COAST : des réseaux de Petri à la planification assistée

Sébastien Bardin
LSV, CNRS UMR 8643
ENS de Cachan
61 avenue du président Wilson
94235 CACHAN Cedex
bardin@lsv.ens-cachan.fr

Laure Petrucci
LIPN, CNRS UMR 7030
Université Paris 13
99 avenue Jean-Baptiste Clément
93430 VILLETANEUSE
petrucci@lipn.univ-paris13.fr

Résumé

COAST est un outil d'assistance à la planification militaire. Son architecture distribuée comprend un serveur constitué d'un moteur d'analyse de réseaux de Petri tandis que l'interface graphique fournie par le client permet de masquer l'utilisation des méthodes formelles. Les synchronisations entre tâches à planifier sont un aspect essentiel de COAST. Dans cet article, après une présentation générale de la problématique et de l'outil, nous décrivons les synchronisations, montrons comment elles sont modélisées et implantées.

Mots-clés : réseaux de Petri, planification, plateforme logicielle, architecture client-serveur

1 Introduction

La planification militaire au niveau opérationnel assure le lien entre les objectifs stratégiques à grande échelle et les décisions tactiques sur le terrain. C'est une tâche à la fois cruciale et non triviale pour les états-majors, compte tenu des ambiguïtés, des incertitudes et de la complexité des situations rencontrées.

La prise de décision dans les états-majors obéit à des processus très stricts. La planification opérationnelle est définie en quatre parties consécutives et itératives dans le JMAP (*Joint Military Appreciation Process*) de l'*Australian Defence Force*. Les autres organisations militaires utilisent des définitions équivalentes. L'analyse de la mission vise à une compréhension claire des objectifs, en définissant notamment l'état courant et l'état souhaité et en identifiant les tâches essentielles qui peuvent permettre de remplir la mission. Les deux étapes suivantes consistent respectivement à proposer différentes *lignes de conduite*, i.e. séquences de tâches, qui permettent d'atteindre l'objectif, puis à les comparer entre elles pour raffiner les plus efficaces. Finalement l'état-major choisit la ligne de conduite à adopter.

Comme dans d'autres domaines critiques, les organismes concernés sont de plus en plus demandeurs d'outils d'assistance fiables. Ainsi, une partie du processus de prise de décision de l'armée australienne a été spécifé formellement dans [KMZB02]. Un logiciel d'aide à la première phase de la planification opérationnelle (analyse de la mission), en utilisant des réseaux bayésiens, est décrit dans [FZD01] tandis que COAST (*Course of Action Support Tool*) [ZKJ+02] assiste les phases de développement et d'analyse des lignes de conduite.

Les problèmes de planification de tâches sont en général abordés par des méthodes de recherche opérationnelle. Cependant ces méthodes ne sont pas adaptées ici. D'une part, les critères d'un ordonnancement optimal ne sont pas définis. Comme le domaine est éminement critique, le programme n'est pas là pour choisir *la* solution, mais pour proposer un ensemble de solutions possibles. Ensuite l'état-major prend ses responsabilités et détermine quel plan sera finalement retenu. D'autre part, les méthodes de recherche opérationnelle rendent difficiles les diagnostics d'erreur, pour comprendre par exemple pourquoi il n'existe pas de solution à un problème donné.

COAST (voir la section 2) est basé sur une architecture client-serveur. Comme l'exécution des tâches d'une ligne de conduite peut être vue comme un système concurrent, le cœur de l'outil a été conçu au moyen de réseaux de Petri colorés [Jen92, Jen94, KCJ98]. Une solution peut être trouvée en examinant l'espace des états accessibles. Le client fournit une interface graphique qui rend l'utilisation des méthodes formelles transparente à l'utilisateur.

La synchronisation de tâches est une fonctionnalité importante des outils de planification, puisque bien souvent les tâches doivent être coordonnées pour atteindre une efficacité maximale. Cependant les synchronisations proposées par COAST sont très limitées. De même les mécanismes de synchronisation fournis par les logiciels civils tels que Microsoft Project [MSP] ou PSN [PSN] ne sont pas adaptés à la complexité de la planification opérationnelle. D'une façon générale, les approches mises en oeuvre souffrent du manque d'une définition générale des synchronisations.

Dans la section 3, une définition rigoureuse des contraintes de synchronisations entre tâches est proposées, basée sur cinq schémas élémentaires de synchronisation. Un vaste éventail de synchronisations peut être décrit, incluant les synchronisations usuelles des outils de planification. Dérivés directement de cette définition, les *scénarios alternatifs* permettent d'envisager différents comportements selon le mode de terminaison des tâches. La section 4 décrit l'implantation dans COAST de ces mécanismes de synchronisation. Les synchronisations sont traduites de façon automatique en préconditions et postconditions adéquates. Comme le modèle de conditions utilisé dans COAST n'est pas assez puissant pour supporter totalement cette traduction, un modèle étendu est implanté.

