Allocation de fonctions de commande de systèmes critiques par recherche d'atteignabilité dans un réseau d'automates communicants

Thibault Lematttre

N° ENSC-2013/454

THESE DE DOCTORAT
DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

présentée par

Monsieur Thibault LEMATTRE

pour obtenir le grade de

DOCTEUR DE L'ÉCOLE NORMALE SUPÉRIEURE DE CACHAN

Domaine :
Électronique, Électrotechnique, Automatique

Titre :

Allocation de fonctions de commande de systèmes critiques par recherche d'atteignabilité dans un réseau d'automates communicants

Thèse présentée et soutenue à Cachan, le 09 juillet 2013, devant le jury composé de :

Kamel Barkaoui	Professeur, CNAM - CEDRIC	Rapporteur
Audine Subias	Maître de conférences, HDR, Université de Toulouse-LAAS	Rapporteur
Benoit Iung	Professeur, Université de Lorraine - CRAN	Examinateur
Patrice Leclaire	Maître de conférences, SUPMECA Paris - LISMMA	Examinateur
Jean-Marc Faure	Professeur, SUPMECA Paris - LURPA	Directeur de Thèse
Jean-François Pétin	Professeur, Université de Lorraine - CRAN	Directeur de Thèse
Patrick Salaün	Ingénieur Chercheur Sénior - EDF R&D	Invité

Laboratoire Universitaire de Recherche en Production Automatisée
ENS de Cachan/ EA 1385/Université de Paris Sud 11
61 avenue du Président Wilson - 94235 Cachan Cedex

Centre de Recherche en Automatique de Nancy
UMR 7039 CNRS - Université de Lorraine
BP 70239, 54506 Vandoeuvre les Nancy, France

Résumé - La conception d'architectures opérationnelles d'un système de contrôle-commande est une phase très importante lors de la conception de systèmes de production d'énergie. Cette phase consiste à projeter l'architecture fonctionnelle sur l'architecture organique tout en respectant des contraintes de capacité et de sûreté, c'est-à-dire à allouer les fonctions de commande à un ensemble de contrôleurs tout en respectant ces contraintes. Les travaux présentés dans cette thèse proposent :
i) une formalisation des données et contraintes du problème d'allocation de fonctions ;
ii) une méthode d'allocation, par recherche d'atteignabilité, basée sur un mécanisme d'appel/réponse dans un réseau d'automates communicants à variables entières ;
iii) la comparaison de cette méthode à une méthode de résolution par programmation linéaire en nombres entiers.

Les résultats de ces travaux ont été validés sur des exemples de taille réelle et ouvrent la voie à des couplages entre recherche d'atteignabilité et programmation linéaire en nombres entiers pour la résolution de problèmes de satisfaction de systèmes de contraintes non linéaires.

Mots Clefs - Architectures de commande, Architecture opérationnelle, Automates communicants, Recherche d'atteignabilité, Vérification formelle.

Abstract - The design of operational control architectures is a very important step of the design of energy production systems. This step consists in mapping the functional architecture of the system onto its hardware architecture while respecting capacity and safety constraints, i.e. in allocating control functions to a set of controllers while respecting these constraints. The work presented in this thesis presents :
i) a formalization of the data and constraints of the function allocation problem ;
ii) a mapping method, by reachability analysis, based on a request/response mechanism in a network of communicating automata with integer variables ;
iii) a comparison between this method and a resolution method by integer linear programming.

The results of this work have been validated on examples of actual size and open the way to the coupling between reachability analysis and integer linear programming for the resolution of satisfaction problems for non-linear constraint systems.

Keywords - Control architectures, Operational architecture, Communicating automata, Reachability Analysis, Formal Verification.

Table des matières

Table des figures

Liste des tableaux

Introduction Générale

Cette thèse a été réalisée dans le cadre d'une convention CIFRE (Convention Industrielle de Formation par la REcherche) entre le Laboratoire Universitaire de Recherche en Production Automatisée (LURPA) de l'École Normale Supérieure de Cachan, le Centre de Recherche en Automatique de Nancy (CRAN) de l'Université de Lorraine et le département Simulation et Traitement de l'information pour l'Exploitation des systèmes de Production (STEP) d'EDF R&D.

La conception de systèmes de contrôle-commande est un point important dans la conception des centrales nucléaires. La conception du système de contrôle-commande suit un cycle de vie dont la première étape est la définition des fonctions du système de contrôle-commande. Ces fonctions ont pour mission de surveiller et de commander le processus. Parmi ces fonctions, certaines sont plus importantes (critiques) que d'autres afin de garantir la sûreté des installations. Une fois les fonctions définies ainsi que leurs différentes caractéristiques, il faut choisir le matériel qui accueillera ces fonctions. Ce matériel comprend des réseaux, des automates programmables industriels, des alimentations, ... Le matériel permet d'exécuter les fonctions du système de contrôle-commande et de transmettre les informations. Pour exécuter ces fonctions, elles doivent être allouées sur le matériel.

Cette allocation de fonctions sur le matériel se fait en respectant certaines contraintes qui peuvent être de plusieurs types :

- des contraintes de sûreté imposant que les fonctions les plus critiques soient allouées dans le matériel le plus fiable ;
- des contraintes de capacité empêchant à du matériel d'accueillir des fonctions s'il n'a plus de place ;
- des contraintes de sûreté exprimant le besoin de redondance fonctionnelle

Lorsqu'on recherche les solutions possibles d'un problème d'allocation de fonctions sous contraintes, qui est un problème de satisfaction de contraintes, plusieurs solutions peuvent être recherchées :

- la non existence de solutions dans le cas où le système est sur-contraint ;
- la recherche d'une unique solution réalisable ou faisable ;
- la recherche d'un ensemble de solutions réalisables dans le but de comparer ces solutions sur des critères de performance ou de fiabilité par exemple ;
- la recherche d'une solution optimale.

Des travaux ont déjà été réalisés dans cette optique au sein d'EDF pour travailler sur l'automatisation de l'allocation de fonctions sous contraintes. Ce travail a été réalisé dans le cadre de la tranche N4 (tranche précédente à l'EPR : European Pressurized Reactor) afin d'aider les personnes effectuant l'allocation manuellement. Ces travaux ont abouti à un prototype nommé THOR qui avait pour application le N4 et s'appuyait sur des données d'un autre outil d'EDF nommé ODIN servant de base de données. Ce travail n'a pas été réutilisé car il était très figé sur les fonctions et le matériel du N4.

Nous avons donc proposé une méthode d'allocation se basant sur des automates communicants par appel/réponse et une recherche de solution faite par recherche d'atteignabilité. Dans cette méthode, un générateur de demandes d'allocation est associé à chaque fonction et un récepteur de demandes acceptant ou refusant les demandes à chaque contrôleur.

Ce mémoire de thèse comporte cinq chapitres qui nous permettent de présenter la problématique scientifique et la formalisation des données du problème, d'exposer nos contributions de résolution, et enfin de valider expérimentalement ces propositions au travers de plusieurs cas d'étude.

Le premier chapitre débute par la description des architectures d'un système de contrôle-commande et de leur conception afin de présenter la problématique scientifique et les objectifs de ces travaux. Une formalisation des données du problème de satisfaction de contraintes est ensuite présentée ainsi qu'un exemple qui sera résolu dans la suite du mémoire.

Le deuxième chapitre présente tout d'abord une brève synthèse de méthodes de résolution des problèmes de satisfactions de contraintes. Par la suite, la première proposition

de cette thèse sera développée sur la résolution du problème d'allocation de fonctions par une approche de programmation linéaire en nombres entiers. Certaines limitations de cette approche nous dirigeront vers une méthode novatrice utilisant la théorie des systèmes à événements discrets pour résoudre un problème de satisfaction de contraintes.

Le troisième chapitre présente la contribution théorique majeure de ce travail de thèse : une méthode originale d'allocation d'un ensemble de fonctions à un ensemble de contrôleurs en respectant des contraintes de capacité et de sûreté. Cette méthode repose sur deux principes :

- le processus d'allocation est modélisé par un ensemble de mécanismes concurrents d'appel/réponse dans un réseau d'automates communicants à variables partagées ;
- une solution d'allocation est obtenue par recherche d'atteignabilité dans ce réseau.

Le quatrième chapitre présente une comparaison des deux méthodes d'allocation décrites dans ce mémoire, la programmation linéaire en nombres entiers décrite au chapitre 2 et l'approche par recherche d'atteignabilité décrite au chapitre 3. Cette comparaison est faite sur la base de deux études de cas (20 fonctions et 200 fonctions) en recherchant

- une solution réalisable ou faisable ;
- une solution optimale.

Le dernier chapitre présente les possibilités d'extension de l'approche proposée par recherche d'atteignabilité avec la capacité de prendre en compte de nouvelles contraintes d'allocation. Ces ajouts reposent sur une nouvelle contrainte et la possibilité de traiter des cas de ré-allocation. Enfin un méta-modèle permettant d'intégrer cette méthode dans un atelier de conception d'architecture de contrôle-commande est proposé.

En conclusion, une synthèse des résultats de ces travaux de thèse est effectuée et des perspectives à court, moyen et long terme sont proposées.

Chapitre 1
Contexte et formalisation du problème

Sommaire

1.1 Introduction

L'objectif de ce chapitre est de présenter le contexte industriel relatif à la conception d'architecture de contrôle-commande de centrale de production d'énergie et d'en dégager une problématique scientifique qui fera l'objet de ce mémoire.
La première partie du chapitre est donc consacrée à la présentation des architectures de contrôle-commande selon trois points de vue relatifs aux architectures fonctionnelle, organique et opérationnelle. L'analyse des méthodes de conception et de vérification de ces architectures fait apparaître un besoin industriel en termes d'aide à l'allocation des fonctions de contrôle-commande sur des supports matériels d'exécution conformément à de nombreuses exigences (capacité, sûreté, performances, ...). Dans un deuxième temps, une formalisation de ce problème industriel sous la forme d'un problème de satisfaction de contraintes est proposée. Ce chapitre se conclut par une synthèse exposant les objectifs de mes travaux de thèse.

1.2 Les architectures de contrôle-commande des systèmes de production d'énergie

1.2.1 Le contrôle-commande des systèmes de production d'énergie

De manière générale, un système de contrôle-commande d'une installation industrielle de production d'énergie est un ensemble de matériels et logiciels réalisant un ensemble de fonctions telles que :

- des fonctions de mesure et d'actionnement qui interagissent avec le procédé ; par exemple, lecture, filtrage et traitement d'un signal et conversion analogique - numérique pour la fonction de mesure ou bien gestion de la puissance et conversion numérique-analogique pour la fonction d'actionnement ;

- des fonctions de commande qui traitent les informations fournies par les fonctions de mesure et les demandes émanant des opérateurs pour calculer une commande appropriée, envoyer des requêtes aux actionneurs ou fournir des informations synthétiques et fiables aux opérateurs ;

- des fonctions de surveillance et de conduite de l'installation permettant, depuis la

salle de commande, de visualiser l'état du procédé et des chaines d'actionnement et de mesure ou de transmettre des requêtes au système de commande ;

- des fonctions de mise à disposition des données d'exploitation vers d'autres applications ; par exemple, de gestion technique ou de maintenance.

L'ensemble de ces fonctions est exécuté sur des matériels très divers tels que des cartes électroniques, des automates programmables industriels, des systèmes de supervision (SCADA), ... pouvant embarquer (ou non) du logiciel et pouvant communiquer à l'aide de réseaux informatiques ou de liaisons filaires électriques.

L'organisation des fonctions de contrôle-commande, en termes d'interactions et de hiérarchisation notamment, est définie dans ce que l'on appelle, par la suite, l'architecture fonctionnelle. L'organisation des matériels, en termes de regroupement et de communication notamment, est définie dans ce que l'on appelle par la suite, l'architecture organique. Enfin, le couplage entre les architectures fonctionnelle et organique correspond à l'allocation des fonctions de contrôle-commande sur les matériels et constitue ce que l'on appelle par la suite, l'architecture opérationnelle.

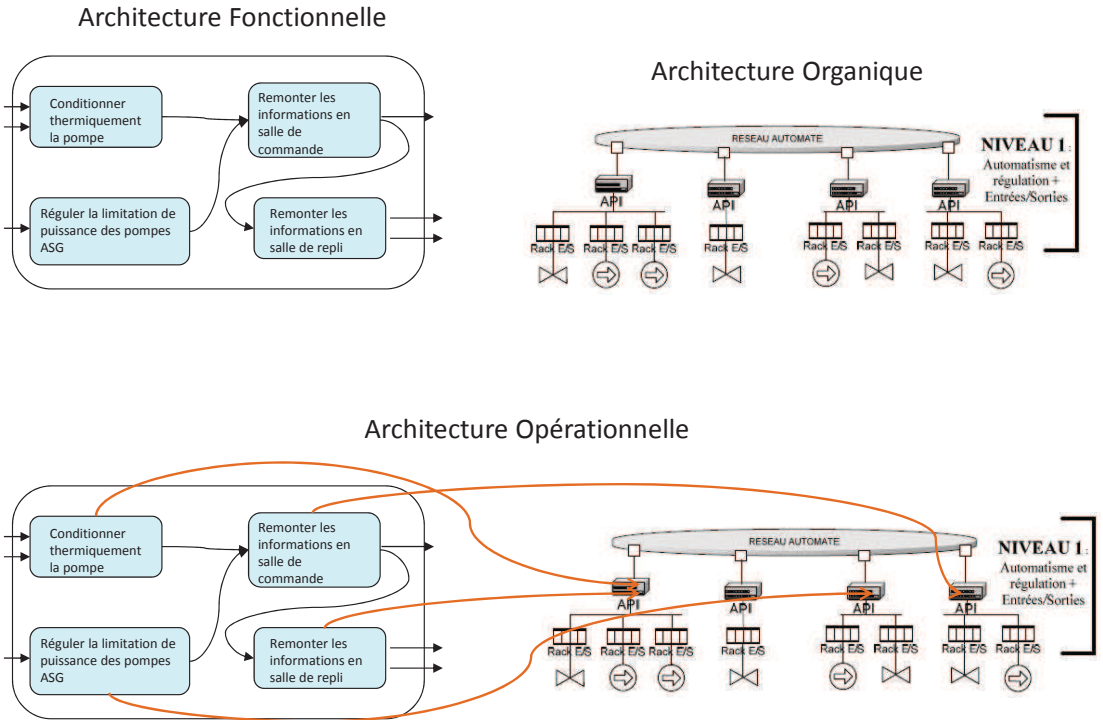


Figure 1.1 – Architectures fonctionnelle, organique et opérationnelle d'un système de contrôle-commande

La figure 1.1 présente un exemple pour illustrer ces différentes architectures :

- l'architecture fonctionnelle comporte une fonction principale "Alimenter en secours les générateurs de vapeurs (ASG)" décomposée en 4 sous-fonctions échangeant de l'information entre elles et avec le procédé ;
- l'architecture organique composée d'Automates Programmables Industriels (API), de cartes d'entrées / sorties (aussi appelées E/S) et de réseaux de communication ;
- l'architecture opérationnelle qui est la projection de l'architecture fonctionnelle sur cette architecture organique.

1.2.2 Architecture fonctionnelle

Une architecture fonctionnelle est définie pour un procédé que l'on suppose connu et a pour objectif :

- d'identifier l'ensemble des services de contrôle-commande nécessaires pour réaliser la mission de l'installation et sa mise en sécurité ;
- de structurer ces fonctions sous la forme d'un réseau de fonctions hiérarchisées et inter-connectées.

La hiérarchisation permet de structurer les fonctions de contrôle-commande selon différents niveaux d'abstraction et d'en proposer une classification efficace. La décomposition de plus haut niveau dans le domaine du contrôle-commande nucléaire fait apparaître 3 familles de fonctions :

- la famille des fonctions du niveau 0 pour les fonctions d'actionnement et de mesure ;
- la famille des fonctions du niveau 1 pour les fonctions réflexes de commande (boucles de régulation, fonctions logiques de protection, ...) ;
- la famille des fonctions du niveau 2 pour les fonctions de conduite des installations.

Les derniers niveaux de décomposition aboutissent à des fonctions élémentaires de contrôle-commande, "élémentaire" signifiant que ces fonctions ne peuvent plus être décomposées en sous-fonctions et, par voie de conséquence, qu'elles seront exécutées sur un seul et même matériel. **Dans la suite du mémoire, nous nous focaliserons sur les fonctions élémentaires de contrôle-commande de niveau 1**. A titre d'information

et pour fixer les ordres de grandeurs, le nombre de fonctions élémentaires de niveau 1 du contrôle-commande est compris entre quelques centaines et un à deux milles selon les tranches de production nucléaire.

L'interconnexion des fonctions au sein d'un réseau a pour objectif d'identifier les échanges d'informations :

- entre fonctions, sous la forme de relations entre les entrées consommées par une fonction donnée et les informations produites par d'autres fonctions ;

- entre les fonctions et le procédé, c'est-à-dire les informations émises (ou reçues) à destination (ou en provenance) du procédé ; ces informations, que nous appellerons par la suite les informations d'entrées/sorties, ont un impact direct sur le dimensionnement de l'architecture de contrôle-commande puisqu'elles induisent, pour chaque fonction, les caractéristiques d'interface avec le procédé (cartes d'entrées/sorties) dont celles-ci auront besoin.

Les installations de production d'énergie nucléaire sont des installations critiques soumises à de fortes contraintes de sûreté. Les analyses de risques préconisées par les normes en vigueur relatives à la conception des systèmes de contrôle-commande conduisent à associer à chacune des fonctions un Facteur de Criticité (noté FC) dépendant de la gravité qu'une défaillance de la fonction fait encourir à l'installation et de sa fréquence [EDF, 2010]. L'association d'un Facteur de Criticité à une fonction traduit explicitement un ensemble d'exigences opérationnelles relatives à l'accès à un service (critère de défaillance unique, tenue aux agressions externes comme les séismes par exemple, maintien dans un état sûr, ...). A titre d'exemple, les fonctions relatives à la mission d'exploitation seront considérées comme moins critiques pour la sûreté que les fonctions de surveillance ou les fonctions de protection du cœur.
Lors de la description des fonctions, un facteur de criticité est donc associé à chaque fonction. Ce facteur est défini comme présenté ci-dessous de la fonction la plus critique (F1A) à la fonction la moins critique (NC) :

- Facteur de criticité F1A : associé aux fonctions de sûreté qui permettent d'atteindre et de maintenir l'état contrôlé (caractérisé par un cœur sous-critique et

l'évacuation de la puissance résiduelle avec inventaire en eau stable) après tout initiateur de type accidentel,

- Facteur de criticité F1B : associé aux fonctions de sûreté qui permettent, à partir de l'état contrôlé, d'atteindre et de maintenir l'état sûr (caractérisé par un cœur sous-critique, des rejets radioactifs maintenus dans des limites prédéfinies et l'évacuation durable de la puissance résiduelle) après tout initiateur de type accidentel,

- Facteur de criticité F2 : associé aux fonctions de sûreté qui permettent d'atteindre et de maintenir l'état final dans les conditions de fonctionnement RRC-A[1], d'empêcher le dépassement du rejet limite dans les conditions RRC-B[2], de réduire les effets d'une agression interne (incendie) ou externe (séisme, chute d'avion), le contrôle de la radioactivité en situation normale,

- Facteur de criticité NC : associé aux fonctions non classées de sûreté, ne faisant pas partie des fonctions précitées.

La définition du facteur de criticité d'une fonction obéit aux règles suivantes :

- toutes les sous-fonctions d'une fonction "mère" donnée ont un facteur de criticité au moins aussi critique que celui de la fonction "mère",

- toutes les fonctions élémentaires sont associées à un facteur de criticité.

Le tableau 1.1 présente pour trois Systèmes Élémentaires (SE) caractéristiques, des exemples de fonctions de contrôle-commande et leur facteur de criticité.

Dans la suite de ce mémoire, nous considérerons que chaque fonction élémentaire peut être décrite par :

- **ses caractéristiques d'interface avec le procédé, en particulier le nombre d'entrées et de sorties consommées et produites,**

- **un facteur de criticité.**

1. Réduction des Risques Catégorie A, guides de conception de sûreté spécifiques du projet EPR ayant pour objectif la diminution du risque de fusion du coeur
2. Réduction des Risques Catégorie B, guides de conception de sûreté spécifiques du projet EPR ayant pour objectif la diminution du risque de rejets accidentels dans l'environnement

Tableau 1.1 – SE, fonctions et facteur de criticité des fonctions

Nom du système élémentaire	Fonctions du système élémentaire	Facteur de criticité
AAD ; Alimentation des générateurs de vapeur	Distribuer le fluide à l'aval des réchauffeurs AHP par le dispositif de pompage AAD	**NC**
	Distribuer le fluide du barillet NC amont AHP utilisé en secours vers la ligne AAD	**NC**
	Conditionner la ligne AAD jusqu'à l'aval des réchauffeurs AHP	**NC**
	Conditionner thermiquement la pompe AAD	**NC**
ASG ; Alimentation de Secours des Générateurs de vapeur	Régulation de limitation de puissance des pompes ASG	**F1A**
	Protection de la pompe i602PO (pompe indicé 602)	**NC**
	Injection ASG sur bas niveau GV	**F1A**
	Injection ASG sur injection de sécurité et perte des alimentations électriques externes	**F1A**
	Isolement ASG sur haut niveau GV	**F1A**
	Délestage des pompes ASG lors du démarrage diesel	**F1A**
DVL ; Ventilation des locaux électriques du bâtiment électrique	Air conditionné et approvisionnement des divisions 1 et 4	**F1B**
	Air conditionné et approvisionnement des divisions 2 et 3	**F1B**
	Distribution et récupération de l'air dans le RSS	**F2**

AHP : Poste d'eau haute pression et réchauffeurs
GV : Générateur de Vapeur
RSS : panneau de repli

1.2.3 Architecture organique

L'architecture organique représente l'organisation des différents matériels et/ou logiciels (Automates Programmable Industriel (API), réseaux, armoires accueillant les API, les alimentations, ...) permettant d'exécuter les fonctions préalablement définies dans l'architecture fonctionnelle. De la même manière que pour les fonctions, l'architecture organique est généralement construite de manière hiérarchisée comme le montre la figure 1.2 :

- au niveau le plus bas, se trouvent les équipements de terrain (capteurs, actionneurs) servant à l'acquisition de mesures et à l'action sur le procédé. Les constantes de temps associées à ce niveau sont de l'ordre de quelques millisecondes (ms).

- le niveau 1 est constitué des automatismes programmables (API, processeurs spécialisés, ...) servant de support d'exécution aux fonctions logiques et aux régulations. Les constantes de temps associées à ce niveau sont de l'ordre de quelques dizaines ou centaines de millisecondes (ms).

- au niveau 2, se trouvent des équipements informatiques plus lourds (calculateurs, serveurs d'applications, système de gestion de base de données, ...) servant de support aux systèmes de conduite et de supervision de la centrale. Les constantes de temps associées à ce niveau sont de l'ordre de quelques secondes.

- le dernier niveau concerne les équipements de mise à disposition des données vers des applications externes telles que la maintenance ou la gestion technique ; ces équipements sont du même type que ceux de niveau 2 mais les constantes de temps sont plutôt de l'ordre de la minute.

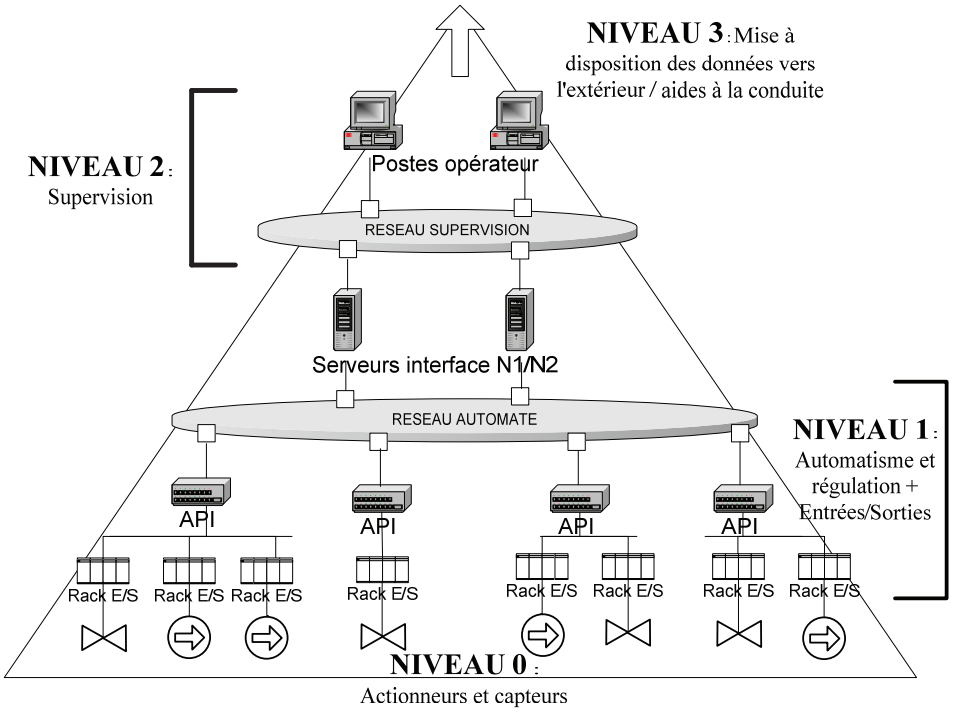


Figure 1.2 – Système de contrôle-commande dans l'industrie nucléaire

La suite de ce mémoire se focalisant sur les fonctions de contrôle-commande de niveau 1, seuls les matériels de niveau 1 seront considérés dans notre étude. Nous appellerons "contrôleur" (encadré sur la figure 1.3) un ensemble comprenant :

- un Automate Programmable Industriel (API) composé de processeurs de traitement et de communication,
- un ou plusieurs racks de modules d'entrées/sorties reliés aux équipements de terrain (actionneurs et capteurs) : il en existe plusieurs catégories selon le type logique ou analogique des signaux émis et/ou reçus,
- un réseau fond de panier reliant les API aux racks d'entrées/sorties.

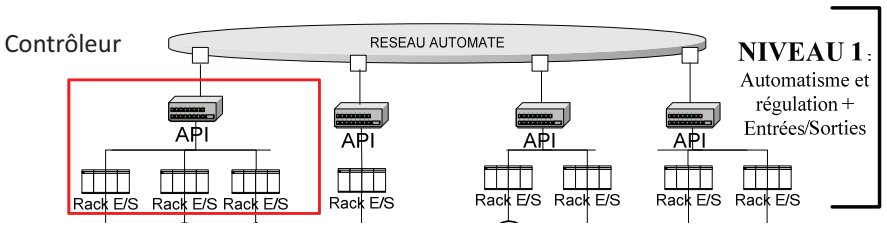


Figure 1.3 – Niveau 1 du système de contrôle-commande

De manière similaire aux facteurs de criticité associés aux fonctions de contrôle-commande, tous les équipements présents dans l'architecture organique sont qualifiés vis-à-vis de critères relatifs à leur sûreté de fonctionnement, principalement à leur fiabilité et/ou leur disponibilité. Cette qualification est une caractéristique intrinsèque des équipements qui dépend de leur tolérance aux fautes et qui peut être, selon les cas, attribuée par les fournisseurs de ces équipements ou bien résultant d'une procédure de qualification par les utilisateurs ou les autorités de sûreté. De manière plus générale, le niveau de sécurité d'un contrôleur résulte :

- de la fiabilité intrinsèque de chacun de ses composants,
- de son architecture matérielle : les constructeurs mettent à disposition sur le marché des automates dits "de sécurité" mettant en œuvre des architectures redondantes incluant des auto-tests ou des voteurs et permettant de palier à une défaillance d'un processeur.

Il va sans dire que le coût d'un contrôleur est directement dépendant de son niveau de sécurité, expliquant ainsi la présence sur le marché d'une large gamme de produits offrant des niveaux de sécurité différents.

Dans la suite de ce mémoire, nous considérerons que les contrôleurs sont caractérisés par :

- **des caractéristiques techniques, notamment les caractéristiques des modules d'entrées/sorties,**
- **leur niveau de sécurité.**

1.2.4 Architecture opérationnelle

L'architecture opérationnelle peut être définie comme la projection de l'architecture fonctionnelle sur l'architecture organique, comme le montre la figure 1.1. Elle contient l'ensemble des informations caractérisant le contexte opérationnel du système de contrôle-commande et en particulier :

- les caractéristiques des fonctions et des contrôleurs telles qu'identifiées dans les sections précédentes,
- une matrice d'allocation représentant les relations entre un ensemble de fonctions élémentaires de contrôle-commande et un ensemble de contrôleurs sous la forme d'une application surjective,
- la caractérisation des flux de communication entre équipements induits par l'allocation des fonctions sur les matériels.

Compte tenu de la hiérarchisation des architectures fonctionnelle et organique, l'architecture opérationnelle peut être définie à plusieurs niveaux d'abstraction :

- on parlera d'architecture opérationnelle de niveau *tranche* lorsque l'on décrira les allocations des grandes fonctions de contrôle-commande sur des groupes d'équipements constituant la plupart du temps un lot "fournisseur". Nous pouvons citer, à titre d'exemple, les systèmes PICS (Process Information and Control System), SICS (Safety Information and Control System), RSS (Remote Shutdown Station), PAS (Process Automation System), SAS (Safety Automation System), PS (Protection System) ou encore RCLS (Reactor Control Limitation System),
- on parlera d'architecture opérationnelle *détaillée* lorsque l'on décrira les allocations des fonctions élémentaires de contrôle-commande sur des contrôleurs tels que ceux définis au paragraphe précédent.

Dans le cadre de ce travail, nous considérerons une architecture opérationnelle simplifiée contenant l'énumération d'un ensemble de fonctions et de contrôleurs (et leurs caractéristiques) ainsi que la matrice d'allocation.

1.3 Conception des architectures opérationnelles

1.3.1 Exigences en conception d'architectures opérationnelles

Les principes généraux de la démarche de conception d'une architecture opérationnelle peuvent se résumer comme suit. Le point de départ consiste à définir :

- l'architecture fonctionnelle, c'est-à-dire un ensemble de fonctions de commande interconnectées ; cette architecture fonctionnelle est supposée connue et non modifiable dans la suite du processus,

- les caractéristiques techniques des différents contrôleurs disponibles (modules d'entrées / sorties, niveau de sécurité) ; ces éléments ne constituent qu'une définition partielle de l'architecture organique puisque le nombre de contrôleurs et des supports de communication est *a priori* inconnu et ne sera fixé que plus tard lors de la conception de l'architecture opérationnelle.

Ces deux points constituent les données d'entrée d'un processus d'allocation de fonctions sur des équipements supportant leur exécution c'est-à-dire sur des contrôleurs dans notre cas. Ce processus d'allocation aboutit à l'architecture opérationnelle. Il doit répondre à des exigences multiples dont :

- des exigences de capacité qui traduisent la capacité technique d'un matériel à supporter une fonction donnée compte tenu du nombres d'entrées / sorties dont il dispose ou encore de sa capacité mémoire, de sa charge CPU, ...,

- des exigences de sûreté qui peuvent être de plusieurs types :

  - des exigences de compatibilité entre le niveau de criticité d'une fonction et le niveau de sécurité d'un contrôleur ; il est par exemple évident que les fonctions les plus critiques ne devront pas être implantées sur les matériels disposant du niveau de sécurité le plus faible,

  - des exigences de séparation qui stipulent que deux fonctions redondantes (définies comme deux fonctions permettant de rendre un service identique par des procédures et des moyens différents) ne devront pas être exécutées sur un même contrôleur pour éviter des modes de défaillance communs ; nous pouvons par exemple citer le cas de deux fonctions de protection d'un même élément de procédé et stimulées respectivement par un dépassement de seuil de température et de pression.

- des exigences de performances telles que le temps de réponse d'une chaine fonctionnelle depuis un stimulus capteur jusqu'à une commande reçue par un actionneur ou encore le volume d'information échangé sur les réseaux de communication, ...) ;

- des exigences de coût dépendant en particulier du nombre de contrôleurs utilisés dans l'architecture opérationnelle et de leur niveau de sécurité.

Ces exigences étant plus ou moins contraignantes, elles sont classées en deux catégories :

- les exigences évaluées de manière binaire (satisfaites ou pas) et que l'architecture opérationnelle doit impérativement respecter ; ces exigences sont de manière générale relatives à la capacité ou à la sûreté,

- les autres exigences seront considérées comme des critères que l'on cherchera à améliorer lors de la conception de l'architecture opérationnelle. On pourra par exemple chercher à minimiser le nombre de contrôleurs, les temps de réponse ou encore la charge réseau.

1.3.2 La conception des architectures opérationnelles au sein d'EDF

La conception du système de contrôle-commande suit généralement le cycle de vie défini à la figure 1.4. Ce cycle de vie est dit en "W" car, en plus des bases d'un cycle en "V" pour les phases d'ingénierie et d'intégration/validation, est ajoutée une branche relative aux modifications réalisées lors des visites décennales (mise à jour de certaines fonctions, gestion de l'obsolescence).

Lors de la phase descendante, la conception de l'architecture opérationnelle se fait actuellement en deux étapes au sein d'EDF :

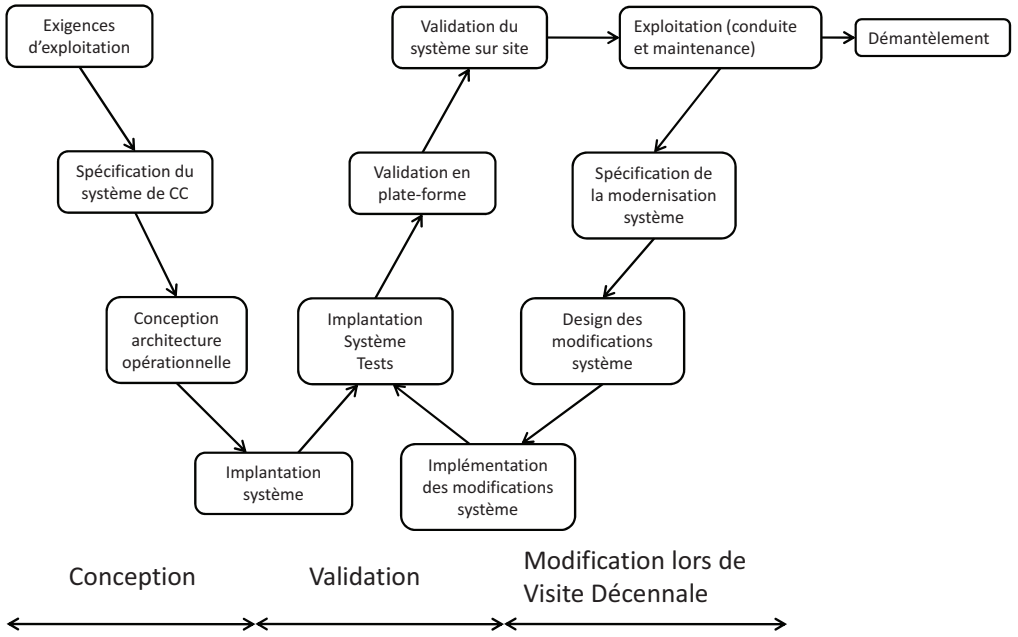


Figure 1.4 – Cycle de vie du système de contrôle-commande

- la première répartition porte sur la conception de l'architecture opérationnelle de niveau *tranche* en ne considérant que les grandes macro-fonctions et l'architecture organique globale exprimée en termes de lignes de défense. Cette première macro-allocation a pour but de définir les stratégies de défense, les classes de systèmes (PAS, PICS, SAS, ...) et les missions du système de contrôle-commande,

- la deuxième étape porte sur la définition de l'architecture opérationnelle *détaillée* en allouant les fonctions élémentaires de contrôle-commande sur les contrôleurs.

Dans la pratique industrielle actuelle, l'allocation réalisée lors de la deuxième étape est réalisée de façon non automatisée en se basant sur le savoir-faire des concepteurs et constitue une tâche fastidieuse et consommatrice de temps. Toute erreur faite lors de cette étape peut s'avérer extrêmement coûteuse par la suite, en particulier si elle est révélée lors d'étapes tardives de vérification et de validation, voire de mise en service.

Pour améliorer son processus de conception des architectures opérationnelles, le besoin industriel exprimé par EDF peut se décliner en deux points relatifs à :

- une méthode automatisée permettant de traiter au plus tôt et de manière déterministe les exigences qui doivent impérativement être satisfaites par l'architecture opérationnelle (capacité, sûreté) ; cette méthode doit conduire à la production d'un ensemble de solutions acceptables vis-à-vis des exigences considérées ;

- une méthode d'amélioration des architectures produites précédemment au regard des critères d'optimisation définis dans la section précédente (nombre de contrôleurs, temps de réponse, charge réseau, ...) ; ces critères ne peuvent être évalués qu'*a posteriori* lorsque l'ensemble de l'architecture est connu.

Dans la mesure où les modèles et méthodes utilisés le permettent, ces deux points pourront être éventuellement intégrés au sein d'une seule et même proposition.

1.3.3 Pratiques actuelles en conception d'architecture opérationnelle

1.3.3.1 Outils de modélisation

Les premiers éléments pouvant apporter une réponse partielle à la problématique posée reposent sur des outils de modélisation permettant aux équipes d'ingénierie de partager les diverses représentations des architectures fonctionnelle, organique et opérationnelle.

