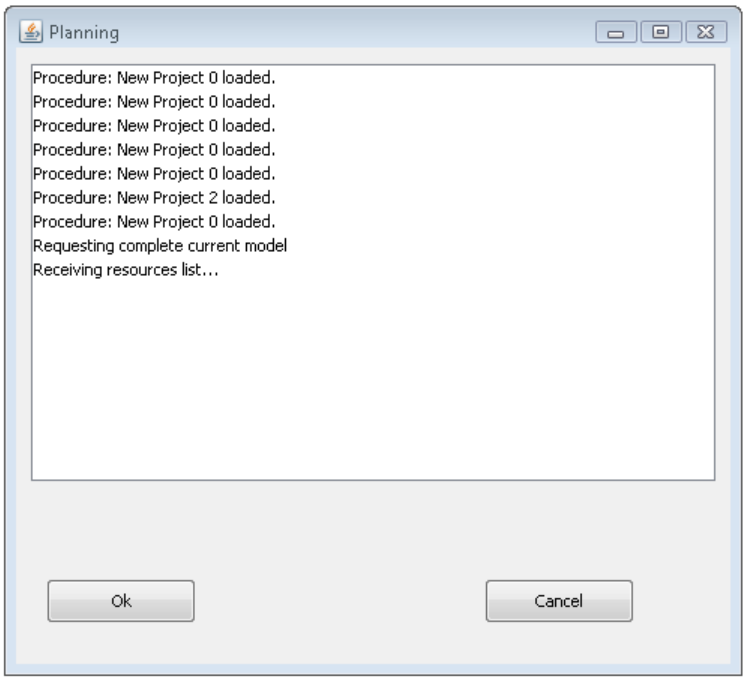

FIGURE 6.4: Interface de la fonction planificateur

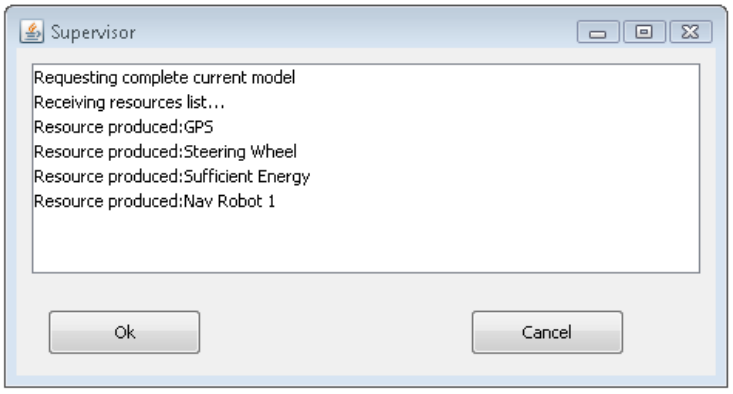

FIGURE 6.5: Interface de la fonction exécutive

– la fonction exécutive (voir figure 6.5) : ce programme sert de destinataire aux événements en provenance du contrôleur de la dynamique de l'autorité, correspondant à l'arrêt / lancement de tâches à réaliser par le robot. L'interface de cette fonction affiche les événements (ordres destinés aux algorithmes des tâches) qui lui parviennent.

3 Déroulement de la mission

Pour le scénario décrit en paragraphe 1, nous présentons le modèle de l'autorité et ses évolutions successives au sein du contrôleur de la dynamique de l'autorité. Le modèle est modifié au fur et à mesure du déroulement de la mission, en particulier à l'occasion de la survenue d'aléas et des reconfigurations consécutives. L'ensemble des figures présentées dans ce paragraphe sont des captures d'écran du logiciel MNMS.

3.1 Phase nominale

La figure 6.6 représente le modèle à l'initialisation de la mission : il correspond au plan d'exécution prévu dans le cas nominal. La première tâche à réaliser est une tâche de navigation réalisée par le robot - ressource *Nav Robot 1* - pour aller au point de passage 1 - ressource *WP1*, suivie d'une tâche de navigation similaire - ressource *Nav Robot 2* - pour aller au point de passage 2 - ressource *WP2*. La dépendance entre les tâches de navigation et les points de passage associés est de type *End-Dep*, le point de passage étant le produit de la tâche de navigation. Ces deux tâches nécessitent la présence des ressources physiques *GPS* (matériel donnant la position), *Steering Wheel* (commande de cap) et *Sufficient Energy* (quantité d'énergie embarquée sur le véhicule suffisante pour effectuer le plan prévu) avec une dépendance de type *Exec-Dep*.

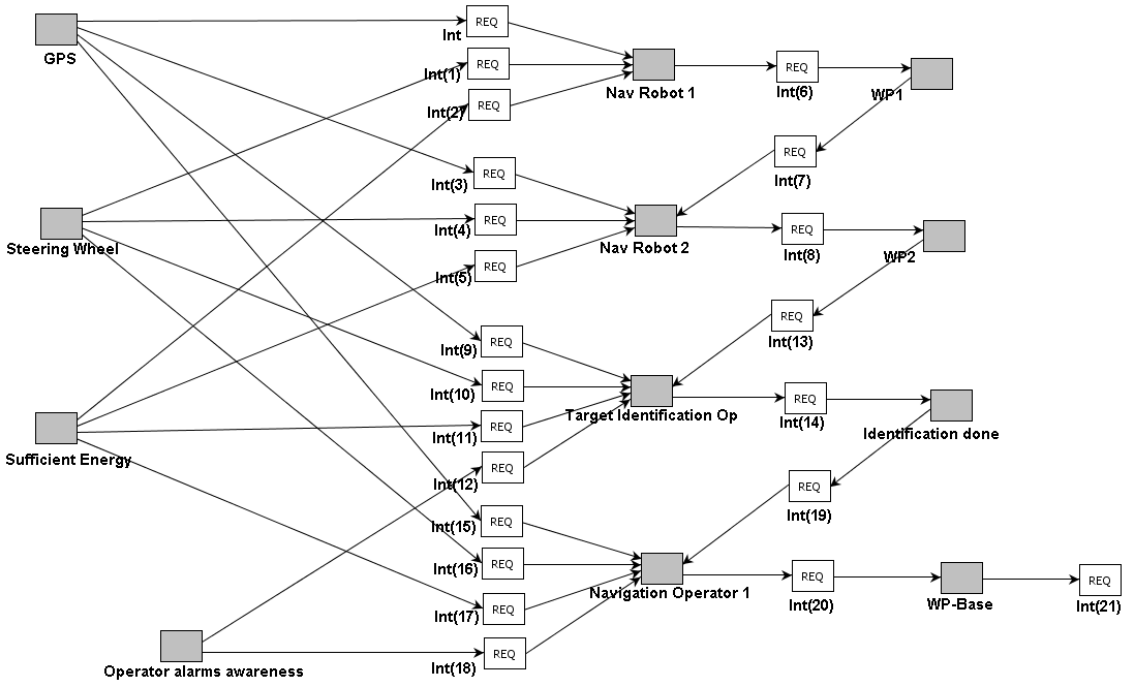

FIGURE 6.6: Modèle à l'initialisation de la mission

Les propriétés des ressources sont les suivantes :
- les ressources *GPS, Sufficient Energy, Operator alarms awareness, WP1, WP2, Identification done, WP-Base* sont partageables ;
- les ressources *Nav Robot 1, Nav Robot 2, Target identification Op, Navigation Operator 1* sont non-partageables, non-préemptables ;
- la ressource *Steering Wheel* est non-partageable et préemptable.

Les propriétés des interfaces sont les suivantes :
- les interfaces *Int, Int(1), Int(2), Int(3), Int(4), Int(5), Int(9), Int(10), Int(11), Int(12), Int(15), Int(16), Int(17), Int(18)* représentent une dépendance de type $Exec{-}Dep$;
- les interfaces *Int(6), Int(8), Int(14), Int(20)* représentent une dépendance de type $End{-}Dep$;
- les interfaces *Int(7), Int(13), Int(19), Int(21)* représentent une dépendance de type $Init{-}Dep$.

Seule $Int(21)$ est telle que $\phi(Int(21)) = operator$, c'est-à-dire que l'interface déterminant le but du plan représenté sous forme de graphe de ressources a été créée par l'opérateur. Toutes les autres interfaces *int* ont été assignées initialement par le robot, c'est-à-dire que $\phi(int) = robot$.

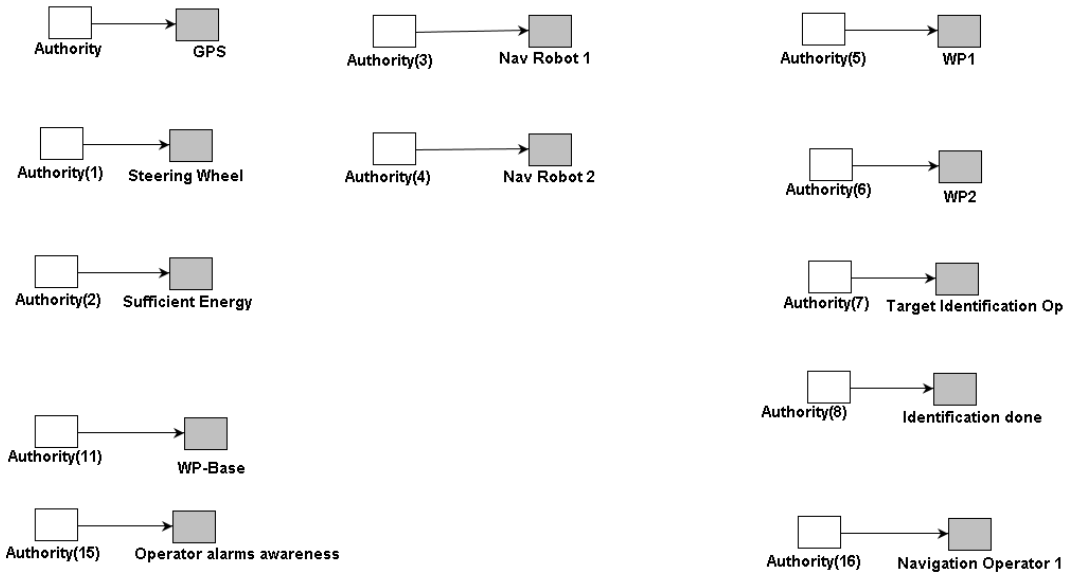

FIGURE 6.7: Relations d'autorité du modèle en début de mission

La figure 6.7 montre les relations d'autorité sur les ressources du modèle (sur une vue séparée) au moment de son initialisation.

Seule la ressource $SteeringWheel$ est telle que $a_{<robot,operator>}(SteeringWheel) = (Access, Preemptability)$: l'opérateur dispose de la préemptabilité vis-à-vis du robot sur cette ressource. Pour toutes les autres ressources r du modèle, les relations d'autorité sont telles que $a_{<robot,operator>}(r) = (Access, Access)$: le robot et l'opérateur ont un droit égal d'accès à chacune des ressources du modèle. Cela implique que le robot peut réaliser l'assignation de ces ressources entre elles (i.e. créer les interfaces les reliant), ce qui est justement le cas sur le graphe de ressources initial. Pour toutes les ressources de cet ensemble qui sont partageables, ou bien non-partageables et non-préemptables, les relations d'autorité associées sont posées de la même manière : en effet, même si le droit de préemption ne peut s'appliquer à ces ressources, à l'extrême il serait toujours possible d'atteindre les états de relation d'autorité tels que $a_{<robot,operator>}(r) = (NoAccess, Access)$ et inversement.

La figure 6.8 présente le modèle de l'autorité juste après le démarrage de la mission, lorsque le suivi de situation a effectué la mise à jour des ressources : les ressources *GPS, Steering Wheel, Sufficient Energy* et *Operator alarms awareness* sont marquées comme présentes (en vert sur la figure). La figure 6.9 représente l'état du modèle immédiatement après : dès que les ressources *GPS*, *Steering Wheel* et *Sufficient Energy* sont simultanément présentes, elles sont affectées à la ressource *Navigation Robot 1* (les ressources affectées apparaissent en bleu), qui est alors produite (ceci correspond à l'envoi à la fonction exécutive de l'ordre de lancement de la tâche correspondante). Toutes les opérations d'affectation sont permises par les relations d'autorité : toutes les interfaces $Int, Int(1)$ et $Int(2)$ ont été créées par le robot, qui dispose du droit *Access* vis-à-vis de l'opérateur sur les ressources *GPS*, *Steering Wheel* et *Sufficient Energy*.