2 COAST

COAST a pour but d'offrir une plateforme de développement et d'analyse de lignes d'action pour la planification opérationnelle. Cet outil doit à la fois être fiable et disposer d'une interface ergonomique permettant ainsi son utilisation à des non-informaticiens. Il permet, à partir d'un ensemble de tâches complexes, de fournir des ordonnacements aboutissant à un but fixé. De plus, les diagnostics d'erreur permettent de comprendre pourquoi une ligne d'action ne convient pas.

2.1 Concepts de base

COAST manipule des *tâches* partageant un ensemble de *ressources*. L'environnement est décrit par un *ensemble de conditions* qui peuvent être satisfaites ou non. L'*état initial* indique les ressources disponibles et les conditions satisfaites au début d'une mission. Un *état terminal* définit les conditions devant être satisfaites pour le succès de la mission.

À chaque tâche est associé un ensemble de paramètres conditionnant son exécution : *pré-conditions*, *post-conditions*, *ressources nécessaires et/ou consommées*. Une tâche peut *se terminer normalement*, *avorter*, ou *se terminer en échec*.

Les pré-conditions des tâches se répartissent en trois catégories : celles qui doivent être satisfaites pour que la tâche puisse commencer à s'exécuter, ou se terminer, ou pendant toute la durée de l'exécution.

Les post-conditions sont également de trois types : les *post-conditions immédiates* deviennent satisfaites dès que la tâche commence son exécution, les *post-conditions de durée* sont satisfaites pendant toute la durée de l'exécution, et les *post-conditions finales* sont valides à la fin de l'exécution si celle-ci s'est terminée normalement.

Les tâches peuvent être reliées les unes aux autres à l'aide de *synchronisations*, une tâche pouvant être ainsi retardée jusqu'à ce que ses contraintes de synchronisation soient satisfaites.

2.2 Modélisation par réseaux de Petri colorés

Habituellement, les outils de planification utilisent des méthodes de recherche opérationnelle. Cependant ces méthodes ne sont pas adaptées aux objectifs de COAST. En effet, la recherche opérationnelle vise à trouver rapidement des solutions optimales ou quasi-optimales. Par contre, aucun retour vers l'utilisateur n'est assuré en cas d'échec du calcul. Dans le cas de COAST, d'une part il n'y a pas de définition d'une meilleure solution et d'autre part le temps de calcul n'est pas un facteur essentiel. De plus, le retour d'informations vers l'utilisateur est essentiel, que ce soit des diagnostics d'erreurs ou des indicateurs sur les lignes de conduite.

Par ailleurs l'exécution des tâches d'une ligne de conduite et leur compétition pour s'approprier les ressources peut être vue comme un système concurrent. Les conditions sont alors considérées comme un type particulier de ressources que les tâches consomment (pré-conditions) ou produisent (post-conditions). Dès lors, l'utilisation des réseaux de Petri colorés [Jen92, Jen94, KCJ98] pour modéliser ces tâches est naturelle. Les lignes de conduite cherchées ainsi que les diagnostics d'erreur peuvent être trouvés en examinant l'espace des états accessibles.

La figure 1 montre un réseau de Petri coloré modélisant simplement les tâches manipulées par COAST, sans prendre en compte ni les synchronisations ni les modes de terminaison. Les couleurs (types) des jetons utilisés sont les tâches, les ressources et les conditions. Les places *Idle*, *Executing* et *Done* correspondent aux états des tâches. Les deux transitions *Begin* et *End* représentent le début et la fin d'une tâche. Une tâche débute (resp. termine) si ses pré-conditions de début et ses ressources (resp. pré-conditions de fin) sont disponibles. Les post-conditions de début (resp. ses post-conditions de fin et ses ressources) sont “produites”.

Le réseau de Petri utilisé dans COAST est plus complexe, mais suit les mêmes idées. Il est présenté dans la section 4.