La modélisation des architectures de contrôle-commande n'est pas une préoccupation spécifique du secteur de la production d'énergie électrique. Les secteurs de l'aéronautique et de l'automobile se sont également intéressés à cette problématique et ont retenu le langage AADL (Architecture Analysis & Design Language) [AADL, 2009] pour y répondre. Ce langage AADL, dédié à l'origine aux systèmes temps réels embarqués, permet de décrire un modèle fonctionnel et un modèle organique. L'architecture opérationnelle est réduite à la définition de relations d'allocation entre les objets des modèles fonctionnel et organique. Les outils d'analyse développés pour ce langage sont souvent limités à la vérification syntaxique ou de cohérence vis-à-vis du méta-modèle d'AADL.

Dans le même ordre d'idée, les approches basées sur les langages UML (Unified Modeling Language) [UML, 2011] ou SysML (System Modeling Language) [SysML, 2008] permettent une représentation des architectures fonctionnelle (sous la forme de diagrammes d'activité par exemple) et organique (sous la forme de diagrammes de block par exemple) et des relations d'allocation entre les objets de ces deux modèles [Willard, 2007]. L'absence de sémantique formelle limite également l'analyse des modèles produits à de simples vérifications de cohérence.

Des efforts pour coupler ces différents modèles statiques à des modèles dynamiques formels ont été entrepris, notamment dans le cadre de projets tels que OOONEIDA [Auinger *et al.*, 2005] ou TORERO [Ferrarini *et al.*, 2005] basés sur les outils comme

FBDK (Calgary University) et CORFU [Tranoris et Thramboulidis, 2006]. Ils font appel à des profils spécifiques tels que UML-PA [Katzke et Vogel-Heuser, 2005] ou Marte [UML MARTE, 2009] ou encore à des environnements intégrés. La plate-forme OpenembeDD [INRIA, 2005] construite autour de Kermeta [Muller *et al.*, 2005], Topcased [TOPCASED, 2010], ..., et interfacée à des outils de modélisation dynamique comme Scicos [Campbell *et al.*, 2010], Syndex [Ghezal *et al.*, 1990], Tina [Berthomieu *et al.*, 2004], ... en est un exemple.

Si ces extensions permettent indéniablement de procéder à des évaluations, par simulation, des architectures opérationnelles, notamment sur des indicateurs de performances (temps de réponse par exemple), elles ne sont d'aucun secours pour la vérification des exigences de capacité et de sûreté et, *a fortiori*, pour la génération automatisée de solutions acceptables.

1.3.3.2 Synthèse et validation d'architecture opérationnelle

La méthode de génération d'un ensemble de solutions d'architectures opérationnelles respectant les contraintes de capacité et de sûreté et optimisant un ensemble de critères peut s'appuyer sur deux techniques complémentaires [Faure et Lesage, 2001] :

- de validation et de vérification permettant *a posteriori* de s'assurer du respect des contraintes et d'évaluer les critères retenus ;
- de synthèse permettant de produire des solutions bonnes *a priori*.

L'utilisation de la première approche consiste à laisser l'expert définir une allocation qui lui paraît satisfaisante puis à vérifier *a posteriori* le respect des contraintes et à évaluer les performances de cette solution. Si ces performances (temps de réponse, charge réseau, ...) sont supérieures ou inférieures à certains seuils, la solution résultant d'expertise est validée ; sinon une nouvelle solution d'allocation doit être proposée. Cette validation fait souvent appel à des langages et outils spécifiques selon les contraintes ou critères considérés. A titre d'exemple, si l'on souhaite évaluer des indicateurs de performances temporelles, les techniques suivantes pourront être envisagées : simulation de Monte-Carlo ou à partir de scénarii de test [Marsal *et al.*, 2006], recherche de bornes [Ruel *et al.*, 2009], méthode des trajectoires [Martin, 2004], [Martin et Minet, 2005], analyse algébrique [Addad et Amari, 2008].

En revanche, si les indicateurs que l'on cherche à quantifier concernent la fiabilité ou la disponibilité des architectures, les modèles utilisés seront souvent de nature stochastique et les techniques basées sur la simulation de Monte-Carlo. Selon la validation à effectuer, les modèles pourront donc être très divers. Des travaux permettant d'intégrer ces analyses dans des modèles de nature différente sont actuellement développés autour du concept d'"Integrated Deterministic and Probablistic Safety Assessment".

Dans la seconde approche de conception dite bonne *a priori*, l'architecture opérationnelle est automatiquement obtenue à partir des deux autres architectures (fonctionnelle et organique) et d'un ensemble de spécifications qui expriment globalement les contraintes sur les possibilités de projection. Elle est donc réputée correcte par construction.
Dans le domaine de la conception des systèmes de contrôle-commande, la principale technique de conception bonne *a priori* est basée sur la théorie de la supervision [Wonham et Ramadge, 1987] et des algorithmes de synthèse tel que celui proposé par Kumar. Dans notre contexte de conception d'architecture opérationnelle, les techniques utilisables sont celles permettant la résolution d'un système de contraintes et qui feront l'objet du chapitre 2. L'inconvénient de cette approche est qu'elle nécessite de mettre en place et de résoudre un système de contraintes. Dans le cadre d'études réelles, la modélisation du système de contraintes reste compliquée dans la mesure où certaines exigences sont implicites et relèvent des règles du métier.

Même s'il peut paraître souhaitable d'intégrer au plus tôt l'ensemble des contraintes et critères dans une démarche de conception bonne *a priori*, il apparait néanmoins clairement que :

- les contraintes qui doivent impérativement être satisfaites par l'architecture opérationnelle ont tout intérêt à être traitées au plus tôt à l'aide de techniques de synthèse ou de résolution de contraintes ;

- les performances temporelles et les indicateurs de fiabilité ou de disponibilité seront préférentiellement évalués *a posteriori* après que la totalité de l'architecture opérationnelle ait été définie ; ils reposeront donc plutôt sur des techniques de validation *a posteriori*.

Au regard des techniques utilisées en conception d'architecture opérationnelle, le problème initialement posé par notre partenaire industriel peut donc être reformulé de la manière suivante :

- les critères relatifs aux performances (temps de réponse, charge du réseau de communication, indicateurs de fiabilité et de disponibilité) seront évalués *a posteriori* sur un ensemble d'architectures opérationnelles admissibles ;
- les contraintes de capacité et de sûreté seront intégrées dans un système de résolution de contraintes adoptant la minimisation du nombre de contrôleurs utilisés comme fonction objectif.

Le travail de cette thèse est une contribution au second point relatif à la résolution d'un système de contraintes. Il faut noter que le problème traité relatif à l'allocation d'un ensemble de fonctions sur un ensemble de contrôleurs constitue une version simplifiée du problème de conception d'architecture opérationnelle. En effet, les informations échangées entre les fonctions ne sont pas prises en compte dans le système de contraintes et seront traitées ultérieurement à l'aide de techniques de validation puisqu'elles impactent essentiellement les performances temporelles.

Enfin, il convient de souligner que, comme pour tout problème d'optimisation de satisfaction de contraintes, plusieurs types de résultats peuvent être recherchés :

- trouver une solution (satisfaisant l'ensemble des contraintes) ;
- trouver un ensemble de solutions au problème ;
- trouver une solution optimale par rapport à un critère (généralement minimisation ou maximisation d'une variable) ;
- prouver la non-existence de solution (dans le cas d'un problème sur-contraint).

Chercher un ensemble de solution est très souvent apprécié par les concepteurs industriels ; cela permet en effet de comparer, à l'aide des techniques d'évaluation et de validation de performances décrites ci-dessus, plusieurs solutions qui respectent toutes les contraintes fondamentales d'allocation.

Après avoir posé de manière informelle le problème scientifique, et avant de proposer au chapitre 2 une analyse critique des techniques de résolution, la section suivante est consacrée à la formalisation du problème.

1.4 Formalisation du problème

Le problème se décrit en une allocation d'un ensemble de fonctions L sur un ensemble de contrôleurs M en respectant plusieurs sortes de contraintes de types arithmétiques ou logiques. Ce problème est souvent défini comme un problème de satisfaction de contraintes (CSP). L'ensemble des fonctions est tiré de l'architecture fonctionnelle et les contraintes des exigences de conception.

L'objectif de cette partie est de présenter l'ensemble des notations utilisées. La formalisation se fera en trois temps :

- description d'une fonction et de ses paramètres ;
- description du contrôleur et de ses paramètres ;
- description des contraintes représentant les liens entre les différents paramètres de la fonction et du contrôleur.

1.4.1 Description d'une fonction f^i

Une fonction f^i est un élément de l'ensemble F représentant l'ensemble des fonctions d'une architecture fonctionnelle. Comme certaines fonctions sont plus critiques que d'autres, un facteur de criticité est défini pour chaque fonction. Il s'agit d'un entier représentant l'impact de la défaillance de cette fonction. De plus, chaque fonction est indivisible et est composée de :

- $RR_k^i \in \mathbb{N}$ nombre de caractéristiques externes requises de type $k \in \mathbb{N}$ par une fonction f^i : chaque fonction a un certain nombre de paramètres qui la décrivent afin de connaître les données qu'elle véhicule et d'avoir ainsi une idée des données caractéristiques de la fonction utiles lors de la conception.

  Dans notre cas, nous avons utilisé quatre types de caractéristiques externes requises par la fonction afin de décrire les entrées et sorties physiques de la fonction. Elles correspondent à des informations émises à un actionneur ou reçues par un capteur. Ces caractéristiques externes sont :

  - $NI_l^i \in \mathbb{N}$, le nombre de données d'entrées logiques de la fonction f^i ;
  - $NO_l^i \in \mathbb{N}$, le nombre de données de sorties logiques de la fonction f^i ;

- $NI_a^i \in \mathbb{N}$, le nombre de données d'entrées analogiques de la fonction f^i ;
- $NO_a^i \in \mathbb{N}$, le nombre de données de sorties analogiques de la fonction f^i ;

- FC^i, le facteur de criticité d'une fonction f^i représentant l'importance de la fonction ; plus la valeur du facteur de criticité est faible, plus la fonction est critique. Le facteur de criticité ayant une valeur : $FC^i \in \{1, ..., r\}$ avec $r \in \mathbb{N}^*$.
  Dans le cas particulier du domaine de l'énergie, le facteur de criticité présenté précédemment comprend 4 valeurs :
  - $FC^i = 1$ correspond aux fonctions classées F1A : traitements correspondant aux fonctions de sûreté qui permettent d'atteindre et de maintenir l'état contrôlé après tout initiateur de type accidentel ;
  - $FC^i = 2$ correspond aux fonctions classées F1B : traitements correspondant aux fonctions de sûreté qui permettent, à partir de l'état contrôlé, d'atteindre et de maintenir l'état sûr après tout initiateur de type accidentel ;
  - $FC^i = 3$ correspond aux fonctions classées F2 : traitements correspondant aux fonctions de sûreté qui permettent d'atteindre et de maintenir l'état final dans les conditions de fonctionnement RRC-A, d'empêcher le dépassement du rejet limite dans les conditions RRC-B, de réduire les effets d'une agression interne (incendie) ou externe (séisme, chute d'avion), le contrôle de la radioactivité en situation normale, les limitations ;
  - $FC^i = 4$ correspond aux fonctions classées NC : traitements correspondant aux fonctions non classées de sûreté, ne faisant pas partie des fonctions précitées.

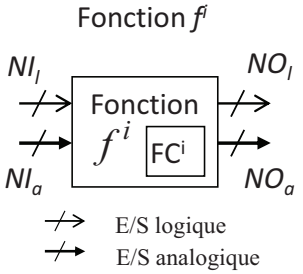


Figure 1.5 – Représentation graphique d'une fonction

La figure 1.5 est la représentation graphique d'une fonction qui sera utilisée dans les différents exemples.

1.4.2 Description d'un contrôleur c_j

Un contrôleur c^j est un élément de l'ensemble C représentant l'ensemble des contrôleurs d'une architecture organique. Chaque contrôleur est composé de :

- $PR_k^j \in \mathbb{N}$ nombre de caractéristiques techniques fournies de type $k \in \mathbb{N}$ par un contrôleur c^j. Dans la suite, nous avons utilisé quatre types de caractéristiques techniques :
  - $LI_{max}^j \in \mathbb{N}$, le nombre maximum d'entrées logiques du contrôleur c^j ;
  - $LO_{max}^j \in \mathbb{N}$, le nombre maximum de sorties logiques du contrôleur c^j ;
  - $AI_{max}^j \in \mathbb{N}$, le nombre maximum d'entrées analogiques du contrôleur c^j ;
  - $AO_{max}^j \in \mathbb{N}$, le nombre maximum de sorties analogiques du contrôleur c^j ;
- NS^j Niveau de Sécurité, ce terme sera défini de manière différente selon les domaines industriels considérés. Le niveau de sécurité ayant une valeur : $NS^j \in \{1, ..., p\}$ avec $p \in \mathbb{N}^*$

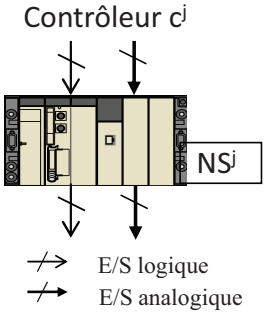


Figure 1.6 – Représentation graphique d'un contrôleur

La figure 1.6 est la représentation graphique d'un contrôleur qui sera utilisée dans les différents exemples.

1.4.3 Description des contraintes

1.4.3.1 Définition et hypothèses de travail

Les notations suivantes seront utilisées par la suite :

- une fonction f^i **pouvant être allouée** à un contrôleur c^j sera notée : $f^i \rightsquigarrow c^j$
- une fonction f^i **étant allouée** à un contrôleur c^j sera notée : $f^i \leftarrow c^j$
- l'ensemble des fonctions f^i allouées à c^j sera noté : $F_j = \{f^i \in F | f^i \leftarrow c^j\}$
- l'ensemble des indices des fonctions qui sont allouées à c^j sera noté : $I_j = \{i \in \{1, ..., L\} | f^i \leftarrow c^j\}$ avec $L = Card(F)$

Dans le cadre de cette étude, un ensemble d'hypothèses ont été émises :

- un contrôleur non vide peut héberger de 1 à n fonctions ($n \in \mathbb{N}$) tant que les contraintes d'allocation sont respectées ;
- une fonction doit être allouée à un et un seul contrôleur ;
- il n'y a pas de hiérarchie entre les fonctions. Les fonctions sont les éléments feuilles de l'architecture fonctionnelle, elles ne peuvent pas être décomposées.

Les deux paragraphes suivant formalisent les contraintes qui seront considérées comme des règles inviolables que doit respecter toute solution d'allocation. Elles permettent ainsi de restreindre l'ensemble des solutions admissibles en empêchant un certain nombre de combinaisons. Ces contraintes peuvent être réparties en deux familles :

1. les contraintes de capacité ; dans notre cas, elles correspondent aux contraintes de capacité en termes d'entrées/sorties des contrôleurs, charges CPU, ... ;
2. les contraintes de sûreté ou de répartition des fonctions.

Ces contraintes sont exprimées, respectivement, sous forme de contraintes arithmétiques et logiques comme précisé ci-dessous.

1.4.3.2 Contraintes de capacité

Ces contraintes sont des équations linéaires entre un type k de caractéristiques externes requises par une fonction f^i (RR^i_k) et un type l de caractéristiques techniques fournies par le contrôleur c^j (PR^j_l). Dans notre cas, les contraintes de capacité prises en compte sont les limites en termes d'entrées / sorties physiques du contrôleur.

Dans le cas général, il faut pour un type donné de caractéristiques que le nombre de caractéristiques externes requises par les fonctions allouées au contrôleur c^j soit inférieur

au nombre de caractéristiques techniques fournies par ce contrôleur. Ce qui s'écrit alors : $\forall i \in \{1, ..., L\}$, $\forall j \in \{1, ..., M\}$, avec $M = Card(C)$

$$f^i \rightsquigarrow c^j \text{ si } \sum_{i' \in I_j} RR_k^{i'} + RR_k^i \leq PR_k^j \tag{1.1}$$

Dans le cas particulier que nous avons choisi, les relations entre les entrées/sorties analogiques ou logiques s'écrivent de la manière suivante : $\forall j \in \{1, ..., M\}$,

$$f^i \rightsquigarrow c^j \text{ si } \sum_{i \in I_j} NI_l^i \leq LI_{max}^j \tag{1.2}$$

L'équation 1.2 représente donc la combinaison linéaire que le contrôleur accepte vis-à-vis de ses entrées logiques. La somme des entrées logiques des fonctions allouées au contrôleur c^j doit être inférieure ou égale à la capacité du contrôleur en entrées logiques.

Les équations 1.3, 1.4 et 1.5 représentent respectivement la combinaison linéaire que le contrôleur accepte vis-à-vis de ses entrées analogiques, de ses sorties logiques et de ses sorties analogiques.

$$f^i \rightsquigarrow c^j \text{ si } \sum_{i \in I_j} NI_a^i \leq AI_{max}^j \tag{1.3}$$

$$f^i \rightsquigarrow c^j \text{ si } \sum_{i \in I_j} NO_l^i \leq LO_{max}^j \tag{1.4}$$

$$f^i \rightsquigarrow c^j \text{ si } \sum_{i \in I_j} NO_a^i \leq AO_{max}^j \tag{1.5}$$

1.4.3.3 Contraintes de sûreté

Les contraintes de sûreté sont des contraintes de sécurité de l'ensemble de l'architecture. Par conséquent, la première contrainte de sûreté concerne le facteur de criticité et le niveau de sécurité. La deuxième contrainte de sûreté est de ne pas allouer deux fonctions redondantes dans un même contrôleur. Dès qu'une fonction effectuant un service donné a été allouée à un contrôleur, il n'est donc plus possible d'attribuer à ce contrôleur une autre fonction redondante pour ce même service.

Les contraintes de sûreté représentent des règles ou relations entre les fonctions et les contrôleurs. Elles permettent aussi de réduire le nombre de possibilités d'allocation. Ces contraintes de distribution peuvent être réparties en deux catégories :

1. les contraintes de sûreté de compatibilité ;

2. les contraintes de sûreté d'exclusion.

Les contraintes de sûreté de compatibilité correspondent à plusieurs relations de compatibilité entre un paramètre d'une fonction et un paramètre d'un contrôleur. Ces relations sont représentées par des matrices. Dans notre cas, nous utilisons une seule relation, et donc une seule matrice.
La matrice de compatibilité X existe et représente une relation entre le facteur de criticité d'une fonction FC et le niveau de sécurité d'un contrôleur NS :
$X : FC \times NS \longrightarrow \mathbb{B} = \{0; 1\}$
Une fonction pourra donc être allouée à un contrôleur si et seulement si la relation entre FC and NS est respectée :
$\forall i \in \{1, ..., L\}$, $\forall j \in \{1, ..., M\}$:

$$f^i \rightsquigarrow c^j \text{ si } x_{(FC^i, NS^j)} = 1 \tag{1.6}$$

La matrice X représentant cette équation peut alors s'écrire :

$$X = \begin{pmatrix} x_{(1,1)} & \cdots & x_{(1,p)} \\ \vdots & \ddots & \vdots \\ x_{(r,1)} & \cdots & x_{(r,p)} \end{pmatrix}$$

Pour des raisons de sûreté, les fonctions doivent être réparties dans les contrôleurs selon leur facteur de criticité.

Dans le cas particulier que nous avons traité par rapport au contexte industriel, les contrôleurs acceptant des fonctions de facteur de criticité $FC^i = 1$ (les plus critiques) ne peuvent accueillir que des fonctions de facteur de criticité $FC^i = 1$ ou $FC^i = 2$. Autrement dit, les fonctions les plus critiques $FC^i = 1$ ne peuvent être regroupées qu'avec des fonctions de facteur de criticité $FC^i = 1$ ou $FC^i = 2$. Les autres fonctions peuvent être rassemblées dans un même contrôleur comme suit :

- les fonctions ayant un facteur de criticité 2 avec des fonctions ayant un facteur de criticité 3 ;
- les fonctions ayant un facteur de criticité 3 avec des fonctions ayant un facteur de criticité 4.

Par conséquent, un contrôleur c^j peut héberger, quand toutes les fonctions sont allouées :

- soit des fonctions de facteur de criticité 1 et des fonctions de facteur de criticité 2 ;
- soit des fonctions de facteur de criticité 2 et des fonctions de facteur de criticité 3 ;
- soit des fonctions de facteur de criticité 3 et des fonctions de facteur de criticité 4.

La relation $\Re$ est alors définie comme suit :

$$\Re = \{(1,1),(2,1),(2,2),(3,2),(3,3),(4,3)\} \subset NS \times FC \tag{1.7}$$

La matrice X peut alors s'écrire :

$$X = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

La contrainte de sûreté d'exclusion correspond à une relation entre deux fonctions f^i et f^k qui ne peuvent être allouées dans le même contrôleur. Dans notre contexte industriel, cela correspond à deux fonctions redondantes. Cette relation est représentée par une matrice d'exclusion Y :
$Y : F \times F \longrightarrow \mathbb{B}$
Une fonction pourra donc être allouée à un contrôleur c^j si et seulement si la relation entre f^i and f^k est respectée :
$\forall i \in \{1,...,L\}, \forall k \in \{1,...,L\}$ et $\forall j \in \{1,...,M\}$:

$$(f^i \rightsquigarrow c^j) \text{ et } (f^k \rightsquigarrow c^j) \text{ si } y_{(f^i,f^k)} = 1 \tag{1.8}$$

La matrice Y représentant cette équation peut alors s'écrire :

$$Y = \begin{pmatrix} 1 & y_{(f^1,f^2)} & \cdots & y_{(f^1,f^L)} \\ y_{(f^2,f^1)} & 1 & \ddots & \vdots \\ \vdots & \cdots & 1 & y_{(f^{L-1},f^L)} \\ y_{(f^L,f^1)} & \cdots & y_{(f^L,f^{L-1})} & 1 \end{pmatrix}$$

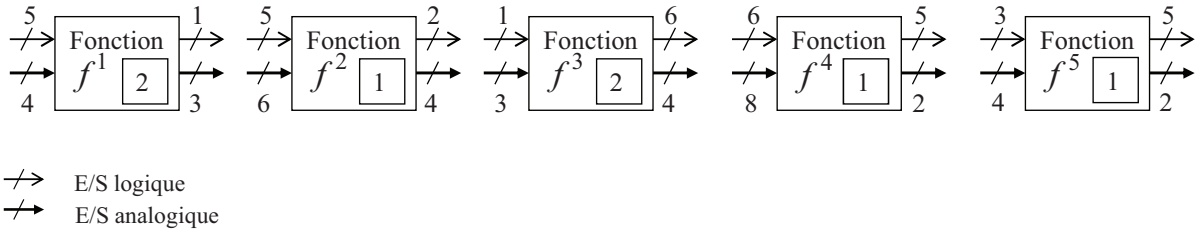
ou différemment : $\left\{y_{(f^i,f^k)} = y_{(f^k,f^i)}; y_{(f^i,f^i)} = 1\right\}$

1.4.4 Exemple à traiter

Afin de bien comprendre la solution que nous cherchons, voici un exemple qui reprend les principes de notre problème. Dans notre cas, le problème à résoudre est de trouver la répartition des fonctions en respectant les contraintes précédentes. Nous décrirons ici un exemple avec 5 fonctions qui doivent être allouées à trois contrôleurs.

1.4.4.1 Les fonctions

La figure 1.7 présente les 5 fonctions de notre exemple qui vont devoir être allouées sur un ensemble de contrôleurs.



Pour chaque bloc représentant une fonction, les entrées sont à gauche et les sorties à droite.

Figure 1.7 – Les cinq fonctions de cet exemple et leurs caractéristiques

Les fonctions ont chacune des caractéristiques et sont donc définies avec leurs connexions sur le procédé, que ce soit en entrée ou en sortie :

- le nombre d'entrées logiques de la fonction f^i : NI_l^i ;
- le nombre d'entrées analogiques de la fonction f^i : NI_a^i ;
- le nombre de sorties logiques de la fonction f^i : NO_l^i ;
- le nombre de sorties analogiques de la fonction f^i : NO_a^i.

Dans cet exemple, les caractéristiques des fonctions sont générées aléatoirement et prennent des valeurs telles que
NI_a^i, NI_l^i, NO_a^i, $NO_l^i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^4$.

Dans cet exemple, le facteur de criticité évolue entre la valeur 1 et 2, d'où $FC^i \in \{1, 2\}$.

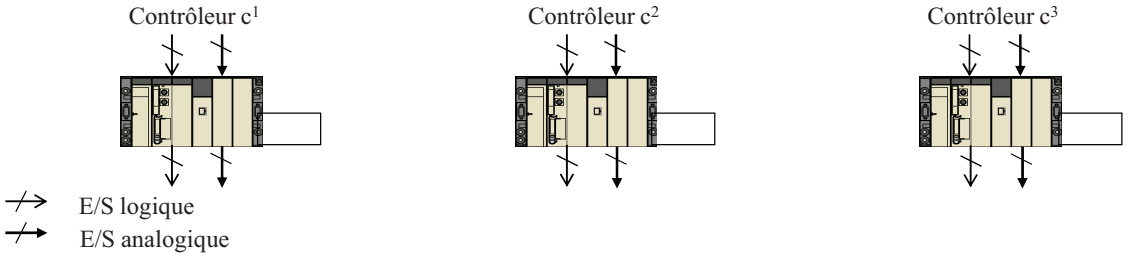
Le détail des caractéristiques de chaque fonction est noté sur la figure 1.7. Pour la fonction f^1 par exemple, ses caractéristiques sont :

- le nombre d'entrées logiques de la fonction f^1 : $NI_l^1 = 5$;
- le nombre d'entrées analogiques de la fonction f^1 : $NI_a^1 = 4$;
- le nombre de sorties logiques de la fonction f^1 : $NO_l^1 = 1$;
- le nombre de sorties analogiques de la fonction f^1 : $NO_a^1 = 3$;
- le facteur de criticité de la fonction f^1 : $FC^1 = 2$.

1.4.4.2 Les contrôleurs

La figure 1.8 présente les 3 contrôleurs de notre exemple qui vont devoir accueillir un ensemble de fonctions. Les caractéristiques de chaque contrôleur sont définies de la manière suivante pour cet exemple :

- le nombre maximum d'entrées logiques du contrôleur est limité à 10 ($LI_{max} = 10$) ;
- le nombre maximum d'entrées analogiques du contrôleur est limité à 10 ($AI_{max} = 10$) ;
- le nombre maximum de sorties logiques du contrôleur limité à 10 ($LO_{max} = 10$) ;
- le nombre maximum de sorties analogiques du contrôleur limité à 10 ($AO_{max} = 10$) ;
- le niveau de sécurité du contrôleur est de 1 ou 2 : $NS^j \in \{1, 2\}$.



Pour chaque bloc représentant un contrôleur, les entrées sont en haut et les sorties en bas.

Figure 1.8 – Les 3 contrôleurs de cet exemple et leurs caractéristiques

La figure 1.8 représente les 3 contrôleurs, avec les entrées et sorties utilisées (initialement nulles). Le rectangle en bas à droite de chaque contrôleur représente le niveau de sécurité du contrôleur (initialement non connu).

1.4.4.3 Allocation à faire

L'exemple d'allocation est donc d'allouer les 5 fonctions ainsi définies sur les 3 contrôleurs. La figure 1.9 représente le point de départ avec les 5 fonctions et les 3 contrôleurs.

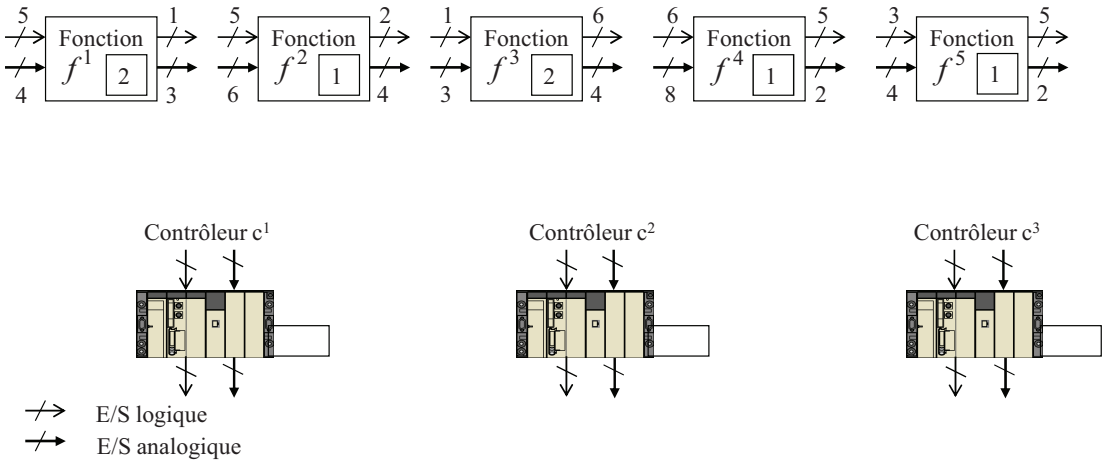


Figure 1.9 – Les cinq fonctions et les trois contrôleurs de cet exemple

L'allocation doit respecter certaines contraintes de capacité présentées dans la section 1.3 et décrites ci-dessous :

$\forall i \in \{1,2,3,4,5\}$, $\forall j \in \{1,2,3\}$:

$$f^i \rightsquigarrow c^j \text{ si } \sum_{i \in I_j} NI_l^i \leq 10 \tag{1.9}$$

$$f^i \rightsquigarrow c^j \text{ si } \sum_{i \in I_j} NI_a^i \leq 10 \tag{1.10}$$

$$f^i \rightsquigarrow c^j \text{ si } \sum_{i \in I_j} NO_l^i \leq 10 \tag{1.11}$$

$$f^i \rightsquigarrow c^j \text{ si } \sum_{i \in I_j} NO_a^i \leq 10 \tag{1.12}$$

L'allocation doit aussi satisfaire une contrainte de distribution de compatibilité. Une fonction pourra donc être allouée à un contrôleur si et seulement si la relation entre FC and NS est respectée :

$\forall i \in \{1,2,3,4,5\}$, $\forall j \in \{1,2,3\}$:

$$f^i \rightsquigarrow c^j \text{ si } x_{(FC^i, NS^j)} = 1 \tag{1.13}$$

Avec la matrice X qui s'écrit dans cet exemple :

$$X = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

L'exemple ci-dessus présente succinctement ce que nous devons faire. D'autres problèmes sont proches de ce genre de situation à résoudre. Ces problèmes répondent au contexte général de problèmes de satisfaction de contraintes et quelques uns sont explicités dans le chapitre suivant.

1.5 Conclusion

Nous avons montré dans ce chapitre que le problème industriel d'allocation de fonctions de contrôle-commande sur un ensemble de matériels supportant leur exécution pouvait se ramener à un problème de satisfaction de contraintes (CSP).
Les données d'entrée sont (Cf. Figure 1.10) :

- un ensemble de fonctions décrites par des caractéristiques externes et un facteur de criticité,
- les caractéristiques externes et le niveau de sécurité des contrôleurs disponibles,
- un ensemble de contraintes arithmétiques et logiques représentant respectivement les contraintes de capacité et de sûreté.

L'objectif du processus d'allocation est de générer automatiquement un ensemble de contrôleurs, chacun d'entre eux étant chargés de l'exécution d'une ou plusieurs fonctions, de telle sorte que (Cf. Figure 1.10) :

- l'allocation fonction/contrôleur respecte la totalité des contraintes arithmétiques et logiques lorsqu'il s'agit de fournir une solution admissible,
- l'allocation fonction/contrôleur minimise le nombre de contrôleur utilisé lorsqu'il s'agit de fournir une solution optimale.

Le chapitre suivant aborde les techniques de résolution de ce problème de satisfaction de contraintes en précisant les familles de problème de CSP les plus proches de notre problème d'allocation et les techniques classiques de résolution, notamment celles basées

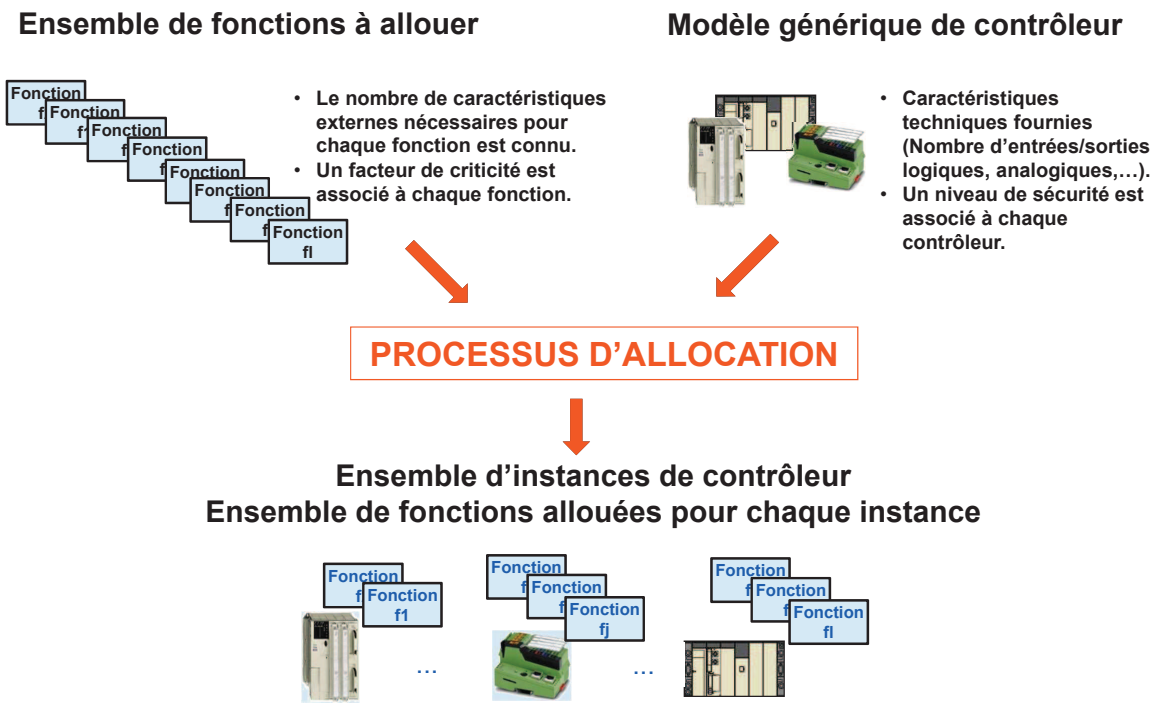


Figure 1.10 – Objectifs des travaux

sur la programmation linéaire. L'application d'une de ces techniques permettra de mettre en évidence leurs limites et nous orientera, dans la dernière partie du chapitre suivant, vers des approches originales basées sur l'analyse de modèles de Systèmes à Événements Discrets.

Chapitre

2

Approches pour la résolution d'un problème de satisfaction de contraintes

Sommaire

2.1 Introduction

Le chapitre précédent ayant montré que le problème d'allocation de fonctions de contrôle-commande pouvait être modélisé sous la forme d'un problème de satisfaction de contraintes, nous allons maintenant présenter une brève synthèse de méthodes de résolution de problème de satisfaction de contraintes issues de la recherche opérationnelle.

Cela nous permettra d'exposer dans un deuxième temps la première contribution développée lors de mes travaux de thèse : la résolution du problème d'allocation de fonctions par programmation linéaire en nombres entiers. L'application de cette proposition à des cas d'étude met cependant en évidence certaines limitations notamment en ce qui concerne l'obtention d'une (d'un ensemble de) solution(s) faisable(s).

Nous présentons enfin, dans la troisième partie de ce chapitre, quelques résultats récents et novateurs de travaux visant à la résolution d'un problème de satisfaction de contraintes à l'aide de la théorie des systèmes à événements discrets.

2.2 Approches issues de la recherche opérationnelle

Les problèmes de satisfaction de contraintes (Constraint Satisfaction Problem : CSP) sont des problèmes mathématiques représentant un certain nombre de contraintes ou de restrictions que doivent satisfaire un ensemble d'objets ou d'états.

Les problèmes de satisfaction de contraintes font l'objet de travaux en recherche opérationnelle et en intelligence artificielle avec, lorsque la complexité des problèmes est trop importante, des enjeux liés à la simplification des problèmes ou à la proposition d'algorithmes de résolution comme les heuristiques ou les méta-heuristiques. En effet, la grande complexité de ce type de problème nécessite souvent une combinaison de méthodes heuristiques et de méthodes de recherche combinatoire pour résoudre le problème dans un délai raisonnable.

Comme nous l'avons indiqué au chapitre précédent, notre problème concerne l'allocation d'un ensemble de L fonctions dans un nombre M de contrôleurs (L et M sont des entiers positifs) tout en respectant des contraintes logiques et arithmétiques modélisables à l'aide de variables entières et booléennes. Comme nous l'avons également évoqué au chapitre précédent, plusieurs recherches de solutions sont envisageables : trouver une solution faisable ou réalisable (satisfaisant l'ensemble des contraintes), trouver

un ensemble de solutions faisables au problème, trouver une solution optimale par rapport à un critère (minimisant ou maximisant une variable, dans notre cas, minimisant le nombre de contrôleurs), prouver la non-existence de solution (dans le cas d'un problème sur-contraint).