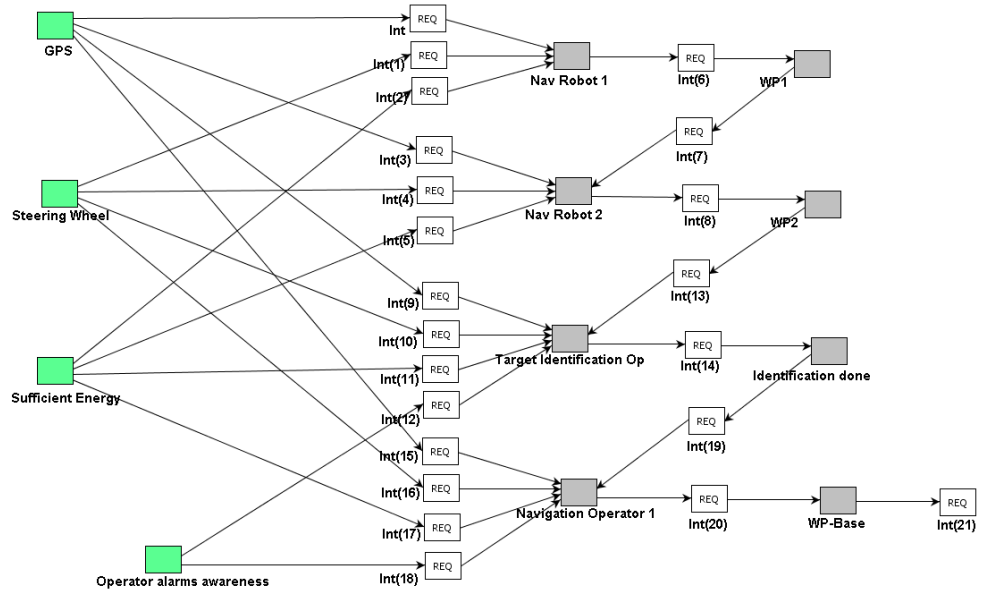

FIGURE 6.8: Modèle initial mis à jour par le suivi de situation

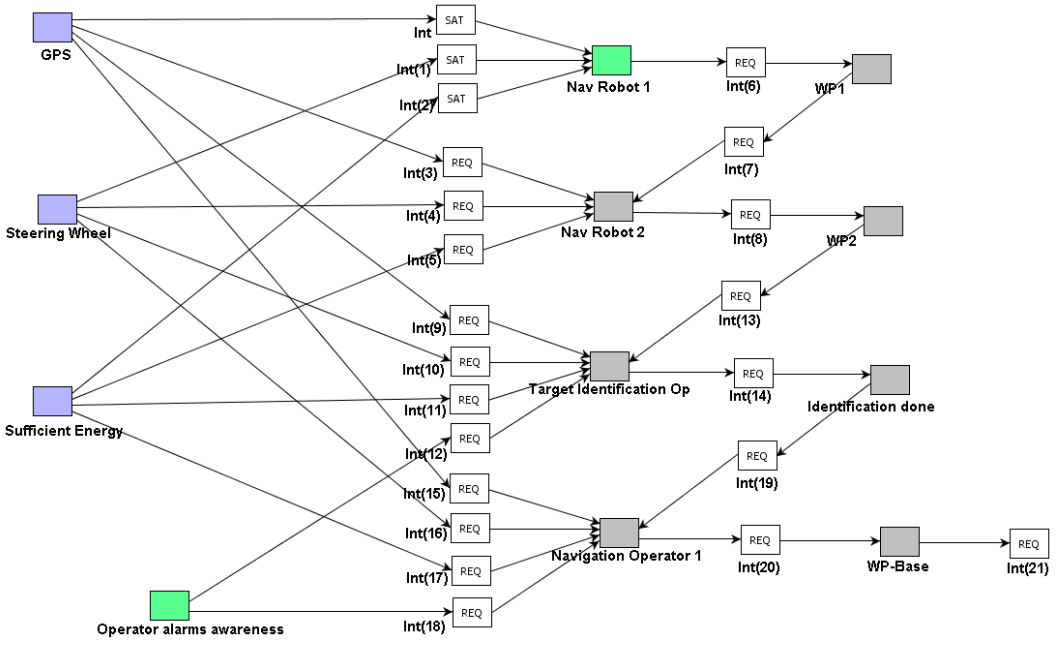

FIGURE 6.9: Production de la ressource *Navigation Robot 1*

La ressource *Navigation Robot 1*, à son tour, est affectée à la ressource *WP1* (voir figure 6.10).

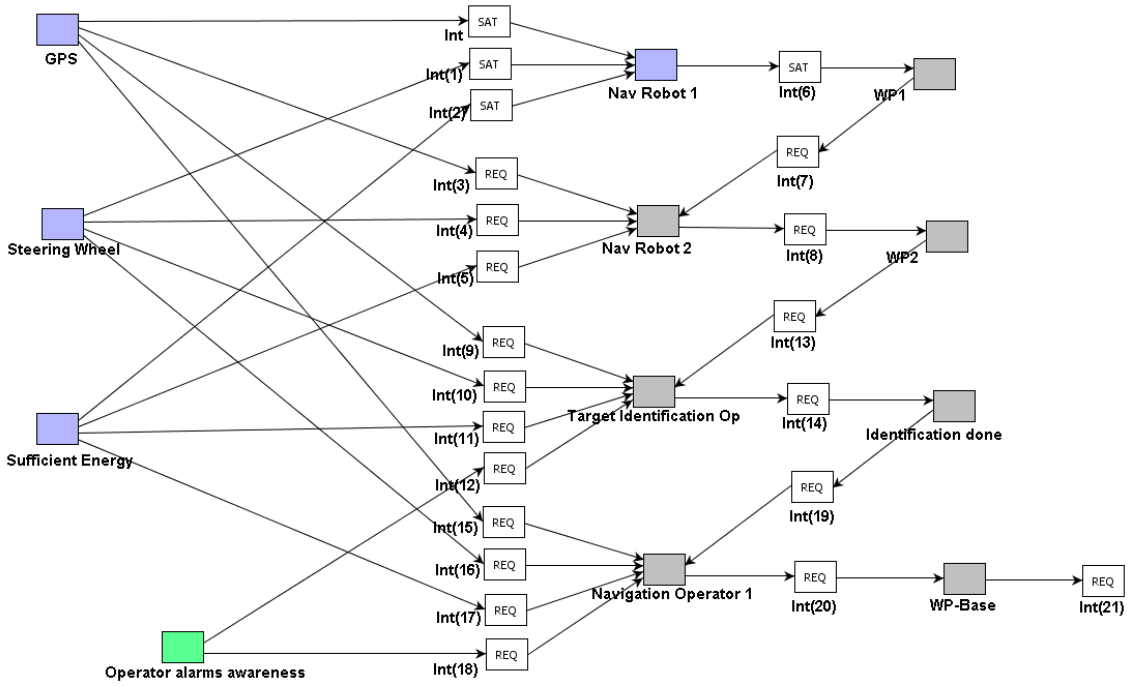

FIGURE 6.10: Affectation à la ressource *WP1*

Lorsque le point de passage représenté par la ressource *WP1* a été atteint, cette ressource est produite ; par le jeu des relations de dépendance, toutes les ressources requises par la réalisation de ce but sont alors relâchées. Toutes les interfaces *Int, Int(1), Int(2), Int(6)* passent en l'état *Request satisfied* et sont retirées du modèle, tout comme la ressource *Navigation Robot 1*, désormais déconnectée de toute autre ressource. La ressource *WP1* est la précondition du lancement de la tâche *Navigation Robot 2 via* une relation de dépendance *Init-Dep*. Après affectation de *WP1, GPS, Steering Wheel* et *Sufficient Energy* à *Navigation Robot 2*, la tâche de navigation vers le point de passage 2 commence, *WP1* est immédiatement relâchée, et *Int(7)* et *WP1* sont supprimées du modèle. *Nav Robot 2* est alors affectée à *WP2*. Le modèle de l'autorité courant est alors celui de la figure 6.11.

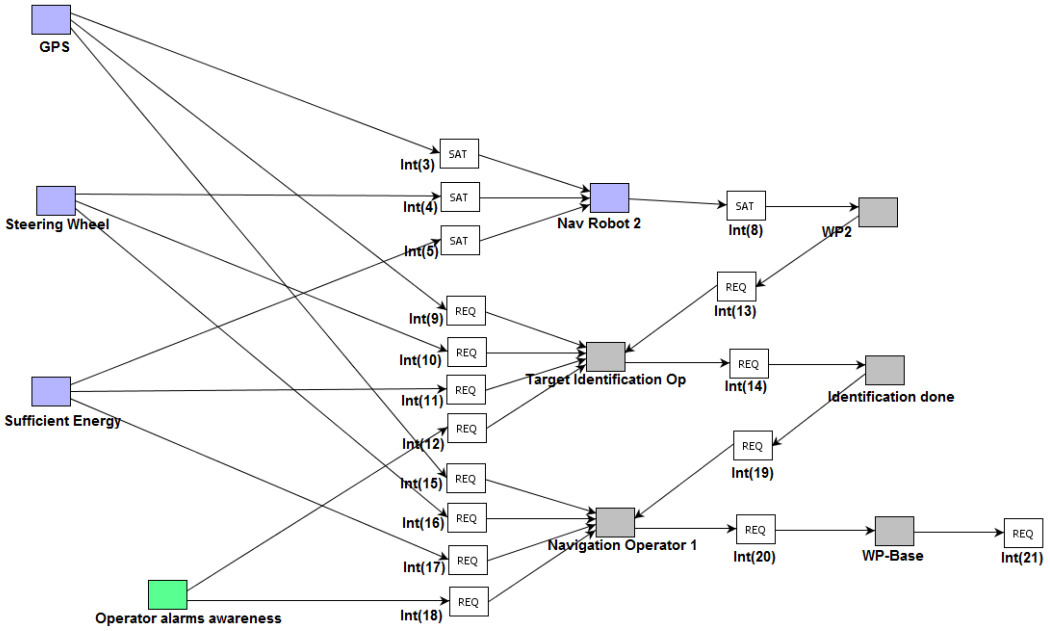

FIGURE 6.11: Affectation de la ressource *Nav Robot 2* à la ressource *WP2*

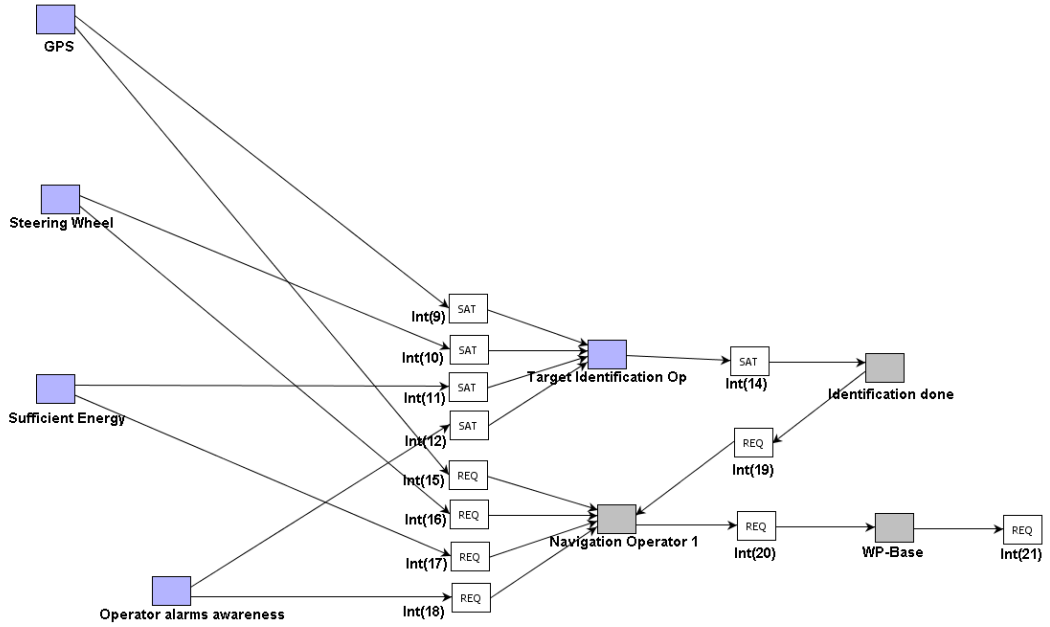

FIGURE 6.12: Identification de cibles par l'opérateur

Une fois que *WP2* a été atteint, la main est rendue à l'opérateur pour qu'il effectue la tâche d'identification de cibles manuellement. Dans le modèle, la ressource *Operator alarms awareness*, représentant le fait que l'opérateur est bien informé de l'état des alarmes du système, est requise par la ressource *Identification Cible Op via* une relation de dépendance de type *Exec-Dep*, car le fait que l'opérateur soit informé des alarmes doit rester vérifié tout le long de l'exécution de la tâche d'identification (voir figure 6.12). Au niveau des relations d'autorité, le principe est toujours le même que précédemment ; on note toutefois ici que, bien que l'opérateur soit chargé de l'exécution de la tâche d'identification de cibles, cette tâche se fait sur requête du robot, puisque c'est le robot qui a créé l'interface $Int(14)$.