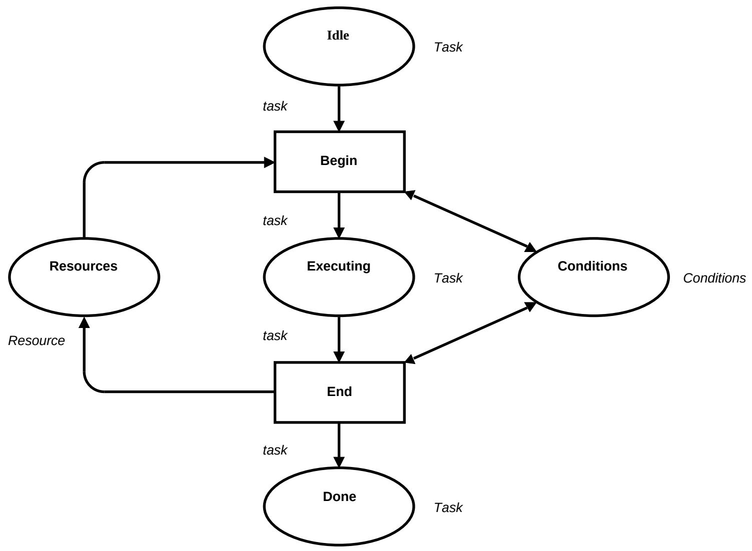


FIG. 1 – Modélisation simple des tâches par réseau de Petri coloré

2.3 Architecture logicielle

L’outil COAST dispose d’une architecture distribuée. Le serveur est un moteur d’analyse de réseau de Petri colorés ([Jen92]) et le client fournit une interface rendant l’utilisation de réseaux colorés et l’analyse par graphes d’états transparente à l’utilisateur. Celà permet également de faire tourner le moteur d’analyse sur une machine puissante et d’utiliser une autre machine avec des performances moindres pour l’interface graphique.

L’architecture de COAST est décrite dans la figure 2. L’entrée de l’outil, via l’interface graphique, est constituée de la description des tâches et des ressources. L’outil calcule des plans d’action et les analyse. Ces résultats peuvent être sauvegardés pour une utilisation ultérieure.

Le client est principalement composé d’un éditeur et d’un analyseur de plans d’action. L’utilisateur peut ainsi utiliser des tâches et ressources soit prédéfinies soit qu’il décrit lui-même. L’analyseur permet ensuite de générer, visualiser et évaluer des lignes de conduite.

Le serveur est constitué de l’implémentation (compilée) sous DESIGN/CPN ([CPN]) d’un modèle générique des tâches, ainsi que d’un ensemble d’algorithmes d’analyse de graphes d’états disponibles dans DESIGN/CPN. Le réseau coloré correspondant sera présenté dans la section 4.

La communication entre le client et le serveur s’effectue grâce à l’utilisation de la bibliothèque COMMS/CPN ([GK01]) qui permet de faire communiquer DESIGN/CPN avec des applications externes via TCP/IP. La couche

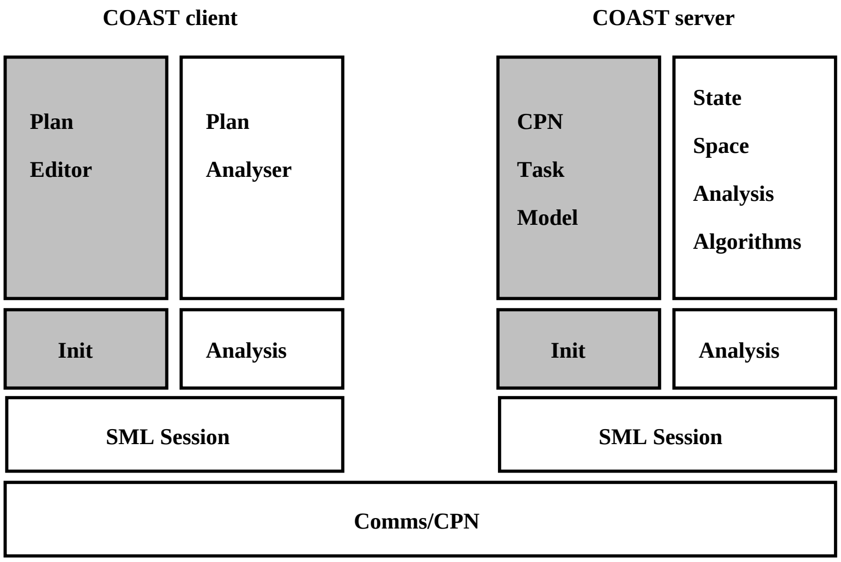


FIG. 2 – L'architecture logicielle de COAST

session SML permet au client d'appeler des fonctions du serveur et de recevoir les résultats. Le client fournit le marquage initial du réseau coloré, et les paramètres correspondant à l'ensemble de tâches et de ressources, via la couche *Init*. L'appel des fonctions d'analyse du graphe d'états s'effectue au travers de la couche *Analyse*.

Dans la section suivante, nous explicitons les différents mécanismes de synchronisation entre tâches que nous avons mis en œuvre.

3 Contraintes de Synchronisation

La synchronisation de tâches est une fonctionnalité importante des outils de planification, puisque bien souvent les tâches doivent être coordonnées pour atteindre une efficacité maximale. Ces synchronisations peuvent prendre différentes formes. Par exemple, des tâches peuvent commencer ou terminer en même temps (voir le premier exemple ci-dessous) (1) ; ou une tâche commence après qu'une autre termine (2) ; ou une tâche commence à une date précise (*trigger*) (3) ; ou une tâche se termine avant une date précise (*deadline*) (4). Dans un environnement réel, une tâche peut avoir des contraintes de synchronisation complexes (5). Voici cinq exemples réalistes de synchronisations.