Le problème d'allocation de fonctions sous contraintes s'apparente dans la littérature à des problèmes connus sous les vocables de "problème du sac à dos" ou "problème du *Bin-Packing*". Les deux paragraphes suivants proposent une description mathématique de ces deux problèmes.

2.2.1 Problème du sac à dos

En algorithmique, le problème du sac à dos est un problème d'optimisation combinatoire. Il modélise le remplissage par des objets d'un sac à dos qui ne peut supporter plus d'un certain poids, chaque objet ayant un poids et une valeur définis. Les objets mis dans le sac à dos doivent alors maximiser la valeur totale, sans pour autant dépasser le poids maximum. Le problème d'allocation de fonctions sous contraintes se rapproche du problème du sac à dos lorsque l'on utilise plusieurs sacs à dos.

Les données des objets et du sac sont définies comme suit :

- $i \in \{1, ..., n\}$, l'indice des objets ;
- p_i, le poids de l'objet i ;
- v_i, la valeur de l'objet i ;
- P_m, le poids total que peut accepter le sac ;
- x_i, l'état de l'objet i, $x_i = 1$ si l'objet est mis dans le sac et $x_i = 0$ sinon ;
- $I = \{i \in \{1, .., n\} \mid x_i = 1\} =\mid \sum_{i=1}^{n} x_i$, représente l'ensemble des indices des objets mis dans le sac ;
- $P = \sum_i p_i = \sum_{i=1}^{n} x_i p_i$, le poids du sac ;
- $V = \sum_i v_i = \sum_{i=1}^{n} x_i v_i$, la valeur du sac.

Pour ce problème, deux contraintes sont prises en compte :

1. $P \leq P_m$: le poids mis dans le sac à dos ne doit pas dépasser le poids maximum admissible par celui-ci ;

2. $P = \sum_i p_i > P_m$: la somme des poids de tous les objets est supérieure au poids admissible par le sac.

Les hypothèses retenues sont :

- $p_i < P_m$: un objet ne peut pas être plus lourd que la charge maximale admissible par le sac ;
- $p_i > 0$: tous les objets ont un poids non nul ;
- $v_i > 0$: tous les objets ont une valeur non nulle.

L'objectif de la résolution de ce problème est de maximiser la valeur tout en respectant les contraintes.
$Obj = MaxV = \sum_{i=1}^{n} x_i v_i$

L'analogie de notre problème avec celui du sac à dos est la suivante : les objets à placer dans le sac à dos correspondent à nos fonctions, le sac à dos correspondant lui à un contrôleur. Les différences sont cependant importantes puisque :

- le problème du sac à dos est limité à un seul sac alors que nous pouvons utiliser plusieurs contrôleurs,
- tous les objets ne sont pas "rangés" dans le sac à dos alors que, par définition, toutes nos fonctions doivent être allouées,
- le critère d'optimisation est lié à l'objet (sa valeur) alors que notre processus d'allocation cherchera à minimiser le nombre de contrôleurs (donc de sacs).

Il n'en demeure pas moins que les techniques utilisées pour résoudre ce type de problème, principalement basées sur la programmation linéaire en nombres entiers ou sur des méta-heuristiques, peuvent contribuer à la résolution de notre problème d'allocation comme nous le montrerons dans la deuxième partie de ce chapitre. Il est à noter que ce problème a été démontré comme étant NP complet [Garey et Johnson, 1979].

2.2.2 Problème du *Bin packing*

Le problème de *Bin packing* consiste à ranger, sans superposition, un ensemble d'objets rectangulaires dans un nombre illimité de rectangles identiques, notés par la suite bins. L'objectif est de déterminer le nombre minimum de bins nécessaire pour ranger tous les objets. La dimension du problème de *Bin packing* correspond à une contrainte de capacité des réceptacles (bins) vis à vis des objets. Par exemple, si les objets sont de dimension 1, le problème revient à placer ces objets sur des segments de droite d'une longueur donnée. S'il existe plusieurs contraintes de capacité corrélées (par exemple contraintes de largeur et de longueur des objets à placer dans une surface donnée), on qualifiera le problème "*Bin packing* à n dimensions". Enfin, s'il existe plusieurs contraintes de capacité non corrélées, on qualifiera le problème de "*Bin packing* à n fois une dimension". Le problème du *Bin packing* à deux dimensions appartient à la famille des rangements et découpes et est noté 2SBSBPP par [Wäscher *et al.*, 2007]. C'est une généralisation du problème à une seule dimension qui est connu pour être déjà un problème NP-difficile [Garey et Johnson, 1979].

Les données du problème de *Bin packing* à deux dimensions sont définies comme suit :

- $i \in \{1, ..., n\}$, l'indice des objets ;
- a_i, le nom de l'objet i ;
- (w_i, h_i), les dimensions de l'objet i ;
- B, le bin ;
- (W, H), les dimensions du bin B ;
- $l_i = min(w_i, h_i)$, la valeur minimale des dimensions d'un objet ;
- $L_i = max(w_i, h_i)$, la valeur maximale des dimensions d'un objet ;
- $l = min(W, H)$, la valeur minimale des dimensions du bin B ;
- $L = max(W, H)$, la valeur maximale des dimensions du bin B .

L'analogie avec le problème de *Bin Packing* est la suivante : les objets à placer sont nos fonctions tandis que les réceptacles (les *Bin*) correspondent à nos contrôleurs. Dans

notre cas, le problème sera qualifié de problème de *Bin Packing* à n fois une dimension (n correspondant aux nombres de contraintes de capacité que nous avons à traiter ; dans notre cas, nous aurons n = 4 pour les contraintes de capacité sur le nombre d'entrées logiques et analogiques et de sorties logiques et analogiques).
En revanche, dans le problème de *Bin packing* classique, il n'existe pas de contraintes entre les objets à placer (tel objet ne peut se retrouver dans le même *Bin* que tel autre objet). Ce type de contrainte est pourtant à traiter dans notre cas puisqu'il correspond à des contraintes de regroupement ou d'exclusion entre fonctions induites par des considérations de sûreté. Il existe, dans la littérature, la description d'un problème de *Bin Packing* étendu [Khanafer, 2010] où des contraintes entre les objets à placer sont introduites par un graphe de conflits. Le problème présente bien sûr un niveau de complexité au moins équivalent au problème de Bin Packing classique, une difficulté supplémentaire étant induite par la non linéarité des contraintes entre objets.

2.2.3 Méthodes de résolution

2.2.3.1 Résolutions du *Bin packing*

A partir de cette description des éléments du problème du *Bin packing*, plusieurs modèles sont utilisés pour du *Bin packing* à 2 dimensions.

Le premier a été proposé par Gilmore et Gomory [Gilmore et Gomory, 1963] : il est une extension de l'approche proposé pour la résolution du problème de *Bin packing* à une dimension [Gilmore et Gomory, 1961]. Il est basé sur une description de tous les sous ensembles qui peuvent être rangés dans un seul bin. Chaque sous ensemble est représenté par un vecteur comprenant n (nombre d'objets) valeurs booléennes représentant si l'objet appartient au sous ensemble ou non. Cette modélisation représente donc toutes les solutions de rangement pour un bin puis utilise une contrainte sur le rangement d'un objet : un objet ne peut être rangé que dans un seul bin.

Le deuxième modèle utilise la théorie des graphes sur un problème de décision [Fekete et Schepers, 2004] cherchant à placer un ensemble d'objets dans un seul bin. En effet, deux graphes de taille n sont utilisés, chacun représentant une dimension. Dans chacun de ses graphes un sommet représente un objet rangé dans le bin et il existe un lien entre deux sommets dans un graphe (par exemple sur la longueur) si les projections des deux objets associés se superposent sur la dimension du graphe (la longueur), de même pour

l'autre graphe (par exemple la largeur). L'intérêt de ce modèle par rapport au précédent est qu'il ne tient pas compte des configurations symétriques.

D'autres modèles existent pour le *Bin packing* mais sont utilisés dans des cas précis, par exemple un modèle sur un problème de découpe modélisé par un modèle de programmation linéaire a été proposé dans [Beasley, 1985].

Comme pour le problème du sac à dos, le problème de *Bin Packing* est NP complet [Garey et Johnson, 1979] et les techniques utilisées sont principalement basées sur la programmation linéaire en nombres entiers ou sur des méta-heuristiques et fournissent des solutions approchées ou exactes.

Les méta-heuristiques utilisées sont de plus confrontées à une difficulté liée à la définition des populations initiales sur lesquelles travaillent les algorithmes. En effet, ces derniers manipulent une (ou plusieurs) solution(s) fournie(s) comme données d'entrée pour progressivement converger vers des solutions améliorant le critère d'optimisation retenu ; ces solutions "initiales" doivent bien entendu être admissibles vis-à-vis de l'ensemble des contraintes, leur production (et la preuve du respect des contraintes) reste une tâche non triviale basée sur l'expertise.

2.2.3.2 Programmation mathématique

Dans de tels problèmes, la programmation mathématique est souvent utilisée afin de répondre à des problèmes nécessitant la programmation linéaire, la programmation en nombres entiers, l'analyse des réseaux ou encore la programmation non linéaire. Le but de la programmation mathématique est l'allocation optimale de ressources suivant plusieurs contraintes qui peuvent être d'ordre technologique, économique, pratique, ... L'approche de programmation linéaire exige que chaque contrainte soit linéaire ou puisse l'être, et d'utiliser un solveur approprié pour ce type de modèle.

La programmation mathématique est utilisée dans plusieurs domaines afin de résoudre des problèmes différents comme :

- l'aménagement d'une usine ;
- la gestion des stocks ;
- l'affectation de personnel ;
- ...

Cette programmation a pour but d'optimiser divers paramètres, comme les coûts, les quantités de matériels ou d'employés, les distances, ...

Pour résoudre ces problèmes, une modélisation est nécessaire. Pour cela, il faut définir l'ensemble des variables ainsi que leur domaine de définition, la ou les fonctions à optimiser et les contraintes nécessaires au problème posé. Une fois la modélisation faite, nous devons définir la fonction objectif, qui est recherchée.

Parmi les modélisations, la programmation linéaire est souvent utilisée afin de résoudre les différents problèmes rencontrés. La méthode du simplexe [Dantzig, 1990] permet de résoudre de tel système en deux phases qui sont proches de notre recherche :

- la première phase consiste à trouver une solution de base faisable (ou montrer l'existence d'aucune solution)

- la deuxième phase est la progression en regardant le voisinage direct et en passant à un sommet voisin pour améliorer la solution.

En revanche, cette méthode du simplexe ne peut pas être utilisée puisque celle-ci permet de traiter uniquement des cas de programmation linéaire à variables continues, ce qui n'est pas notre cas. En effet, le problème d'allocation de fonctions sous contraintes nécessite un modèle de programmation à variables entières, voire binaires.

Il existe aussi les algorithmes génétiques [Goldberg, 1989] qui sont une analogie avec la génétique et qui utilisent les approches utilisées dans ce domaine comme la mutation, le croisement et la sélection. Le principe est de sélectionner les meilleures solutions trouvées à une étape et de les faire évoluer par croisement des solutions sélectionnées avec possibilité de mutations. En revanche, cette méthode a souvent des temps de calculs importants, n'est pas évidente a mettre en œuvre et les extréma locaux posent problème.

La programmation en nombres entiers (Integer Linear Programing : ILP) est utilisée quand le résultat attendu doit être un entier, comme dans notre cas. Cette programmation diffère de la programmation à variables réelles par l'ajout des contraintes imposant aux variables d'être entières. Il existe plusieurs programmations en nombres entiers :

- Pure Integer Linear Programming où l'ensemble des variables sont entières ; ce genre de problèmes se résout par des méthodes de résolutions arborescentes comme branch & bound [Lawler et Wood, 1966] ;

- Mixed Integer Linear Programming où les variables sont définies soient comme des entiers soient comme des réelles. Ce genre de problèmes se résout par des méthodes de résolutions arborescentes comme branch & bound [Lawler et Wood, 1966] ou l'algorithme de Gomory [Gomory, 1960].

Il existe aussi des modèles de programmation à variables binaires, cette programmation va être utilisée dans la suite du problème d'allocation de fonctions sous contraintes pour toutes les variables binaires et les contraintes associées. En effet, certaines contraintes d'allocation empêchent les fonctions d'être allouées plusieurs fois. Comme elles ne sont allouées qu'une seule fois, la variable le représentant sera une variables binaire.

Ces techniques peuvent donc contribuer à la résolution de notre problème d'allocation et seront donc exploitées dans la deuxième partie de ce chapitre.

2.2.3.3 Travaux antérieurs effectués au sein d'EDF

Un travail a été réalisé dans le cadre du N4 par EDF R&D sur un projet baptisé THOR. Ce travail intervient à un stade de la conception du contrôle-commande où les traitements sont déjà répartis en groupes d'armoires à l'intérieur desquels on considère qu'il n'y a pas de problème de saturation de réseau. L'optimisation est réalisée uniquement sur un critère de répartition, c'est-à-dire pour un groupe d'armoires donné, sur la minimisation du coût, ou en pratique, sur la minimisation du nombre d'automates. Par ailleurs, la répartition doit satisfaire un ensemble de contraintes d'ordre matériel (comme par exemple : la place dans les armoires, la charge des processeurs) et fonctionnel (pour satisfaire la sûreté de fonctionnement).

La résolution d'un dimensionnement revient à un placement de tâches sur un ensemble de processeurs, sujet de nombreuses recherches théoriques. Pour la plupart ils se basent en particulier sur un nombre de processeurs fixé à l'avance alors que ce nombre est inconnu lors des activités de dimensionnement. Il a donc été choisi d'utiliser la technologie de Programmation Par Contraintes qui est, par nature, adaptée aux problèmes de forte combinatoire tels que le dimensionnement. Elle offre de plus une programmation souple dans les cas où on ne connaît pas d'algorithme déterministe. THOR utilise une bibliothèque de programmation par contraintes orientées objet basée sur le langage C++ et SOLVER de la société ILOG. Le prototype THOR comporte deux parties, une partie algorithmique et une partie interface homme-machine animée permettant de suivre

au fur et à mesure les essais de l'algorithme, ce qui a permis d'en mettre en évidence certaines imperfections.

Le prototype THOR a donc montré la faisabilité d'un outil industriel de dimensionnement automatique. Ce projet a été basé sur de l'allocation et non sur de la conception d'architecture opérationnelle. Ce problème est donc aussi compliqué et sa résolution est typée aux problèmes du N4 ce qui nous empêche de reprendre les mêmes outils/méthodes, nous devons donc regarder quelles sont les résolutions possibles pour des problèmes d'allocation de fonctions sous contraintes.

2.3 Allocation de fonctions de contrôle-commande par une approche de programmation linéaire en nombres entiers

Une modélisation du problème d'allocation a été faite. Cette modélisation utilise les mêmes notations que celles présentées dans la partie 1.3 et qui seront rappelées ci-dessous, ainsi qu'une modélisation sous la forme de la programmation linéaire en nombres entiers (noté ILP pour : Integer Linear Programming).

2.3.1 Notations utilisées

Les notations utilisées pour la programmation linéaire en nombres entiers sont les mêmes que celles définies dans la description du problème, c'est-à-dire :

L'approche par programmation linéaire en nombres entiers a besoin, en plus des notations décrites précédemment, de variables pour résoudre le problème afin de définir l'objectif de l'étude, c'est-à-dire l'obtention d'une solution au problème d'allocation. Il y a deux ensembles de variables à ajouter :

- $z : F \times C \rightarrow \mathbb{B} = \{0;1\}$ représente l'état de la fonction f^i : allouée ou non au contrôleur c^j, la variable $z_{i,j} = z(f^i, c^j) \in \mathbb{N}$ représente les éléments de la matrice Z :

$$z_{i,j} = \begin{cases} 1 & \text{si la fonction } f^i \text{ est allouée au contrôleur } c^j \\ 0 & \text{sinon} \end{cases}$$

- $U_j \in \mathbb{N}$ représente l'état du contrôleur c^j : utilisé ou non.

Tableau 2.1 – Notations utilisées

$L \in \mathbb{N}$	nombre de fonctions
$i \in L$	indice de la fonction f^i
FC^i	facteur de criticité de la fonction f^i
NI_a^i	nombre de données d'entrées analogiques de la fonction f^i
NI_l^i	nombre de données d'entrées logiques de la fonction f^i
NO_a^i	nombre de données de sorties analogiques de la fonction f^i
NO_l^i	nombre de données de sorties logiques de la fonction f^i
$M \in \mathbb{N}$	nombre de contrôleurs
$j \in M$	indice de contrôleur c^j
NS^j	niveau de sécurité du contrôleur c^j
AI_{max}^j	nombre maximum d'entrées analogiques du contrôleur c^j
LI_{max}^j	nombre maximum d'entrées logiques du contrôleur c^j
AO_{max}^j	nombre maximum de sorties analogiques du contrôleur c^j
LO_{max}^j	nombre maximum de sorties logiques du contrôleur c^j

$$U_j = \begin{cases} 1 & \text{si le contrôleur } c^j \text{ est utilisé} \\ 0 & \text{sinon} \end{cases}$$

2.3.2 Réécriture de l'ensemble des six contraintes

Pour mettre en place le modèle de programmation linéaire en nombres entiers, il faut réécrire les contraintes spécifiées dans le chapitre précédent. Avec ces deux ensembles de variables, les équations utiles pour la méthode ILP :

- permettent de respecter les équations de capacité 1.1 en termes d'entrées logiques 1.2, d'entrées analogiques 1.3, de sorties logiques 1.4 et de sorties analogiques 1.5. $\forall j \in \{1, ..., M\}$

$$\sum_{i=1}^{L} NI_l^i . z_{i,j} \leq LI_{max}^j \tag{2.1}$$

$$\sum_{i=1}^{L} NI_a^i . z_{i,j} \leq AI_{max}^j \tag{2.2}$$

$$\sum_{i=1}^{L} NO_l^i . z_{i,j} \leq LO_{max}^j \tag{2.3}$$

$$\sum_{i=1}^{L} NO_a^i . z_{i,j} \quad \leq \quad AO_{max}^j \tag{2.4}$$

- permettent de respecter la compatibilité entre le facteur de criticité de la fonction et le niveau de sécurité du contrôleur par l'équation 1.6 représenté par la matrice X.

$$X = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

  Cette contrainte doit être linéarisée pour être prise en compte lors de la résolution, car on ne peut pas mettre de condition dans une contrainte de programmation linéaire, tel que :
  Si $FC^i = 2$ alors $NS^j = 1$ ou 2.
  Nous utilisons donc un pré-traitement qui nécessite de connaitre le niveau de sécurité de tous les contrôleurs pour la mettre en place. Il faut donc que le niveau de sécurité du contrôleur soit déjà affecté. Dans la suite des études, les niveaux de sécurité des contrôleurs seront donc prédéterminés et répartis de façon homogène entre les différentes valeurs :

  - 1/3 du nombre de contrôleurs auront un niveau de sécurité $NS = 1$;
  - 1/3 du nombre de contrôleurs auront un niveau de sécurité $NS = 2$;
  - 1/3 du nombre de contrôleurs auront un niveau de sécurité $NS = 3$.

  Lors de cette répartition, il sera possible d'avoir un écart de un contrôleur entre chaque catégorie afin d'avoir des nombres entiers, par exemple pour 200 contrôleurs, la répartition sera :

  - 67 contrôleurs auront un niveau de sécurité $NS = 1$;
  - 67 contrôleurs auront un niveau de sécurité $NS = 2$;
  - 66 contrôleurs auront un niveau de sécurité $NS = 3$.

  La contrainte représentant la compatibilité se résume donc à :
  $\forall i \in \{1, ..., L\}, \forall j \in \{1, ..., M\}$ tel que $x_{ij} = 0$

$$z_{i,j} = 0 \tag{2.5}$$

En effet, le pré-traitement met uniquement les valeur de $z_{i,j}$ à zéro quand la relation entre le facteur de criticité d'une fonction donnée f^i et un contrôleur donné c^j est à zéro dans la matrice X, ce qui revient à $x_{ij} = 0$.

- permettent de respecter la contrainte de sûreté d'exclusion 1.8 :
  $\forall (i,k) \in \{1,...,L\}^2, \forall j \in \{1,...,M\}$ tel que $y(f^i, f^k) = 1$

$$z_{i,j} + z_{k,j} \leq 1 \tag{2.6}$$

Cette contrainte s'écrit 2.6 pour chaque contrôleur c^j, comme $z_{i,j}$ représente l'allocation de f^i à c^j, on impose qu'il soit impossible d'avoir les deux dans un même contrôleur. Cette contrainte est donc exprimée par la somme $z_{i,j} + z_{k,j}$ qui doit être inférieure ou égale à 1, en effet elle ne peut prendre que la valeur 0 ou 1 d'après cette contrainte alors que sans celle-ci, cette valeur pouvait être égale à 2 si les deux fonctions étaient allouées dans ce même contrôleur.

2.3.3 Ajouts de contraintes

Le problème d'allocation de fonctions sous contraintes nécessite de mettre en place d'autres contraintes pour la programmation linéaire en nombres entiers qui :

- permettent de respecter l'hypothèse que les fonctions ne peuvent être allouées qu'à un seul contrôleur : $\forall i \in \{1,...,L\}$

$$\sum_{j=1}^{M} z_{i,j} = 1 \tag{2.7}$$

- imposent qu'un contrôleur soit dit utilisé si au moins une fonction est allouée à ce contrôleur. Cette contrainte lie deux ensembles de variables :
  $\forall i \in \{1,...,L\}, \forall j \in \{1,...,M\}$

$$z_{i,j} \leq U_j \tag{2.8}$$

Les variables $z_{i,j}$ peuvent être considérées comme des entiers puisque que la somme de ces variables est au maximum égale à 1 d'après l'équation 2.6. Comme ces variables ne peuvent prendre que les valeurs 0 ou 1, cela signifie que parmi ces variables (considérées en somme) l'une d'elles prend la valeur entière 1 tandis que les autres sont mises à zéro.

Les U_j peuvent aussi être considérées comme des nombres entiers. En effet, un contrôleur est utilisé si au moins une fonction lui est allouée. L'équation 2.8 veille à ce que ces variables soient minorées par 0 si le contrôleur correspondant n'est pas utilisé, c'est-à-dire sans aucune fonction allouée, ou 1 si au moins une fonction est allouée. De plus, l'objectif est de minimiser la somme de ces variables U_j, une variable U_j reçoit donc la valeur maximum des bornes inférieures des contraintes $z_{i,j} \in U_j$ concernant un contrôleur c^j, c'est-à-dire 0 ou 1. Par conséquent, ces variables peuvent être déclarées comme entières mais n'auront comme valeurs possibles que les valeurs 0 ou 1.

2.3.4 Description de la fonction objectif

L'objectif de l'étude et de trouver une solution faisable au problème d'allocation puis de trouver une solution optimisant le nombre de contrôleurs. Pour cela, la fonction *objectif* est exprimée :

$$\text{Min } Z = \sum_{j=1}^{M} U_j \tag{2.9}$$

Une solution obtenue grâce au modèle mathématique de la programmation linéaire en nombres entiers, décrit par les équations 2.7 à 2.8 et la fonction objectif 2.9, est une allocation possible, donnée par la valeur des variables de $z_{i,j}$, de l'ensemble des fonctions L sur un sous-ensemble de M contrôleurs. La solution optimale est la solution ayant la meilleure valeur de Z, soit le nombre minimal de contrôleurs utilisés.

2.3.5 Cas d'étude

Dans cette partie, deux types de cas d'étude sont traités : un sur le cas jouet présenté dans la fin du chapitre précédent et l'autre sur des cas d'étude de 200 fonctions plus proche du cadre réél de l'étude. Le cas jouet permettra de voir l'ensemble des contraintes sur un cas précis et demandant peu de variables vu le nombre de fonctions et de contrôleurs. Pour pouvoir résoudre ce problème un outil de programmation linéaire en nombres entiers (CPLEX) est utilisé.

2.3.5.1 Cas d'étude de 5 fonctions

Dans l'exemple présenté, à la fin du chapitre précédent, un ensemble de 5 fonctions doit être alloué à 3 contrôleurs. Dans ce cas d'étude, le niveau de sécurité du contrôleur c^1 est fixé à 2 et les deux autres c^2, c^3 ont un niveau de sécurité de 1. Cette répartition est obligatoire dans ce cas jouet car l'inverse (2 contrôleurs avec $NS = 2$ et 1 contrôleur avec $NS = 1$) conduit à ne trouver aucune solution et donc à une impossibilité d'allocation. En effet, avec cette affectation de NS, les fonctions ayant un facteur de criticité de 1 ne peuvent pas toutes être allouées dans un seul contrôleur.

Le problème revient donc d'allouer 5 fonctions à 3 contrôleurs comme le montre la figure 2.1 en respectant les équations 1.9, 1.10, 1.11, 1.12, 1.13 avec la matrice X présentée dans la section 1.4.5.

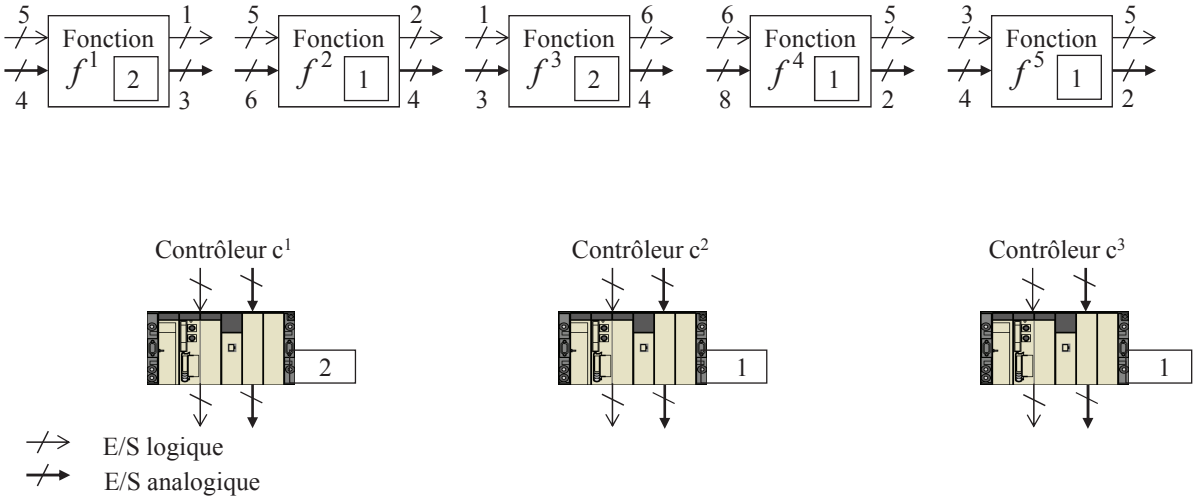


Figure 2.1 – Exemple avec cinq fonctions et trois contrôleurs (NS fixé)

2.3.5.1.1 Description de toutes les variables de cet exemple jouet

Les variables utiles dans cet exemple sont :

$\forall i \in \{1, ..., 5\}, \forall j \in \{1, 2, 3\}$

- les variables représentant l'état des contrôleurs : U_1, U_2, U_3 ;
- les variables représentant l'état de la fonction f^i ; allouée ou non au contrôleur c^j :

  $z_{1,1}$, $z_{1,2}$, $z_{1,3}$

  $z_{2,1}$, $z_{2,2}$, $z_{2,3}$

  $z_{3,1}$, $z_{3,2}$, $z_{3,3}$

  $z_{4,1}$, $z_{4,2}$, $z_{4,3}$

  $z_{5,1}$, $z_{5,2}$, $z_{5,3}$

Une fois les variables définies, il faut écrire les contraintes.

2.3.5.1.2 Description de toutes les contraintes de cet exemple jouet

Voici l'ensemble des contraintes pour cet exemple qui :

- permettent de respecter les équations de capacité en termes d'entrées logiques 1.9, et analogiques 1.10, de sorties logiques 1.11 et analogiques 1.12 pour chaque contrôleur :
  - pour c^1 :

    $5z_{1,1} + 5z_{2,1} + 1z_{3,1} + 6z_{4,1} + 3z_{5,1} \leq 10$

    $4z_{1,1} + 6z_{2,1} + 3z_{3,1} + 8z_{4,1} + 4z_{5,1} \leq 10$

    $1z_{1,1} + 2z_{2,1} + 6z_{3,1} + 5z_{4,1} + 5z_{5,1} \leq 10$

    $3z_{1,1} + 4z_{2,1} + 4z_{3,1} + 2z_{4,1} + 2z_{5,1} \leq 10$
  - pour c^2 :

    $5z_{1,2} + 5z_{2,2} + 1z_{3,2} + 6z_{4,2} + 3z_{5,2} \leq 10$

    $4z_{1,2} + 6z_{2,2} + 3z_{3,2} + 8z_{4,2} + 4z_{5,2} \leq 10$

    $1z_{1,2} + 2z_{2,2} + 6z_{3,2} + 5z_{4,2} + 5z_{5,2} \leq 10$

    $3z_{1,2} + 4z_{2,2} + 4z_{3,2} + 2z_{4,2} + 2z_{5,2} \leq 10$
  - pour c^3 :

    $5z_{1,3} + 5z_{2,3} + 1z_{3,3} + 6z_{4,3} + 3z_{5,3} \leq 10$

    $4z_{1,3} + 6z_{2,3} + 3z_{3,3} + 8z_{4,3} + 4z_{5,3} \leq 10$

$$1z_{1,3} + 2z_{2,3} + 6z_{3,3} + 5z_{4,3} + 5z_{5,3} \leq 10$$
$$3z_{1,3} + 4z_{2,3} + 4z_{3,3} + 2z_{4,3} + 2z_{5,3} \leq 10$$

- permettent de respecter la compatibilité entre le facteur de criticité de la fonction et le niveau de sécurité du contrôleur par l'équation 1.13. Comme nous connaissons les niveaux de sécurité des contrôleurs ($NS^1 = 2$, $NS^2 = 1$, $NS^3 = 1$), les contraintes sont alors :

  $z_{2,1} = 0$; $z_{4,1} = 0$; $z_{5,1} = 0$

  $z_{1,2} = 0$; $z_{3,2} = 0$

  $z_{1,3} = 0$; $z_{3,3} = 0$

- permettent de respecter l'hypothèse que les fonctions ne peuvent être allouées qu'à un seul contrôleur :

  - pour f^1 :

    $z_{1,1} + z_{1,2} + z_{1,3} = 1$

  - pour f^2 :

    $z_{2,1} + z_{2,2} + z_{2,3} = 1$

  - pour f^3 :

    $z_{3,1} + z_{3,2} + z_{3,3} = 1$

  - pour f^4 :

    $z_{4,1} + z_{4,2} + z_{4,3} = 1$

  - pour f^5 :

    $z_{5,1} + z_{5,2} + z_{5,3} = 1$

- imposent qu'un contrôleur soit dit utilisé si au moins une fonction est allouée à ce contrôleur :

  $z_{1,1} \leq U_1$; $z_{1,2} \leq U_2$; $z_{1,3} \leq U_3$

  $z_{2,1} \leq U_1$; $z_{2,2} \leq U_2$; $z_{2,3} \leq U_3$

  $z_{3,1} \leq U_1$; $z_{3,2} \leq U_2$; $z_{3,3} \leq U_3$

  $z_{4,1} \leq U_1$; $z_{4,2} \leq U_2$; $z_{4,3} \leq U_3$

  $z_{5,1} \leq U_1$; $z_{5,2} \leq U_2$; $z_{5,3} \leq U_3$

2.3.5.1.3 Description de la fonction "objectif" de cet exemple jouet

La fonction objectif est reprise de l'équation 2.9 dans le cas de cet exemple, donc :

$$\text{Min } Z \quad = \quad \sum_{j=1}^{3} U_j = U_1 + U_2 + U_3 \tag{2.10}$$

2.3.5.1.4 Solution du cas jouet

La solution faisable et optimale trouvée d'allocation de ces 5 fonctions sur les 3 contrôleurs lors de l'utilisation de l'outil CPLEX est donnée à la figure 2.2. En effet, en respectant l'ensemble des contraintes, la fonction f^4 ne peut être allouée avec aucune autre fonction. Les fonctions f^1 et f^3 ne sont pas du même classement que la fonction f^4 et ne peuvent donc pas être dans le même contrôleur afin de respecter l'équation 1.13, elles sont allouées au contrôleur c^1 qui a un niveau de sécurité compatible avec leurs facteurs de criticité. Les fonctions f^2 et f^5 ne peuvent pas être allouées avec f^4 même si leurs facteurs sont compatibles car :

- pour la fonction f^5, la contrainte 1.10 ne serait pas respectée ;
- pour la fonction f^2, les contraintes 1.9 et 1.10 ne seraient pas respectées ;

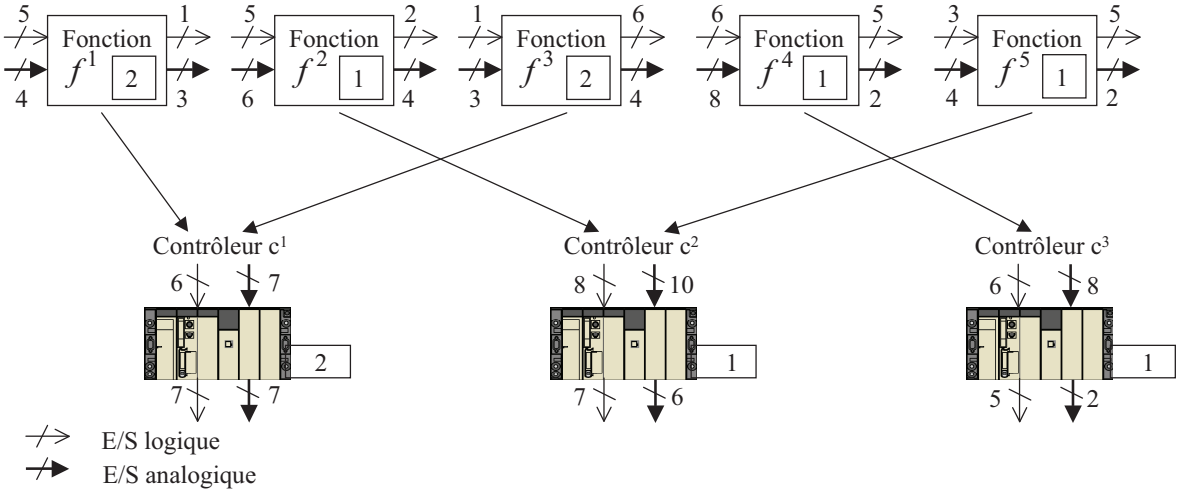


Figure 2.2 – Une solution d'allocation optimale de cinq fonctions sur trois contrôleurs par MILP

2.3.5.2 Cas d'étude de 200 fonctions

Dans le cadre industriel, un ensemble de 200 fonctions doit pouvoir être traité. Cet ensemble de 200 fonctions doit être alloué dans un ensemble de contrôleurs que nous prendrons de 200 contrôleurs avec une répartition comme suit :

- 67 contrôleurs avec un niveau de sécurité de 1 $NS = 1$;
- 67 contrôleurs avec un niveau de sécurité de 2 $NS = 2$;
- 66 contrôleurs avec un niveau de sécurité de 3 $NS = 3$.

L'allocation est faite pour respecter l'ensemble des contraintes définies dans la section 1.4. Pour la modélisation mathématique des trois exemples qui suivent, toutes les valeurs d'entrées/sorties logiques/analogiques ont été générées automatiquement et sont comprises entre 0 et 10, d'où :
$\forall i \in \{1, 2, 3, 4, 5\}$,
NI_a^i, NI_l^i, NO_a^i, $NO_l^i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^4$.

Chaque exemple est entré dans le logiciel avec l'ensemble de ses variables, contraintes et sa fonction objectif. La recherche de la première solution faisable est effectuée en utilisant une option du logiciel CPLEX et permet d'avoir les résultats sur les 3 exemples de 200 fonctions. La recherche du nombre de contrôleur optimal est aussi effectuée. Pour le premier cas d'étude, le nombre de contrôleur de la première solution est de 83 et le temps d'obtention de cette solution est de 71,37 secondes (comme le montre le tableau 2.2). Pour ce cas d'étude, les données des fonctions sont présentées en annexe pour les 200 fonctions. Pour la solution optimale, nous avons limité l'étude à un temps de calcul de 1200 secondes (20min) et la meilleure solution que l'on trouve est de 40 contrôleurs utiles, mais ce n'est pas la solution optimale.

Les résultats des deux autres cas d'étude sont présentés dans le tableau 2.2

2.3.6 Discussion

Le modèle mathématique de programmation linéaire en nombres entiers permet de résoudre le problème d'allocation de fonctions sous contraintes même si quelques points restent problématiques après cette résolution :

- au niveau des résultats :

Tableau 2.2 – Résultats des 3 cas d'études de 200 fonctions

Cas d'étude de 200 fonctions n°	1	2	3
Nombre de contrôleurs de la première solution faisable	83	121	87
Temps d'obtention de la première solution faisable (secondes)	71,37	15,5	219,8
Nombre de contrôleurs de la meilleure solution	40 (NO)	32 (O)	39 (NO)
Temps d'obtention de la meilleure solution (secondes)	1200	852	1200

NO : Non Optimale, atteignant la limite de 1200 secondes fixée avant la recherche
O : Optimale

  - Un nombre de pas de calcul très important entre la première solution faisable et la meilleure solution. Un écart très important du nombre de contrôleurs est trouvé entre ces deux solutions ;
  - La sensibilité des populations initiales admissibles qui impose de fixer les niveaux de sécurité des contrôleurs. La définition initiale est un problème connu des méta-heuristiques ;
  - le temps pour obtenir la première solution faisable est long.