3.2 Défaillance de la batterie

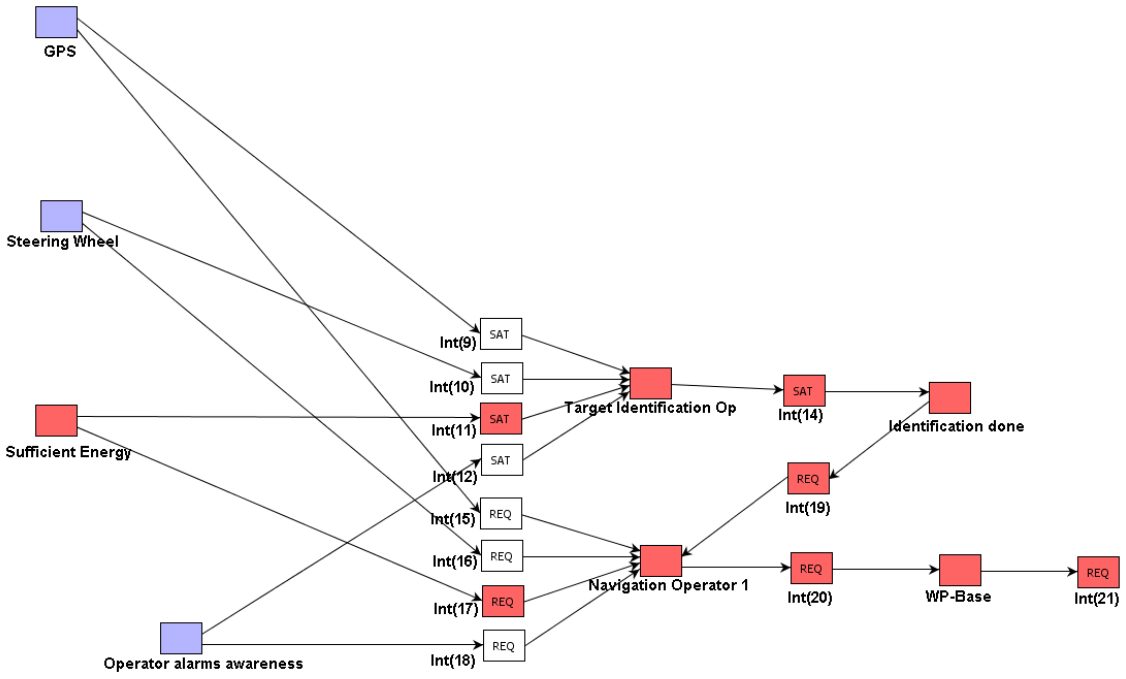

FIGURE 6.13: Conflit de destruction suite à la panne batterie

Cependant, suivant le déroulement du scénario, lorsque l'opérateur réalise la tâche *Identification Cible Op*, un aléa d'insuffisance de batterie est déclenché. Ceci génère l'affichage correspondant sur l'interface de l'opérateur, et la ressource *Sufficient Energy* est détruite par le suivi de situation, provoquant un conflit de destruction (voir figure 6.13).

3.2.1 Résolution sans changement d'autorité

Le graphe de ressources affecté par le conflit est représenté en rouge sur la figure 6.13. Le but affecté est la ressource *WP-Base*, et toutes les interfaces du modèle ayant été créées par le robot, il n'y a pas de sous-but dont il faut tenir compte (voir chapitre 4).

Au conflit de défaillance sur l'énergie, le processus de résolution faisant appel au planificateur (simulé, et agissant sur la base de recettes préétablies) apporte la solution représentée sur la figure 6.14, qui présente le graphe de ressource tel qu'inséré dans le modèle d'autorité avant son exécution : le robot est chargé de ramener le véhicule à la base de manière automatique, sous la forme d'une tâche spécifique (ressource *Return To Base Robot*), et sous supervision de l'opérateur. L'énergie restant dans la batterie est jugée suffisante par le suivi de situation pour accomplir la tâche demandée. La supervision opérateur requiert que l'opérateur soit conscient des informations relatives aux alarmes ; pour s'en assurer, l'envoi d'une contre-mesure cognitive est intégré, sur la base des informations concernant les alarmes.

Toutes les interfaces introduites (*Int(22), Int(23), Int(24), Int(25), Int(32), Int(33), Int(34), Int(35)* sont créées par le robot ($\phi(int) = robot$). Voici le détail des propriétés de dépendance :

- les interfaces *Int(22), Int(23), Int(24), Int(32), Int(34), Int(35)* représentent une dépendance de type $Exec{-}Dep$;
- les interfaces $Int(33), Int(25)$ représentent une dépendance de type $End{-}Dep$;

L'interface $Int(33)$ est du type $End{-}Dep$ car le produit de la tâche d'envoi de contre-mesure est la ressource *Operator alarms awareness*. C'est la fonction de suivi de situation qui se charge de marquer cette ressource comme présente ou absente. Par défaut, le suivi de situation peut la marquer comme présente immédiatement après l'envoi de la contre-mesure, mais peut venir la détruire immédiatement après si le comportement de l'opérateur n'est pas estimé cohérent avec la conscience de situation sur les alarmes.

La résolution du conflit a pu se faire sans modification des relations d'autorité existantes : en effet, toutes les interfaces présentes dans le modèle ont été créées par le robot, celui-ci peut donc les modifier sans contrainte. Le robot exécute ainsi une tâche de retour automatique à la base sous supervision de l'opérateur.

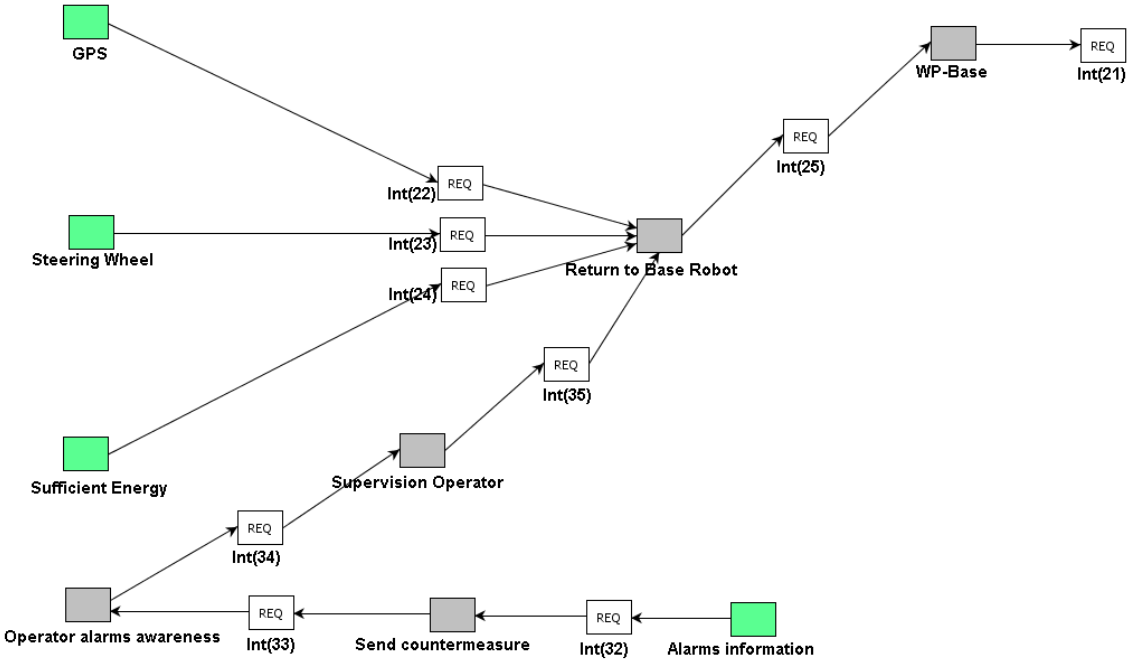

FIGURE 6.14: Modèle restauré après panne batterie

3.2.2 Résolution avec changement d'autorité

Le conflit de défaillance sur la batterie correspond à un problème sérieux sur la mission : il s'agit d'un cas où une redistribution de l'autorité entre robot et opérateur pourrait être envisagée, en faisant appel à la fonction de *méta-autorité* définissant les règles de changement d'autorité entre les agents. Le concept de méta-autorité sera développée dans les perspectives de ce travail ; cependant, sur cet exemple, nous avons établi une règle *ad hoc* très simple : en cas de défaillance de la batterie, le robot gagne le droit de préemption sur la ressource *Steering Wheel*. On a alors : $a_{<robot,operator>}(r_{SteeringWheel}) = (Preemptability, Access)$.

En ce qui concerne la résolution du conflit, elle aboutit exactement à la solution précédente présentée figure 6.14. L'intérêt de modifier les relations d'autorité *via* la règle de méta-autorité décrite précédemment se montre par la suite : dans le cas où l'opérateur, malgré l'envoi de contre-mesures, persévère et cherche à reprendre le contrôle du véhicule *via* son joystick, il ne pourra pas le faire. En effet, la nouvelle relation d'autorité lui impose d'attendre la fin de l'utilisation de la ressource *Steering Wheel* par le robot. Ceci devrait être signifié à l'opérateur depuis son interface de contrôle, le joystick de commande pouvant devenir très lourd à manipuler par exemple, dans le cas d'une interface de type haptique.

4 Discussion

4.1 Règle de méta-autorité alternative

Plusieurs règles de changement des relations d'autorité entre agents constituant la méta-autorité peuvent être envisagées et leur mise en application en situation de mission peut conduire à des solutions très différentes lors de la résolution de conflit, solutions qui dépendent également des savoir-faire dont dispose le planificateur. Sur l'exemple du conflit de défaillance sur la ressource *Sufficient Energy*, postulons désormais qu'il n'y a pas de règle correspondante au niveau de la méta-autorité et que, en conséquence, les relations d'autorité restent inchangées par rapport au départ de la mission. On en revient à la situation où : $a_{<robot,operator>}(r_{SteeringWheel}) = (Access, Preemptability)$.

En conférant au robot une procédure de retour à la base ne nécessitant pas explicitement de supervision opérateur, la solution au conflit proposée serait celle de la figure 6.15.