(1) *Déclencher une attaque simultanée sur une cible unique.*

(2) *Déclencher un assaut après une tâche de renseignement.*

(3) *Déclencher un assaut à 15h.*

(4) *Achever une mission de reconnaissance avant le troisième jour.*

(5) *Mener deux bombardements aériens sur une même cible en une heure.*

À notre connaissance, aucun outil de planification ne peut gérer tous ces exemples de planification. Habituellement, les triggers (3) et les deadlines (4) sont proposées. Certains outils proposent des synchronisations entre tâches (commencer ou finir en même temps (1), commencer après une autre tâche (2)). Mais des synchronisations complexes telles que (5) ne sont pas gérées. La version initiale de COAST propose un seul mécanisme de synchronisation : des groupes de tâches peuvent commencer ou terminer simultanément.

Dans la suite, nous proposons une définition précise des synchronisations, qui englobe les synchronisations usuelles mais aussi des synchronisations plus complexes. Nous définissons tout d'abord trois groupes de propriétés indépendants entre eux. Ces propriétés sont ensuite combinées pour dégager des *schémas élémentaires de synchronisation*. Nos synchronisations sont celles qui s'obtiennent par instantiation et composition de ces schémas élémentaires. Nous montrons l'expressivité de ces schémas sur les cinq exemples introduits plus haut. Finalement, une utilisation judicieuse de ces schémas et de la notion d'événement de tâche amène à la notion de *scénario alternatif*, qui s'avère utile pour le raffinement des missions.

3.1 Définitions

Tout d'abord nous introduisons les notions d'*événement de tâche* et de *délai*. Les événements de tâches sont des événements relatifs au déroulement d'une tâche, par exemple son début ou sa fin. En examinant les exemples donnés plus haut, il apparaît que les synchronisations se font entre événements de tâches et non entre tâches. Les délais servent à affiner les événements de tâches, en leur ajoutant une quantité de temps (par exemple *fin de A + 10 min*). Ils permettent de rapprocher les synchronisations relatives aux tâches et celles liées à l'horloge globale.

Nous définissons trois types de propriétés sur les synchronisations. Premièrement nous distinguons entre *synchronisation forte* (notée S), quand un événement de tâche est lié à un instant précis, et *synchronisation faible* (W), quand un événement de tâche est lié à un intervalle de temps, par exemple "avant 15h" ou "après la fin de A".

Deuxièmement nous distinguons entre *synchronisation absolue* (A), quand un événement de tâche est synchronisé avec l'horloge globale, et *synchronisation relative* (R), quand un événement de tâche est synchronisé avec un autre événement de tâche.

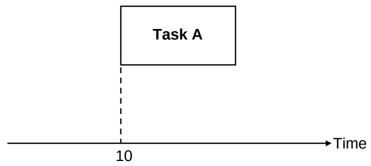
Troisièmement nous distinguons, dans le cas des synchronisations relatives, entre *synchronisation bilatérale* (B), quand les tâches impliquées ont toutes effectivement une contrainte (contraintes mutuelles), et *synchronisation unilatérale* quand une seule des tâches a une contrainte par rapport aux autres. Cela permet de distinguer entre "A débute ssi B débute" (bilatérale, A et B démarrent ensemble ou ne démarrent pas) et "A débute seulement au démarrage de B" (unilatérale, seul A a une contrainte).

3.2 Schémas de synchronisation

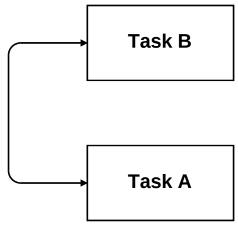
Pour extraire les schémas de synchronisation, nous combinons les propriétés précédentes. Mais certains schémas obtenus sont incohérents à cause des synchronisations bilatérales. Considérons par exemple une synchronisation bilatérale forte où A doit commencer à "commencement de B + 10 heures". Cela signifie que B débute si et seulement si A debute avec certitude 10 heures plus tard. Le commencement de B est donc soumis à une condition future.

Pour éviter ces problèmes, la notion de synchronisation bilatérale est restreinte de deux façons : une synchronisation bilatérale ne peut être que forte, et ne peut avoir de délai (c.à-d. que le délai vaut 0). Les synchronisations avec des délais restreints sont dites *à faible précision* (L).

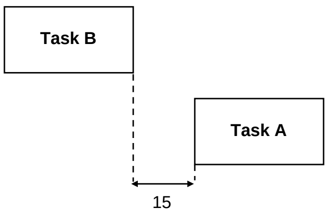
Cette méthode nous amène à distinguer cinq schémas élémentaires de synchronisations : SA-Sync, BSRL-Sync, SH-Sync, WA-Sync,WRH-Sync. Les figures ci-après décrivent une instantiation particulière de chacun des schémas. Il doit être clair que les événements de tâches (“début”, “fin”), pour les synchronisations relatives, et les relation temporelles (“avant”, ”après”), pour les synchronisations faibles, peuvent être quelconques.