- au niveau de la modélisation :
  - les ajouts d'autres contraintes, nécessiteront une étape préliminaire de linéarisation, ce qui n'est pas toujours possible ;
  - la difficulté de modifier les contraintes ou données pour permettre au concepteur de pouvoir manipuler les outils facilement. En effet, les fichiers d'entrée ".lp" pour un cas de 200 fonctions font plus de 100 000 lignes, ce qui est difficile à lire et à modifier ;

Pour toutes ces raisons, nous nous sommes intéressés à une autre solution de résolution de ce problème de satisfaction de contraintes, qui a été présenté dans [Behrmann *et al.*, 2005] et [Subbiah et Engell, 2010] afin de résoudre des problèmes d'ordonnancement.

2.4 Recherche d'atteignabilité

Certains auteurs ont proposé d'utiliser des méthodes basées sur la théorie des systèmes à événements discrets (SED) pour résoudre les problèmes qui peuvent être décrits

comme des problèmes de satisfaction de contraintes. L'analyse d'atteignabilité sur un réseau d'automates a été proposée dans [Behrmann *et al.*, 2005] et [Subbiah et Engell, 2010] pour résoudre les problèmes d'ordonnancement. [Kobetski et Fabian, 2009] Kobetski propose de combiner des techniques de contrôle de surveillance et de programmation linéaire en nombres entiers mixte (Mixed Integer Linear Programming) afin d'optimiser la coordination des éléments de systèmes de fabrication.

En effet, Subbiah [Subbiah et Engell, 2010] utilise les automates temporisés pour modéliser et résoudre un problème d'ordonnancement de procédure de modifications. Ils proposent de modéliser le problème d'ordonnancement par les automates temporisés et de résoudre l'optimisation de ce problème en utilisant la recherche d'atteignabilité. Deux types de modèles d'automates sont proposés : l'un représentant les gammes opératoires (un automate par gamme) et l'autre représentant les machines (un automate pour chaque machine) qui sont définies par deux états (libre et occupée). Les gammes opératoires contiennent la séquence des opérations pour obtenir le produit final ainsi que les machines exécutant ces opérations. La recherche d'atteignabilité permet ensuite d'atteindre des états préalablement définis dans ces modèles, qui permettent de satisfaire toutes les demandes et d'atteindre un coût minimal. La trace donnée par la suite donne l'ordonnancement recherché.
Deux cas d'études sont présentés dans ce papier et une comparaison avec des modèles de programmation linéaire en nombres entiers mixte (mixed integer linear programming : MILP) est présentée sur le premier cas d'étude.
Les conclusions de leur comparaison sont que l'approche basée sur les automates temporisés et la recherche d'atteignabilité ont des résultats similaires voire meilleurs que l'approche MILP.

Ce ne sont pas les seuls à avoir utilisé les systèmes à événements discrets, Behrmann [Behrmann *et al.*, 2005] les a aussi utilisé dans le cadre du projet AMETIST dont le but est de développer une méthode de modélisation sur des outils de résolution de problèmes pour des systèmes temps réels distribués. Ils utilisent les automates temporisés pour représenter un problème d'ordonnancement sur de la production de laque. De même que précédemment, chaque gamme opératoire est modélisée par un automate et chaque machine est aussi modélisée par un automate. La composition parallèle de l'ensemble de ces automates construit le modèle du système. Cette construction assure

d'avoir l'ensemble des possibilités d'ordonnancement. Ils utilisent par la suite la recherche d'attignabilité pour trouver l'ordonnancement qu'ils recherchent et qui vérifie que la production demandée soit effectuée avant une date donnée.

Marangé [Marangé *et al.*, 2011] utilise aussi les automates temporisés pour faire de l'ordonnancement. Sa modélisation diffère des deux précédentes car ils utilisent des gammes logiques et non des gammes opératoires. En effet, les gammes opératoires nécessitent de connaitre sur quelles machines le produit doit passer. Dans leur cas où l'objectif est de faire de la reconfiguration dynamique, les machines qui effectueront les opérations ne sont pas connues. Dans ce travail, chaque machine (ressource) est modélisée par un automate, ainsi que toutes les gammes logiques. Tous ces modèles négocient grâce à la synchronisation entre automates communicants. Cette technique permet de ne définir aucune stratégie d'allocation des opérations sur les ressources (machines) et permet au model-checker de parcourir l'espace d'état pour trouver un état de cet espace où les gammes logiques sont toutes réparties sur les machines. Ceci donne une allocation possible au problème d'ordonnancement.
Dans leur cas, les états recherchés sont des état bloquants et non les états marqués comme pour Behrmann [Behrmann *et al.*, 2005] et Subbiah [Subbiah et Engell, 2010]. En effet, les gammes logiques se terminent par des états bloquants et la propriété recherchée pour la recherche d'atteignabilité est :
"il n'existe aucun état bloquant".
Si une allocation est possible, que toutes les gammes logiques sont allouées à une machine et arrivent dans leurs états bloquant, alors la propriété ne sera pas vérifiée et la trace est obtenue par le contre exemple.

Tous les problèmes d'ordonnancement qui viennent d'être présentés sont des problèmes de satisfactions de contraintes, avec comme contraintes la disponibilité des machines, les contraintes de dimension de la machine, ...

Les problèmes traités dans ces articles ne sont pas exactement ceux que nous rencontrons, mais les résultats présentés sont intéressants et sont appliqués à des problèmes de satisfaction de contraintes . Ils sont donc probablement applicables pour le problème d'allocation de fonctions sous contraintes.

Nous allons adopter le principe de l'utilisation des systèmes à événements discrets pour le problème d'allocation de fonctions sous contraintes en utilisant des automates

communicants pour la modélisation et la recherche d'atteignabilité.

2.5 Conclusion

Nous avons montré dans ce chapitre que le problème d'allocation sous contraintes de fonctions de contrôle-commande peut être résolu par une approche de programmation linéaire en nombres entiers. Cette approche permet d'obtenir une solution faisable et une solution minimisant le nombre de contrôleurs. Cependant, la première solution faisable obtenue est très éloignée de la solution minimisant le nombre de contrôleurs et nécessite un temps de calcul assez long.

Nous avons également montré que des travaux récents visant à résoudre des problèmes de satisfaction de contraintes à l'aide de la théorie des systèmes à événements discrets ont permis d'obtenir des résultats intéressants. Sur la base de ces travaux, le chapitre suivant présentera une nouvelle approche pour la résolution du problème d'allocation sous contraintes de fonctions de contrôle-commande.

Chapitre
3
Allocation de fonctions par recherche d'atteignabilité

Sommaire

3.1 Introduction

Ce chapitre présente la contribution théorique majeure de ce travail de thèse : une méthode originale d'allocation d'un ensemble de fonctions, dont les caractéristiques externes et les facteurs de criticité sont connus, à un ensemble de contrôleurs dont les caractéristiques techniques sont également connues, en respectant des contraintes de capacité et de sûreté ([Lemattre *et al.*, 2011a],[Lemattre *et al.*, 2011b]). Les hypothèses retenues sont les suivantes :

1. les fonctions sont insécables, c'est-à-dire. qu'il n'est pas possible de décomposer une fonction quelconque en plusieurs sous-fonctions qui seraient allouées à plusieurs contrôleurs ;

2. les caractéristiques techniques des contrôleurs sont telles qu'il est toujours possible d'allouer une fonction à un contrôleur auquel aucune autre fonction n'a été précédemment allouée ;

3. les niveaux de sécurité des contrôleurs ne sont pas connus *a priori*.

Cette méthode repose sur deux principes :

- le processus d'allocation est modélisé par un ensemble de mécanismes concurrents d'appel/réponse dans un réseau d'automates communicants à variables partagées ;

- une solution d'allocation est obtenue par recherche d'atteignabilité dans ce réseau.

Le formalisme de représentation retenu pour le réseau d'automates est présenté dans la deuxième section de ce chapitre. La troisième section explique comment le processus d'allocation de fonctions peut être décrit par des mécanismes d'appel/réponse entre deux classes d'automates. La quatrième section s'intéresse à la définition de la propriété d'atteignabilité d'un état caractéristique de l'allocation de toutes les fonctions ; il est également montré dans cette section comment la vérification, itérative ou non, de cette propriété sur le réseau d'automates permet d'obtenir une solution faisable si elle existe, un ensemble de solutions faisables ou une solution minimisant le nombre de contrôleurs qui assurent l'allocation de toutes les fonctions. Enfin, la mise en œuvre de cette proposition à l'aide d'un outil de vérification formelle existant est détaillée dans la cinquième section.

3.2 Présentation du formalisme retenu

Le formalisme retenu est celui d'un réseau d'automates non temporisés communiquant par des variables partagées et synchronisées par des étiquettes de transition. Ce formalisme a été retenu car :

- les variables partagées permettent de modéliser les caractéristiques des fonctions et contrôleurs ;
- les étiquettes de synchronisation assurent la cohérence des évolutions entre les automates représentant les demandes d'allocation et ceux représentant les acceptations/refus d'allocation.

3.2.1 Définition d'un automate communicant A

Un automate A est un N-uplet $A = (S, X, L, T, S_m, s_0, v_0)$, où :

- S est un ensemble fini de localités ;
- X est un ensemble fini de variables entières ;
- L est un ensemble d'étiquettes décomposable en 2 ensembles disjoints L_i, L_o, où
  - L_i est l'ensemble des étiquettes de réception ;
  - L_o est l'ensemble des étiquettes d'émission.
- T est un ensemble de transitions $(s, l, g, m, s') \in S \times L \times G \times M \times S$, avec G l'ensemble des gardes (conditions sur les variables de X) et M l'ensemble des mises à jour sur les valuations des variables ;
- $S_m \subseteq S$ est un ensemble de localités marquées ;
- $s_0 \in S$ est la localité initiale ;
- $v_0 : X \leftarrow \mathbb{N}$ est la valuation initiale des variables.

Il convient de souligner que l'état d'un automate à un instant donné est défini par la localité active et les valeurs des variables à cet instant.

Une trace (ou exécution) de A est une succession d'évolutions à partir de l'état initial :
$(s_0, v_0) \xrightarrow{t1} (s_1, v_1) \xrightarrow{t2} (s_2, v_2) ... \xrightarrow{tn} (s_n, v_n)$

3.2.2 Définition d'un réseau d'automates communicants NA

Un réseau d'automates $NA = A^1 \| A^2 \| ... \| A^n$ est défini par $NA = (S, X, L, T, S_m, s_0, v_0)$, avec :

- $S \subseteq S^1 \times S^2 \times ... \times S^n$
- $X = X^1 \cup X^2 \cup ... \cup X^n$
- $L = L^1 \cup L^2 \cup ... \cup L^n$
- $T \subseteq S \times L \times G \times M \times S'$
- $S_m = S_m^1 \times S_m^2 \times ... \times S_m^n$
- $s_0 = s_0^1 \times s_0^2 \times ... \times s_0^n$
- $v_0 : X \leftarrow \mathbb{N}$

De manière similaire, l'état d'un réseau d'automates communicants à un instant donné est défini par l'ensemble des localités actives $s \in S$ et les valeurs des variables v à cet instant.

Une évolution du réseau d'automates NA $(s, v) \rightarrow (s', v')$ est possible si :

- une évolution se produit dans un seul des automates par franchissement d'une transition t dont la localité source ($s^i \in S$) est active, la garde satisfaite et qui ne porte pas d'étiquette de synchronisation :
  $(s^1 \dots s^i \dots s^n, v^1 \dots v^j \dots v^m) \xrightarrow{t} (s^1 \dots s'^i \dots s^n, v'^1 \dots v'^j \dots v'^m)$

- deux transitions t_k^γ, t_m^β d'un couple d'automates (A^γ, A^β) avec t_k^γ contenant l'étiquette $l_k^\gamma \in L^\gamma$ et t_m^β contenant l'étiquette $l_m^\beta \in L^\beta$ telles que $l_k^\gamma = l_m^\beta$ sont franchies simultanément, les états sources de ces transitions étant actifs et leurs gardes étant satisfaites :
  $(s^1 \dots s^i \dots s^l \dots s^n, v^1 \dots v^m) \xrightarrow{(t_k^\gamma, t_m^\beta)} (s^1 \dots s'^i \dots s'^l \dots s^n, v'^1 \dots v'^m)$
  avec : $(t_k^\gamma, t_m^\beta) \in T^2 \mid (s.t_k^\gamma = s^i, s.t_m^\beta = s^l) \wedge (l.t_k^\gamma = l.t_m^\beta)$ la notation $a.B$ désignant l'élément a du n-uplet B.

Il importe de souligner qu'aucune de ces deux sémantiques n'est prioritaire sur l'autre. Les évolutions d'un réseau NA sont donc intrinsèquement non déterministes.

Les conventions suivantes sont utilisées dans les modèles graphiques :

- les localités initiales sont désignées par un arc source comme le montre la figure 3.1 ;

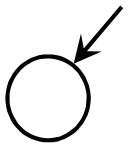


Figure 3.1 – Localité initiale

- les localités marquées sont représentées par deux cercles concentriques comme le montre la figure 3.2 ;

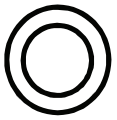


Figure 3.2 – Localité marquée

- les localités actives seront visualisées comme sur la figure 3.3 ;

Figure 3.3 – Exemples de localités actives

- **les noms des localités sont en gras** ;
- *les noms des étiquettes sont en italique* et suivis d'un ! (resp. ?) pour les étiquettes d'émission (de réception) : *Sync* ! ou *Sync* ? pour l'étiquette Sync ;
- [les noms des gardes sont entre crochets] ;
- les mises à jour de variables sont soulignées.

La figure 3.4 présente un exemple de réseau. Dans l'état initial, les automates sont dans les localités A1, B1, C1. De cet état si $x < 2$, il y a une seule évolution possible, mais si $x \geq 2$, deux franchissements exclusifs sont possibles (cf. figure 3.5) :

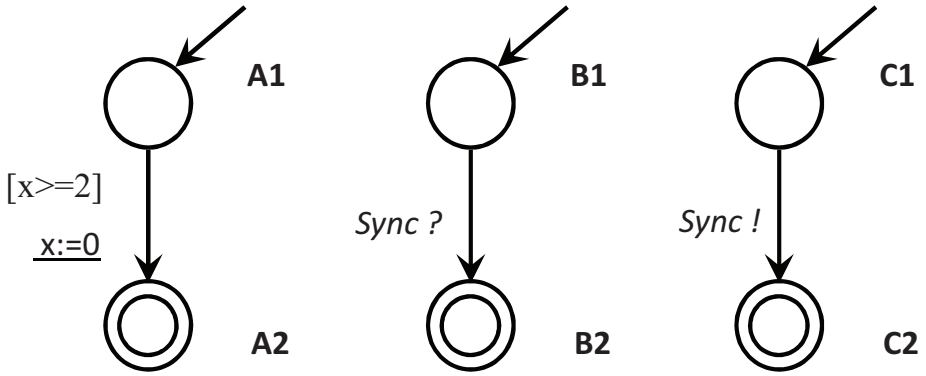


Figure 3.4 – Un exemple de réseau d'automates

- le franchissement de la transition de A1 vers A2, il y a alors une mise à zéro de x ;

- le franchissement simultané des transitions ayant comme localité source et localité destination (B1,B2) et (C1,C2) grâce à l'étiquette « Sync » qui est émise lors du franchissement de la transition de C1 vers C2 et reçu de B1 vers B2.

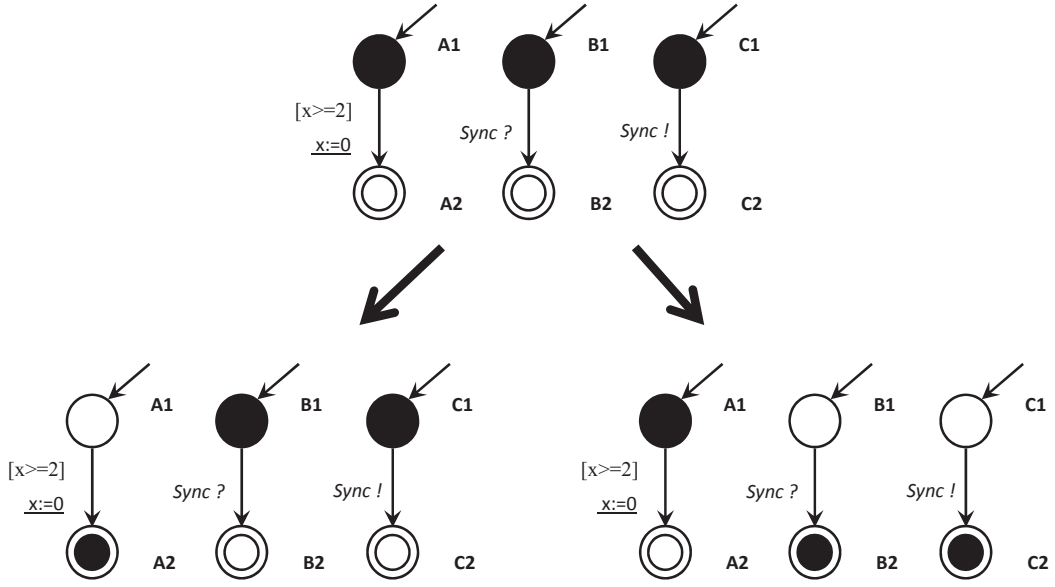


Figure 3.5 – Évolution de l'exemple de réseau d'automates

Comme indiqué à la section 3.2.2, ces évolutions sont non déterministes.

3.3 Modélisation du problème d'allocation de fonctions

3.3.1 Modèles génériques

Afin de modéliser le problème d'allocation sous la forme d'un ensemble de mécanismes concurrents d'appel/réponse entre des modèles sous forme d'automates communicants, deux classes d'automates génériques ont été définies :

- la classe des demandes d'allocation d'une fonction ;
- la classe des acceptations de demande par un contrôleur.

Chaque classe représente un automate générique dont les instances peuvent communiquer avec les instances de l'autre classe par appel/réponse.

Les figures 3.6 et 3.7 présentent les modèles génériques de demande d'allocation d'une fonction et d'acceptation/refus d'une demande par un contrôleur, notés respectivement δ et α.

3.3.1.1 Modèle de demande d'allocation

Le modèle de demande d'allocation représente l'état dans lequel peut se trouver la fonction. Pour l'allocation des fonctions, nous partons du principe qu'une fonction est insécable, elle ne peut donc être allouée qu'à un unique contrôleur.

3.3.1.1.1 Description des éléments de ce modèle

Le modèle de demande d'allocation comprend les éléments suivants :

Localités

- **Fonction non allouée** ; représente le fait que la fonction n'a pas encore été allouée,
- **Attente réponse contrôleur** ; représente le fait que la fonction est en attente d'une réponse d'un automate acceptation/refus d'une demande,
- **Fonction allouée** ; représente le fait que la fonction est allouée.

Étiquettes

- *Demande_ Allocation*, cette demande montre qu'une fonction fait un appel vers un contrôleur afin de lui demander d'étudier sa possibilité d'allocation.
- *Allocation_ refusée*, ce refus montre que les contraintes de capacité et de sûreté ne sont pas satisfaites si la fonction est allouée au contrôleur.
- *Allocation_ Ok*, cette acceptation montre que toutes les contraintes de capacité et de sûreté sont satisfaites si la fonction est allouée au contrôleur.

[Gardes]

- L'automate de demande d'allocation n'a aucune garde.

Mise à jour

- L'automate de demande d'allocation ne comporte aucune mise à jour.

3.3.1.1.2 Évolutions du modèle

Le but du problème est d'allouer l'ensemble des fonctions. Dans l'état initial, toutes les fonctions sont non allouées. La localité initiale du modèle représentant la demande

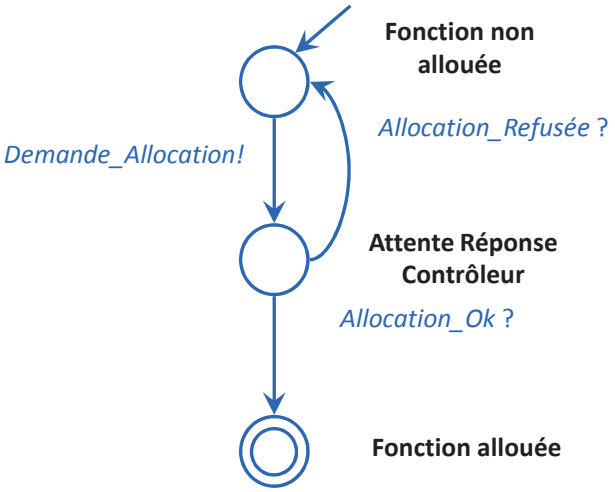


Figure 3.6 – Modèle générique (δ) de demande d'allocation

d'allocation est donc " **Fonction non allouée**". Une seule transition, qui correspond à l'émission d'une demande d'allocation, est possible à partir de cette localité. Une fois

cette demande émise, le modèle attend dans la localité "**Attente Réponse Contrôleur**" la réponse d'un modèle d'acceptation/refus d'un contrôleur qui peut être :

- *Allocation_ refusée*, le modèle retourne alors en localité initiale ;
- *Allocation_ Ok*, le modèle évolue vers la localité marquée « Fonction allouée » qui est une localité puits.

3.3.1.2 Modèle d'acceptation/refus d'une demande

Le modèle d'acceptation/refus d'une demande représente l'état dans lequel peut se trouver un contrôleur. Le contrôleur a la possibilité d'accueillir plusieurs fonctions tant que les contraintes sont respectées.

3.3.1.2.1 Description des éléments de ce modèle

Le modèle d'acceptation/refus d'une demande comprend les localités et les transitions suivantes :

Localités

- **Contrôleur en attente** ; représente le fait que le contrôleur est en attente d'une demande d'allocation,
- **Vérification des contraintes** ; représente le fait que toutes les contraintes doivent être vérifiées si on alloue la fonction émettant une demande à ce contrôleur,
- **Affectation de NS** ; représente l'état où le niveau de sécurité du contrôleur acceptant la fonction est défini.

Étiquettes

- *Demande_ Allocation*, cette demande montre qu'une fonction fait un appel vers un contrôleur afin de lui demander d'étudier sa demande d'allocation.
- *Allocation_ refusée*, ce refus montre que les contraintes de capacité et de sûreté ne sont pas satisfaites si la fonction est allouée au contrôleur.
- *Allocation_ Ok*, cette acceptation montre que toutes les contraintes de capacité et de sûreté sont satisfaites si la fonction est allouée au contrôleur.

[Gardes]

- La garde [Capacité_ok] représente le fait que la fonction peut être acceptée en termes de capacité et est représentée par l'équation 1.1, en l'occurrence les capacités d'entrées/sorties représentées dans les équations 1.2, 1.3, 1.4, 1.5,

- La garde [Exclusion_ok] représente le fait que deux fonctions f^i, f^j, qui ne doivent pas être dans le même contrôleur, ne le soient pas. Si la fonction f^i fait une demande sur ce contrôleur alors que la fonction f^j est déjà allouée dans ce contrôleur alors l'allocation de f^i sera refusée. L'équation qui représente [Exclusion_ok] est l'équation 1.8,

- La garde [Compatibilité_ok] représente l'équation générale 1.6 et dans notre cas, le fait que la fonction f^i, qui fait la demande d'allocation, a son facteur de criticité FC^i compatible avec le niveau de sécurité NS^j du contrôleur c^j. Notre cas particulier est représenté par l'équation 1.7.

- La garde [NS_affecté] ou [NS_non_affecté] représente le fait que le niveau de sécurité du contrôleur a déjà été affecté ou non.

Mises à jour

- caractéristiques_maj, les caractéristiques sont mises à jour, donc les variables $\sum_{i \in I_j} NI_l^i$; $\sum_{i \in I_j} NI_a^i$; $\sum_{i \in I_j} NO_l^i$; $\sum_{i \in I_j} NO_a^i$, sont actualisées.

- $NS := 1$, met à jour le niveau de sécurité du contrôleur à la valeur 1 si l'équation de compatibilité 1.6 n'est pas respectée

- $NS := 2$, met à jour le niveau de sécurité du contrôleur à la valeur 2 si l'équation de compatibilité 1.6 n'est pas respectée

- $NS := 3$, met à jour le niveau de sécurité du contrôleur à la valeur 3 si l'équation de compatibilité 1.6 n'est pas respectée

Les trois dernières mises à jour permettent de générer aléatoirement le niveau de sécurité : NS (CF. Figure 3.7).

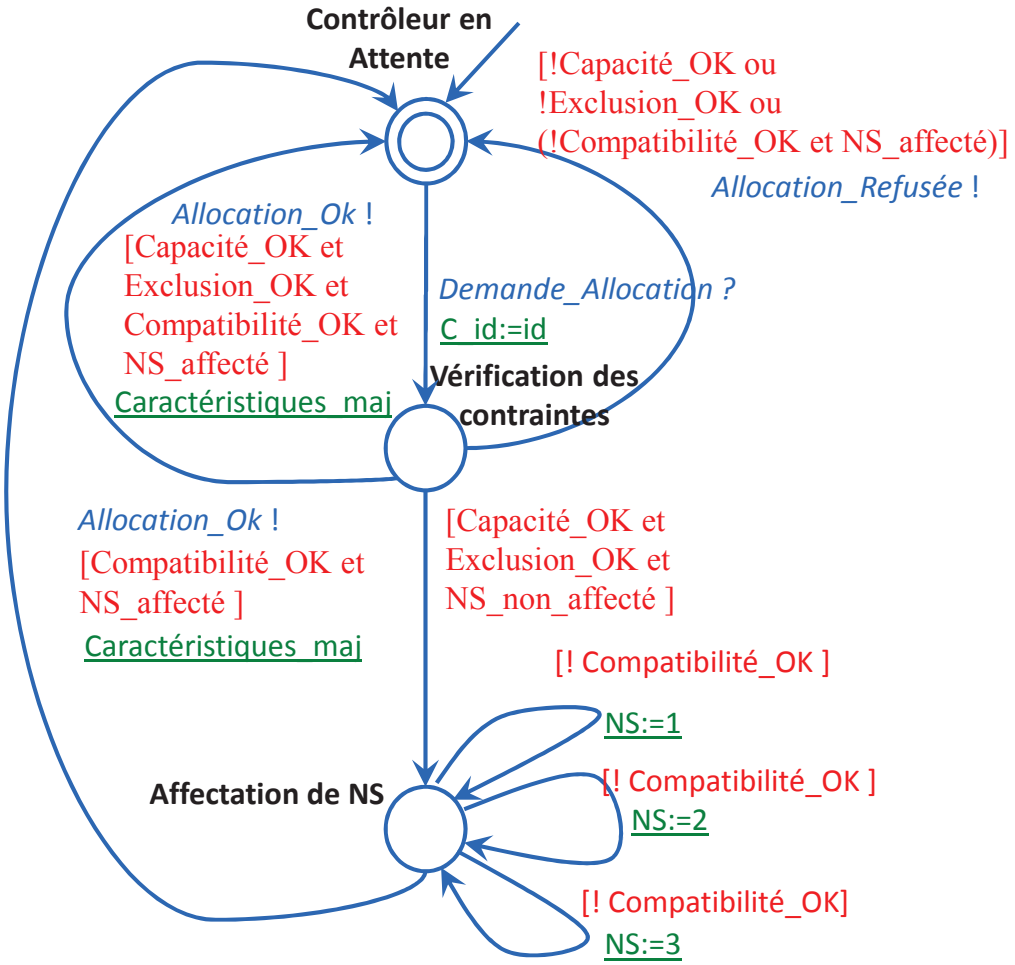


Figure 3.7 – Modèle générique (α) d'acceptation/refus d'une demande

3.3.1.2.2 Évolutions du modèle

Le modèle d'acceptation/refus d'une demande est initialement en attente d'une demande d'allocation d'une fonction. A partir de la localité initiale "**Contrôleur en attente**", qui est également une localité marquée, ce modèle ne peut évoluer que vers la localité " **Vérification des contraintes** " s'il y a réception d'une demande d'allocation. Cette localité permet de vérifier si le contrôleur peut accepter la fonction qui est en cours d'allocation. Pour cela, les contraintes d'allocation sont vérifiées ([Capacité_ok], [Exclusion_ok], [Compatibilité_ok], [NS_affecté] ou [NS_non_affecté]).

Les trois transitions partant de la localité " **Vérification des contraintes** " correspondent à :

- la violation d'une des contraintes d'allocation ; l'étiquette *Allocation_refusée* est alors émise ;
- l'acceptation d'une demande alors qu'aucune autre fonction n'a été préalablement

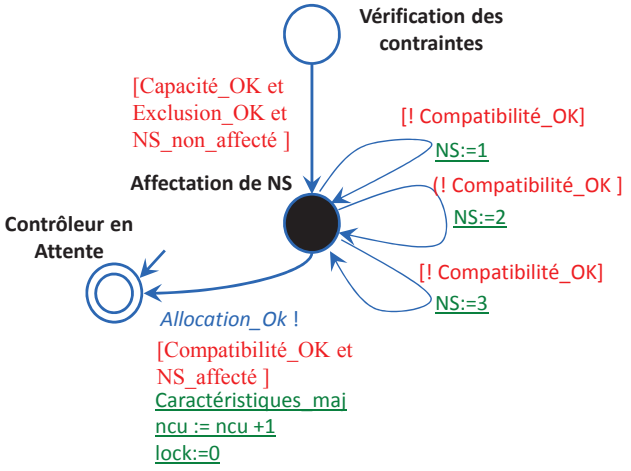
allouée au contrôleur (garde [Capacité_ok et Exclusion_ok et NS_non_affecté]). Le contrôleur n'a initialement aucun niveau de sécurité. Le modèle du contrôleur passe alors dans la localité " Affectation de NS ".

- l'acceptation d'une demande alors qu'au moins une autre fonction a été préalablement allouée (garde [Capacité_ok et Exclusion_ok et Compatibilité_ok et NS_affecté] vraie). L'étiquette *Allocation_ Ok* est alors émise ;

A partir de la localité " **Affectation de NS** " , quatre évolutions sont possibles (cf. Figure 3.8) qui correspondent à deux actions différentes :

- la première action correspond à trois évolutions possibles. Chaque évolution parmi les trois possibles permettent d'affecter le niveau de sécurité du contrôleur à une valeur différente. Ces évolutions se font tant que le niveau de sécurité du contrôleur n'est pas compatible avec le facteur de criticité de la fonction (Cf. matrice présentée ci-après représentant l'équation 1.6). Ce niveau de sécurité est donc affecté aléatoirement tout en respectant la compatibilité avec le facteur de criticité de la fonction.

- la deuxième action n'est possible que lorsque le niveau de sécurité du contrôleur est compatible avec le facteur de criticité de la fonction. Cette évolution permet de mettre à jour toutes les caractéristiques. L'étiquette *Allocation_ Ok* est alors émise et le modèle évolue vers la localité marquée et initiale "**Contrôleur en Attente**".



$$X : FC \times NS \longrightarrow \mathbb{B}$$

$$X = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 3.8 – Évolutions possibles depuis "**Affectation de NS**"

3.3.2 Instanciation des modèles génériques

Le modèle instancié pour un problème d'allocation de L fonctions comporte L automates de demande d'allocation et M automates d'acceptation/refus d'une demande. Ce modèle instancié est donc un réseau d'automates communicants NA qui comporte :

- autant d'instances $(\delta^1, \delta^2, ..., \delta^L)$ du modèle de la figure 3.6 qu'il y a de fonctions à allouer ;
- M instances $(\alpha^1, \alpha^2, ..., \alpha^M)$ du modèle de la figure 3.7 . Pour la suite, nous prendrons comme valeur initiale du nombre de contrôleurs $M = L$ afin d'avoir toujours une possibilité d'allocation avec une fonction par contrôleur.

Il vient donc $NA = \delta^1 \| \delta^2 \| ... \| \delta^L \| \alpha^1 \| \alpha^2 \| ... \| \alpha^M$.
Une évolution synchrone de deux automates n'est possible que si ces deux automates émettent et reçoivent l'un des couples d'étiquettes suivantes :

- *Demande_ Allocation* ! et *Demande_ Allocation* ? ;
- *Allocation_ Ok* ! et *Allocation_ Ok* ? ;
- *Allocation_ Refusée* ! et *Allocation_ Refusée* ?.

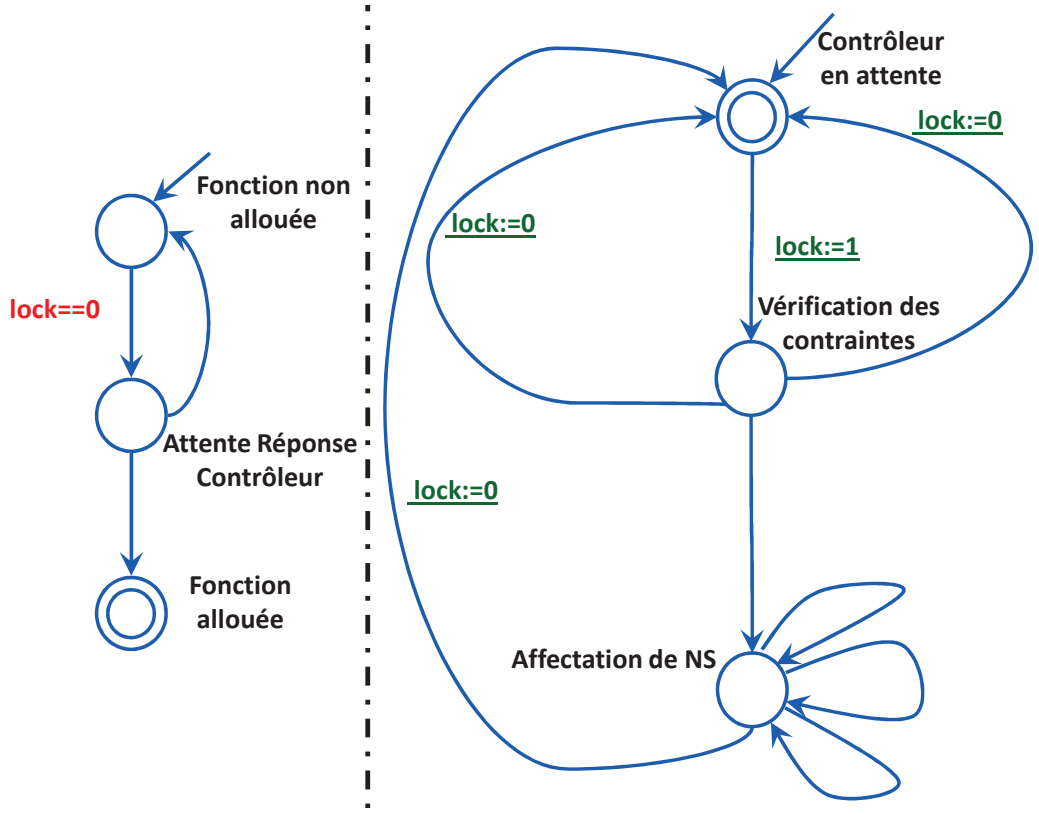
3.3.2.1 Introduction de sémaphores

Afin d'éviter des incohérences consistant à ce qu'une instance α^j émette une réponse à destination d'une instance δ^i autre que celle ayant émis la requête d'allocation, le mécanisme d'appel/réponse doit être conçu comme une section critique protégée par un sémaphore. La réalisation de cette section critique est réalisée avec le sémaphore **lock** présenté dans la figure 3.9.

La variable "lock" permet à un modèle de demande d'allocation d'une fonction d'effectuer une demande uniquement si :

$$lock = 0 \tag{3.1}$$

Ceci permet une allocation de fonction une à une, puisque la garde pour qu'une fonction émette une demande est que l'équation 3.1 soit vraie. Dès qu'une fonction fait une demande et qu'un contrôleur la reçoit, la variable "lock" est mise à 1 pour éviter toute interaction avec une autre fonction.



Les gardes, les étiquettes et les mises à jour sont omises par souci de lisibilité.

Figure 3.9 – Mise en place du sémaphore

La variable est relâchée quand le contrôleur accepte la fonction ou la rejette, par la mise à jour de "lock".

3.3.2.2 Ajout pour les données de l'allocation

Les figures 3.10 et 3.11 montrent les deux modèles génériques utilisés par la suite afin de permettre la récupération de données après l'allocation.