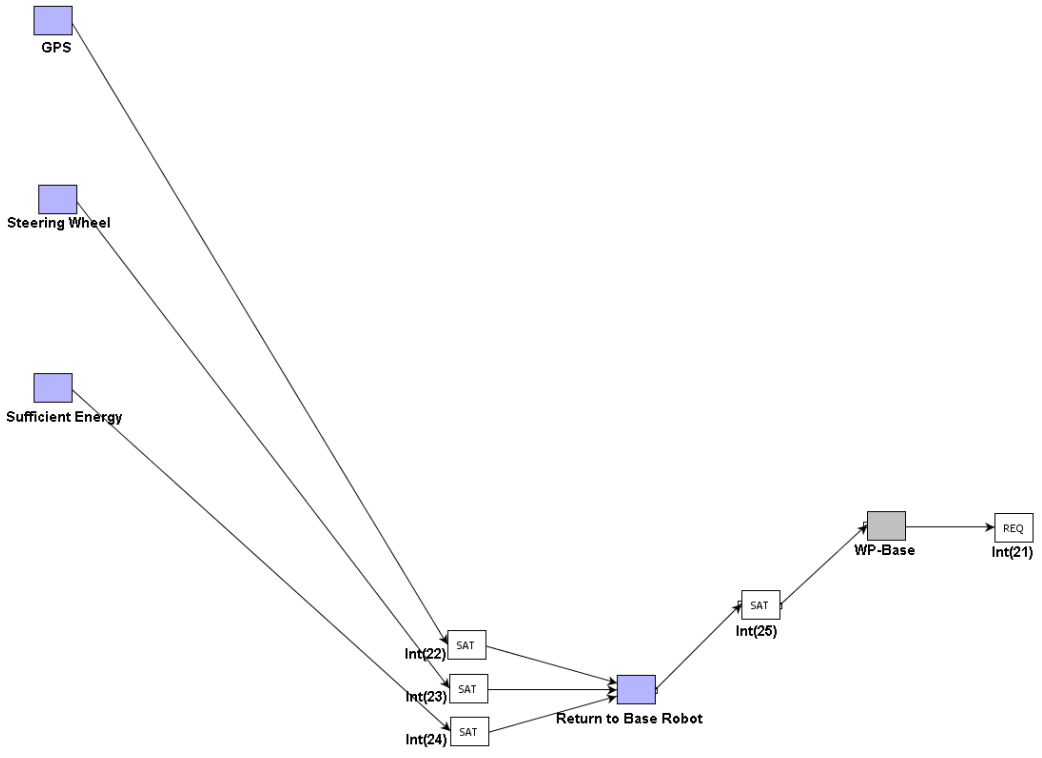

FIGURE 6.15: Modèle alternatif restauré après panne batterie

Les propriétés des composants du modèle de l'autorité sont les suivantes :
- les interfaces $Int(22), Int(23), Int(24)$ représentent une dépendance de type $Exec{-}Dep$;
- l'interface $Int(25)$ représente une dépendance de type $End{-}Dep$;
- toutes ces interfaces sont créées par le robot ;
- on pose $a_{<robot,operator>}(r_{ReturntoBaseRobot}) = (Access, Preemptability)$.

4.1.1 Intervention de l'opérateur

Après la reprise en main par le robot, l'opérateur désire reprendre le contrôle du véhicule, pour des raisons inconnues du robot. L'opérateur disposant de la préemption sur le robot pour la ressource *Steering Wheel* par les relations d'autorité dans le modèle, cette ressource lui est affectée, après insertion du modèle de tâche correspondant à l'action de l'opérateur sur son interface (l'opérateur a pris le joystick de contrôle en main). Un conflit de préemption en résulte sur la ressource *Steering Wheel*, qui est non partageable. Le modèle résultant est présenté figure 6.16. On remarque que le point de destination de l'opérateur *WP Operator* est inconnu du robot.

On a également les propriétés suivantes :
- les interfaces $Int(26), Int(27), Int(28), Int(29)$ représentent une dépendance de type $Exec{-}Dep$;
- l'interface $Int(30)$ représente une dépendance de type $End{-}Dep$;
- toutes ces interfaces sont créées par l'opérateur ;
- on pose $a_{<robot,operator>}(r_{NavigationOperator1}) = (Access, Access)$;
- de même, on pose $a_{<robot,operator>}(r_{WP-Operator}) = (Access, Access)$.

Pour respecter les relations d'autorité sur les ressources, la résolution de conflit laisse la main à l'opérateur pour rejoindre son point de destination inconnu, mais lui impose dans le même temps de rejoindre la base. Il est en effet présupposé que l'opérateur a bien perçu l'alarme de panne de batterie, et qu'il va donc de lui-même reconduire le robot à sa base (voir figure 6.17).

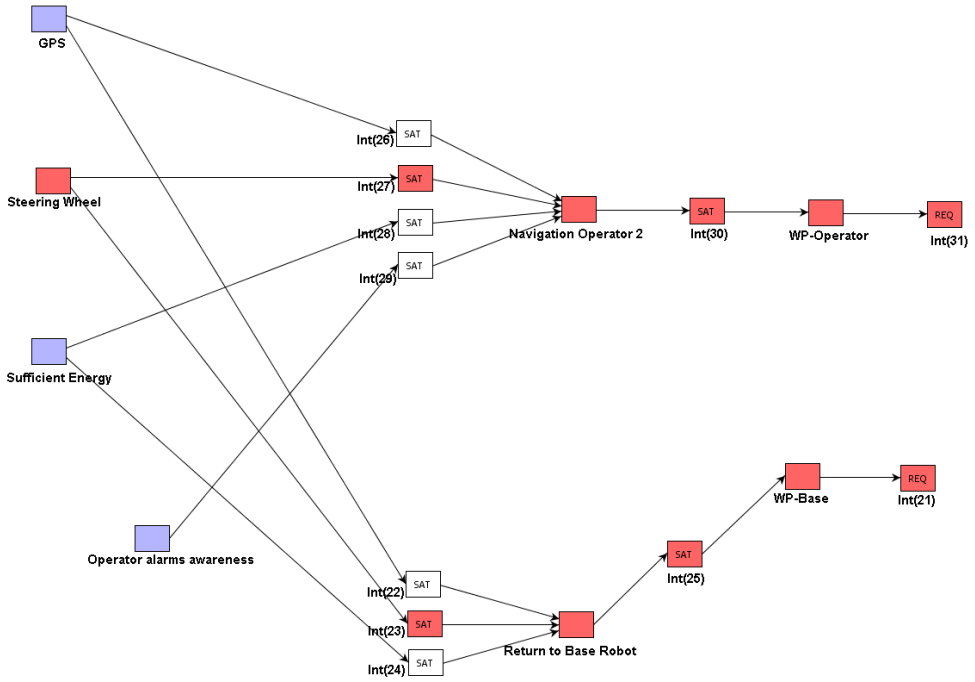

FIGURE 6.16: Reprise en main par l'opérateur du contrôle du véhicule

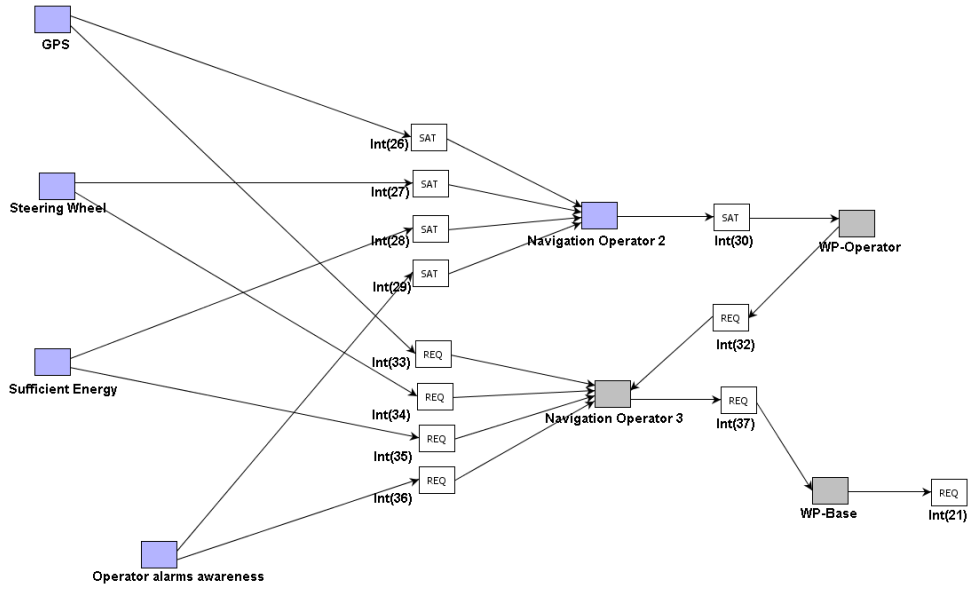

FIGURE 6.17: Résolution du conflit de préemption opérateur

4.1.2 Persévération de l'opérateur

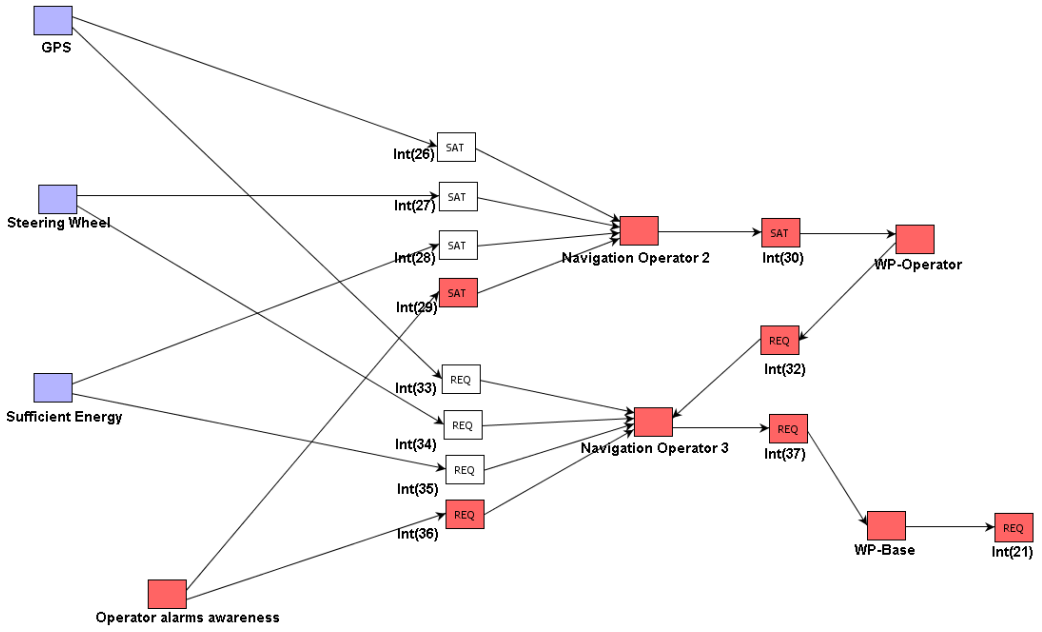

FIGURE 6.18: Persévération opérateur

Cependant, l'opérateur ne prend pas la direction de la base malgré l'alarme de batterie, ce qui ne correspond pas au comportement qui était attendu de lui. Il peut être en état de persévération sur sa tâche d'identification de cibles. Le suivi de situation estime alors que l'opérateur n'a pas perçu l'alarme de défaillance de batterie ; le suivi de situation détruit alors la ressource *Operator alarms awareness*, ce qui provoque un conflit de destruction (voir figure 6.18).

La résolution laisse la main à l'opérateur, mais, dans le but de restaurer la conscience de situation de l'opérateur au niveau des alarmes, inclut une tâche d'interaction de la part du robot en l'envoi d'une contre-mesure cognitive. Il est attendu que grâce à cette contre-mesure, affichée sur l'interface utilisateur, l'opérateur va pouvoir décrocher de sa tâche précédente et voir l'alarme sur la batterie : c'est le suivi de situation qui est chargé d'estimer si cela a fonctionné ou non (par exemple par l'usage d'un oculomètre) (voir figure 6.19).