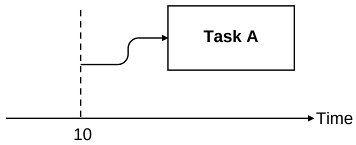
(a) SA-Sync : A débute à t=10



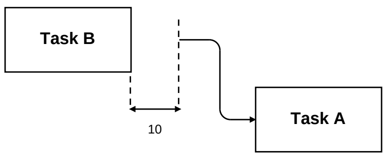
(b) BSRL-Sync : A et B débutent ensemble



(c) SRH-Sync : A débute à (fin de B + 15)



(d) WA-Sync : A débute après t=10



(e) WRH-Sync : A débute après (fin de B + 10)

FIG. 3 – Exemples de chacun des 5 schémas de synchronisation définis

3.3 Exemples d'utilisation

Nous montrons comment ces schémas de synchronisation peuvent être utilisés pour modéliser les cinq exemples introduits au début de cette section.

1. *Déclencher une attaque simultanée sur une cible unique :* les commencements des assaults sont synchronisés avec une BSRL-Sync (figure 3(b)).
2. *Déclencher un assaut après une tâche de renseignement :* le début de l'assaut est lié à la fin de la tâche de renseignements par une WRH-Sync de type "après" (figure 3(e)).
3. *Déclencher un assaut à 15h :* le début de la tâche est lié à l'horloge globale par une SA-Sync (figure 3(a)).
4. *Achever une mission de reconnaissance avant le troisième jour :* la fin de la mission de reconnaissance est liée à l'horloge globale par une WA-Sync de type "avant" (figure 3(d)).
5. *Mener deux bombardements aériens sur une même cible en une heure :* le premier bombardement n'a pas de contraintes tandis que le second en a deux. Son commencement est lié à la fin du premier assaut par une WRH-Sync, sa fin est liée au début du premier bombardement par une WRH-Sync avec délai d'une heure (figure 3(e)). L'ensemble de ces synchronisation est représenté dans la figure 4.

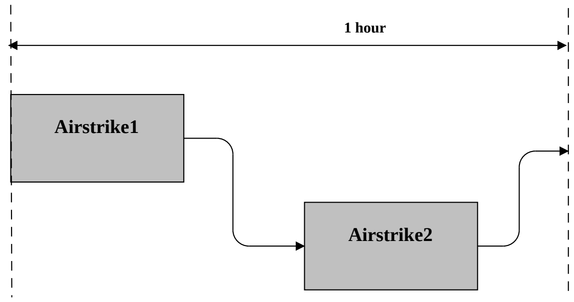


Fig. 4 – Schéma du cinquième exemple de synchronisation

3.4 Scénarios alternatifs

Pour le moment nous n'avons considéré que des tâches se terminant normalement. Nous envisageons maintenant le cas où une tâche est synchronisée avec une autre tâche qui échoue ou est interrompue. Dans le cas d'une synchronisation bilatérale, la tâche synchronisée doit clairement être interrompue à son tour. Pour une synchronisation unilatérale, les choses sont plus complexes. La distinction entre différentes terminaisons peut ne pas être pertinente, ou au contraire il peut être nécessaire de distinguer différents cas selon que la tâche réussit, échoue ou est interrompue. Pour ce faire, nous introduisons deux nouveaux événements de tâches : *échec* et *interruption*. Ils permettent de spécifier des *scénarios alternatifs*, c'est-à-dire des synchronisations dépendantes du mode de terminaison des tâches condidérées. Des applications réalistes sont par exemple des plans d'urgence ou des missions de sauvetage (figure 5). Cette notion montre tout son intérêt dans la phase d'analyse des lignes de conduite : une fois que quelques lignes de conduites ont été définies, elles peuvent être raffinées avec des scénarios alternatifs pour comparer leur robustesse.

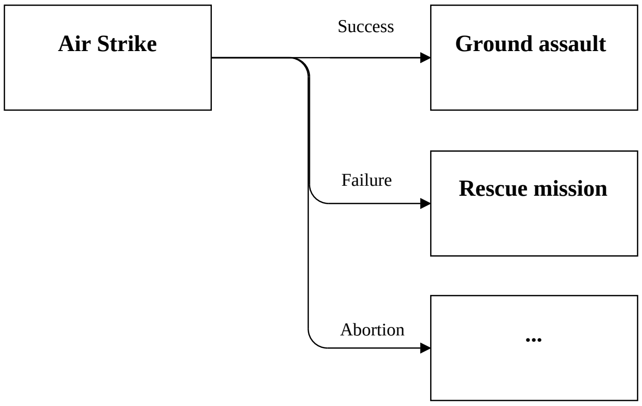


FIG. 5 – Scénarios alternatifs pour une attaque aérienne

3.5 Bilan sur les synchronisations

Considérant les différentes remarques de cette section, nous proposons la grammaire de la figure 6 pour définir les contraintes de synchronisation entre tâches.