3.3.2.3 Ajout sur le modèle de demande de l'allocation

Il est utile de connaitre des données relatives aux fonctions, par exemple quelle fonction fait la demande (indice de la fonction) ou encore dans quel contrôleur la fonction est allouée. Quelques variables sont donc introduites pour savoir dans quel contrôleur c_id chaque fonction f_id est allouée : $Alloc\_c[f\_id] := c\_id$

Dans le modèle complet de demande d'allocation, la formalisation finale est donc celle présentée précédemment avec les ajouts suivants :

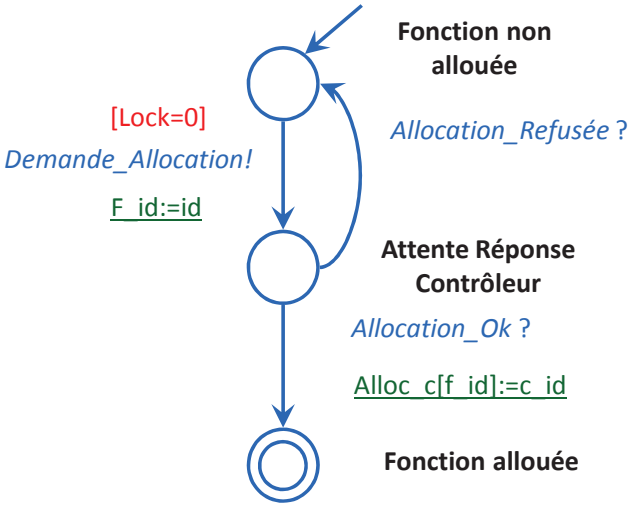


Figure 3.10 – Représentation du modèle δ de demande d'allocation complet

Localité

- Idem que précédemment

Étiquettes

- Idem que précédemment

[Gardes]

- $[Lock = 0]$ permet d'assurer la cohérence entre appels et réponses

Mises à jour

- $Alloc\_c[f\_id] := c\_id$ permet de savoir dans quel contrôleur la fonction qui fait la demande est allouée.

- $f\_id := id$ permet de dire que la fonction qui fait la demande f^{id} est le numéro de l'instance actuelle.

3.3.2.4 Ajout sur le modèle d'acceptation/refus d'une demande

Il est utile de connaitre le nombre de contrôleurs utilisés réellement une fois l'allocation terminée. Pour cela, une variable est ajoutée : *ncu* (Nombre de Contrôleurs Utilisés). Cette variable est initialement nulle et est incrémentée dès qu'un nouveau contrôleur est utilisé (cf. figure 3.11).

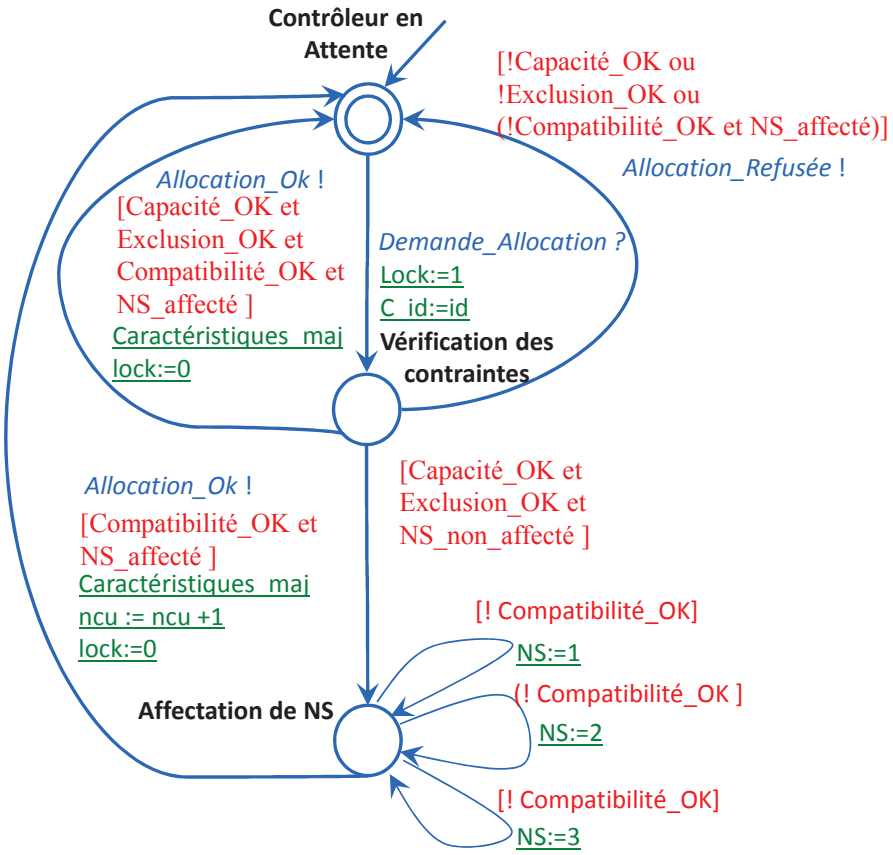


Figure 3.11 – Représentation du modèle complet d'acceptation/refus d'une demande

Dans le modèle complet d'acceptation/refus d'une demande, la formalisation finale est donc :

Localité

- Idem que précédemment

Étiquette

- Idem que précédemment

[Gardes]

- Idem que précédemment

Mises à jour

- $\underline{Lock := 1}$, met à jour le sémaphore ce qui empêche d'avoir une nouvelle demande d'allocation

- $\underline{Lock := 0}$, met à jour le sémaphore ce qui permet d'avoir une nouvelle demande d'allocation
- $\underline{ncu := ncu + 1}$, met à jour le nombre de contrôleurs utilisés, à chaque fois qu'un contrôleur accepte sa première fonction, il doit affecter son NS et il augmente de 1 le nombre de contrôleurs utilisé.
- $\underline{C\_id := id}$ permet de dire que le contrôleur qui vérifie la demande est le numéro de l'instance actuelle.

3.3.3 Exemple d'allocation de 5 fonctions sur 3 contrôleurs

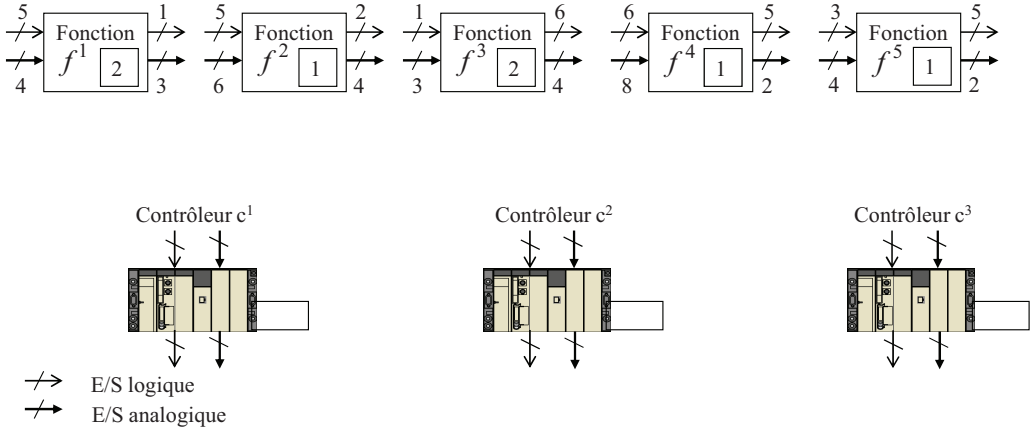


Figure 3.12 – Les cinq fonctions et trois contrôleurs de l'exemple traité

L'exemple présenté dans la section 1.4 consiste à allouer 5 fonctions sur un ensemble de 3 contrôleurs (cf. figure 3.12).

Le réseau d'automates communicants permettant de résoudre ce problème est donné à la figure 3.13.

Ces fonctions sont à répartir sur un ensemble de trois contrôleurs c^1, c^2, c^3, dont les capacités sont considérées dans ce cas jouet comme identiques :

$\forall j \in \{1,2,3\}, LI^j_{max} = AI^j_{max} = LO^j_{max} = AO^j_{max} = 10$

Une solution possible d'allocation des fonctions sur les trois contrôleurs est décrite dans la figure 3.14.

Cette solution peut être obtenue par le scénario suivant. Supposons que le modèle associé à la fonction f^1 fasse en premier une demande d'allocation. Supposons également que, parmi tous les contrôleurs c^1 ; c^2 ; c^3, le modèle associé au contrôleur c^1 est celui qui reçoit cette demande.

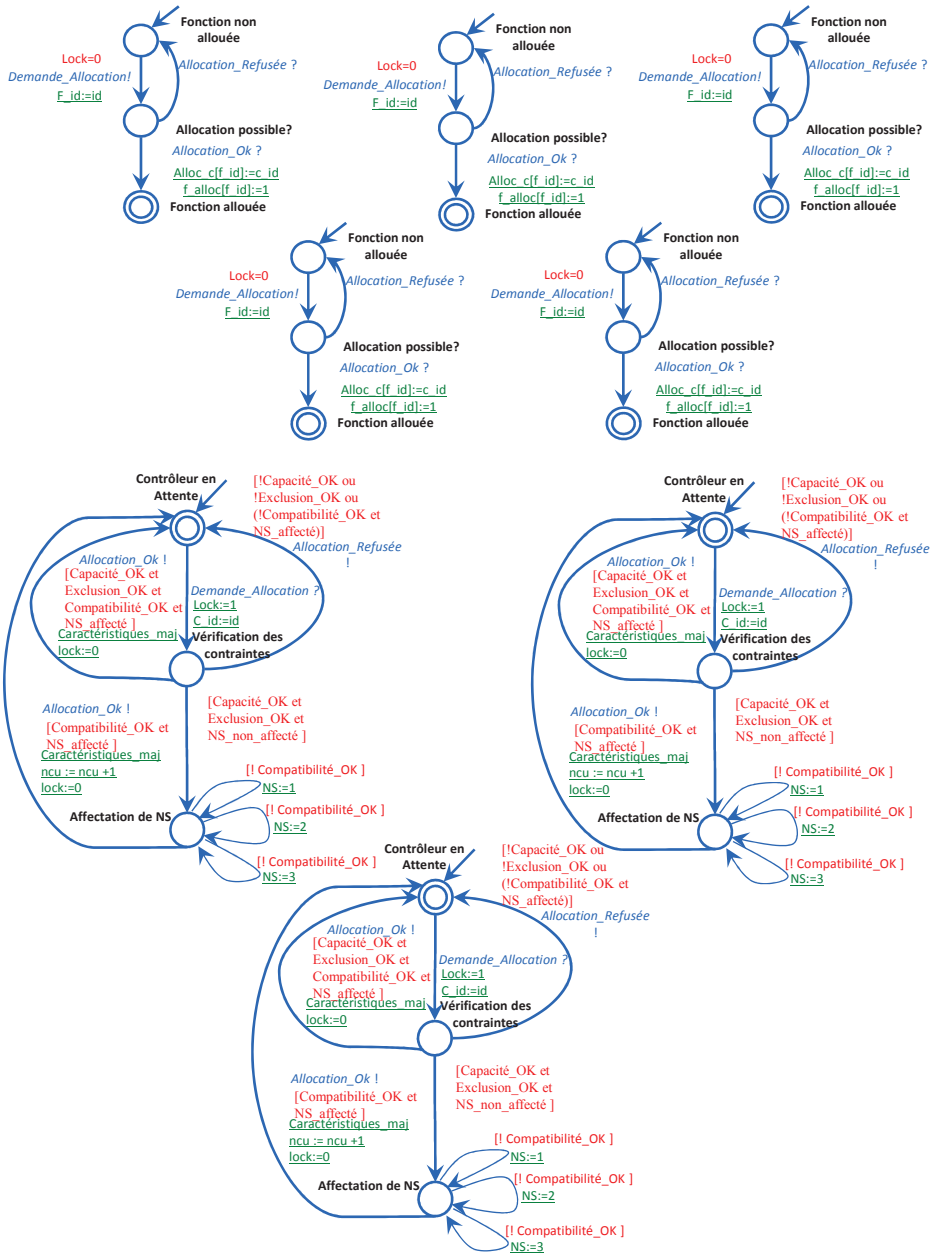


Figure 3.13 – Modèle de l'exemple de 5 fonctions et 3 contrôleurs

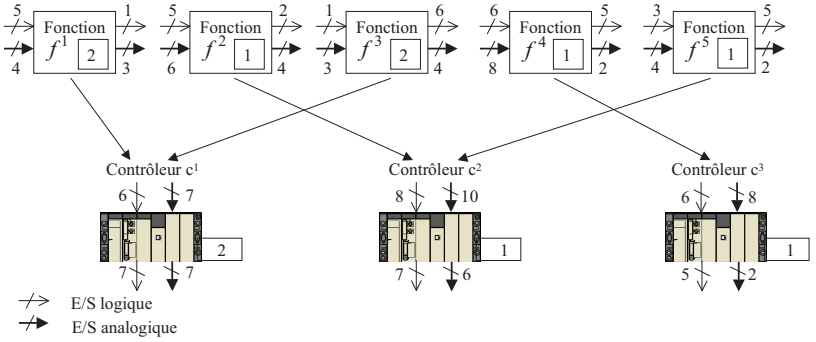


Figure 3.14 – Une solution d'allocation de cinq fonctions sur trois contrôleurs

Comme ce contrôleur n'héberge aucune fonction et que son niveau de sécurité est indéfini, la fonction f^1 peut être allouée à ce contrôleur. Le niveau de sécurité est déterminé lorsque l'une des transitions de type self-loop issues de l'état **Affectation de NS** est franchie et que ce franchissement fournit une valeur de ce niveau compatible avec le facteur de criticité de la fonction (figure 3.15). Le modèle de la fonction 1 passe alors en "**fonction allouée**", la variable Lock devient égale à 0 et l'automate d'acceptation/refus d'une demande associé à c^1 passe en la localité "**Contrôleur en Attente**".

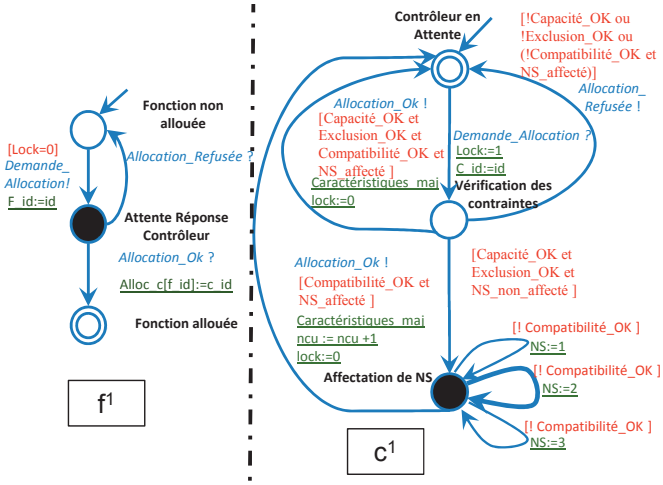


Figure 3.15 – Évolutions de la fonction f^1 et du contrôleur c^1

La fonction f^2 peut ensuite être allouée au contrôleur c^2, dont le niveau de sécurité sera fixé à $NS^2 = 1$. Puis la fonction f^3 peut être allouée au contrôleur c^1 car les contraintes 1.9, 1.10 , 1.11, 1.12, et 1.13 sont respectées.

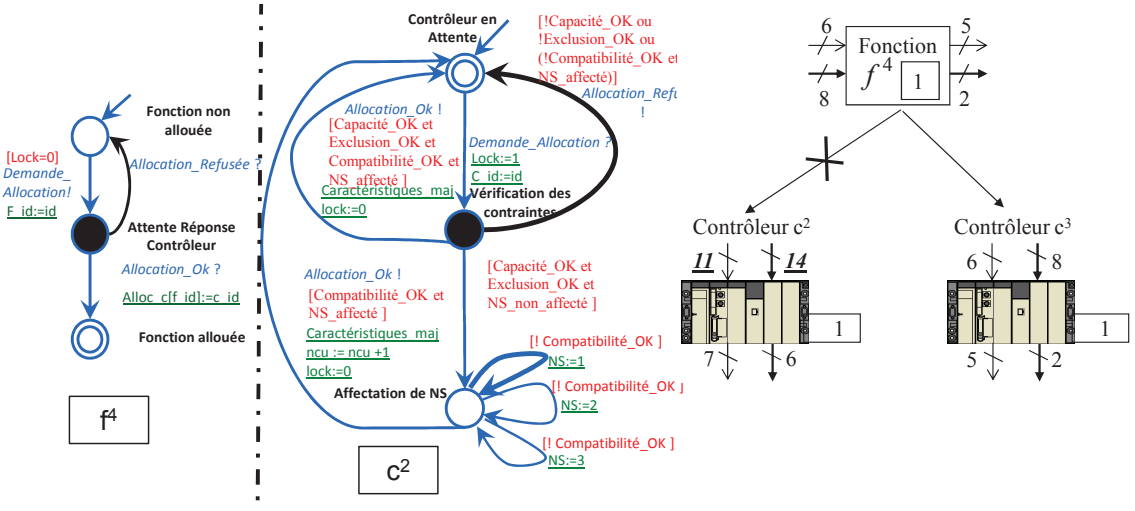


Figure 3.16 – Évolutions de la fonction f^4 et du contrôleur c^2

La fonction f^4 ne peut pas être alors allouée au contrôleur c^2, car l'allocation des fonctions f^2 et f^4 à ce contrôleur violerait le système de contraintes, comme le montre la figure 3.16. Elle peut par contre être allouée au contrôleur c^3.

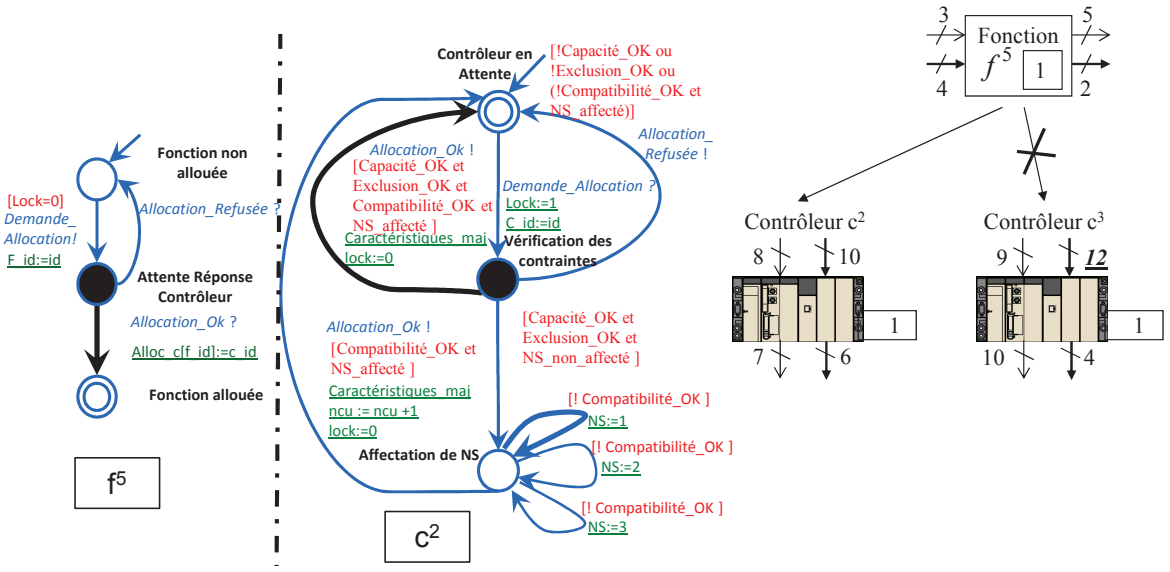


Figure 3.17 – Évolutions de la fonction f^5 et du contrôleur c^2

La fonction f^5 ne peut pas être allouée au contrôleur c^3, car la contrainte des entrées analogiques 1.3 ne serait pas respectée. L'automate de demande d'allocation de la fonction f^5 fait donc une demande à l'automate d'acceptation/refus d'une demande du contrôleur c^2 qui vérifie toutes les contraintes et accepte la demande, comme le montre la figure 3.17.

Il convient de noter que cette solution n'est pas unique. Comme les contrôleurs possèdent les mêmes nombres d'entrées-sorties, d'autres solutions satisfaisantes peuvent être obtenues par permutation circulaire des contrôleurs ou en permutant simplement deux contrôleurs. Ces solutions sont équivalentes à celle détaillée ci-dessus si aucune autre contrainte n'est introduite.

3.4 La recherche d'atteignabilité

3.4.1 Principe d'analyse

Dans le cadre de cette thèse, nous cherchons à réaliser une allocation de l'ensemble des fonctions sur un ensemble des contrôleurs. Le but est donc d'aller rechercher un état parmi ceux existants où toutes les fonctions sont allouées. La recherche d'atteignabilité

n'est utilisée que sur des automates temporisés ou non, elle est aussi utilisée dans d'autres domaines comme par exemple sur des réseau de petri [Boucheneb et Barkaoui, 2012]. Pour faire de la recherche d'atteignabilité nous avons besoin d'un outil de vérification formelle.

Les techniques de vérification formelle par model-checking [Bérard *et al.*, 2001] ont pour objectif de prouver qu'un modèle satisfait une propriété formelle qui peut être par exemple une propriété d'atteignabilité. Il est donc naturel d'envisager la mise en œuvre de la méthode proposée à l'aide d'une telle technique.

Pour cela, le model-checker doit avoir un modèle formel à vérifier et une propriété formelle à vérifier (cf. figure 3.18). En explorant l'espace d'états du modèle, le model-checker va vérifier si la propriété est vérifiée ou non et donner un exemple ou un contre exemple, s'il est demandé. La propriété est écrite en logique temporelle temporisée ou

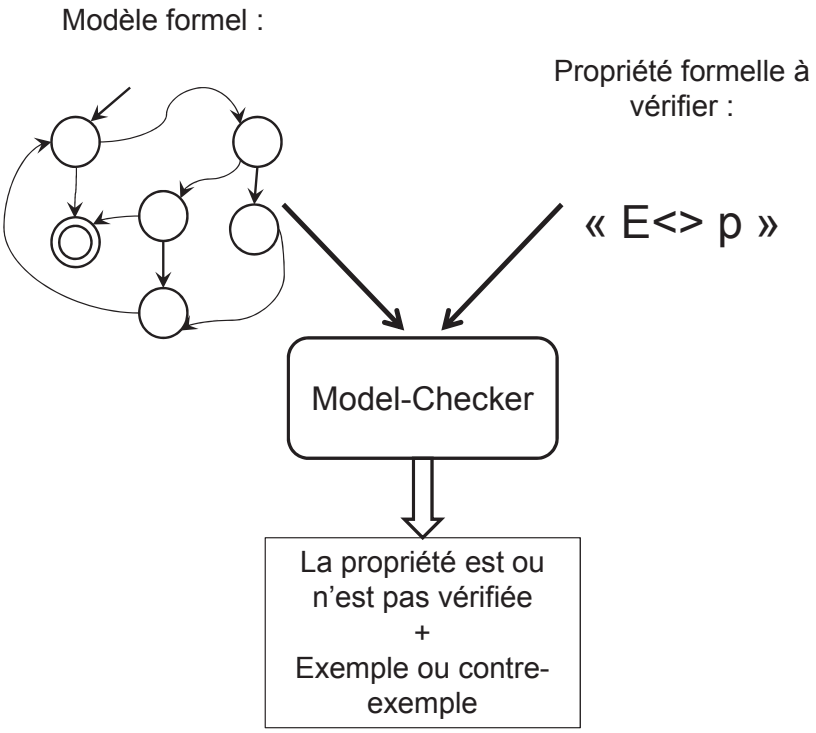


Figure 3.18 – Principe du model-checking

non temporisée. Dans notre cas, nous avons des modèles non temporisés, nous utiliserons donc des logiques temporelles non temporisées.

La vérification formelle par model-checking est souvent utilisée pour des recherches dans les domaines de l'évaluation de performances [Ruel, 2009], de l'analyse prévisionnelle de fautes [Perin et Faure, 2008] ou encore de la vérification de contrôleur([Lindahl *et al.*, 1998],[Rossi et Schnoebelen, 2000],[Gourcuff *et al.*, 2008]).

Le principe d'utiliser ici la recherche d'atteignabilité sur un mécanisme d'appel/réponse permet de n'avoir aucune stratégie d'allocation et de laisser le model-checker explorer l'espace d'état pour trouver une trace qui correspond à une allocation admissible. Nous cherchons donc à vérifier que toutes les fonctions sont allouées par la vérification d'une propriété. Cette propriété, si elle est vérifiée, permettra d'avoir un exemple ou trace donnant une solution d'allocation.

3.4.2 Définition de la propriété d'atteignabilité recherchée

Toutes les fonctions sont allouées lorsque la localité marquée est atteinte dans toutes les instances du modèle générique de demande d'allocation. Lorsqu'il en est ainsi, les instances du modèle générique d'acceptation/refus se trouvent dans la localité initiale, qui est marquée. La propriété d'atteignabilité à vérifier peut donc s'énoncer, de manière informelle, de la manière suivante :
Est-il possible d'atteindre, à partir de l'état initial, un état du réseau d'automates tel que la localité active soit une localité marquée dans tous les automates du réseau ?

La figure 3.19 montre un exemple de l'état recherché par cette propriété, sur l'exemple de 5 fonctions et 3 contrôleurs.

3.4.3 Mise en œuvre à l'aide d'un outil de vérification formelle

La mise en œuvre à l'aide d'un outil de vérification formelle exige en premier lieu d'énoncer formellement la propriété recherchée, donnée de manière textuelle à la section précédente. Il existe deux principales logiques temporelles permettant la description de propriétés, comme la LTL (Linear Temporal Logic [Pnueli, 1981]) ou la CTL (Computation Tree Logic ([Clarke et Emerson, 1981], [Emerson et Halpern, 1986])). La LTL permet de définir une propriété le long d'un chemin. Une propriété écrite dans ce langage utilise uniquement des quantificateurs d'état et une proposition logique à vérifier. En revanche, le CTL permet de définir une propriété dans le cadre de l'exploration d'un futur décrit sous la forme d'un arbre. Une propriété écrite dans ce langage utilise des couples de quantificateurs (un de chemin et un d'état) et une proposition logique à vérifier.

Les opérateurs de chemin et d'état sont :

- A pour *All*, opérateur de chemin décrivant tous les chemins,

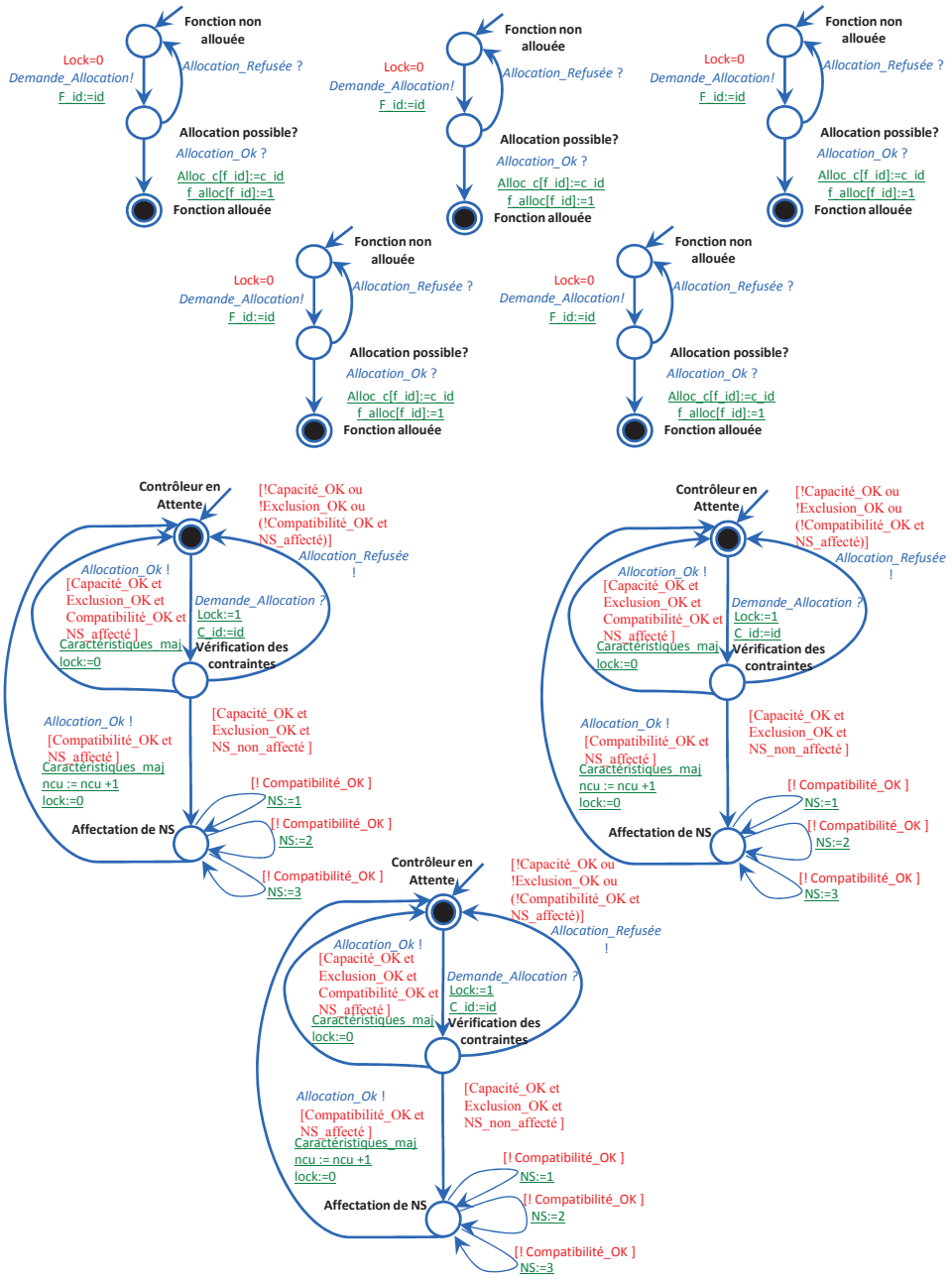


Figure 3.19 – État recherché par la propriété sur l'exemple

- E pour *Exists*, opérateur de chemin décrivant au moins un chemin,
- F pour *Finally*, opérateur temporel décrivant un état du futur,
- G pour *Globally*, opérateur temporel décrivant tous les états futurs.

Pour la logique LTL, un autre opérateur temporel existe : l'opérateur X, pour *neXt*, qui correspond au prochain état.

En utilisant les quantificateurs de la logique temporelle CTL, car nous recherchons l'existence d'un chemin, nous allons utiliser la possibilité :

La propriété EF p est vraie si et seulement si il existe au moins une séquence d'évolutions partant de l'état initial s_0 et atteignant un état s_n tel que s_n satisfait p.

La propriété qui nous intéresse s'écrit donc :

$$P : \text{EF } \textit{Allocation totale}$$

Allocation totale désigne l'état du réseau tel que la localité active soit une localité marquée pour tous les automates. Cette propriété est vérifiée s'il existe au moins une trace partant de l'état initial du réseau et atteignant *Allocation totale*.

La recherche d'une solution d'allocation peut donc être réalisée en prouvant que le réseau d'automates NA satisfait la propriété ci-dessus, ce qui sera noté : $NA \models P$. Il est évident que cette preuve dépend du nombre M d'instances du modèle d'acceptation/refus α. Il est par exemple illusoire de chercher à allouer à un seul contrôleur ($M = 1$) deux fonctions ($L = 2$) si ces fonctions ont des facteurs de criticité incompatibles. On peut cependant remarquer que, si : $\forall i$, $\forall j$, $NI_l^i \leq LI_{max}^j$, $NI_a^i \leq AI_{max}^j$, $NO_l^i \leq LO_{max}^j$, $NO_a^i \leq AO_{max}^j$, le cas $M = L$ conduit à coup sûr à une preuve positive. L'examen de la trace conduisant à l'état *Allocation totale* montre en général dans ce cas que tous les contrôleurs ne sont pas utilisés ; seul un nombre $N < M$ hébergent une ou plusieurs fonctions.

3.4.3.1 Proposition d'un ensemble de solutions

A partir d'une première solution obtenue, il est donc possible de générer d'autres solutions satisfaisant toujours l'ensemble des contraintes. En effet, une contrainte d'exclusion entre fonctions est ajoutée (soit directement dans le système de contraintes, soit dans l'écriture de la propriété d'atteignabilité à vérifier) pour obtenir une solution différente de la précédente.

3.4.3.2 Réduction du nombre de contrôleurs

Afin de réduire le nombre de contrôleurs utilisés dans l'architecture opérationnelle, la démarche par preuves itératives de l'algorithme 1 a donc été développée. Cette démarche consiste, lorsqu'une solution a été trouvée, à s'assurer qu'il n'existe pas de solution si on réduit le nombre de contrôleurs.

Dans cet algorithme :

Algorithme 1 Recherche d'une solution d'allocation réduisant le nombre de contrôleurs

Inputs:
- Définition de la fonction à allouer :
  $f^i \in F = (FC^i, NI_l^i, NI_a^i, NO_l^i, NO_a^i), \forall i \in \{1, ..., L\}$
- Caractéristiques techniques du contrôleur :
  $LI_{max}^j, AI_{max}^j, LO_{max}^j, AO_{max}^j, \forall j \in \{1, ..., M\}$

Outputs:
- N : Nombre de contrôleurs utilisés pour allouer l'ensemble des fonctions ;
- Niveau de Sécurité de chaque contrôleur utilisé :
  $NS^j, \forall j \in \{1, ..., N\}$
- Liste des fonctions allouées à chaque contrôleur :
  $I_j = \{i \in \{1, ..., L\} | f^i \leftarrow c^j\}$

1: /* Initialisation : */
2: $M \leftarrow L; N \leftarrow M$
3: $NA = \delta^1 \| \delta^2 \| ... \| \delta^L \| \alpha^1 \| \alpha^2 \| ... \| \alpha^M$
4: /* Construction itérative : */
5: **while** $NA \models P$ **do**
6: **if** $\exists I_j = \emptyset | j \in \{1, ..., N\}$ **then**
7: $N \leftarrow N - Card(I) - 1$ avec $I = \{I_j, j \in \{1, .., N\} | I_j = \emptyset\}$
8: **else**
9: $N \leftarrow N - 1$
10: **end if**
11: $NA = \delta^1 \| \delta^2 \| ... \| \delta^L \| \alpha^1 \| \alpha^2 \| ... \| \alpha^N$
12: **end while**
13: $N \leftarrow N + 1$
14: /* Afficher : */

- N ;
- $NS^j, \forall j \in \{1, ..., N\}$;
- I_j , $\forall j \in \{1, ..., N\}$

- $I_j = \{f^i \in F | c^j \leftarrow f^i\}$ représente l'ensemble des indices des fonctions f^i qui sont allouées au contrôleur c^j ;
- $I_j = \emptyset$ représente le fait qu'aucune fonction n'est allouée au contrôleur c^j, le contrôleur est donc vide ;
- $I = \{I_j, j \in \{1, .., N\} | I_j = \emptyset\}$ représente l'ensemble des I_j tel que $I_j = \emptyset$, il représente donc l'ensemble des contrôleurs vides ;
- $Card(I)$ représente le nombre de contrôleurs vides.

Cet algorithme commence avec un nombre de contrôleurs $M = L$: nombre de fonc-

tions (ligne 2 de l'algorithme). A la ligne suivante, nous créons le modèle complet comprenant L modèles de demande d'allocation et M modèles d'acceptation/refus d'une demande. Nous avons donc : $NA = \delta^1 \| \delta^2 \| ... \| \delta^L \| \alpha^1 \| \alpha^2 \| ... \| \alpha^L$ et nous cherchons à vérifier P : EF *Allocation totale*

Cette propriété est alors vérifiée et soit elle est positive, soit elle est négative :

- si elle est positive, une solution est alors trouvée pour un nombre de contrôleurs $M = L$ mais certains de ces contrôleurs ne sont pas utilisés (contrôleur qui ne contient aucune fonction : $I_j = \emptyset$ (ligne 6 de l'algorithme)). Deux cas sont alors possibles :

  1. s'il existe un ou plusieurs contrôleurs vides $I_j = \emptyset$ alors le nombre de contrôleurs vides est $Card(I)$ et donc nous posons la nouvelle valeur de $N \leftarrow N - Card(I) - 1$. N prend cette valeur car nous connaissons déjà une solution pour $N - Card(I)$ (celle que nous venons de trouver et qui est égale à *ncu* définit dans les modèles) ;
  2. s'il n'existe pas de contrôleurs vides alors nous enlevons un contrôleur au modèle précédent $N \leftarrow N - 1$ (ligne 9 de l'algorithme).

  La preuve est vérifiée, à nouveau, à la suite de ces deux cas, avec
  $NA = \delta^1 \| \delta^2 \| ... \| \delta^L \| \alpha^1 \| \alpha^2 \| ... \| \alpha^N$

- si la propriété est négative alors nous posons $N \leftarrow N + 1$ car pour N nous n'avons pas trouvé de solution.

Dans le cadre industriel, un ensemble de $L = 200$ fonctions doit pouvoir être traité. Cet ensemble de $M = L = 200$ fonctions doit être alloué dans un ensemble de contrôleurs que nous prendrons de 200 contrôleurs. Pour la modélisation de cet exemple (le même exemple que le premier présenté dans le chapitre précédent), toutes les valeurs d'entrées/sorties logiques/analogiques ont été générées automatiquement avec des valeurs comprises entre 0 et 10, donc $\forall i \in \{1, ..., 200\}$,
NI_a^i, NI_l^i, NO_a^i, $NO_l^i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^4$.

Nous avons donc :$NA = \delta^1 \| \delta^2 \| ... \| \delta^{200} \| \alpha^1 \| \alpha^2 \| ... \| \alpha^{200}$ et nous cherchons à vérifier P : EF *Allocation totale.*
Lors de la première recherche, la propriété est vérifiée et le nombre de contrôleurs utilisés

est de 34 en 4,318 secondes ($card(I) = 166$, 166 contrôleurs vides). Une nouvelle valeur est donc affectée à N en tenant compte du nombre de contrôleurs vides dans la solution précédente :

$N \leftarrow 200 - 166 - 1$ d'où $N \leftarrow 33$

La même propriété est à nouveau recherchée, avec : $NA = \delta^1 \| \delta^2 \| ... \| \delta^{200} \| \alpha^1 \| \alpha^2 \| ... \| \alpha^{33}$

Lors de cette nouvelle recherche, la propriété n'est pas vérifiée. Le nombre final de contrôleurs est donc $N = 33 + 1 = 34$.