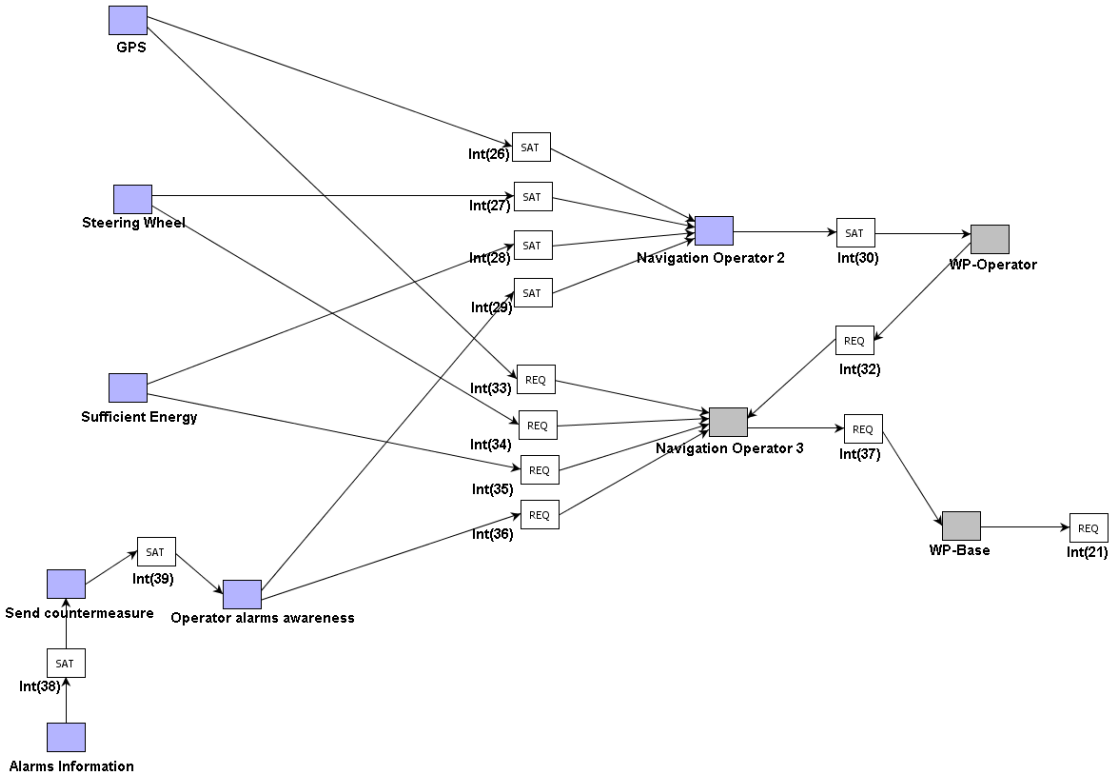

FIGURE 6.19: Envoi de contre-mesure

4.2 Connexion au robot Emaxx et discussion technique

Lors de cette phase d'application, nous avons expérimenté la connexion du contrôleur de la dynamique de l'autorité à une véritable plate-forme robotique, la plate-forme Emaxx développée par l'ISAE : celle-ci est basée sur un véhicule terrestre de type Rover possédant une électronique embarquée complète, incluant des capteurs tels que le GPS, l'odométrie, des capteurs ultrasons et une caméra panoramique destinée à la vision de l'opérateur. Ce robot peut être téléopéré manuellement ou peut effectuer des tâches de navigation de manière automatique, en évitant les obstacles.

Techniquement, la connexion du contrôleur de la dynamique de l'autorité au robot ne pose aucun problème, puisqu'il s'agit d'une simple connexion réseau à effectuer entre le système de commande au sol du robot avec le programme « fonction exécutive » présenté au paragraphe 2.2. Le programme « fonction exécutive » se charge de convertir les commandes en provenance du contrôleur de la dynamique de l'autorité en commandes interprétables par le robot, mais il s'agit simplement d'un relais de communication sans algorithmique particulière. Il en va de même pour la connexion entre le programme

« Interface opérateur » et la véritable interface de commande du robot Emaxx : encore une fois, il s'agit uniquement d'un relais de communication. Pour le reste de l'architecture, le planificateur et le suivi de situation restent simulés. Lors des tests effectués, les connexions avec la plate-forme sont apparues tout à fait fonctionnelles.

Cependant, nous n'avons pas pu manipuler directement le robot Emaxx en raison des performances du joueur de Petri et de la logique de contrôle de l'autorité du logiciel MNMS, qui sont apparues trop lentes pour permettre un contrôle suffisamment efficace du robot. Une raison à cette lenteur de traitement est que le logiciel MNMS est basé sur un éditeur OpenSource [2] sur lequel de nombreuses fonctions ont été ajoutées : création de projets pour gérer de nombreux réseaux, gestion des couleurs, des événements, des structures de fusion, des connexions vers les programmes externes, etc. Toutefois, un travail de mise à jour de l'architecture et d'optimisation du code pourrait très vraisemblablement permettre de contourner cet obstacle logiciel.

Les performances ont toutefois été suffisantes pour réaliser une manipulation en boucle ouverte : sur le scénario de l'application, nous avons pu connecter le contrôleur de la dynamique de l'autorité de telle manière qu'il ne soit pas chargé de l'envoi des commandes sur le système, mais qu'il en récupère les événements (état des ressources, prise de joystick par l'opérateur). Nous avons pu constater que le modèle évolue suivant le scénario attendu, et est capable de générer les bons messages de commande destinés aux actionneurs.

2. Pipe, Platform Independent Petri Editor 2.5

Chapitre 7

Conclusion et perspectives

1 Bilan

1.1 Autonomie variable sans « niveaux d'autonomie »

Nous avons proposé dans le cadre de cette thèse un contrôleur de la dynamique de l'autorité destiné à être embarqué sur un robot interagissant avec un opérateur humain pour la réalisation d'une mission. Ce contrôleur réalise la détection de conflits provoqués dans le plan des agents, par les aléas ou les mauvaises coordinations des agents. Le conflit est utilisé comme déclencheur d'une réorganisation du plan des agents, conduisant à une adaptation de l'autonomie du robot vis-à-vis de l'opérateur, sans qu'il soit fait appel à des « niveaux d'autonomie ». L'affectation des ressources entre le robot et l'opérateur humain n'est en effet pas réalisée de manière prescriptive, définie *a priori* dans un ensemble restreint de niveaux d'autonomie ; elle est en revanche le résultat de la combinaison des capacités courantes des deux agents vis-à-vis des buts poursuivis. Ces capacités sont exprimées sous la forme de ressources et d'autorité relative sur chaque ressource.

1.2 Modèle de l'autorité

Le contrôleur de la dynamique de l'autorité repose sur la gestion d'un *modèle de l'autorité* mis à jour en permanence. Celui-ci se décompose en deux niveaux : le macro-modèle, fondé sur les concepts de ressources, interfaces et relations d'autorité, qui permet de représenter le plan des agents sous la forme d'un graphe de dépendances ; le micro-modèle, mise en œuvre du macro-modèle par des réseaux de Petri synchronisés, qui permet l'exécution du macro-modèle. Le formalisme utilisé nous permet de donner une définition de l'autorité sur une ressource pour un couple d'agents, représentant les droits respectifs de ces agents quant à l'usage de la ressource.

1.3 Mise en œuvre du contrôleur de la dynamique de l'autorité

Le contrôleur de la dynamique de l'autorité se décompose en trois parties : le modèle de l'autorité, un joueur de Petri pour exécuter ce modèle, et un superviseur pour gérer la mise à jour du modèle et la détection de conflits. Pour être opérationnel, le contrôleur de la dynamique de l'autorité doit lui-même s'insérer dans l'architecture décisionnelle du robot composée d'un planificateur, d'une fonction de suivi de situation, d'une fonction exécutive et de l'interface opérateur. L'outil MNMS que nous avons proposé permet l'écriture des modèles de l'autorité requis ainsi que leur exécution, remplissant ainsi toutes les fonctions du contrôleur de la dynamique de l'autorité. Nous avons également simulé les fonctions nécessaires pour compléter l'architecture fonctionnelle afin de montrer le fonctionnement du système complet.

2 Discussion et perspectives

2.1 Évaluation et métriques

Une fois le déploiement du contrôleur de la dynamique de l'autorité rendu effectif par une mise en œuvre du logiciel MNMS, il sera possible de procéder à l'évaluation de différents modèles de l'autorité, sur une plate-forme robotique et un échantillon d'opérateurs en situation de mission. La mise en place d'un protocole de test sera nécessaire, ce qui passe par la définition de métriques pour évaluer les performances du système homme-robot et la pertinence du contrôleur de la dynamique de l'autorité. [Pina *et al.*, 2008] présentent ainsi un ensemble de métriques généralisables pour l'évaluation des équipes homme-robot, couvrant les aspects tels que l'efficacité de l'équipe, l'efficacité des comportements humains (efficacité de prise de décision, mise en œuvre d'actions) et robotiques (robustesse, tendance à générer des erreurs), la collaboration interne à l'équipe, les précurseurs physiologiques (fatigue, stress) et cognitifs (conscience de situation, charge de travail, confiance en l'automatisme). La confiance en l'automatisme de l'opérateur vis-à-vis du robot paraît d'ailleurs un indicateur particulièrement pertinent [Crocquesel *et al.*, 2010], en ce sens où c'est bien l'autonomie « apparente » du robot vis-à-vis de l'opérateur qui est évaluée, puisque définie relativement à celui-ci. Enfin, [Zieba *et al.*, 2009] proposent des métriques pour évaluer la résilience d'un système homme-robot, c'est-à-dire la robustesse du système face à la survenue d'aléas (tolérance, récupération d'erreurs). Les métriques pertinentes pour l'évaluation d'un système homme-robot intégrant le contrôleur de la dynamique de l'autorité que nous avons réalisé pourraient ainsi être les performances en mission (temps de complétion de la mission, objectifs atteints), le taux d'erreurs commises par le système, la résilience effective, ainsi qu'une évaluation de la confiance de l'opérateur vis-à-vis du robot.

2.2 Liens avec l'interface opérateur

Le principe de fonctionnement du modèle de l'autorité, mis à jour en fonction de l'état du système et des plans des agents, permet d'organiser et de maintenir accessibles un grand nombre d'informations sur le système. Le concept de conflit permet d'identifier les différences entre la réalisation *attendue* du plan et l'exécution *observée*. Grâce aux relations de dépendance, les conséquences des conflits sont propagées, permettant d'anticiper les problèmes et d'identifier les buts de la mission menacés. Ainsi, l'opérateur peut constater l'effet actuel et prédit de ses actions. De plus, en intégrant la partie déclarée du plan de l'opérateur, observée ou qui lui est assignée, le modèle de l'autorité permet au robot d'organiser les tâches de l'opérateur et éventuellement de l'assister.

Toutes ces informations peuvent être particulièrement pertinentes pour améliorer l'interaction avec l'opérateur *via* l'IHM. L'IHM constitue une fonction hors du cadre de travail de cette thèse, mais qui pourrait bénéficier de la mise en œuvre du contrôleur de la dynamique de l'autorité. Une interface dont la configuration des entrées / sorties est ajustable dynamiquement, sur le modèle des travaux de [Navarre *et al.*, 2010], permettrait d'obtenir une interface s'ajustant à la tâche de l'opérateur, à sa conscience de situation, au conflit détecté. De plus, en intégrant des capacités telles que la détection de l'état de persévération chez l'opérateur ainsi que l'envoi de contre-mesures cognitives [Dehais *et al.*, 2003], le contrôleur de la dynamique de l'autorité permettrait au robot d'ajouter à ses actions possibles des tâches d'interaction avec l'opérateur *via* l'IHM, permettant au robot de ne pas agir que sur lui-même mais également sur l'opérateur pour résoudre un conflit.

2.3 Extension à des systèmes multiagents

Les travaux réalisés dans le cadre de cette thèse se sont limités à la présence d'un seul couple d'agents, l'opérateur humain et le robot. Cependant, la question du contrôle de l'autorité est centrale dans les systèmes faisant intervenir un plus grand nombre d'agents, dès lors que coexistent plusieurs boucles décisionnelles, que ce soit par la présence de plusieurs opérateurs humains (éventuellement organisés entre eux suivant des rapports de hiérarchie), de plusieurs agents robotiques et artificiels (robots hétérogènes sur un champ de bataille, agents logiciels). Notre modèle de l'autorité, à son niveau macro, devrait pouvoir être étendu pour supporter ces configurations ; cependant, une réflexion devrait être conduite particulièrement au niveau micro du modèle, afin de s'assurer de sa cohésion lorsque coexistent plusieurs couples d'agents pouvant utiliser une même ressource. En particulier, il faut vérifier si le modèle reste cohérent dans les cas extrêmes, si par exemple un agent x, par le jeu des relations d'autorité, est interdit d'accéder à une ressource par un agent y mais y est autorisé par un agent z, que se passe-t-il alors ? Comment le modèle devrait-il se comporter ? Ce genre de relations dans les systèmes multiagents doit être clarifié et implémenté.

2.4 Extensions du modèle de l'autorité et autres applications

Le formalisme que nous proposons a été appliqué dans un autre contexte, pour réaliser des simulations et des estimations de performance pour différentes configurations de constellations de satellites, ces satellites devant se partager des ressources communes comme des stations-sol [Bolis, 2010]. La représentation par ressources et dépendances entre ressources permet de modéliser différents systèmes fonctionnels ; l'introduction des relations d'autorité permet alors de réguler le partage des ressources entre systèmes, en évitant l'apparition de conflits.