Synchronization	: :=	BilateralSync
		| UnilateralSync
Tasks	: :=	task | Tasks, task
BilateralSync	: :=	BILATERALSYNCH (Tasks, Mode1)
UnilateralSync	: :=	Relation (task1, Mode1, task2, Mode2, delay)
		| Relation(task1, Mode1, delay)
Relation	: :=	AFTER
		| BEFORE
		| AT
Mode1	: :=	BEGIN
		| NORMALEND
Mode2	: :=	Mode1
		| FAILURE
		| ABORTION

FIG. 6 – Grammaire des synchronisations

Dans la section suivante, nous expliquons comment ces contraintes de synchronisation ont été intégrées au sein de COAST.

4 Implantation

L'architecture logicielle de COAST comportant un client et un serveur, la nouvelle représentation des synchronisations doit être reflétée des deux côtés. Elle se traduit par une *représentation logique* dans le client, similaire à la grammaire présentée dans la section précédente. Le serveur est un réseau coloré hiérarchique ([Jen92]) générique pour tout système de tâches. Il doit par conséquent être en mesure de recevoir les informations du client et les traduire de manière appropriée dans le réseau de Petri pour former une *vue opérationnelle*.

Les synchronisations gérées au préalable par COAST étaient relativement simples : des ensembles de tâches devant toutes commencer ou toutes se terminer simultanément. Ceci se traduit dans le réseau coloré par des transitions qui ne sont franchissables que si toutes les tâches concernées sont prêtes à commencer ou à se terminer. Il a donc été nécessaire de transformer le modèle de manière à gérer les contraintes de synchronisation plus complexes que nous avons introduites.

Les synchronisations envoyées par le client sont d'abord traduites automatiquement par le serveur en une représentation adaptée à leur gestion. Les synchronisations bilatérales (groupes de tâches) sont gérées comme expliqué ci-dessus. Les synchronisations unilatérales sont traduites en ajoutant des préconditions et des post-conditions adéquates aux tâches concernées. L'horloge est vue comme une tâche qui démarre à t=0 et qu'on ne peut arrêter.

La hiérarchie initiale du modèle est présentée dans la figure 7. On remarque qu'outre les comportements des tâches et la modélisation de l'environnement, trois pages de la hiérarchie sont utilisées pour l'initialisation : *Initialization*, *Synchronization* et *Resources*. Ce sont ces pages qui permettent d'instancier le réseau de Petri générique avec un marquage initial correspondant à l'ensemble de tâches et de ressources à étudier. La traduction des synchronisations s'effectue donc naturellement ici.

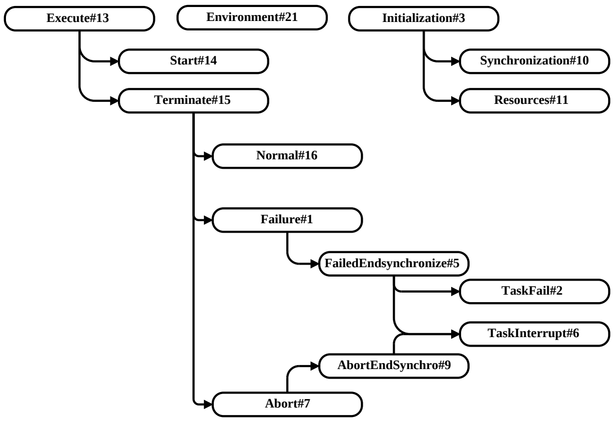


FIG. 7 – La hiérarchie initiale du modèle

La nouvelle hiérarchie d'initialisation, présentée dans la figure 8, prend en compte les nouveaux types de synchronisation. La figure 9 détaille le réseau coloré de la page d'initialisation. Le marquage initial du réseau ne dépend pas de l'ensemble de tâches et de ressources à étudier : en effet, il ne comprend qu'un seul jeton, de valeur élémentaire e dans la place *TasksInit*. L'outil utilisé, Design/CPN ([CPN]), permet d'associer du code ML à la transition *Tasks Init* pour charger un fichier contenant la description des tâches. L'ensemble des tâches est donc instancié lors du franchissement de cette transition et les contraintes de synchronisation sont ensuite initialisées par la transition de substitution *Synchronisation Init*. Le fait d'utiliser toujours le même modèle et le même marquage initial permet de générer un exécutable, basé sur le réseau coloré et contenant les outils de construction et d'analyse du graphe d'états. Cet exécutable est le cœur opérationel de COAST.