Tableau 3.1 – Résultats des 3 cas d'études de 200 fonctions par recherche d'atteignabilité

Cas d'étude de 200 fonctions n°	1	2	3
Nombre de contrôleurs de la 1^{re} solution faisable	34	36	35
Temps d'obtention de la 1^{re} solution faisable	4.31 s	4.32 s	4.37 s
Nombre de contrôleurs de la meilleure solution	34 (NO)	35 (NO)	35 (NO)
Temps d'obtention de la meilleure solution	627 s	852 s	659 s

NO : Non Optimale, atteignant la limite mémoire de l'ordinateur utilisé
Les temps d'obtention sont obtenus en secondes

Les résultats des deux autres cas d'étude utilisés dans le chapitre précédent sont dans le tableau 3.1.

3.5 Mise en œuvre de la démarche proposée

3.5.1 Mise en œuvre de l'approche par recherche d'atteignabilité sous UPPAAL

Plusieurs outils de vérification formelle par model-checking, tels que NuSMV [Cimatti *et al.*, 2002], SPIN [Holzmann, 2004], UPPAAL [Behrmann *et al.*, 2006], peuvent être envisagés pour réaliser l'analyse d'atteignabilité sur laquelle s'appuie la recherche d'une solution d'allocation. L'outil UPPAAL [Behrmann *et al.*, 2002] a été retenu car il possède une interface graphique très ergonomique et peut fournir une trace d'exécution en cas de preuve positive. Il importe de souligner que seules ces caractéristiques ont motivé ce choix ; la capacité de cet outil à vérifier des propriétés sur des modèles temporisés ne constitue en aucun cas un critère de sélection, les automates communicants utilisés dans ce travail n'étant pas temporisés.

UPPAAL utilise une sous classe de la logique CTL limitant à cinq le nombre de

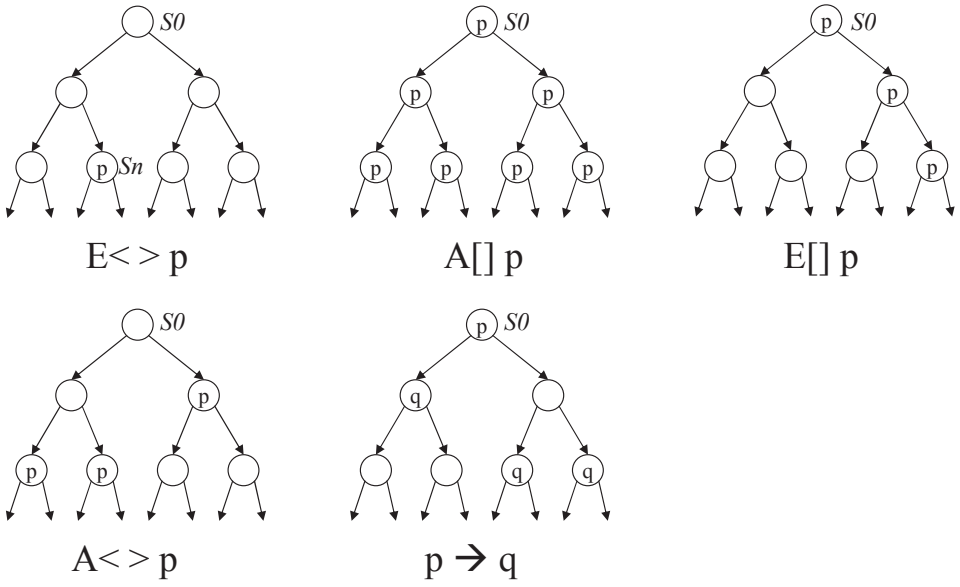


Figure 3.20 – Propriétés logiques d'UPPAAL [Ruel, 2009]

types de propriétés utilisables. Soient p et q, deux propositions logiques qui sont des expressions booléennes sur les localités du modèle formel (&&, ||, !) et des conditions sur des variables (expressions utilisant $<$, $>$, $\leq$, $\geq$, $=$), alors les 5 types de propriétés utilisables dans UPPAAL sont (cf.figure 3.20) :

- Possibily : la propriété E<> p est vraie si et seulement s'il existe au moins une séquence d'évolutions partant de l'état initial s_0 et atteignant un état s_n où la proposition p est vérifiée.

- Invariantly : la propriété A[] p est vraie si et seulement si tous les états atteignables depuis l'état initial vérifient p.

- Potentially always : la propriété E [] p est vraie si et seulement s'il existe au moins une séquence d'évolutions partant de l'état initial pour laquelle p est vérifiée pour tous les états. Cette séquence doit en plus être infinie ou aboutissant sur un blocage.

- Eventually : la propriété A <> p est vraie si et seulement si toutes les séquences d'évolutions possibles partant de l'état initial atteignent un état vérifiant p.

- Leads to : la syntaxe $p \rightarrow q$ décrit une propriété signifiant que si p est vérifiée, alors q devra être vérifiée dans le futur pour toutes les séquences possibles depuis cet état.

Dans cette partie, les déclarations faites dans UPPAAL ainsi que les modèles d'UPPAAL sont présentés.

3.5.1.1 Déclaration des constantes dans UPPAAL

Dans un premier temps, voici la définition de toutes les constantes utilisées dans le logiciel et décrites précédemment pour un exemple d'allocation de 20 fonctions sur 20 contrôleurs.

Le nombre de contrôleurs, égal au nombre de fonctions, est définit dans UPPAAL :

```
const int NUMBER_OF_CONTROLLER = 20;
const int NUMBER_OF_FUNCTION = 20;
```

Une fois les nombres de fonctions et contrôleurs définis, toutes les variables utilisées sont définies et :

UPPAAL	représentant
meta int f_id;	f^i
meta int c_id;	c^j
meta int alloc_c[NUMBER_OF_FUNCTION];	lieu de l'allocation
meta bool f_alloc[NUMBER_OF_FUNCTION];	fonction allouée ou non
meta int ncu = 0;	nombre de contrôleurs utilisés
int FC, NS;	FC et NS
int lock;	le sémaphore

Lorsque les variables sont définies, les caractéristiques des fonctions sont intégrées, sous la forme d'une structure :

- le facteur de criticité : noté FC
- le nombre d'entrées logiques : noté LI
- le nombre d'entrées analogiques : noté AI
- le nombre de sorties logiques : noté LO
- le nombre de sorties analogiques : noté AO

```
const struct { int FC; int LI; int AI; int LO; int AO; }
fp[NUMBER_OF_FUNCTION] = {
{1, 6, 6, 4, 0}, {1, 2, 3, 9, 1}, {1, 5, 6, 9, 7}, {1, 5, 5, 2, 3}, {1, 5, 4, 3, 4},
{2, 0, 7, 6, 9}, {2, 0, 5, 4, 3}, {2, 4, 4, 1, 3}, {2, 0, 4, 8, 4}, {2, 9, 7, 9, 2},
{3, 4, 6, 8, 3}, {3, 2, 6, 0, 9}, {3, 8, 8, 6, 4}, {3, 4, 0, 9, 8}, {3, 6, 6, 0, 1},
{4, 9, 1, 6, 4}, {4, 4, 9, 3, 6}, {4, 5, 1, 5, 4}, {4, 2, 4, 6, 9}, {4, 0, 8, 2, 5} };
```

3.5.1.2 Définition des étiquettes dans UPPAAL

Pour permettre les communications par appel/réponse, les étiquettes doivent être définies et reprennent les trois étiquettes présentées précédemment :

```
chan Demande_allocation;
chan allocation_OK;
chan allocation_refusee;
```

3.5.1.3 Définition des gardes dans UPPAAL

Dans cette partie, nous expliquons chaque terme utilisé dans les modèles au niveau des gardes.

3.5.1.3.1 Compatibilité

La compatibilité reprend l'équation 1.6 relative au compatibilité requise entre le facteur de criticité de la fonction et le niveau de sécurité du contrôleur.

```
bool Compatibilite_ok(int FC, int NS) {
const bool FC_NS[4][3] = {
{1, 0, 0},
{1, 1, 0},
{0, 1, 1},
{0, 0, 1} };
meta bool result;
if (NS != 0) result = FC_NS[FC-1][NS-1]; else result = 0;
return result; }
```

3.5.1.3.2 NS_ affecté

L'affectation de NS doit se faire. Dans le logiciel, initialement NS est nul, pour que NS soit affectée, sa valeur doit donc être différente de zéro.

```
bool NS_affecte(int NS){
return NS != 0 ; }
```

3.5.1.3.3 Capacité

La contrainte de capacité permet de respecter les équations 1.2, 1.3, 1.4, 1.5 et est décrite avec :

- les caractéristiques de la fonction ($fp.LI$ ou $fp.AI$ ou $fp.LO$ ou $fp.AO$) avec
  - $fp.LI$ correspond à NI_l^i de la fonction actuelle ;
  - $fp.AI$ correspond à NI_a^i de la fonction actuelle ;
  - $fp.LO$ correspond à NO_l^i de la fonction actuelle ;
  - $fp.AO$ correspond à NO_a^i de la fonction actuelle.
- les caractéristiques actuelles du contrôleur qui représentent la somme des caractéristiques des fonctions qui lui sont allouées. Par exemple, $cp.LI = \sum_{i \in I_j} fp.LI^i$ et $I_j = \{i \in \{1, ..., L\} | f^i \leftarrow c^j\}$;
- les caractéristiques externes du contrôleur qui sont déjà à la valeur : 32.

```
bool capacite_OK (struct int FC ; int LI ; int AI ; int LO ; int AO ;
fp, struct int NS ; int LI ; int AI ; int LO ; int AO ;  cp )
{return (cp.LI + fp.LI <= 32) and (cp.AI + fp.AI <= 32)
and (cp.LO + fp.LO <= 32) and (cp.AO + fp.AO <= 32) ;}
```

3.5.1.3.4 Séparation

Voici la définition d'une contrainte d'exclusion entre 2 fonctions sous UPPAAL. La fonction f^1 ne peut pas être allouée avec la fonction f^2 et vice versa.

```
bool func_constraints_are_OK(int f_id, int c_id,
int alloc_c[NUMBER_OF_FUNCTION])
{ bool result = 1;
if ((f_id == 1) and (alloc_c[ 2] == c_id)) result = 0;
if ((f_id == 2) and (alloc_c[ 1] == c_id)) result = 0;
return result;}
```

3.5.1.4 Définition de la mise à jour des caractéristiques dans UPPAAL

La mise à jour des caractéristiques dans UPPAAL permettent de mettre à jour les caractéristiques actuelles du contrôleur.

```
void caractéristiques_maj (
struct int FC; int LI; int AI; int LO; int AO;  fp,
struct int NS; int LI; int AI; int LO; int AO; & cp )
{ cp.LI = cp.LI + fp.LI;
cp.AI = cp.AI + fp.AI;
cp.LO = cp.LO + fp.LO;
cp.AO = cp.AO + fp.AO; }
```

3.5.1.5 Modèle de l'automate de demande d'allocation dans UPPAAL

Cette figure représente le modèle comprenant toutes les gardes, mises à jour, et étiquettes présentés précédemment. Le terme f_id permet de récupérer l'indice de la fonction, cette donnée est ainsi utilisée dans d'autres variables comme :

- $alloc\_c(f\_id)$ qui permet de récupérer l'indice du contrôleur où la fonction a été allouée.
- $f\_alloc(f\_id)$ qui permet de savoir si la fonction a été allouée.

3.5.1.6 Modèle de l'automate d'acceptation/refus d'une demande d'allocation dans UPPAAL

Cette figure 3.22 représente le modèle comprenant toutes les gardes, les mises à jour et les étiquettes présentées précédemment. c_id permet de récupérer l'indice du contrôleur actuellement utilisé, cette donnée est ainsi utilisée dans la contrainte d'exclusion.

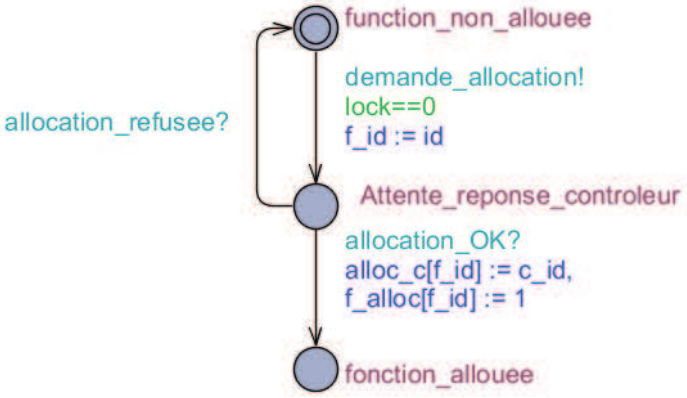


Figure 3.21 – Modèle UPPAAL d'une demande d'allocation

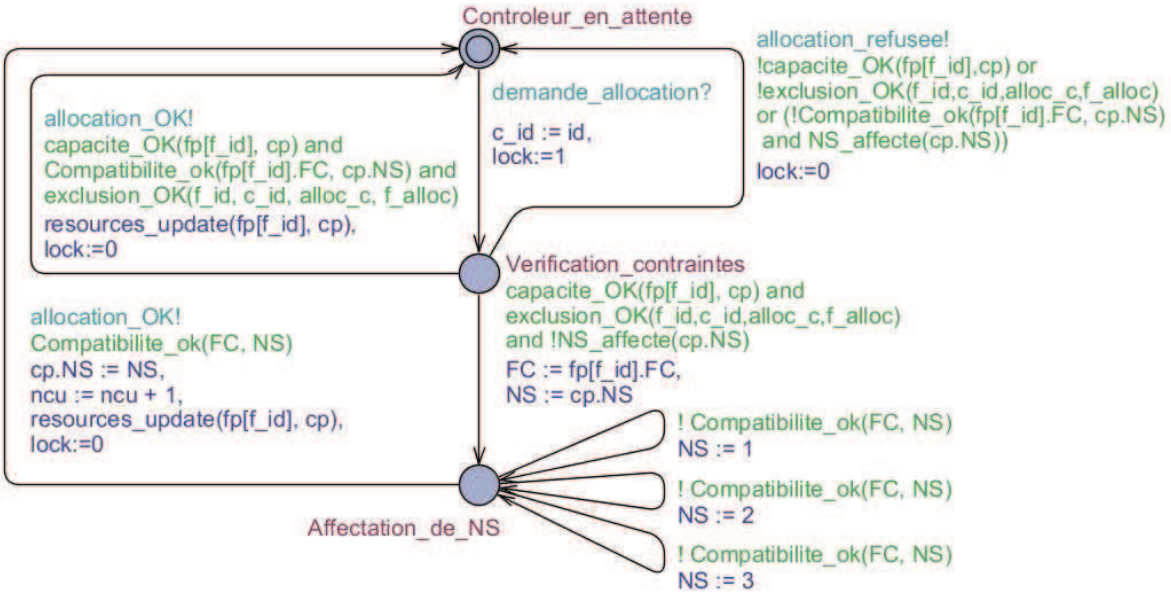


Figure 3.22 – Modèle UPPAAL d'acceptation/refus d'une demande d'allocation

3.5.2 Mise en œuvre des algorithmes sous Python

Afin de faciliter la prise en main de l'ensemble de la méthode, des scripts ont été réalisés pour permettre de n'avoir que les données d'entrée à insérer dans un tableau

excel, et de récupérer aussi les données de sortie dans un tableau excel.

3.5.2.1 Recherche d'un ensemble de solutions

Une possibilité de recherche d'un ensemble de solutions existe avec UPPAAL. L'option de recherche "random depth first" permet d'explorer aléatoirement différentes branches de l'arbre d'état. Ainsi plusieurs solutions seront données en lançant plusieurs recherches d'atteignabilité.

Il est aussi possible d'ajouter des contraintes d'exclusion dans le modèle. C'est la solution que nous avons utilisée pour créer des solutions différentes de manière automatique. En effet lors de l'utilisation du script Python présenté en annexe, des exclusions sont ajoutées entre les fonctions. A chaque exemple supplémentaire, une contrainte d'exclusion est ajoutée.

3.5.2.2 Réduction du nombre de contrôleurs

La recherche d'une solution qui réduit le nombre de contrôleurs est faite itérativement comme l'indique l'algorithme 1. Pour la première solution, le nombre de contrôleurs utilisés (ncu) est récupéré en sortie et la deuxième passe débute ainsi avec un nombre de contrôleur : $ncu - 1$.

3.6 Conclusion

Nous avons montré dans ce chapitre que le problème d'allocation de fonctions de contrôle-commande sous contraintes peut être résolu par l'utilisation de la recherche d'atteignabilité sur un ensemble de modèles avec des mécanismes d'appel/réponse. Cette approche permet d'obtenir une solution faisable, un ensemble de solutions faisables et une solution minimisant le nombre de contrôleurs. La solution faisable (ou première solution) est obtenue rapidement et le résultat en nombre de contrôleurs est très intéressant.

Cette proposition se doit d'être comparée à une approche plus classique comme celle présentée dans le chapitre 2. Sur cette base, le prochain chapitre présentera une comparaison entre cette méthode et la programmation linéaire en nombres entiers (ILP) sur un ensemble de cas d'étude.

Chapitre
4
Comparaison des deux méthodes

Sommaire

4.1 Introduction

Dans ce chapitre, nous allons comparer les deux méthodes d'allocation décrites dans ce mémoire, la programmation linéaire en nombres entiers décrite au chapitre 2 et l'approche par recherche d'atteignabilité décrite au chapitre 3. Cette comparaison est faite sur la base de deux études de cas.

4.2 Objectifs

L'objectif de cette partie est de comparer les résultats obtenus par la recherche d'atteignabilité à travers l'outil UPPAAL et par la programmation linéaire en nombres entiers (ILP) à travers CPLEX. Cette étude permet, par la même occasion, de voir comment l'approche par recherche d'atteignabilité se comporte vis-à-vis de l'explosion combinatoire lors de l'étude de cas importants.

La comparaison est réalisée expérimentalement sur 600 cas d'étude afin de voir les différences entre l'approche par Recherche d'Atteignabilité (aussi notée RA pour Reachability Analysis) et l'approche par programmation linéaire en nombres entiers (aussi notée ILP pour Integer Linear Programming). Les comparaisons sont faites sur les solutions trouvées par ces deux approches. Plusieurs paramètres sont alors analysés :

- la possibilité de trouver une première solution faisable (réalisable) ;
  - le temps nécessaire pour obtenir cette première solution faisable ;
  - le nombre de contrôleurs utilisés dans cette solution.
- la possibilité de trouver une meilleure solution ; pour cela, nous comparerons :
  - le temps nécessaire pour obtenir la meilleure solution ;
  - le nombre de contrôleurs utilisés dans la meilleure solution.

4.3 Conditions expérimentales

Les deux méthodes ont été comparées en utilisant les outils UPPAAL et CPLEX fonctionnant sur le même ordinateur avec un seul thread. Pour obtenir des résultats dans des temps raisonnables, des limites ont été fixées à la fois sur le temps de traitement

et sur la consommation de mémoire ; ces limites sont respectivement de 1200 secondes et de 12 Gbytes. Dès que l'une de ces bornes est atteinte, l'exécution est interrompue. Cette stratégie peut empêcher de trouver la solution optimale mais est nécessaire pour pouvoir effectuer un grand nombre d'expériences.

4.4 Description des études de cas

La campagne expérimentale est basée sur un ensemble de 600 cas d'étude répartis en deux ensembles :

- le premier est un ensemble de cas d'étude comprenant 20 fonctions,
- le second est un ensemble de cas d'étude comprenant 200 fonctions.

4.4.1 Étude de cas n°1 (20 fonctions)

L'ensemble des 300 cas d'étude de 20 fonctions a été généré aléatoirement c'est-à-dire que les caractéristiques de chaque fonction (les nombres d'entrées et sorties) ont été générées aléatoirement entre les valeurs 0 et 9 (incluses), d'où :
$\forall i \in \{1, ..., L = 20\}$; NI_a^i, NI_l^i, NO_a^i, $NO_l^i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^4$

En outre, l'ensemble des 300 cas d'étude a été divisé en trois sous-ensembles de 100 cas. Chaque sous-ensemble correspond à une distribution différente des facteurs de criticité des fonctions :

- le premier sous-ensemble, noté D1, correspond à une répartition homogène des facteurs de criticité. Il comprend :
  - 25% de fonctions avec leur facteur de criticité égal à 1 ($FC = 1$) ;
  - 25% de fonctions avec leur facteur de criticité égal à 2 ($FC = 2$) ;
  - 25% de fonctions avec leur facteur de criticité égal à 3 ($FC = 3$) ;
  - 25% de fonctions avec leur facteur de criticité égal à 4 ($FC = 4$).
- le deuxième sous-ensemble, noté D2, correspond à une répartition décroissante des facteurs de criticité. Il comprend :
  - 40% de fonctions avec $FC = 1$;

  - 30% de fonctions avec $FC = 2$;
  - 20% de fonctions avec $FC = 3$;
  - 10% de fonctions avec $FC = 4$.

- le troisième sous-ensemble, noté D3, correspond à une répartition gaussienne des facteurs de criticité. Il comprend :
  - 15% de fonctions avec $FC = 1$;
  - 35% de fonctions avec $FC = 2$;
  - 35% de fonctions avec $FC = 3$;
  - 15% de fonctions avec $FC = 4$.

Ces trois sous-ensembles sont représentés sur la figure 4.1 :

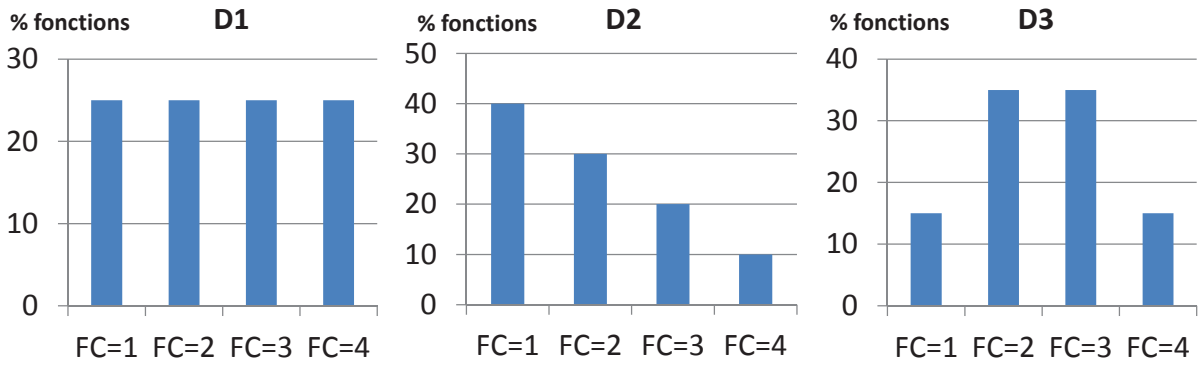


Figure 4.1 – Répartition des fonctions entre les différents facteurs de criticité dans chacun des sous-ensemble

Comme présenté dans les chapitres précédents, le nombre initial de contrôleurs est égal au nombre de fonctions (initialement $M = L$). De plus, les contrôleurs sont initialement vides, c'est-à-dire qu'ils n'accueillent aucune fonction dans l'état initial.

Dans le cadre de cette comparaison, les caractéristiques techniques prévues pour les entrées et sorties logiques ou analogiques sont les mêmes pour tous les contrôleurs.
$\forall j \in \{0, \ldots, M = L = 20\}^4$; $LI_{max} = AI_{max} = LO_{max} = AO_{max} = 32$
Le niveau de sécurité, quant à lui, n'est pas défini dans l'approche par recherche d'atteignabilité (pas de contrainte sur la distribution initiale de cette fonctionnalité). En revanche, dans le cas de l'approche par programmation linéaire en nombres entiers, il doit obligatoirement être défini au départ. Pour la programmation linéaire en nombres

entiers, les facteurs de criticité ont été distribués de façon homogène. Ainsi, pour 20 contrôleurs :

- 7 contrôleurs ont un niveau de sécurité de 1 :$NS = 1$;
- 7 contrôleurs ont un niveau de sécurité de 2 :$NS = 2$;
- 6 contrôleurs ont un niveau de sécurité de 3 :$NS = 3$.

4.4.2 Étude de cas n°2 (200 fonctions)

L'ensemble des 300 cas d'étude de 200 fonctions ont été généré aléatoirement c'est-à-dire que les caractéristiques de chaque fonction (les nombres d'entrées et sorties) ont été générées aléatoirement entre les valeurs 0 et 9 (incluses), d'où :
$\forall i \in \{1, ..., L = 200\}$; NI_a^i, NI_l^i, NO_a^i, $NO_l^i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^4$
De même que dans l'étude de cas n°1, l'ensemble des 300 cas d'étude de 200 fonctions a aussi été divisé en trois sous-ensembles de 100 cas avec des répartitions différentes des facteurs de criticité des fonctions. Les répartitions dans ces 3 sous-ensembles sont les mêmes que dans l'étude de cas n°1 et présentées dans la figure 4.1.

Les caractéristiques techniques des contrôleurs sont les mêmes que celles de l'étude de cas précédente avec :
$\forall j \in \{0, \ldots, M\}^4$ $LI_{max} = AI_{max} = LO_{max} = AO_{max} = 32$

De même que pour l'étude de cas n°1, le niveau de sécurité des 200 contrôleurs doit être défini pour l'approche de la programmation linéaire en nombres entiers. Ils sont répartis de manière homogène. Ainsi, pour 200 contrôleurs :

- 67 contrôleurs ont un niveau de sécurité de 1 :$NS = 1$;
- 67 contrôleurs ont un niveau de sécurité de 2 :$NS = 2$;
- 66 contrôleurs ont un niveau de sécurité de 3 :$NS = 3$.

4.5 Résultats de la comparaison

4.5.1 Résultats de l'ensemble des cas de 20 fonctions

Les figures suivantes présentent les résultats obtenus pour les 100 cas d'étude de 20 fonctions suivant les différents types de distribution (D1, D2, D3).

La figure 4.2 représente le temps nécessaire (en seconde) de la première solution faisable (réalisable) trouvée pour chaque cas étudié (numéro de cas en abscisse) dans le cas de la distribution D1.

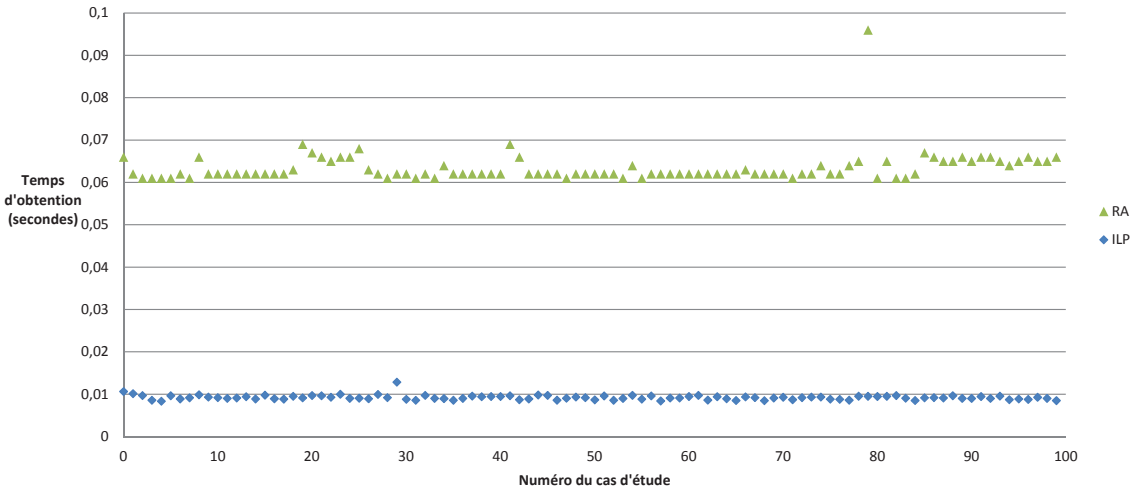


Figure 4.2 – Temps d'obtention de la première solution faisable trouvée pour 20 fonctions avec la distribution D1

La figure 4.3 représente le nombre de contrôleurs utilisés (en ordonnée) pour la première solution faisable (réalisable) trouvée de chaque cas étudié (numéro de cas en abscisse) dans le cas de la distribution D1.

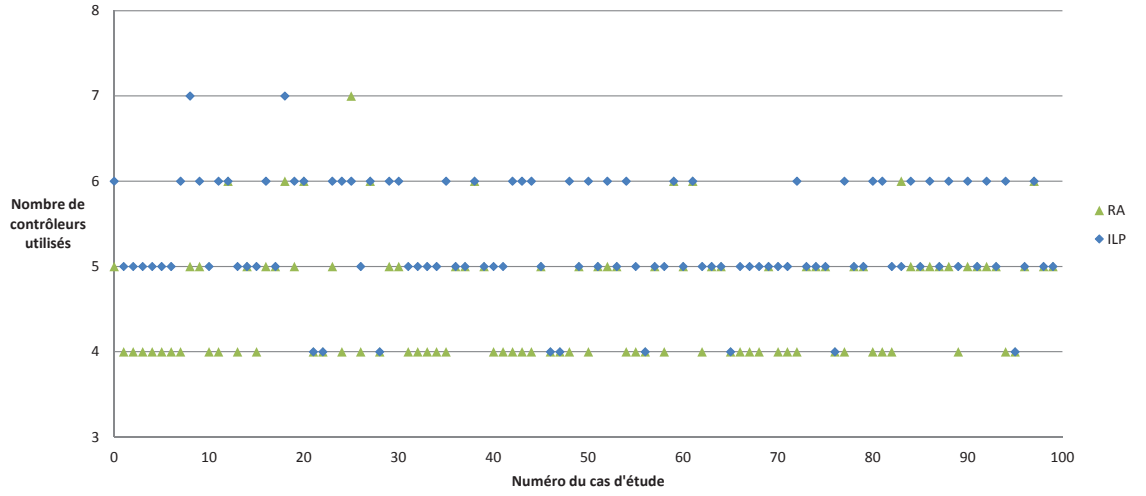


Figure 4.3 – Nombre de contrôleurs de la première solution faisable trouvée pour 20 fonctions avec la distribution D1

Les figures 4.4 et 4.5 représentent les résultats obtenus pour la première solution faisable trouvée dans le cas de la distribution D2.

Les figures 4.6 et 4.7 représentent les résultats obtenus pour la première solution faisable trouvée dans le cas de la distribution D3.

La figure 4.8 représente le nombre de contrôleurs de la meilleure solution pour chacun des cas avec la distribution D1. On peut remarquer sur la figure 4.8 que les deux

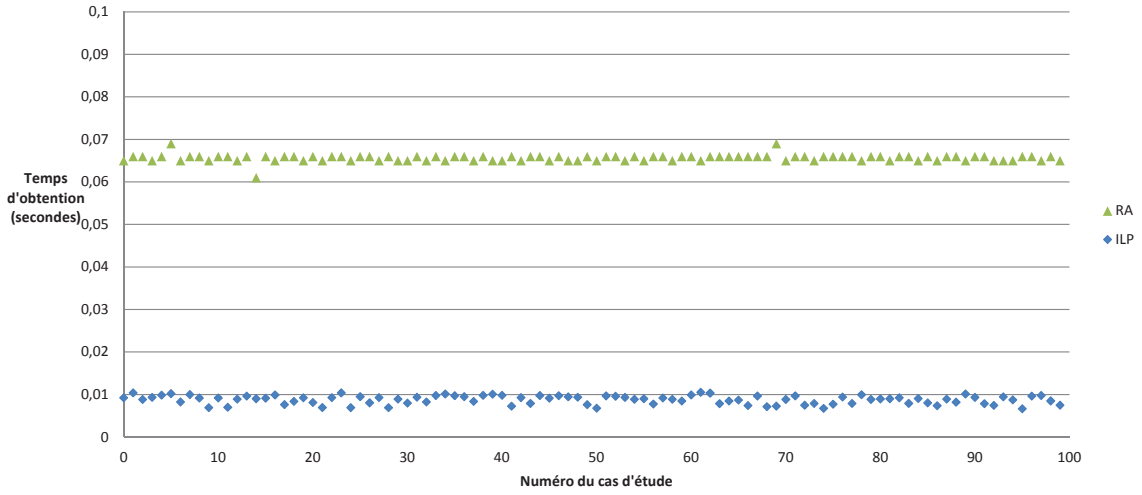


Figure 4.4 – Temps d'obtention de la première solution faisable trouvée pour 20 fonctions avec la distribution D2

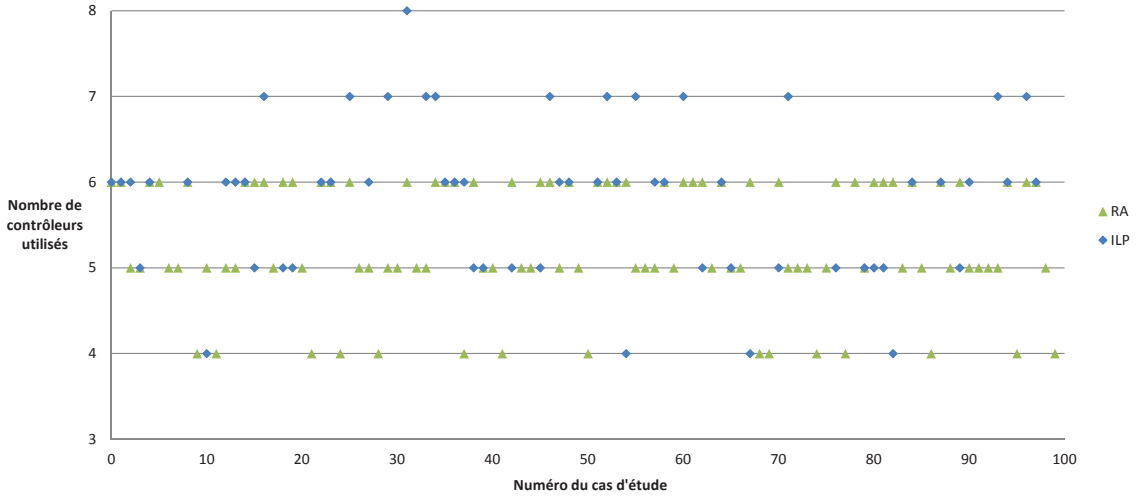


Figure 4.5 – Nombre de contrôleurs de la première solution faisable trouvée pour 20 fonctions avec la distribution D2

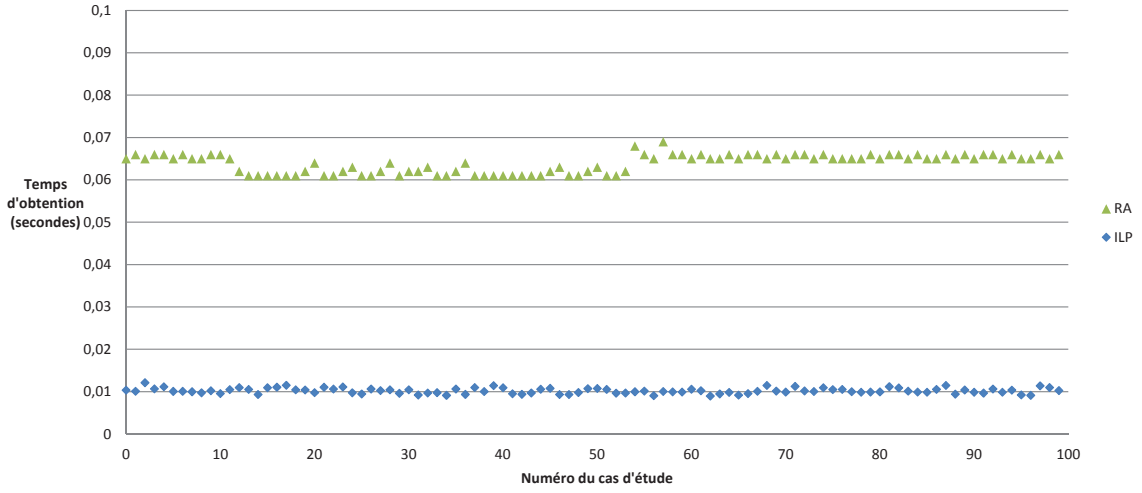


Figure 4.6 – Temps d'obtention de la première solution faisable trouvée pour 20 fonctions avec la distribution D3

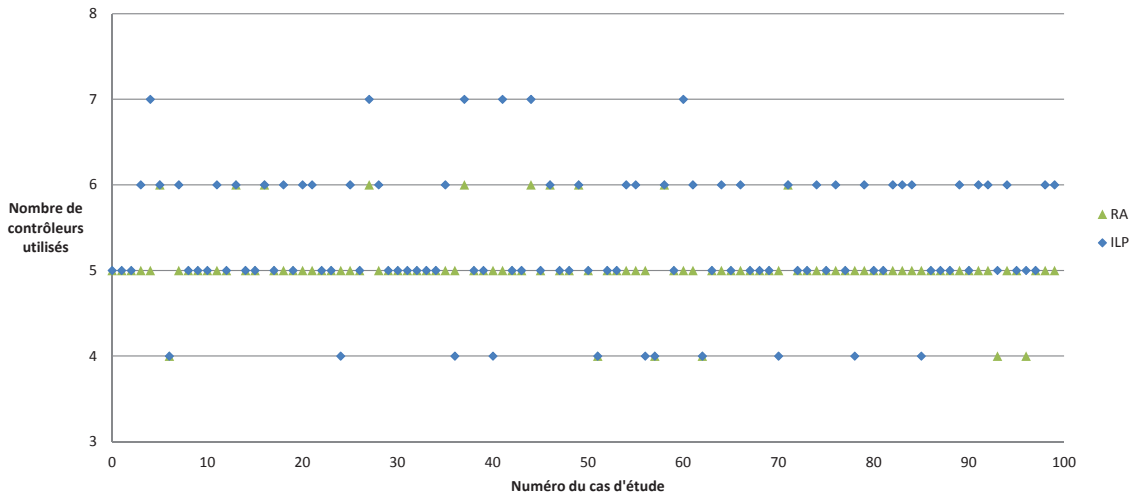


Figure 4.7 – Nombre de contrôleurs de la première solution faisable trouvée pour 20 fonctions avec la distribution D3

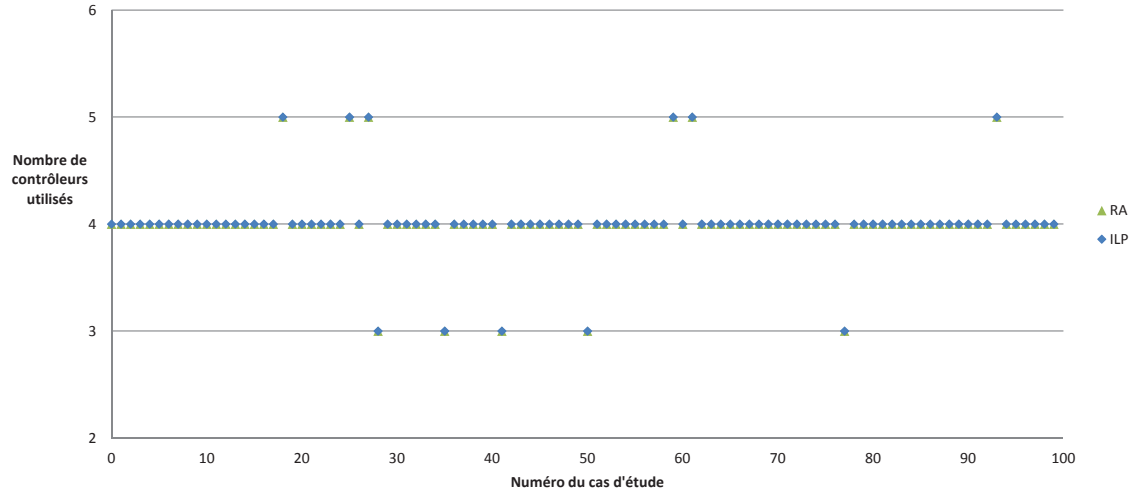


Figure 4.8 – Nombre de contrôleurs de la meilleure solution trouvée pour 20 fonctions avec la distribution D1

approches donnent des résultats identiques sur la distribution D1.