L'utilisation du modèle de l'autorité en dehors du cadre des interactions homme-robot pourrait ainsi être développée et appliquée dans des domaines tels que la simulation de systèmes fonctionnels, ou la sûreté de fonctionnement : il s'agirait par exemple d'ajouter dans le modèle d'autres propriétés de ressources (exemple : consommabilité), d'autres types de dépendance (exemple : dépendances alternatives (OU logique), une ressource, ou l'autre, suffisant à elle seule). Le travail sur les constellations de satellites a par exemple mis en lumière l'intérêt d'avoir une propriété de consommabilité sur les ressources (une ressource consommable devrait disparaître d'elle-même dès qu'elle est relâchée par la ressource dépendante).

2.5 Méta-autorité

Les exemples présentés au paragraphe 2.3.2 ainsi qu'au chapitre 6 illustrent le besoin de poser des règles établissant précisément les conditions de changement d'état des relations d'autorité sur les ressources : c'est le rôle de ce que nous appelons la *méta-autorité*.

Le modèle de l'autorité constitue une base de la méta-autorité, sur laquelle les règles de modification de l'autorité relative des agents sur les ressources peuvent être construites. Une règle de méta-autorité pourrait par exemple prendre la forme suivante : « En cas de défaillance de la batterie, tant que l'opérateur n'a pas perçu l'alarme correspondante, le robot peut acquérir l'autorité sur les ressources nécessaires pour amorcer le retour du véhicule à la base. ».

La méta-autorité se pose alors comme une entité neutre dans l'architecture du système, ne portant le sceau d'aucun agent. La méta-autorité apparaît ainsi comme un « arbitre », faisant appliquer en cours de mission des règles prédéfinies et connues de tous les agents du système. La méta-autorité intervient sur requête, c'est-à-dire uniquement lorsqu'un agent fait appel à elle. C'est en replanification que le planificateur, ne trouvant pas de solution au vu des relations d'autorité existantes, essaie d'élargir son espace de recherche en faisant appel à la méta-autorité pour modifier ces relations d'autorité. L'opérateur doit pouvoir faire de même, si l'accès à certaines ressources ne lui est pas permis mais qu'il considère devoir les contrôler.

Les règles de méta-autorité sont fortement dépendantes du système considéré et des implications pratiques et philosophiques qu'elles induisent : dans quelles circonstances est-il possible de retirer le contrôle du véhicule à l'opérateur humain pour le donner à l'algorithme de navigation du robot ? Cela peut dépendre de la fiabilité de l'algorithme de navigation du robot, des capteurs qu'il utilise, de ses performances en situation réelle, de l'acceptabilité par l'opérateur de cette reprise en main, des considérations éthiques associées (i.e. que dire du robot reprenant la main sur un système d'armes ?)

En fin de compte, ces règles de méta-autorité ne peuvent être établies qu'en étroite collaboration entre les concepteurs du système, les opérateurs de ce système et les autorités de régulation et après une phase importante de validation expérimentale, dans les conditions typiques de déploiement du système.

2.6 Éthique et autorité

La question de la répartition de l'autorité entre un opérateur humain et un robot ne va pas sans implications éthiques, en particulier si le robot peut prendre la main sur l'opérateur humain pour le contrôle de ressources ; il s'agit du statut de la machine vis-à-vis de l'opérateur humain.

Le point de vue de la réglementation juridique, bien que parfois quelque peu déphasée par rapport à l'état de l'art technologique [Balke, 2010], ne laisse pourtant aucun doute sur la considération actuelle de la société sur ce qu'est une machine : il s'agit, malgré tous les dispositifs complexes la constituant, uniquement d'un outil et en tous les cas, la responsabilité morale incombe entièrement à l'opérateur qui déploie une machine pour l'assister, même si le concepteur de la machine peut voir sa responsabilisé engagée également [Sheridan et Parasuraman, 2006]. Ceci pourrait peut-être évoluer avec le développement d'architectures robotiques intégrant des considérations « éthiques" » au sein même du robot, celui-ci respectant un code de conduite identique à celui des humains ; les travaux de [Arkin, 2008] sur l'intégration du code militaire relatif aux règles d'engagement du combat au sein de drones est significatif, et soulève de nombreuses questions, avec en perspective la possibilité de robots respectant ces règles mieux que les humains eux-mêmes. Cependant, au vu des considérations précédemment avancées, la réglementation devra peut-être évoluer, mais pour l'instant un robot dit autonome est vu comme un logiciel comme un autre.

Dans le cadre de la collaboration homme-machine et de la prise d'autorité par le robot, il s'agit essentiellement de rétablir une forme de symétrie dans la relation homme-machine quant aux rôles des agents dans la mission et surtout de faire tomber le statut « sacralisé » de l'opérateur humain sur la machine. Il convient d'être lucide sur ce qu'apporte l'opérateur humain : ses qualités d'analyse sont essentielles et inégalées, pour autant il

n'est pas parfait et commet des erreurs ; c'est un problème qui ne peut être négligé si l'on souhaite améliorer objectivement les systèmes homme-machine. La machine est une aide au potentiel formidable pour étendre les possibilités humaines, tant au niveau de l'exécution de tâches dans le monde physique qu'au niveau de la cognition [Brezillon et Pomerol, 1997]. Cette thèse s'inscrit dans la perspective d'équilibre des rôles entre l'humain et la machine.

Annexe A

Réseaux de Petri synchronisés

1 Réseaux de Petri

Un *réseau de Petri* $< P, T, F, B >$ [David et Alla, 2005] est un graphe biparti avec P, ensemble fini de places, T, ensemble fini de transitions, les arcs orientés représentant respectivement la fonction d'incidence avant $F : P \times T \rightarrow \mathbb{N}$ et la fonction d'incidence arrière $B : P \times T \rightarrow \mathbb{N}$. Un *réseau de Petri synchronisé* est tel que le marquage $M : P \rightarrow \mathbb{N}$ représente les conditions vérifiées et des événements sont associés aux transitions ; l'évolution des jetons dans le réseau suit les règles de franchissement des transitions : une transition est franchissable si elle est validée (ses places amont sont suffisamment marquées) et l'événement associé se produit.

Réseaux colorés [Jensen et Kristensen, 2009] : il est possible d'associer une couleur aux jetons d'un réseau de Petri, qui correspond à un type. Si un réseau est marqué par des jetons de couleurs différentes, les couleurs n'interfèrent pas : le réseau coloré se comporte comme autant de réseaux non colorés indépendants qu'il y a de couleurs de jetons.

Couleur neutre : nous définissons la couleur de jeton *neutre* comme la couleur générique, pouvant être considérée comme toute autre couleur de jeton existant pour valider une transition.

Places monochromatiques : nous utilisons le concept de places *monochromatiques* : il s'agit d'un type de place dans lequel tout jeton est obligatoirement converti (si nécessaire) en un jeton de couleur neutre. Il ne peut y avoir de jeton de couleur non neutre sur une place monochromatique.

2 Synchronisation des réseaux de Petri

Les mécanismes de fusion de transition et d'envoi d'événements permettent de synchroniser des réseaux *a priori* non connectés par des arcs. Cependant, il est toujours possible de déplier ces réseaux et de se ramener à un seul et unique réseau correspondant à ces différents réseaux connectés par fusions ou événements.

2.1 Passage d'événements et réseaux de Petri synchronisés

On appelle *Réseau de Petri synchronisé* le triplet $< R, E, sync >$ où
- R est un Réseau de Petri marqué ;
- E est un ensemble d'événements externes ;
- $sync : T \rightarrow E \cup \{e\}$ où
  - T est l'ensemble des transitions de R ;
  - e est l'événement toujours occurrent.

Le réseau de Petri est totalement synchronisé si aucune transition n'est associée à e.

Dans un réseau de Petri synchronisé, une transition validée par le marquage de ses places amont n'est pas forcément franchissable. Elle deviendra franchissable quand l'événement externe associé à la transition par la relation *sync* se produit : elle est alors immédiatement franchie. La figure A.1 illustre ce principe.

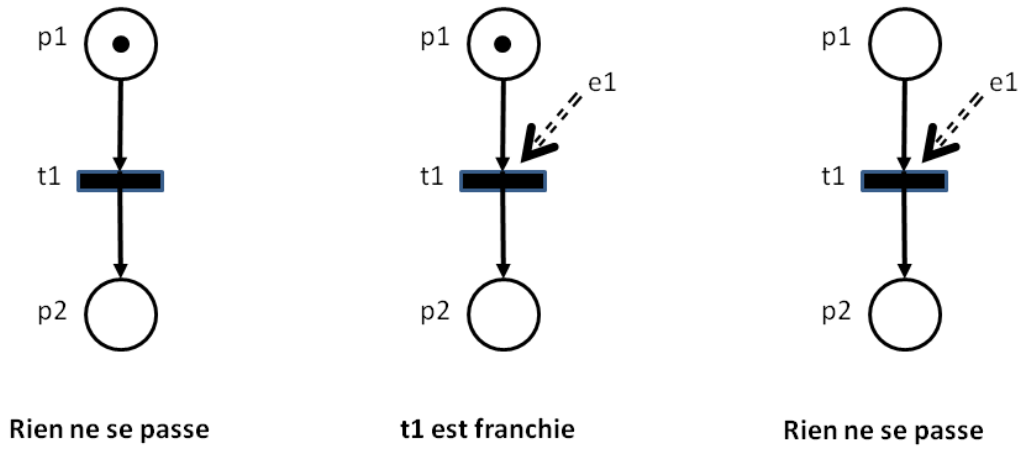

FIGURE A.1: Principe du franchissement d'une transition dans un RdP synchronisé

Pour le réseau considéré, on a la relation $sync(t1) = e1$: la transition $t1$ ne peut être franchie (ou tirée) que si :
- elle est validée par le marquage de ses places amont, c'est-à-dire si la place $p1$ contient un jeton ;
- l'événement $e1$ se produit.

Si l'événement $e1$ se produit lorsque $t1$ n'est pas validée, alors $e1$ n'a aucun effet et disparaît.

Lien avec les réseaux colorés

Une couleur peut être associée à un événement. Dans ce cas, le franchissement de la transition associée à l'événement ne se fait que si elle est validée par un marquage de la même couleur. Cependant, la couleur *neutre* peut s'associer à toute autre couleur pour compléter un marquage ou tirer une transition. La figure A.2 illustre différentes configurations de couleurs de jetons et d'événements.

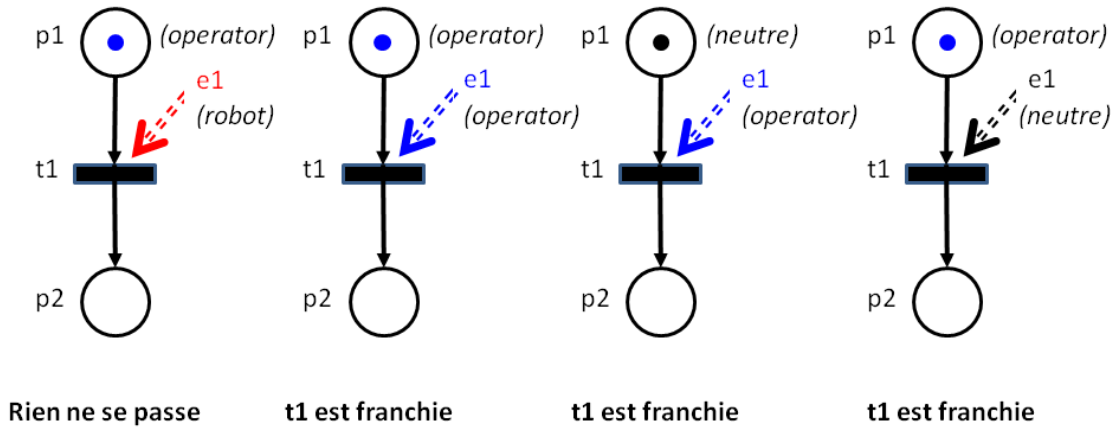

FIGURE A.2: Franchissement de transition dans un RdP synchronisé coloré

2.2 Fusion de transitions

La fusion de transitions permet de synchroniser des réseaux de Petri non connectés en un endroit précis, là où les transitions ont été fusionnées. Prenons deux transitions, t_1 sur un réseau de Petri $R1$ et t_2 sur un réseau de Petri $R2$; les deux transitions sont fusionnées. Dans ce cas, chacune de ces deux transitions doit être validée par le marquage respectif de ses places amont pour que les deux transitions soient simultanément tirées. Si le marquage en amont d'une des transitions n'est pas suffisant pour la valider, aucune des deux transitions ne sera franchie. La figure A.3 montre l'équivalence entre la fusion de transitions de deux réseaux distincts et la construction d'un réseau unique.