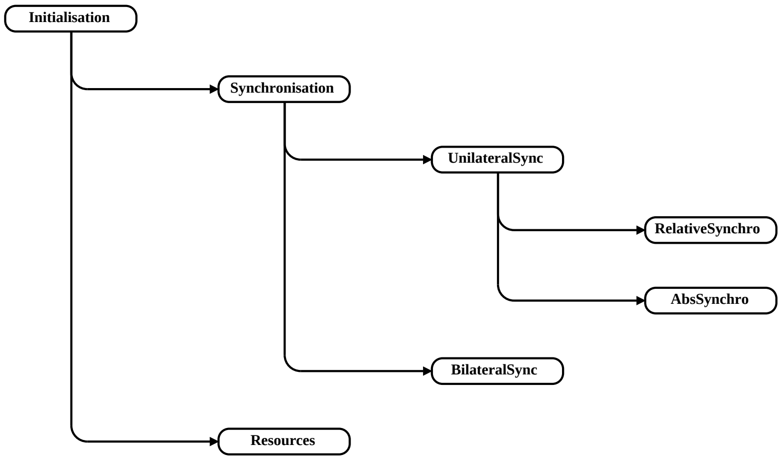


Fig. 8 – L'initialisation dans le nouveau modèle

À chaque type de synchronisation que nous avons défini correspond une page du réseau coloré hiérarchique. Ces nouvelles synchronisations faisant aparaître des contraintes de temps qui n'existaient pas précédemment, nous avons dû ajouter des temporisations. Le modèle de réseaux temporisés utilisé dans Design/CPN est celui défini dans [Jen92]. Aux jetons temporisés est associée la date de consommation du jeton au plus tôt. L'horloge est globale. La figure 10 montre la modélisation de *synchronisation relative*. Chacune des transitions correspond à un type parmi BEFORE, AT et AFTER. L'utilisation du langage ML dans Design/CPN permet de construire des fonctions de haut niveau pour exprimer les gardes et la transformation des jetons.

COAST vise à trouver des plans d'actions pour atteindre un but. Ceci se traduit par l'existance d'un chemin permettant d'atteindre ce but dans le graphe d'états du réseau coloré. Le résultat obtenu est alors renvoyé par le serveur au client, affichant ainsi une (ou plusieurs) ligne(s) de conduite à l'utilisateur.

5 Conclusion

La planification militaire est une tâche cruciale et non triviale. Nous avons présenté COAST, un outil d'aide à la planification militaire, destiné plus particulièrement à assister les phases de développement et d'analyse des lignes de conduites. COAST possède une architecture client-serveur. Le cœur de COAST est fondé sur les méthodes formelles (réseaux de Petri colorés) tandis que l'inter-