De la même façon, la figure 4.9 (respectivement 4.10) présente les résultats obtenus pour la distribution D2 (respectivement D3) qui sont à nouveau, identiques pour les deux approches.

Le tableau 4.1 présente les résultats obtenus pour les 100 cas d'étude de 20 fonctions suivant les différents types de distribution (D1, D2, D3).

Il propose une synthèse de l'ensemble des résultats présentés précédemment sous formes de figures et montre pour chaque paramètre étudié (temps d'obtention de la première solution faisable trouvée, nombre de contrôleurs de la première solution faisable trouvée, nombre de contrôleurs de la meilleure solution) :

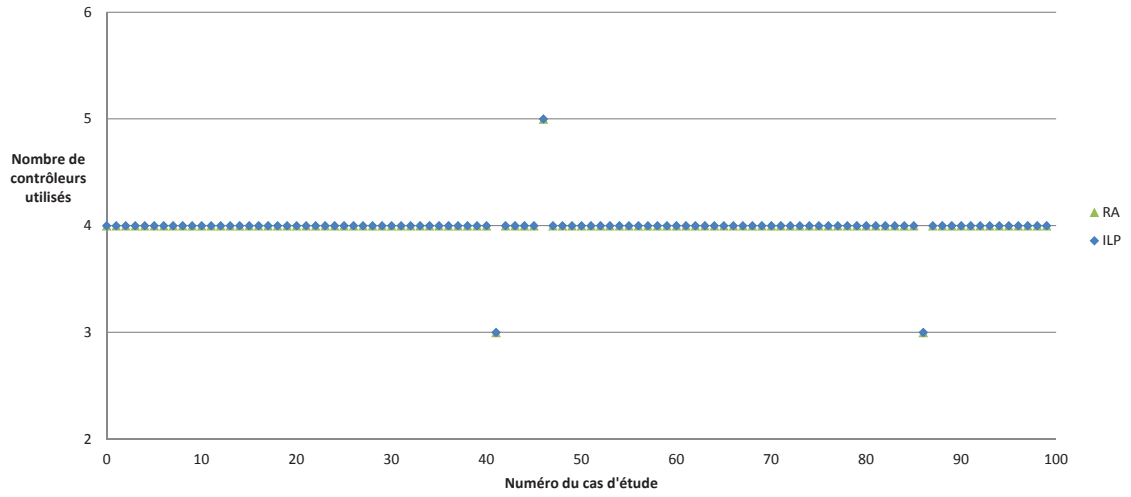


Figure 4.9 – Nombre de contrôleurs de la meilleure solution trouvée pour 20 fonctions avec la distribution D2

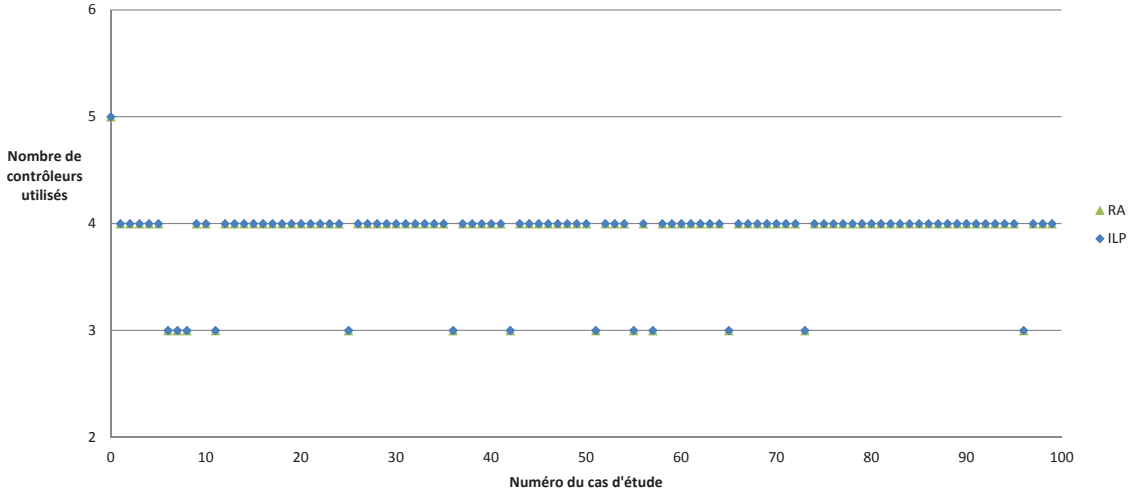


Figure 4.10 – Nombre de contrôleurs de la meilleure solution trouvée pour 20 fonctions avec la distribution D3

- le minimum obtenu sur les 100 cas ;
- la moyenne des 100 cas ;
- le maximum obtenu sur les 100 cas.

4.5.2 Résultats de l'ensemble des cas de 200 fonctions

Les figures suivantes présentent les résultats obtenus pour les 100 cas d'étude de 200 fonctions suivant les différents types de distribution (D1, D2, D3).

La figure 4.11 (respectivement 4.13,4.15) représente le temps d'obtention de la première solution faisable trouvée de chaque cas étudié avec la distribution D1 (respective-

Tableau 4.1 – Comparaison de la recherche atteignabilité (RA) et de la programmation linéaire en nombres entiers (ILP) pour 20 fonctions

Paramètres	Distribution	Type	faisable		meilleure	
			RA	ILP	RA	ILP
Temps d'obtention (secondes)	$D1$ (homogène)	Minimum	0.061	0.008	0.125	0.037
		Moyen	0.063	0.009	30.1	0.055
		Maximum	0.096	0.012	516	0.081
	$D2$ (décroissante)	Minimum	0.061	0.006	0.124	0.030
		Moyen	0.065	0.008	31.4	0.051
		Maximum	0.069	0.010	170	0.110
	$D3$ (gaussienne)	Minimum	0.061	0.008	0.158	0.042
		Moyen	0.064	0.010	147	0.065
		Maximum	0.069	0.012	490	0.167
Nombres de contrôleurs	$D1$ (homogène)	Minimum	4	4	3	3
		Moyen	5	5	4	4
		Maximum	7	7	5	5
	$D2$ (décroissante)	Minimum	4	4	3	3
		Moyen	5	11	4	4
		Maximum	6	20	5	5
	$D3$ (gaussienne)	Minimum	4	4	3	3
		Moyen	5	5	4	4
		Maximum	6	7	5	5

ment D2, D3).

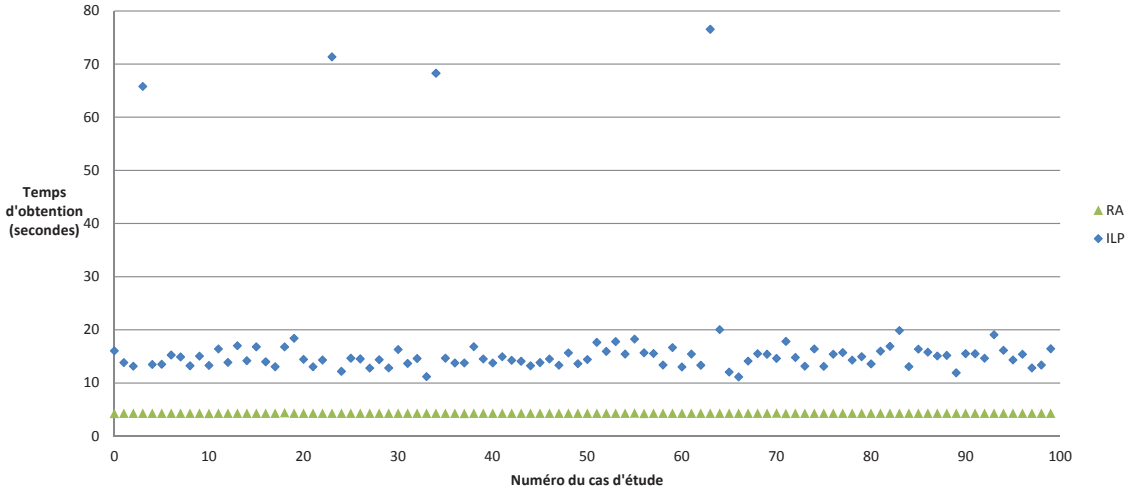


Figure 4.11 – Temps d'obtention de la première solution faisable trouvée pour 200 fonctions avec la distribution D1

La figure 4.12 (respectivement 4.14,4.16) représente le nombre de contrôleurs utilisés de la première solution faisable trouvée de chaque cas étudié avec la distribution D1 (respectivement D2, D3).

La figure 4.17 (respectivement 4.18,4.19) représente le nombre de contrôleurs utilisés

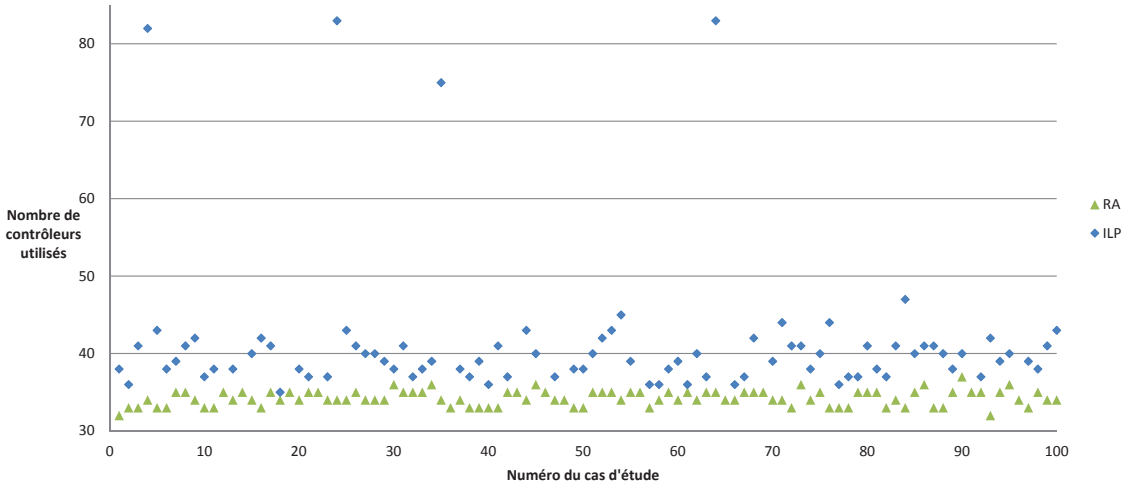


Figure 4.12 – Nombre de contrôleurs de la première solution faisable trouvée pour 200 fonctions avec la distribution D1

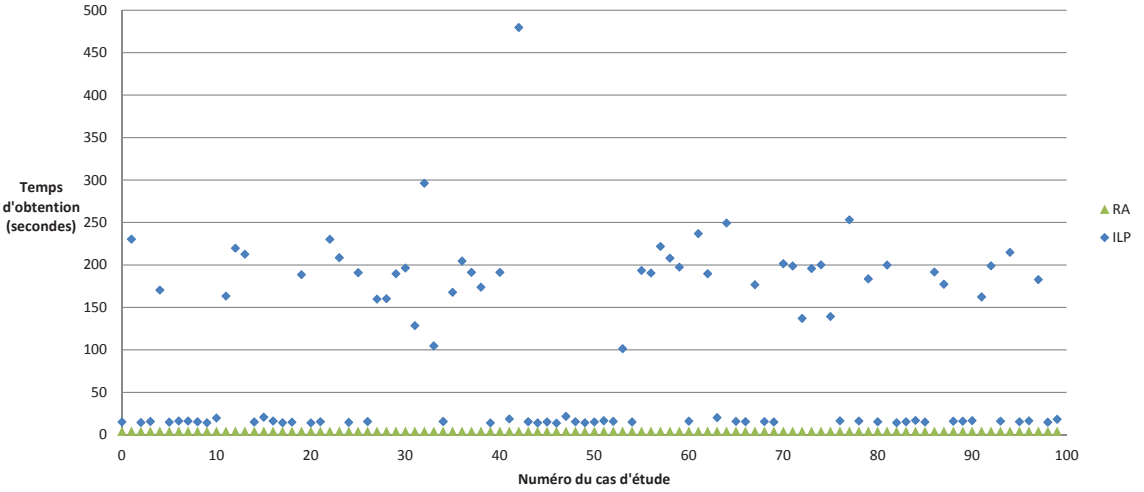


Figure 4.13 – Temps d'obtention de la première solution faisable trouvée pour 200 fonctions avec la distribution D2

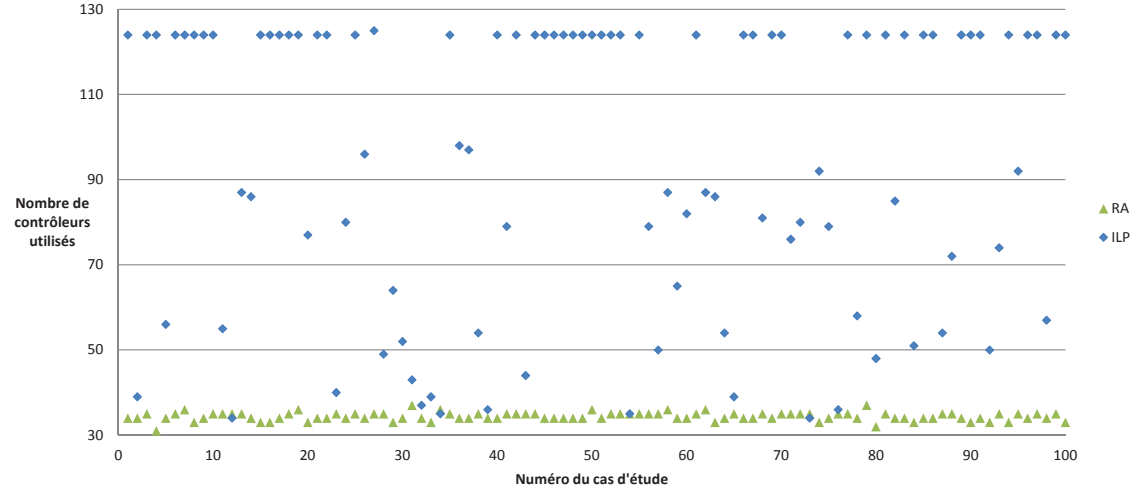


Figure 4.14 – Nombre de contrôleurs de la première solution faisable trouvée pour 200 fonctions avec la distribution D2

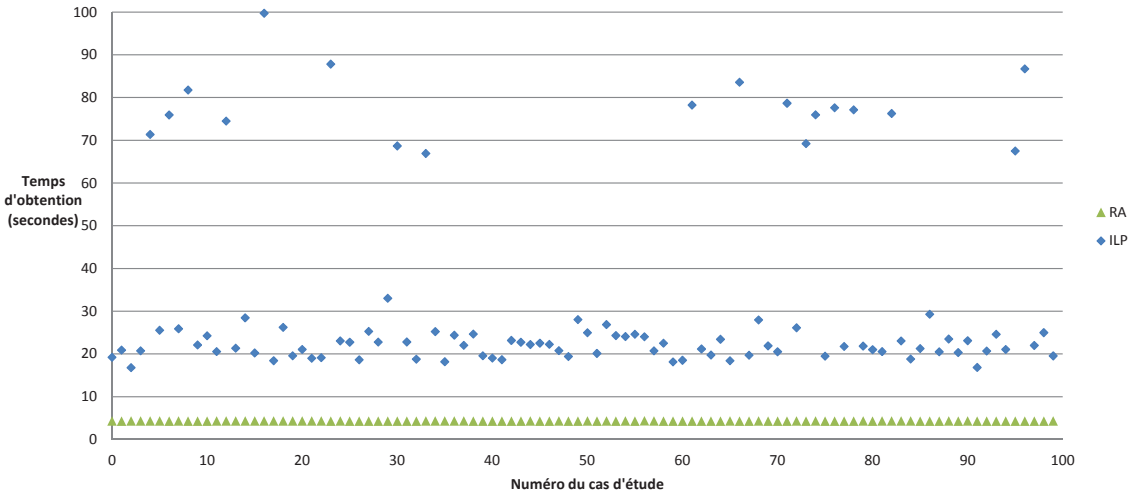


Figure 4.15 – Temps d'obtention de la première solution faisable trouvée pour 200 fonctions avec la distribution D3

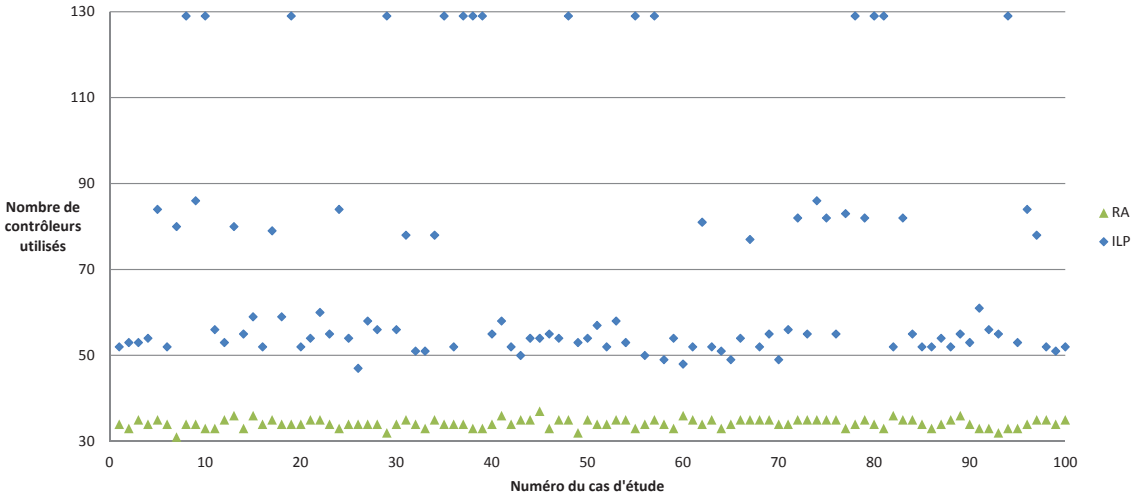


Figure 4.16 – Nombre de contrôleurs de la première solution faisable trouvée pour 200 fonctions avec la distribution D3

de la meilleure solution trouvée pour chacun des cas avec la distribution D1 (respectivement D2, D3).

Le tableau 4.2 présente les résultats obtenus pour les 100 cas d'étude de 200 fonctions suivant les différents types de distribution (D1, D2, D3).

Il propose une synthèse de l'ensemble des résultats présentés précédemment sous formes de figures et montre pour chaque paramètre étudié (temps d'obtention de la première solution faisable trouvée, nombre de contrôleurs de la première solution faisable trouvée, nombre de contrôleurs de la meilleure solution) :

- le minimum obtenu sur les 100 cas ;
- la moyenne des 100 cas ;

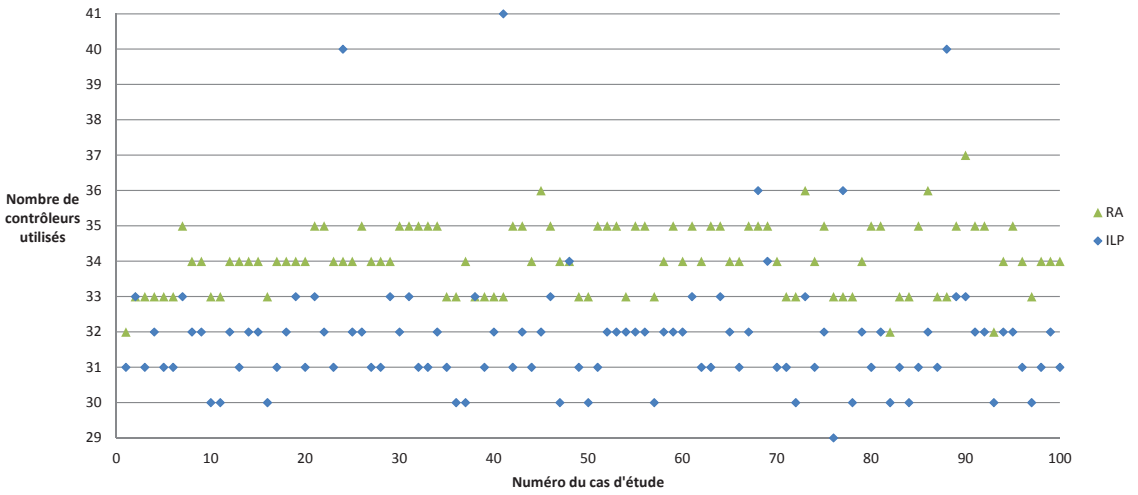


Figure 4.17 – Nombre de contrôleurs de la meilleure solution trouvée pour 200 fonctions avec la distribution D1

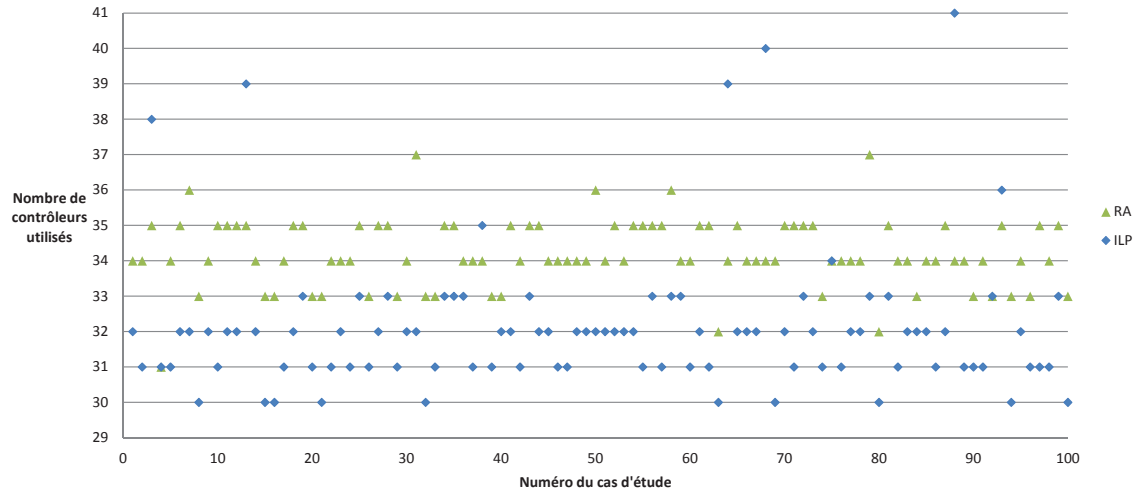


Figure 4.18 – Nombre de contrôleurs de la meilleure solution trouvée pour 200 fonctions avec la distribution D2

- le maximum obtenu sur les 100 cas.

4.6 Discussion

4.6.1 Discussion sur l'étude de cas n°1 (20 fonctions)

Les résultats des cas d'étude de 20 fonctions sont donnés assez rapidement par les deux méthodes pour la première solution trouvée (temps d'obtention de l'ordre du centième de secondes). L'approche par programmation linéaire en nombres entiers est un peu plus rapide avec des temps d'environ 1 centième de secondes contre environ 7 centièmes pour la recherche d'atteignabilité.

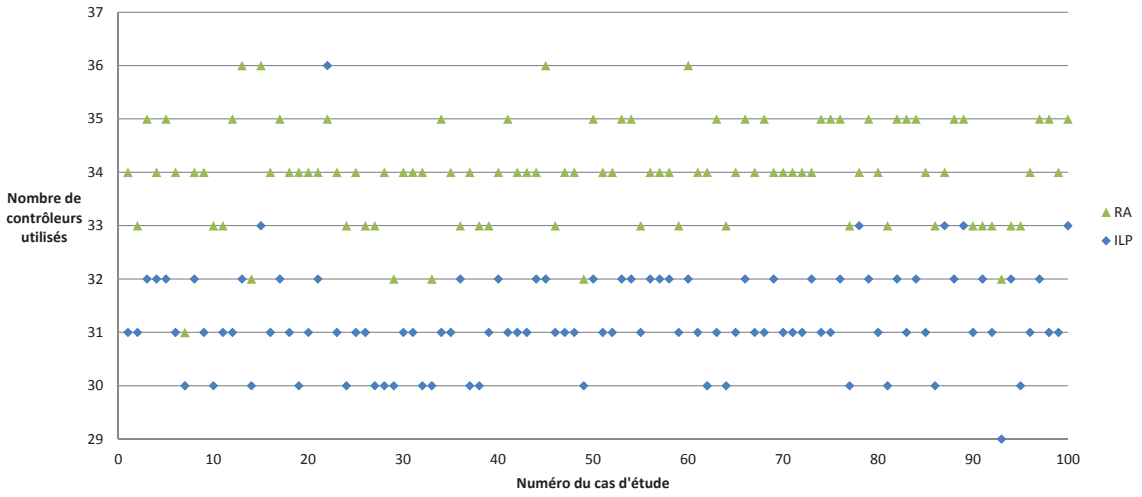


Figure 4.19 – Nombre de contrôleurs de la meilleure solution trouvée pour 200 fonctions avec la distribution D3

Tableau 4.2 – Comparaison de la recherche atteignabilité (RA) et de la programmation linéaire en nombres entiers (ILP) pour 200 fonctions

Paramètres	Distribution	Type	faisable		meilleure	
			RA	ILP	RA	ILP
Temps d'obtention (secondes)	$D1$ (homogène)	Minimum	4,3	11	400	460
		Moyen	4,3	17	600	1100
		Maximum	4,4	77	770	1200
	$D2$ (décroissante)	Minimum	4,3	14	300	440
		Moyen	4,3	100	590	1150
		Maximum	4,4	480	730	1200
	$D3$ (gaussienne)	Minimum	4,2	17	410	490
		Moyen	4,3	32	600	1100
		Maximum	4,3	100	1100	1200
Nombres de contrôleurs	$D1$ (homogène)	Minimum	32	35	32	29
		Moyen	34	52	34	32
		Maximum	37	120	37	41
	$D2$ (décroissante)	Minimum	31	34	31	30
		Moyen	34	94	34	32
		Maximum	37	125	37	41
	$D3$ (gaussienne)	Minimum	31	47	31	29
		Moyen	34	70	34	31
		Maximum	37	129	36	36

Le nombre de contrôleurs dans la première solution faisable trouvée est identique dans les deux approches pour les distribution D1 et D3 entre les deux approches (environ 5 contrôleurs). En revanche, pour la distribution D2, l'approche par programmation linéaire en nombres entiers donne des résultats moins intéressants que l'approche par

recherche d'atteignabilité (11 contrôleurs en moyenne pour la programmation linéaire en nombres entiers contre 5 seulement pour recherche d'atteignabilité).

Le nombre de contrôleurs dans la meilleure solution sont identiques dans les deux approches quelque soit la distribution, il est même possible de remarquer sur les figures 4.8, 4.9 et 4.10 que les points se superposent.

Par contre, la meilleure solution est trouvée plus rapidement avec la programmation linéaire en nombres entiers qu'avec l'approche par recherche d'atteignabilité même si les résultats trouvés sont les mêmes avec les deux méthodes.

Les résultats de cette étude de cas n°1 sont de même ordre pour les deux approches mais ne sont pas représentatifs de cas réels, nous allons donc discuter des résultats de l'étude de cas n°2, plus proche des cas industriels.

4.6.2 Discussion sur l'étude de cas n°2 (200 fonctions)

La première solution des cas d'étude de 200 fonctions est donnée assez rapidement par la recherche d'atteignabilité. En effet, dans tous les cas testés, quelque soit la distribution des fonctions, le temps d'obtention de la première solution varie entre 4.3 et 4.4 secondes pour l'approche par recherche d'atteignabilité alors qu'il varie entre 11 et 480 secondes pour l'approche par programmation linéaire en nombres entiers, avec une moyenne de 17 secondes pour la distribution $D1$, 100 secondes pour la distribution $D2$ et 32 secondes pour la distribution $D3$. Les figures 4.11, 4.13, 4.15 montrent bien la différence de temps pour obtenir la première solution avec les deux approches.

Le nombre de contrôleurs de la première solution trouvée est aussi meilleure par la recherche d'atteignabilité que par la programmation linéaire en nombres entiers. En effet, dans tous les cas testés, quelque soit la distribution des fonctions, le nombre de contrôleurs de la première solution trouvée varie entre 31 et 37 pour l'approche par recherche d'atteignabilité alors qu'il varie entre 34 et 129 pour l'approche par programmation linéaire en nombres entiers, avec une moyenne de 52 pour la distribution $D1$, 94 pour la distribution $D2$ et 70 pour la distribution $D3$. Les figures 4.12, 4.14, 4.16 montrent bien la différence du nombre de contrôleurs dans la première solution trouvée avec les deux approches.

En revanche, la programmation linéaire en nombres entiers est plus efficace pour la recherche de la meilleure solution par rapport à l'approche par recherche d'atteignabilité

avec un écart de deux contrôleurs en moyenne entre les deux approches pour les distributions $D1$ et $D2$ et un écart de trois contrôleurs en moyenne pour la distribution $D3$. Les figures 4.17, 4.18, 4.19 montrent les résultats en terme de nombre de contrôleurs trouvés par les deux approches. On peut remarquer sur ces figures que l'approche par programmation linéaire ne trouve pas toujours la meilleure solution. En effet il existe certains cas où l'approche par recherche d'atteignabilité trouve une meilleure solution.

L'obtention de la meilleure solution est, en général, obtenue en atteignant les bornes temporelles ou de mémoire imposées lors de la comparaison. En effet, la recherche d'atteignabilité atteignait la limite en terme de mémoire et la programmation linéaire en nombres entiers atteignait la limite de temps.

4.7 Conclusion

Le temps nécessaire pour obtenir la première solution est toujours plus petit (et parfois de un ou deux ordres de grandeur) avec la recherche d'atteignabilité. La première solution faisable trouvée par la recherche d'atteignabilité est proche de la meilleure solution trouvée, quelque soit la distribution en entrée. Le temps pour obtenir la meilleure solution est quant à lui approximativement la même pour les deux procédés. La meilleure solution obtenue avec la programmation linéaire utilise un plus petit nombre de contrôleurs (alors que la solution obtenue avec la recherche d'atteignabilité n'est pas si loin de cette solution ($< 10\%$)).

Une possibilité de couplage de ces deux approches pourrait donner des résultats intéressants en interfaçant ces deux approches dans le domaine où elles sont les meilleures. La recherche d'atteignabilité pourrait donc être utilisée pour trouver une première solution faisable qui serait ensuite traitée en la programmation linéaire afin de trouver la meilleure solution possible. Quelques cas ont été repris de cette comparaison et exécutés pour voir si cette perspective était intéressante. Cette approche mixte en passant d'abord par la recherche d'atteignabilité puis par la programmation linéaire en nombres entiers a donné des résultats permettant d'arriver plus rapidement à des solutions meilleures que celles trouvées par la programmation linéaire. Ce couplage devrait être poursuivi pour voir les résultats qui peuvent en découler sur l'ensemble des cas.

Pour résumer, la programmation linéaire est la meilleure méthode pour trouver

le nombre minimum de contrôleurs sans pour autant être facilement modifiable lors d'ajouts de contraintes à cause de la linéarisation des contraintes. En revanche, la recherche d'atteignabilité est une solution efficace pour trouver rapidement une solution proche de la meilleure solution et permet de prendre en compte facilement des ajouts de contraintes, qui fera l'objet chapitre suivant.

Chapitre 5
Cas d'application

Sommaire

5.1 Introduction

Le chapitre 5 présente la capacité de l'approche proposée par recherche d'atteignabilité à prendre en compte de nouvelles contraintes d'allocation. En effet, au fur et à mesure de la conception d'architecture opérationnelle, l'architecture organique devient connue de plus en plus finement, ce qui implique, dans notre cas, des ajouts de contraintes. Les possibilités de passage à l'échelle et l'évolutivité de l'approche par recherche d'atteignabilité ayant été démontrées, un méta-modèle permettant d'intégrer cette méthode dans un atelier de conception d'architecture de contrôle-commande est proposé.

5.2 Étude de cas industrielle

Dans le cadre de l'allocation de fonctions sous contraintes (présenté dans le chapitre 1), la méthode par recherche d'atteignabilité (développée dans le chapitre 3) s'applique à un niveau donné de granularité de conception de l'architecture opérationnelle. Cette méthode a été développée dans le but d'être facilement adaptable et de pouvoir prendre en compte des ajouts pour la modélisation du processus d'allocation. Au début de la phase de conception, la description de l'architecture organique est limitée. Cette description devient de plus en plus fine au cours de la conception, engendrant des modifications dans le processus de conception et des possibilités d'ajouts de contraintes ou de modifications de données. Nous allons maintenant traiter un exemple où les contraintes ci-dessous ont dû être ajoutées :

- une limitation technique a été ajoutée aux modèles précédents afin de prendre en compte une restriction de l'allocation. En effet, un contrôleur ne peut avoir au maximum que 32 connexions avec d'autres contrôleurs, il faut donc empêcher toute allocation où un contrôleur pourrait avoir plus de 32 connexions avec d'autres contrôleurs.

- lors de la conception de systèmes tels que les systèmes de contrôle-commande des centrales nucléaires, nous avons vu dans le chapitre 1 qu'une phase de visite décennale existe dans le cycle de vie en "W" d'EDF. Cette phase engendre aussi une partie d'allocation pour toutes les fonctions qui ont été modifiées et/ou ajoutées lors de cette remise à niveau du système. Il faut donc que la ré-allocation ou

allocation partielle soit possible. Cette allocation part d'un état où une partie des fonctions est déjà allouée et où l'autre partie doit l'être.

Pour prendre en compte ces deux contraintes, un nouveau cas d'étude est modélisé. La description des fonctions, des contrôleurs et des contraintes est donc reprise par la suite.