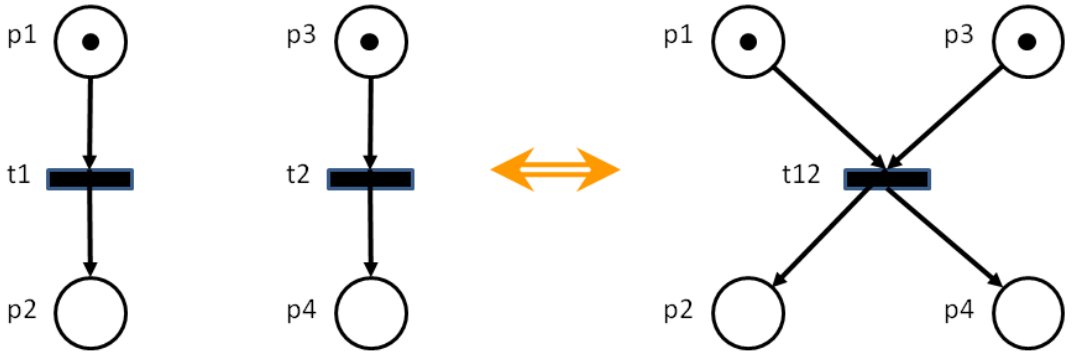

FIGURE A.3: Équivalence de la fusion de transition

Il est possible de fusionner plus de deux transitions ; dans ce cas, sur le même principe chacune des transitions $(t_1, t_2, ..., t_n)$ fusionnées doit être individuellement validée par le marquage de ses places amont pour que l'ensemble des transitions fusionnées soit simultanément tiré.

Lien avec les réseaux colorés

Dans un groupe de transitions fusionnées, chacune des transitions doit individuellement être validée par le marquage de ses places amont afin que le groupe dans son ensemble soit franchi. Dans le cas d'utilisation de jetons colorés, cette condition doit toujours être vérifiée, avec la contrainte supplémentaire que ce doit être *la même couleur de jeton* utilisée pour valider chaque marquage individuel. Si une transition t_1 est validée par un marquage utilisant la couleur *robot*, et qu'une transition t_2 est validée par un marquage utilisant la couleur *operator*, alors l'ensemble des transitions fusionnées (t_1, t_2) ne peut être tiré.

Bibliographie

C. Alwood (1984). Error detection processes in statistical problem solving. *Cognitive science*, 8 :413–437.

R. Amalberti (1996). La conduite de systemes à risques. Paris, Presses Universitaires de France. *Aviation human factors and the safety investigation process.*

R. Amalberti (2002). Une réflexion sur le rôle des hommes dans les systèmes intelligents et automatisés. In *Le rôle de l'être humain dans les systèmes automatisés intelligents, RTO HFM.*

R. Arkin (2008). Governing lethal behavior : embedding ethics in a hybrid deliberative/reactive robot architecture. In *Proceedings of the 3rd ACM/IEEE international Conference on Human Robot interaction*, pages 121–128. ACM.

T. Balke (2010). "Entity" and "Autonomy" - The conclusion of contracts by software agents in the eyes of the law. A software agent definition-based analysis. *Revue d'Intelligence Artificielle*, 24(3) :391–413.

D. Barber, L. Davis et D. Nicholson (2008). The mixed initiative experimental (MIX) testbed for human robot interactions with varied levels of automation. In *26th Army Science Conference.*

S. Barton (1988). Pilot control. In *Human factors in aviation*, chapter 11. Academic Pr.

BEA (2000). "Objectif : destination". Technical report.

B. Berberian, P. Le Blaye, V. Marchand et J. Sarrazin (2010). A preliminary experiment on the concepts of authority sharing and agency in UAS supervisory control. In *2nd HUMOUS (HUMans Operating Unmanned Systems)*, Toulouse, France.

A. Berthoz (2003). *La Décision.* Odile Jacob, Paris.

E. Billings (1996). Aviation automation. *Lawrence Erlbaum Associates, Inc., Mahwah, USA. Nov 1996.*

B. Bolis (2010). Développement d'une interface de simulation de réseaux de satellites autonomes. Master's thesis, Université Paul Sabatier, Toulouse, France.

G. Bonnet et C. Tessier (2007). Collaboration among a satellite swarm. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 54. ACM.

J. Bradshaw, M. Sierhuis, A. Acquisti, R. Feltovich, R. Hoffman, R. Jeffers, D. Prescott, N. Suri, A. Uszok et R. Van Hoof (2003). Adjustable autonomy and human-agent teamwork in practice : An interim report on space application. In [Hexmoor *et al.*, 2003], chapter 11.

S. Brainov et H. Hexmoor (2003). Quantifying relative autonomy in multiagent interaction. In [Hexmoor *et al.*, 2003], chapter 4.

P. Brezillon et J. Pomerol (1997). Joint cognitive systems, cooperative systems and decision support systems : a cooperation in context. In *Proc. of the European Conference on Cognitive Science*, pages 129–139. Citeseer.

J. Brookshire, S. Singh et R. Simmons (2004). Preliminary results in sliding autonomy for coordinated teams. In *AAAI04 Spring Symposium Series*, Stanford, CA.

T. Callantine (2002). Activity tracking for pilot error detection from flight data. Technical report, NASA.

C. Carabelea et O. Boissier (2006). Coordinating agents in organizations using social commitments. *Electronic Notes in Theoretical Computer Science, Elsevier*.

C. Castelfranchi (2000). Conflict ontology. In H.-J. Müller et R. Dieng, editors, *Computational conflicts - Conflict modelling for distributed intelligent systems*, pages 21–40. Springer Verlag.

C. Castelfranchi et R. Falcone (2003). From automaticity to autonomy : the frontier of artificial agents. In [Hexmoor *et al.*, 2003], chapter 6.

L. Chaudron, F. Dehais, P. Le Blaye et L. Wioland (1999). Human activity modelling for flight analysis. In *Proceedings of HCP'99, Intl. Conf. on Human Centered Processes*, Brest, France.

L. Chaudron, H. Fiorino, H.-J. Müller et C. Tessier (2000). Agent's conflict : new issues. In C. Tessier, L. Chaudron et H.-J. Müller, editors, *Conflicting agents : conflict management in multi-agent systems*, chapter 1, pages 1–30. Kluwer Academic Press.

C. Chopinaud, A. Seghrouchni et P. Taillibert (2006). Prevention of harmful behaviors within cognitive and autonomous agents. In *ECAI 2006 : 17th European Conference on Artificial Intelligence*, page 205, Riva del Garda, Italy.

K. Christoffersen et D. Woods (2002). 1. How to make automated systems team players. *Advances in human performance and cognitive engineering research*, 2 :1–12.

B. T. Clough (2002). Metrics, Schmetrics ! How the Heck do you determine a UAV's Autonomy anyway ? In *Performance Metrics for Intelligent Systems Workshop*, Gaithersburg, Marylan.

G. Coppin, F. Legras et S. Saget (2009). Supervision of autonomous vehicles : Mutual modelling and interaction management. In *HCI International 2009*, San Diego, CA, USA.

T. Coye de Brunélis et P. Le Blaye (2009). Towards a human centered methodology for the dynamic allocation of functions. *RIA, Revue des Sciences et Technologies de l'Information, Human-centered processes*, 23/4.

J. Crandall et M. Cummings (2008). A Predictive Model for Human-Unmanned Vehicle Systems : Final Report.

C. Crocquesel, F. Legras et G. Coppin (2010). Dynamic trust evaluation in supervisory control. In *2nd HUMOUS (HUMans Operating Unmanned Systems)*, Toulouse, France.

M. Cummings et P. Mitchell (2008). Predicting controller capacity in remote supervision of multiple unmanned vehicles. *IEEE Systems, Man, and Cybernetics, Part A Systems and Humans*, 38(2) :451–460.

R. David et H. Alla (2005). *Discrete, continuous and hybrid Petri nets*. Springer.

V. De Keyser et D. Woods (1990). Fixation errors : Failures to revise situation assessment in dynamic and risky systems. *Systems reliability assessment*, pages 231–251.

P. de Vries, C. Midden et D. Bouwhuis (2003). The effects of errors on system trust, self-confidence, and the allocation of control in route planning. *Int. J. Hum.-Comput. Stud.*, 58 :719–735.

F. Dehais (2004). *Modélisation des conflits dans l'activité de pilotage*. Thèse de doctorat, ENSAE, Toulouse.

F. Dehais, A. Goudou, C. Lesire et C. Tessier (2005). Towards an anticipatory agent to help pilots. In *AAAI 2005 Fall Symposium "From Reactive to Anticipatory Cognitive Embodied Systems"*, Arlington, Virginia.

F. Dehais et P. Pasquier (2000). Approche générique du conflit. In ESTIA, editor, *ErgoIHM*. D.L.Scapin et E. Vergisson.

F. Dehais, C. Tessier et L. Chaudron (2003). Ghost : experimenting countermeasures for conflicts in the pilot's activity. In *IJCAI03*, Acapulco, Mexico.

D. Donath, A. Rauschert et A. Schulte (2010). Human supervisory control of multiple UAVs by use of task based guidance. In *2nd HUMOUS (HUMans Operating Unmanned Systems)*, Toulouse, France.

B. Donmez, P. Pina et M. Cummings (2009). Evaluation criteria for human-automation performance metrics. *Performance Evaluation and Benchmarking of Intelligent Systems*, pages 21–40.

G. Dorais, P. Bonasso, D. Kortenkamp, B. Pell et D. Schreckenghost (1999). Adjustable autonomy for human-centered autonomous systems. In *IJCAI'99 Workshop on Adjustable Autonomy Systems*, Stockholm, Sweden.

M. Endsley (1995). Toward a theory of situation awareness in dynamic systems. *Human Factors*, 37 :32–64.

M. Endsley (1996). Automation and situation awareness. *Automation and human performance : Theory and applications*, pages 163–181.

J. Ferber et J. Perrot (1995). *Les systèmes multi-agents : vers une intelligence collective*, volume 522. InterEditions Paris.

A. Finzi et A. Orlandini (2005). Human-robot interaction through mixed-initiative planning for rescue and search rovers. In *AIIA*, pages 483–494, Milan, Italy.

F. Flemisch, J. Kelsch, C. Löper, A. Schieben, J. Schindler et M. Heesen (2008). Cooperative control and active interfaces for vehicle assistance and automation. In *FISITA World automotive Congress ; Munich*.

T. Fong, C. Thorpe et C. Baur (2002). Collaboration, dialogue and human-robot interaction. In *10th International Symposium on Robotics Research*, Lorne, Victoria, Australia.

S. Franklin et A. Graesser (1997). Is it an Agent, or just a Program ? : A Taxonomy for Autonomous Agents. *Intelligent Agents III Agent Theories, Architectures, and Languages*, pages 21–35.

K. Funk, B. Lyall, J. Wilson, R. Vint, M. Niemczyk, C. Suroteguh et G. Owen (1999). Flight deck automation issues. *The International Journal of Aviation Psychology*, 9(2) :109–123.

O. Giroud-Fliegner (2001). *Tout le monde peut se tromper. Essai sur l'erreur*. Seuil, Paris.

M. Goodrich, T. McLain, J. Crandall, J. Anderson et J. Sun (2007). Managing autonomy in robot teams : Observations from four experiments. In *ACM/IEEE International Conference on Human-robot interaction*, Washington, DC.

M. Goodrich, D. Olsen, J. Crandall et T. Palmer (2001). Experiments in adjustable autonomy. In *Ijcai'01 Workshop on Autonomy, Delegation and Control : Interacting with autonomous agents*, Seattle, WA.

P. Hancock (1996). Teleology for technology. *Automation and human performance : Theory and applications*, pages 461–497.

B. Hardin et M. Goodrich (2009). On using mixed-initiative control : a perspective for managing large-scale robotic teams. In *HRI'09, 4th ACM/IEEE International Conference on Human-Robot Interaction*, San Diego, CA.

B. Hasslacher et M. Tilden (1995). Living machines. *Robotics and Autonomous Systems*, 15(1-2) :143–169.

H. Hexmoor, C. Castelfranchi et R. Falcone (2003). *Agent Autonomy*. Kluwer Academic Publishers.

H. Hexmoor, B. McLaughlan et G. Tuli (2009). Natural human role in supervising complex control systems. *Journal of Experimental & Theoretical Artificial Intelligence*, 21(1) :59–77.