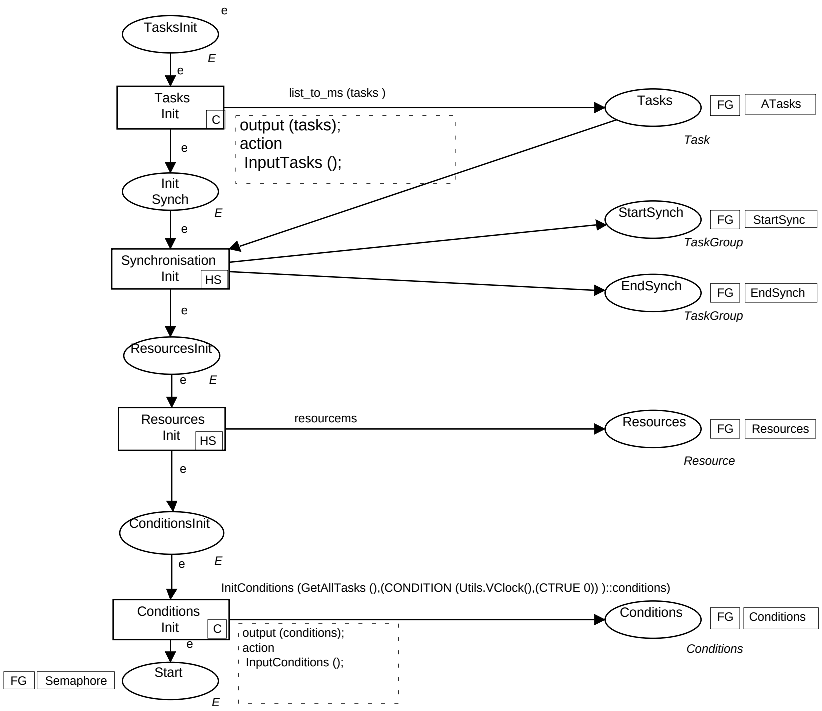


Fig. 9 – La page d'initialisation

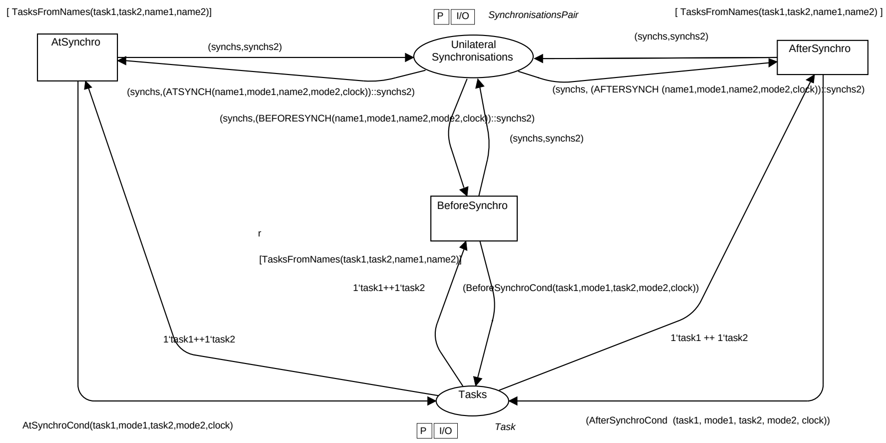


Fig. 10 – La gestion des synchronisations relatives

face utilisateur masque leur utilisation aux usagers. Les synchronisations sont un élément essentiel pour la planification. Nous proposons cinq schémas élémentaires de synchronisation, à partir des notions d'événement de tâche et de délai. Une large palette de synchronisations peuvent être exprimées en combinant ces schémas, y compris les synchronisations usuelles disponibles dans d'autres outils, mais aussi des synchronisations originales mixant événements de tâche et délais. De plus, la notion d'événement de tâche permet naturellement de différencier les modes de terminaison d'une tâche, ce qui nous amène aux scénarios alternatifs permettant de spécifier des missions de sauvetage ou des plans de secours. Ces mécanismes de synchronisation ont été implantés dans COAST. Les synchronisations sont traduites de façon automatique en préconditions et postconditions adéquates. Comme le modèle de conditions utilisé dans COAST n'est pas assez puissant pour supporter totalement cette traduction, un modèle étendu par ajout de temporisations est implanté. L'analyse du graphe d'états du modèle permet de proposer à l'utilisateur des lignes de conduite.

Les travaux futurs devront permettre aux utilisateurs de disposer des nouvelles possibilités de synchronisation au travers d'une interface graphique ergonomique. D'autre part, la génération de graphe d'états et l'analyse de propriétés pourront être rendues plus performantes en incluant des techniques de réduction.

Remerciements :

Nous tenons à remercier Lars Michael Kristensen, Jonathan Billington et Lin Zhang grâce auxquels ce travail a pu être mené. Ils ont participé à diverses discussions nous permettant de mieux cerner le problème.

Références

[CPN] DESIGN/CPN *online*. `http://www.daimi.au.dk/designCPN`.

[FZD01] L. Falzon, L. Zhang, et M. Davies. *Hierarchical Probabilistic Model for Operational Level Course of Action Development.* Dans Proc. 6th Command and Control Research and Technology Symposium, Annapolis, juin 2001.

[GK01] G. Gallash et L. Kristensen. *Comms/CPN : a communication infrastructure for external communication with Design/CPN.* Dans Proc. 3rd Workshop on Practical Use of Coloured Petri Nets, Aarhus, Denmark, août 2001.

[Jen92] K. Jensen. *Coloured Petri Nets : Basic concepts, analysis methods and practical use. Volume 1 : basic concepts.* Monographs in Theoretical Computer Science. Springer, 1992.

[Jen94] K. Jensen. *Coloured Petri Nets : Basic concepts, analysis methods and practical use. Volume 2 : analysis methods.* Monographs in Theoretical Computer Science. Springer, 1994.

[KCJ98] Lars M. Kristensen, Søren Christensen, et Kurt Jensen. The practitioner's guide to coloured Petri nets. *International Journal on Software Tools for Technology Transfer : Special section on coloured Petri nets*, 2(2) :98–132, 1998.

[KMZB02] L. Kristensen, B. Mitchell, L. Zhang, et J. Billington. *Modelling and Analysis of Operational Planning Processes using Coloured Petri Nets.* Dans Proc. Workshop on Formal Methods Applied to Defence Systems, Adelaide, Australia, volume 12 de *Conferences in Research and Practice in Information Technology.* Australian Computer Society, juin 2002.

[MSP] *Microsoft Project Home Page.* `http://www.microsoft.com/office/project/default.asp`.

[PSN] *PSN Home Page.* `http://www.sciforma.com/products/ps_suite/ps_suite.htm`.

[ZKJ+02] L. Zhang, L. Kristensen, C. Janczura, G. Gallash, et J. Billington. *A Coloured Petri Net based Tool for Course of Action Development and Analysis.* Dans Proc. Workshop on Formal Methods Applied to Defence Systems, Adelaide, Australia, volume 12 de *Conferences in Research and Practice in Information Technology.* Australian Computer Society, juin 2002.