M. Heymann et A. Degani (2001). On formal abstraction and verification of human-machine interfaces : the discrete event case. In *NASA Technical Memorandum*.

J. Hoc (2000). From human–machine interaction to human–machine cooperation. *Ergonomics*, 43(7) :833–843.

E. Hollnagel (2003). *Handbook of cognitive task design*. CRC.

E. Hollnagel et D. Woods (1983). Cognitive systems engineering : New wine in new bottles. *International Journal of Man-Machine Studies*, 18(6) :583–600.

H. Huang, K. Pavek, B. Novak, J. Albus et E. Messin (2005). A framework for autonomy levels for unmanned systems ALFUS. In *AUVSI's Unmanned Systems North America 2005*, Baltimore, Maryland.

T. Inagaki et M. Itoh (2010). Theoretical framework for analysis and evaluation of human's overtrust in and overreliance on advanced driver assistance systems. In *Proceedings of the European conference on human centered design for intelligent transport systems*.

K. Jensen et L. Kristensen (2009). *Coloured Petri Nets : Modeling and Validation of Concurrent Systems*. Springer-Verlag New York Inc.

D. B. Kaber, M. R. Endsley et E. Onal (2000). Design of automation for telerobots and the effect on performance, operator situation awareness and subjective workload. *Human Factors and Ergonomics in Manufacturing*, 10(4) :409–430.

G. Klein, D. Woods, J. Bradshaw, R. Hoffman et P. Feltovich (2004). Ten Challenges for Making Automation a" Team Player. *Joint Human-Agent Activity, IEEE Intelligent Systems*, 19(6) :91–95.

D. Kortenkamp, P. Bonasso, D. Ryan et D. Schreckenghost (1997). Traded control with autonomous robots as mixed initiative interaction. In *AAAI-97 Spring Symposium on Mixed Initiative Interaction*, Stanford University, CA.

J. Laprie, J. Arlat, J. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun *et al.* (1995). *Guide de la sûreté de fonctionnement*. Cépaduès.

C. Layton, P. Smith et C. Mc Coy (1994). Design of a cooperative problem-solving system for en-route flight planning : An empirical evaluation. *Human Factors : The Journal of the Human Factors and Ergonomics Society*, 36(1) :94–119.

M. Le Marc (1999). *Psychologie sociale*, chapter Du conflit. Bréal : collection Grand Amphi.

J. Lee et K. See (2004). Trust in automation : Designing for appropriate reliance. *Human Factors : The Journal of the Human Factors and Ergonomics Society*, 46(1) :50.

F. Legras et C. Tessier (2003). Lotto : Group formation by overhearing in large teams. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 425–432. ACM.

J. Leplat (1985). *Erreur humaine, fiabilité humaine dans le travail*. Paris, A. Colin.

J. Leplat (1997). *Regards sur l'activité en situation de travail : contribution à la psychologie ergonomique*. Paris, PUF.

J. Leplat et J. Hoc (1983). Tâche et activité dans l'analyse psychologique des situations. *Cahiers de Psychologie Cognitive*, 3 :49–63.

D. Luzeaux (2004). La Bulle Opérationnelle Aéroterrestre : la démarche de simulation pour l'acquisition : Exploitation intelligente des senseurs dans les systèmes complexes. *REE. Revue de l'électricité et de l'électronique*, (6-7) :46–51.

S. Mercier, F. Dehais, C. Lesire et C. Tessier (2008a). Resources as basic concepts for authority sharing. In *1st HUMOUS (HUMans Operating Unmanned Systems)*, Brest, France.

S. Mercier, F. Dehais et C. Tessier (2009). Adjustable autonomy without levels. In *NATO SCI-202 Symposium on "Intelligent uninhabited vehicle guidance systems"*, Neubiberg, Germany.

S. Mercier et C. Tessier (2008). Some basic concepts for shared autonomy : a first report. In *Proceeding of the 2008 conference on Collaborative Decision Making : Perspectives and Challenges*, pages 40–48. IOS Press.

S. Mercier, C. Tessier et F. Dehais (2008b). Basic concepts for shared authority in heterogenous agents. In *AAMAS 2008 Workshop on Coordination, Organisations, Institutions and Norms in agent systems (COIN 2008), Estoril, Portugal.*

S. Mercier, C. Tessier et F. Dehais (2008c). Premières pistes pour l'autonomie adaptative sans niveaux. *ERGO IA.*

S. Mercier, C. Tessier et F. Dehais (2010a). Authority management in human-robot systems. In *11th IFAC/IFIP/IFORS/IEA Symposium on Analysis, Design, and Evaluation of Human-Machine Systems*, Valenciennes, France.

S. Mercier, C. Tessier et F. Dehais (2010b). Authority sharing in human-robot systems. In *2nd HUMOUS (HUMans Operating Unmanned Systems)*, Toulouse, France.

S. Mercier, C. Tessier et F. Dehais (2010c). Détection et résolution de conflits d'autorité dans un système homme-robot. *RIA, Revue d'Intelligence Artificielle, numéro spécial "Droits et Devoirs d'Agents Autonomes"*, RSTI série RIA Volume 24 - 3/2010.

P. Mitchell, M. Cummings et T. Sheridan (2005). Mitigation of human supervisory control wait times through automation strategies. *HAL Reports ; HAL2005-02.*

N. Moray (1986). Monitoring behavior and supervisory control. *Handbook of perception and human performance.*, 2 :40–1.

N. Moray (2000). Culture, politics and ergonomics. *Ergonomics*, 43(7) :858–868.

N. Moray et T. Inagaki (2000). Attention and complacency. *Theoretical Issues in Ergonomics Science*, 1(4) :354–365.

K. Mosier, L. Skitka, S. Heers et M. Burdick (1998). Automation bias : Decision making and performance in high-tech cockpits. *The International journal of aviation psychology*, 8(1) :47–63.

R. Mumaw, N. Sarter et C. Wickens (2001). Analysis of pilots' monitoring and performance on an automated flight deck. In *11th International Symposium on Aviation Psychology*, Colombus, OH.

K. Myers et D. Morley (2001). Human directability of agents. In *K-CAP 2001, 1st International Conference on Knowledge Capture*, Victoria, Canada.

D. Nau, H. Muñoz-Avila, Y. Cao, A. Lotem et S. Mitchell (2001). Total-order planning with partially ordered subtasks. In *IJCAI-2001*, Seattle, WA.

D. Navarre, P. Palanque, E. Barboni, J. Ladry et C. Martinie (2010). Designing for resilience to hardware failures in interactive systems : A model and simulation-based approach. *Reliability Engineering & System Safety*.

H. Nwana (1996). Software agents : An overview. *The Knowledge Engineering Review*, 11(03) :205–244.

R. Parasuraman (2000). Designing automation for human use : empirical studies and quantitative models. *Ergonomics*, 43(7) :931–951.

R. Parasuraman et E. Byrne (2003). Automation and human performance in aviation. *Principles and practice of aviation psychology*, pages 311–356.

R. Parasuraman et V. Riley (1997). Humans and automation : Use, misuse, disuse, abuse. *Human Factors*, 39(2).

R. Parasuraman, T. Sheridan et C. Wickens (2000). A model for types and levels of human interaction with automation. *IEEE Transactions on Systems, Man and Cybernetics, Part A : Systems and Humans*, 30(3) :286–297.

P. Pina, M. Cummings, J. Crandall et M. Della Penna (2008). Identifying generalizable metric classes to evaluate human-robot teams. In *Proceedings of Metrics for Human-Robot Interaction Workshop at the 3rd Annual Conference on Human-Robot Interaction*.

J. Rasmussen (1983). Skills, rules, and knowledge ; signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on SMC*, 13(3) :257–266.

J. Reason (1990). *Human error*. Cambridge University Press.

A. Rizzo, S. Bagnara et M. Visciola (1987). Human error detection processes. *International journal of Man-Machine studies*, pages 36 :253,259.

W. Rouse (1981). Human-computer interaction in the control of dynamic systems. *ACM Computing Surveys (CSUR)*, 13(1) :71–99.

S. Russell et P. Intelligence (1995). A modern approach. *Artificial Intelligence. Prentice-Hall*.

N. Sarter et R. Amalberti (2000). *Cognitive engineering in the aviation domain*. CRC.

N. Sarter, R. Wickens, S. Kimball, R. Marsh, M. Nikolic et W. Xu (2003). Modern flight deck automation : pilot's mental model and monitoring patterns and performance. In *Proceedings of the International Symposium on Aviation Psychology*, Dayton.

N. Sarter et D. Woods (1994). Pilot interaction with cockpit automation II. An experimental study of pilots' model and awareness of the flight management system. *International Journal of Aviation Psychology*, 4 :1–28.

N. Sarter, D. Woods et C. Billings (1997). Automation surprises. *Handbook of human factors and ergonomics*, 2 :1926–1943.

P. Scerri, D. Pynadath et M. Tambe (2003). Adjustable autonomy for the real world. In [Hexmoor *et al.*, 2003], chapter 10.

D. Schreckenghost, D. Ryan, C. Thronesbery, P. Bonasso et D. Poirot (1998). Intelligent control of life support systems for space habitat. In *AAAI-IAAI Conference*, Madison, WI.

N. Schurr, J. Marecki et M. Tambe (2009). Improving adjustable autonomy strategies for time-critical domains. In *AAMAS'09*, Budapest, Hungary.

N. Schurr, P. Patil, F. Pighin et M. Tambe (2006). Using multiagent teams to improve the training of incident commanders. In *AAMAS'06 Industry Track*, Hakodate, Japan.

B. Sellner, F. Heger, L. Hiatt, R. Simmons et S. Singh (2006). Coordinated multi-agent teams and sliding autonomy for large-scale assembly. In *Proceedings of the IEEE, 94(7)*.

T. Sheridan (2000). Function allocation : algorithm, alchemy or apostasy ? *International Journal of Human-Computer Studies*, 52(2) :203–216.

T. Sheridan et R. Parasuraman (2006). Human-automation interaction. In *Reviews of Human Factors and Ergonomics, 1*, chapter 2, pages 89–129.

T. Sheridan et W. Verplank (1978). Human and computer control of undersea teleoperators. Technical report, MIT Man-Machine Systems Laboratory, Cambridge, MA.

G. Simmel (2003). *Le conflit*. Circé/poche.

L. Steels (1995). When are robots intelligent autonomous agents ? *Journal of Robotics and Autonomous Systems*, pages 15 :3–9.

W. Sweet (1995). The glass cockpit. In *IEEE Spectrum*.

T. Villaren, C. Madier, F. Legras, A. Leal, B. Kovàcs et G. Coppin (2010). Towards a method for context-dependant allocation of functions. In *2nd HUMOUS (HUMans Operating Unmanned Systems)*, Toulouse, France.

M. Visciola, A. Armando et S. Bagnara (1992). Communication patterns and errors in flight simulation. *Reliability Engineering & System Safety*, 36(3) :253–259.

J. Wanner et N. Wanner (1999). Opérateur et sécurité. Technical report, 5/2.

C. Wickens, A. Kramer, L. Vanasse et E. Donchin (1983). Performance of concurrent tasks : a psychophysiological analysis of the reciprocity of information-processing resources. *Science (New York, NY)*, 221(4615) :1080.

C. D. Wickens (2008). Situation awareness : review of Mica Endsley's 1995 articles on situation awareness theory and measurement. *Human Factors*, 50(3) :397–403.

E. Wiener (1985). Human factors in cockpit automation : A field study of flight crew transition.

L. Wioland (1997). *Etudes des mécanismes de protection et de détection des erreurs. Contribution à un modèle de sécurité écologique.* Thèse de doctorat, Université Paris V.

D. Woods (1996). Decomposing automation : Apparent simplicity, real complexity. *Automation and human performance : Theory and applications*, pages 3–17.

D. Woods, E. Roth et K. Bennett (1990). Explorations in joint human-machine cognitive systems. *Cognition, computing and cooperation*, pages 123–158.

S. Zieba, P. Polet, F. Vanderhaegen et S. Debernard (2009). Resilience of a human-robot system using adjustable autonomy and human-robot collaborative control. *International Journal of Adaptive and Innovative Systems*, 1(1) :13–29.