5.2.1 Nouvelle description des fonctions

Dans cet exemple, 1200 fonctions sont présentes et sont notées f^i avec $i \in \{1, ..., 1200\}$. La fonction présentée dans le chapitre 1 évolue dans cet exemple et ses caractéristiques deviennent :

- $NESf^i \in \mathbb{N}$ nombre d'entrées / sorties physiques de la fonction f^i, d'où : $NESf^i = NEf^i + NSf^i$ avec :
  - NEf^i nombre d'entrées physiques de la fonction f^i correspondant à la somme des anciennes valeurs logiques et analogiques :
    $NEf^i = NI_l^i + NI_a^i$;
  - NSf^i nombre de sorties physiques de la fonction f^i correspondant à la somme des anciennes valeurs logiques et analogiques :
    $NSf^i = NO_l^i + NO_a^i$.
- FC^i, facteur de criticité de la fonction f^i tel que : $FC^i \in \{1, 2, 3\}$ dans ce cas.
- $\{f^k, f^l, f^m, ...\}$ liste des fonctions avec lesquelles la fonction f^i échange des informations. La matrice de connexions entre fonctions ou matrice d'échange entre fonctions (E) est ainsi créée, et s'exprime :
  $E : F \times F \longrightarrow \mathbb{N}$

$$E = \begin{pmatrix} e_{(1,1)} & \cdots & e_{(1,L)} \\ \vdots & \ddots & \vdots \\ e_{(L,1)} & \cdots & e_{(L,L)} \end{pmatrix}$$

$\forall (i, k) \in \{1, ..., L\}^2$

$$e_{(i,k)} = \begin{cases} 1 & \text{si il existe une ou plusieurs connexions entre } f^i \text{ et } f^k \\ 0 & \text{sinon} \end{cases} \quad (5.1)$$

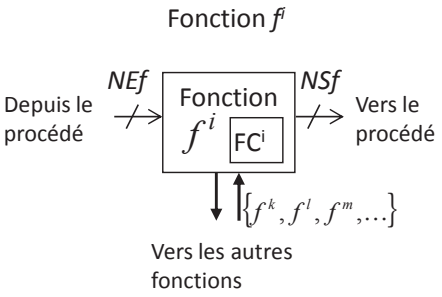


Figure 5.1 – Représentation d'une fonction dans le cas industriel

- "$f\_alloc[id]$", variable permettant de savoir si la fonction a déjà été allouée.
  $\forall id \in \{1, ..., L\}$

$$f\_alloc[id] = \begin{cases} 1 & \text{si la fonction } f^{id} \text{ est allouée} \\ 0 & \text{sinon} \end{cases} \tag{5.2}$$

5.2.2 Nouvelle description des contrôleurs

Le contrôleur présenté dans le chapitre 1 évolue dans cet exemple et ses caractéristiques deviennent :

- $NESc^j \in \mathbb{N}$ nombre d'entrées / sorties proposés par le contrôleur c^j d'où : $NESc^j = NEc^j + NSc^j$ avec :
  - NEc^j nombre d'entrées physiques du contrôleur c^j, correspondant à la somme des anciennes valeurs logiques et analogiques :
    $NEc^i = LI^j_{max} + AI^j_{max}$;
  - NSc^j nombre de sorties physiques du contrôleur c^j, correspondant à la somme des anciennes valeurs logiques et analogiques :
    $NSc^i = LO^j_{max} + AO^j_{max}$.
- NS^j niveau de sécurité tel que : $NS^j \in \{1, 2, 3\}$
- des connexions entre contrôleurs sont nécessaires dans ce cas d'étude. La matrice de connexions entre contrôleurs (MC) s'exprime donc :
  $MC : C \times C \longrightarrow \mathbb{B}$
  $\forall (j, l) \in \{1, ..., M\}^2$

$$mc_{(j,l)} = \begin{cases} 1 & \text{si il existe une connexion entre } c^j \text{ et } c^l \\ 0 & \text{sinon} \end{cases} \tag{5.3}$$

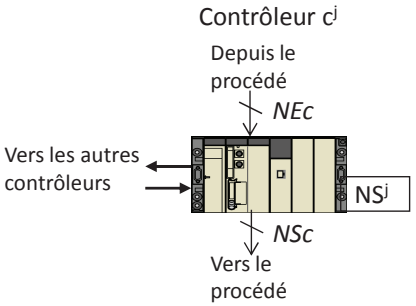


Figure 5.2 – Représentation graphique d'un contrôleur

5.2.3 Contraintes prises en compte

Dans le chapitre 1, nous avons présenté un ensemble de contraintes prises en compte dans le cas général. Nous allons détailler ici les contraintes utilisées pour l'exemple industriel.

5.2.3.1 Contraintes de capacité

Dans le cas industriel, les contraintes de capacité se définissent en une seule contrainte correspondant au respect du nombre d'entrées / sorties proposés par le contrôleur et notée :

$$f^i \rightsquigarrow c^j \text{ si } \sum_{i \in I_j} NESf^i \leq NESc^j \tag{5.4}$$

5.2.3.2 Contraintes de sûreté

Deux contraintes de sûreté sont mises en place dans l'exemple traité. Elles sont identiques sur le principe à celles présentées dans le chapitre 1, à savoir :

- la contrainte de compatibilité 1.6, cette contrainte est légèrement modifiée, la matrice X a évolué et est spécifiée ci-dessous :

$$f^i \rightsquigarrow c^j \text{ si } x_{(FC^i, NS^j)} = 1$$

La matrice X peut alors s'écrire :

$$X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- la contrainte d'exclusion 1.8 correspond à la relation entre deux fonctions f^i et f^k qui ne peuvent être allouées dans le même contrôleur. Cette relation est représentée par une matrice d'exclusion Y :
  $Y : F \times F \longrightarrow \mathbb{B}$
  Une fonction f^i pourra donc être allouée à un contrôleur c^j contenant déjà la fonction f^k si et seulement si la relation entre f^i et f^k est respectée :
  $\forall i \in \{1, ..., L\}$,$\forall k \in \{1, ..., L\}$ et $\forall j \in \{1, ..., M\}$:

$$(f^i \rightsquigarrow c^j) \text{ et } (f^k \rightsquigarrow c^j) \text{ si } y_{(f^i, f^k)} = 1$$

  La matrice Y représentant cette équation peut alors s'écrire :

$$Y = \begin{pmatrix} 1 & y_{(f^1,f^2)} & \cdots & y_{(f^1,f^L)} \\ y_{(f^2,f^1)} & 1 & \ddots & \vdots \\ \vdots & \cdots & 1 & y_{(f^{L-1},f^L)} \\ y_{(f^L,f^1)} & \cdots & y_{(f^L,f^{L-1})} & 1 \end{pmatrix}$$

5.2.3.3 Contrainte technique

Une contrainte technique a du être ajoutée pour cette étude pour tenir compte du fait que l'architecture organique est connue de plus en plus finement. Chaque contrôleur ne peut envoyer et recevoir des informations que de 32 contrôleurs maximum. Il faut donc mettre en place une contrainte limitant les connexions entre contrôleurs.

Cette contrainte sur la limite de connexions entre contrôleurs s'exprime donc :
$\forall (j, l) \in \{1, ..., M\}^2$, $\forall i \in \{1, ..., L\}$

$$f^i \rightsquigarrow c^j \text{ si } \sum_{l \in \{1,...,M\}} mc_{(j,l)} \leq 32 \tag{5.5}$$

5.2.4 Modifications des modèles

Afin de prendre en compte ces contraintes et la possibilité de ré-allocation, les modèles ont dû être modifiés. Les deux nouveaux modèles génériques de demande d'allocation et d'acceptation/refus d'allocation sont présentés ci-dessous.

5.2.4.1 Modèle de demande d'allocation

Le modèle de demande d'allocation est présenté à la figure 5.3. Ce modèle est complété par rapport au modèle de la figure 3.10 par l'ajout d'un arc entre la localité "Fonction non allouée" et la localité "Fonction allouée" avec comme garde : $f\_alloc[id] = 1$ qui représente l'état de la fonction. En effet, une fonction déjà allouée ne doit pas être considérée lors de l'allocation.

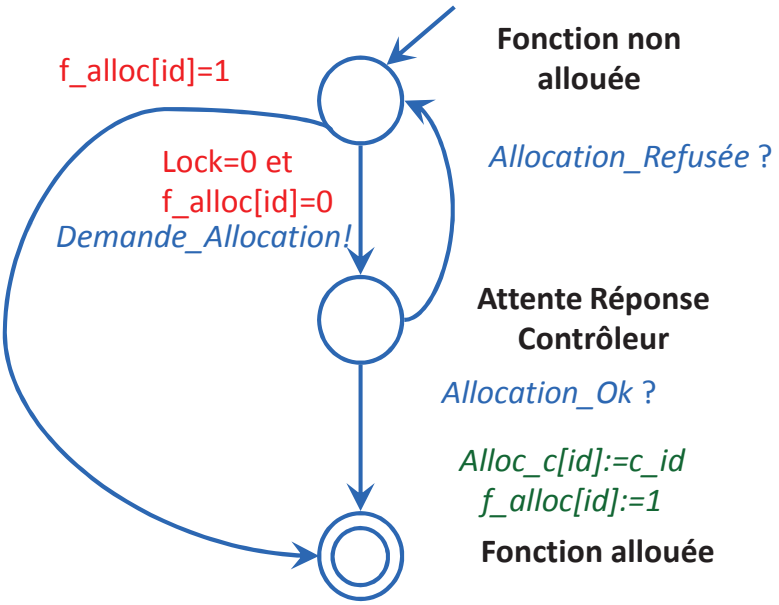


Figure 5.3 – Modèle de demande d'allocation après modification

5.2.4.2 Modèle d'acceptation/refus d'une demande d'allocation

Le modèle d'acceptation/refus d'une demande d'allocation a été modifié par rapport au modèle de la figure 3.11 pour prendre en compte l'ensemble des contraintes. Celles-ci sont respectées grâce aux gardes et aux mises à jour du modèle comme le montre la figure 5.4.

Étant donné que la contrainte de compatibilité se traduit par une matrice diagonale ce modèle peut être simplifié. Les changements apportés sont présentés à la figure 5.5, ils permettent d'enlever la localité "Affectation de NS" et de mettre à jour NS de la manière suivante $NS := FC$

Une fois l'ensemble des contraintes et des modèles mis à jour dans le logiciel, la recherche d'atteignabilité reste la même. Nous recherchons un état où toutes les fonctions sont allouées. Ce cas d'étude a permis de trouver une solution faisable d'allocation de

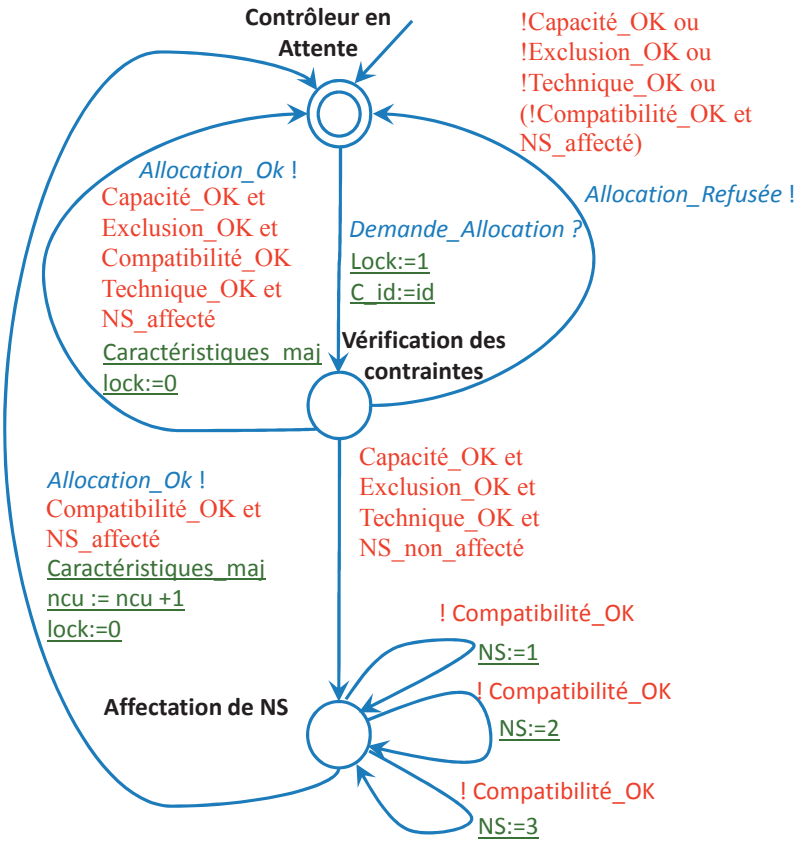


Figure 5.4 – Modèle d'acceptation/refus d'une demande après modification

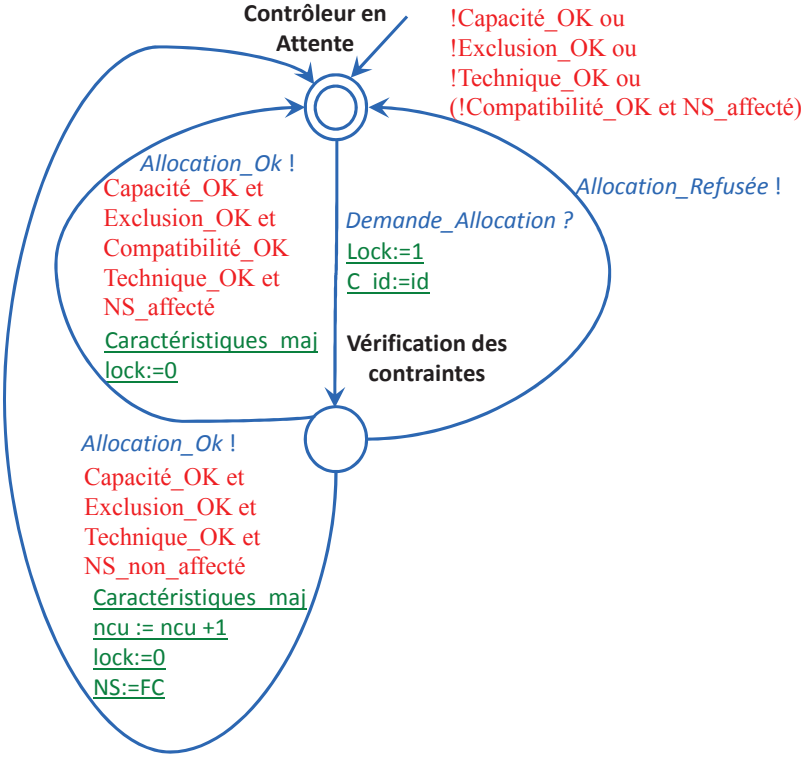


Figure 5.5 – Modèle simplifié d'acceptation/refus d'une demande après modification

l'ensemble des fonctions.

5.2.5 Synthèse

L'approche de modélisation par appel/réponse couplée à la recherche d'une propriété d'atteignabilité a permis de traiter ce cas industriel. Ce cas a été modélisé à partir du modèle présenté au chapitre 3 en modifiant des gardes et ajoutant un arc supplémentaire. Ces modifications sont assez simples à mettre en place.

5.3 Représentation sous forme de méta-modèle

L'outil UPPAAL et les scripts Python offrent un environnement permettant de générer un ensemble d'architectures satisfaisant des contraintes de capacité et de sûreté (contrainte d'exclusion, contraintes de compatibilité). Dans le cadre du projet CONNEXION faisant suite à ce travail de thèse, cet ensemble outillé, dénommé ATHENA, constitue une brique de base [Lemattre *et al.*, 2011c] (parties en gras de la figure 5.6) qui sera enrichie en intégrant :

- les performances temporelles, et en particulier les temps de réponse, de cascades de fonctions réparties sur différents contrôleurs ; cette dimension inclut les contraintes de charge sur les médium permettant la communication entre contrôleurs,
- les attributs de sûreté (fiabilité, disponibilité) globaux des architectures opérationnelles générées.

Ces critères supplémentaires devront être évalués lors ou à l'issue du processus d'allocation à l'aide de modèles temporisés et/ou stochastiques. Pour préparer cette extension et faciliter l'intégration de notre outil/méthode à d'autres environnements utilisés par notre partenaire industriel pour la vérification de performances ou les études probabilistes, il nous est apparu opportun de développer un méta-modèle permettant de capitaliser les concepts généraux relatifs à nos architectures ainsi que la formalisation des contraintes. Le premier point a été modélisé sous la forme d'un diagramme de classe UML tandis que le second point a été représenté sous la forme de contraintes OCL.
Ce méta-modèle reprend les différentes parties présentées précédemment afin de pouvoir utiliser les instances du méta-modèle comme données du problème d'allocation sous

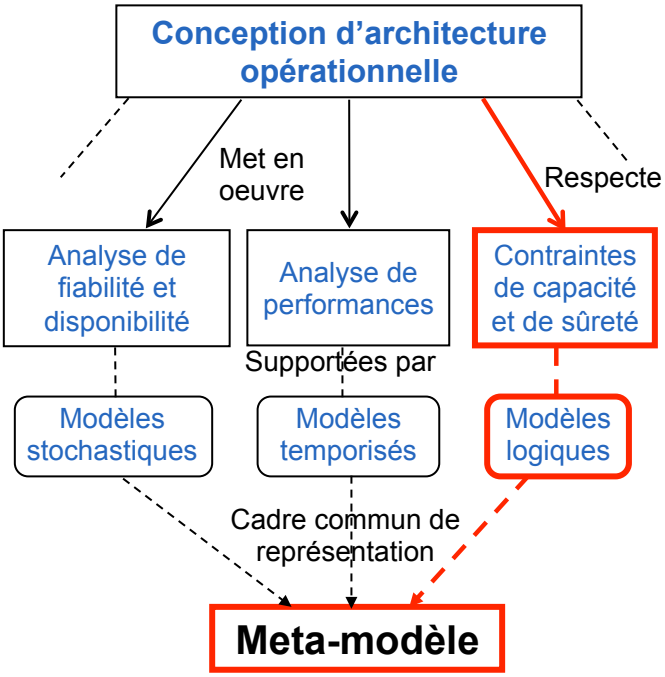


Figure 5.6 – Extensions de la brique ATHENA (Projet Connexion)

contraintes et devra être enrichi en fonction des domaines d'applications utilisateurs.

5.3.1 L'ensemble des fonctions

L'architecture fonctionnelle est un ensemble de fonctions de commande inter-connectées qui expriment les besoins du système de contrôle-commande. La modélisation de cette architecture est représentée sur la figure 5.7.

Cette architecture est composée de :

- fonctions correspondant aux fonctions de service du système de contrôle-commande ;
- données venant du procédé. Ces données sont de types logiques ou analogiques. Elles sont donc réparties en :
  - données d'entrée d'une fonction, elles viennent directement du procédé ;
  - données de sortie d'une fonction, elles vont directement au procédé .

Dans le méta-modèle, les fonctions sont connectées au procédé par des données d'entrée ou des données de sortie. Ces données sont représentées par un arc liant la classe "fonction" à "donnée d'entrée" ou "donnée de sortie". Le lien est pondéré à son entrée et sa sortie, représentant le fait qu'une fonction peut avoir plusieurs données d'entrée mais une donnée d'entrée n'est liée qu'à une seule fonction.

Il existe le même lien entre la classe "fonction" et "données de sortie" ; ce lien permet à une fonction d'avoir plusieurs données de sortie mais une donnée de sortie ne peut être liée qu'à une seule fonction.

Par conséquent, si une information venant du procédé doit aller à deux endroits différents (fonctions différentes), il faut alors représenter deux données d'entrée différentes.

La classe "fonction" est composée d'attributs :

- NILp, entier représentant le nombre d'entrées logiques venant du procédé (équivalent à NI_l) ;
- NOLp, entier représentant le nombre de sorties logiques allant vers le procédé (équivalent à NO_l) ;
- NIAp, entier représentant le nombre d'entrées analogiques venant du procédé (équivalent à NI_a) ;
- NOAp, entier représentant le nombre de sorties analogiques allant vers le procédé (équivalent à NO_a) ;
- i, entier représentant l'indice de la fonction ;
- FC, représentant le facteur de criticité de la fonction. Il est énuméré parmi les valeurs définies dans la formalisation du problème $FC \in \{1, ..., r\}$ avec $r \in \mathbb{N}^* - \{1\}$ et dans notre cas $FC \in \{1, 2, 3, 4\}$.

Les données d'entrée, qui viennent du procédé, peuvent être de deux types :

- les entrées logiques, représentant les remontées des capteurs TOR ;
- les entrées analogiques, représentant les remontées des autres capteurs.

Les données de sortie, qui vont vers le procédé, peuvent aussi être de deux types :

- les sorties logiques, représentant les actions sur les actionneurs TOR ;
- les sorties analogiques, représentant les actions sur les autres actionneurs.

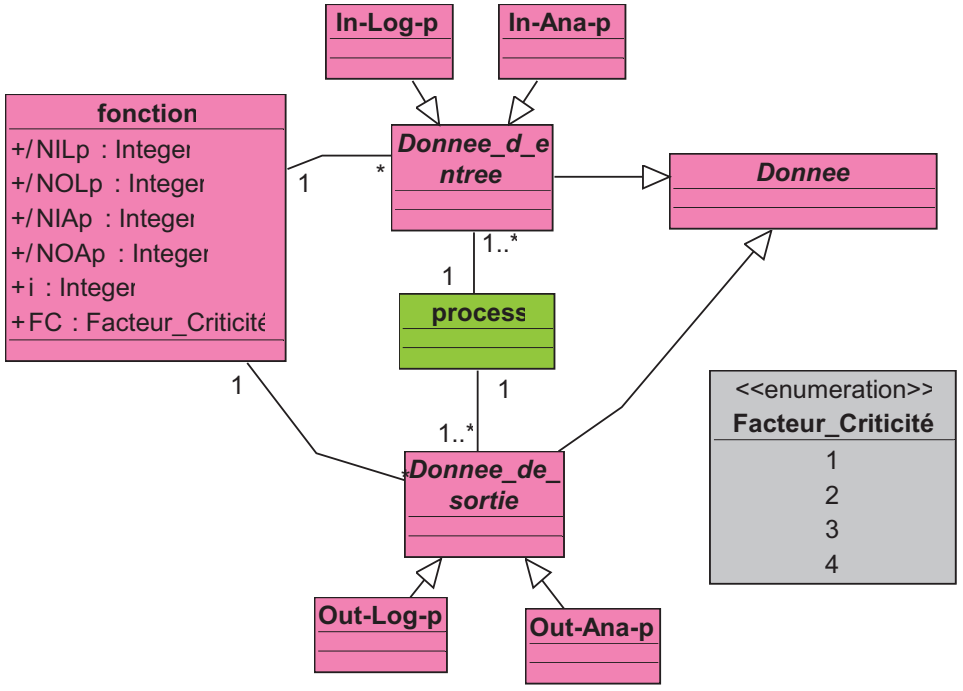


Figure 5.7 – Méta-modèle de l'ensemble des fonctions

5.3.2 L'ensemble des matériels

L'architecture matérielle (cf figure 5.8) est composée :

- de contrôleurs : un contrôleur comprend un automate et ses racks d'entrées/sorties. Chaque contrôleur possède un niveau de sécurité.

- d'un réseau représentant les liens entre les contrôleurs et permettant les échanges d'informations ; le réseau n'est pas représenté dans le méta-modèle car il ne fait pas partie de l'étude de dimensionnement.

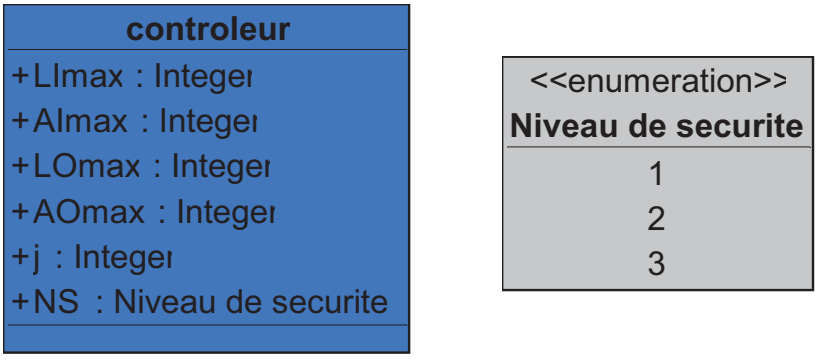


Figure 5.8 – Méta-modèle des matériels

Le méta-modèle de l'architecture matérielle reprenant les paramètres du contrôleur est représenté dans la figure 5.8, avec :

- LImax, entier représentant le nombre d'entrées logiques maximum du procédé ;
- LOmax, entier représentant le nombre de sorties logiques maximum du procédé ;
- AImax, entier représentant le nombre d'entrées analogiques maximum du procédé ;
- AOmax, entier représentant le nombre de sorties analogiques maximum du procédé ;
- j, entier représentant l'indice du contrôleur ;
- NS, représentant le niveau de sécurité du contrôleur. Il est énuméré parmi les valeurs définies dans la formalisation du problème $NS \in \{1, ..., p\}$ avec $p \in \mathbb{N}^* - \{1\}$et dans notre cas $NS \in \{1, 2, 3\}$;

Ces contrôleurs sont donc représentés avec leurs caractéristiques dans la figure 5.8. Dans notre cas d'étude, seulement trois niveaux de sécurité existent :

- 1 (ou E1A) représentant les matériels les plus critiques ;
- 2 (ou E1B) ;
- 3 (ou E2).

5.3.3 L'architecture opérationnelle

L'architecture opérationnelle est construite par projection de l'architecture fonctionnelle sur l'architecture matérielle. Chaque élément de l'architecture fonctionnelle, que nous avons défini comme étant une fonction, est alloué sur les contrôleurs de l'architecture matérielle.

Le lien entre la classe "fonction" et la classe "contrôleur" représente l'allocation des fonctions sur les contrôleurs. On peut remarquer sur la figure 5.9 qu'un contrôleur peut recevoir plusieurs fonctions sur ce lien et qu'un contrôleur ne peut pas être vide (1–1..*) dans l'architecture opérationnelle, il contient donc au minimum une fonction.

Le méta-modèle (figure 5.9) représente la vue opérationnelle avec la description des classes "fonction" et "contrôleur". Cependant, pour respecter les contraintes d'allocation, un ajout de contraintes Object Constraint Langage (OCL) est nécessaire pour

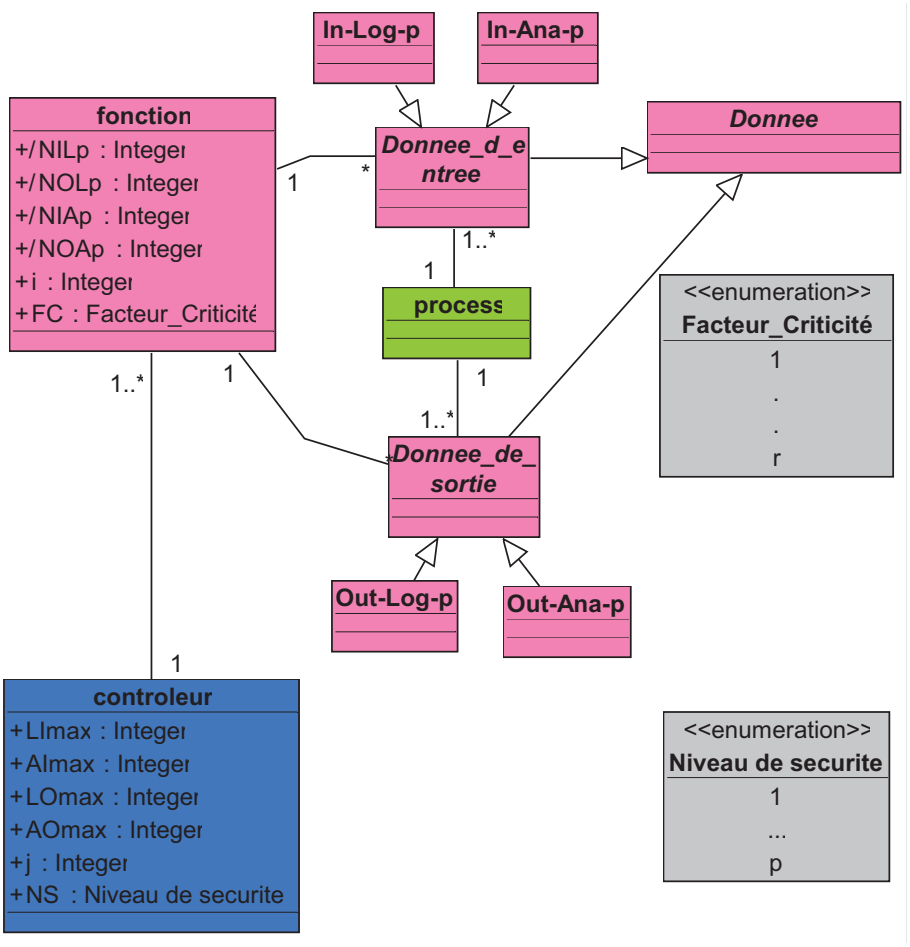


Figure 5.9 – Méta-modèle de l'architecture opérationnelle

compléter le diagramme de classe. Ces contraintes OCL permettent à la fois de récupérer les données intéressantes sur les diagrammes d'instance et de spécifier les contraintes de conception définies en 1.4.4. On peut remarquer que les contraintes OCL portent sur les fonctions et leurs paramètres ou sur les contrôleurs et ce qui peut leur être alloué.

Les contraintes OCL sur les fonctions permettent de définir le nombre d'entrées/sorties logiques et analogiques d'une fonction que ce soit entre fonctions ou avec le procédé.

Le nombre d'entrées/sorties d'une fonction donnée venant ou allant au procédé est défini par le nombre de relations existant entre cette fonction et ses entrées/sorties associées. Pour une architecture fonctionnelle, cette information est récupérée depuis les diagrammes d'instance en utilisant la fonction OCL : *size()*. Cette fonction *size()* est utilisée en se plaçant au niveau de la fonction de contrôle-commande avec :

context fonction

Une fois le contexte défini, nous allons construire pour chaque fonction un N-uplet. Ce

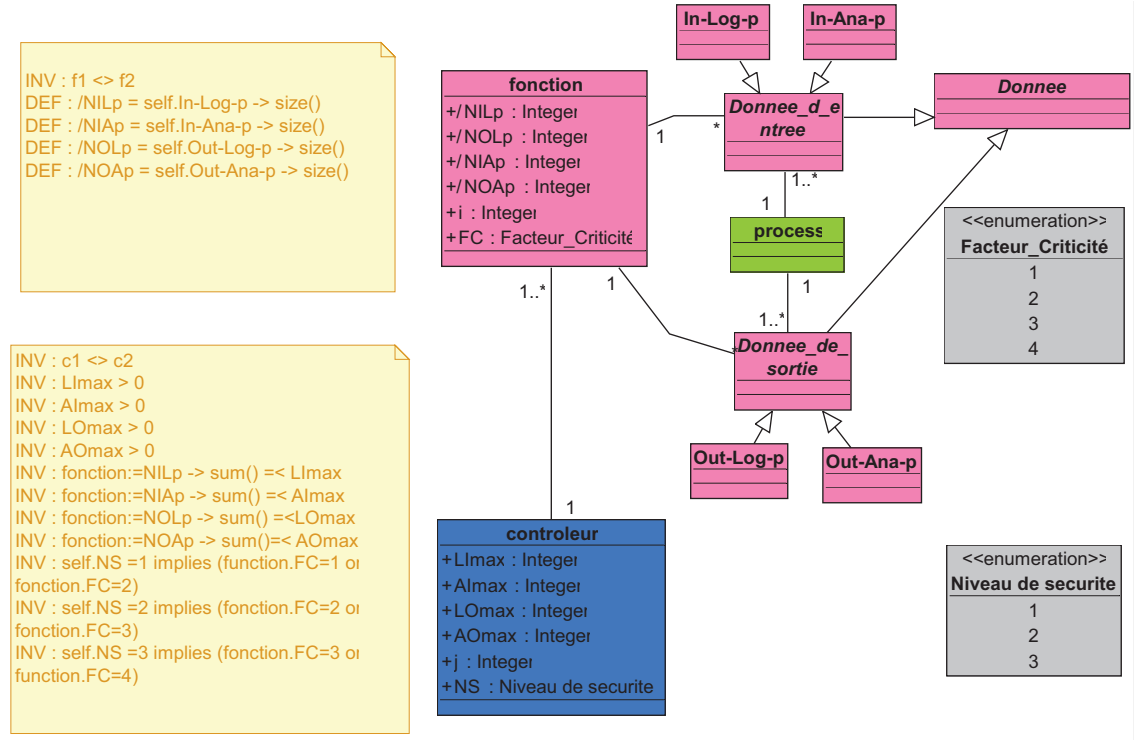


Figure 5.10 – Méta-modèle de l'architecture opérationnelle avec contraintes OCL

N-uplet est composé des paramètres de la fonction :

- le nombre d'entrées logiques que la fonction a de relier avec le procédé ;

$$\textbf{DEF} : /\text{NILp} = \text{self.In-Log-p} -> \text{size}() \tag{5.6}$$

- le nombre d'entrées analogiques que la fonction a de relier avec le procédé ;

$$\textbf{DEF} : /\text{NIAp} = \text{self.In-Ana-p} -> \text{size}() \tag{5.7}$$

- le nombre de sorties logiques que la fonction a de relier avec le procédé ;

$$\textbf{DEF} : /\text{NOLp} = \text{self.Out-Log-p} -> \text{size}() \tag{5.8}$$

- le nombre de sorties analogiques que la fonction a de relier avec le procédé ;

$$\textbf{DEF} : /\text{NOAp} = \text{self.Out-Ana-p} -> \text{size}() \tag{5.9}$$

Les autres contraintes OCL sont au niveau de la classe "contrôleur". Nous les définissons dans ce contexte :

context controleur

Ces contraintes au niveau du contrôleur permettent :

- de définir les paramètres des contrôleurs ; la définition des paramètres des contrôleurs revient à définir le nombre d'entrées/sorties logiques et analogiques du contrôleur.

$$\textbf{INV} : LImax > 0 \tag{5.10}$$

$$\textbf{INV} : AImax > 0 \tag{5.11}$$

$$\textbf{INV} : LOmax > 0 \tag{5.12}$$

$$\textbf{INV} : AOmax > 0 \tag{5.13}$$

- de définir les différentes contraintes de capacité à prendre en compte lors de l'allocation des fonctions sur les contrôleurs. Ces contraintes de capacité traduisent le fait que la somme des entrées/sorties des fonctions allouées à un contrôleur donné doit être plus petite ou égale à la capacité de ce contrôleur en terme d'entrées/sorties. Ces contraintes sont modélisées en utilisant la fonction OCL *sum()*

$$\textbf{INV} : function./NIL- > sum() \leq LImax \tag{5.14}$$

$$\textbf{INV} : function./NIA- > sum() \leq AImax \tag{5.15}$$

$$\textbf{INV} : function./NOL- > sum() \leq LOmax \tag{5.16}$$

$$\textbf{INV} : function./NOA- > sum() \leq AOmax \tag{5.17}$$

- de définir la contrainte de compatibilité 1.6, qui est une relation entre le niveau de sécurité du contrôleur et le facteur de criticité de la fonction. En donnant les diagrammes d'instances de l'architecture fonctionnelle et de l'architecture matérielle, ces contraintes sont alors exprimées :

$$\textbf{INV} : \text{self}.NS = 1 \text{ implies fonction}.FC = 1 \text{ or fonction}.FC = 2) \tag{5.18}$$

$$\textbf{INV} : \text{self}.NS = p \text{ implies}(\text{fonction}.FC = r - 1 \text{ or fonction}.FC = r) \tag{5.19}$$

- de définir les contraintes de séparation de fonctions qui interdisent à deux fonctions f^k et f^l de pouvoir être allouées dans un même contrôleur.

$$\textbf{INV} : \text{self}.j \text{ implies}(\text{fonction}.i = k \text{ or } \text{fonction}.i = l) \tag{5.20}$$

Les contraintes de sûreté, qui sont des relations entre le niveau de sécurité du contrôleur et le facteur de criticité de la fonction, sont définies pour notre cas de production d'énergie. En donnant les diagrammes d'instances de l'architecture fonctionnelle et de l'architecture matérielle, ces contraintes sont alors exprimées :

$$\textbf{INV} : \text{self}.NS = 1 \text{ implies } \text{fonction}.FC = 1 \text{ or } \text{fonction}.FC = 2) \tag{5.21}$$

$$\textbf{INV} : \text{self}.NS = 2 \text{ implies}(\text{fonction}.FC = 2 \text{ or } \text{fonction}.FC = 3) \tag{5.22}$$

$$\textbf{INV} : \text{self}.NS = 3 \text{ implies}(\text{fonction}.FC = 3 \text{ or } \text{fonction}.FC = 4) \tag{5.23}$$

5.4 Conclusion

Le cas d'étude présenté en début de chapitre permet de voir que le modèle générique présenté au chapitre 3 peut être facilement modifié pour prendre en compte d'autres problèmes d'allocations tels que ceux apparaissant lors de la modification d'une architecture existante.

Dans le cadre du projet CONNEXION faisant suite à ce travail de thèse, cet ensemble outillé, dénommé ATHENA, constitue une brique de base qui sera enrichie en intégrant des contraintes de performances temporelles ainsi que de fiabilité et de disponibilité. Le méta-modèle développé permet de capitaliser les données relatives aux architectures sous la forme d'un diagramme de classe UML et de contraintes OCL. Ce méta-modèle est une base et devra être enrichi en fonction du domaine d'application considéré.

Conclusion

Ce mémoire expose les contributions réalisées au cours de ces recherches doctorales. Celles-ci permettent de résoudre les problèmes d'allocation de fonctions sous contraintes en proposant une solution faisable, un ensemble de solutions réalisables et une solution minimisant le nombre de contrôleurs.

Notre contribution majeure est la proposition d'une méthode d'allocation de fonctions sous contraintes dont une formalisation a été effectuée. Cette proposition est présentée et développée tout au long du mémoire et repose sur deux principes :

- le processus d'allocation est modélisé par un ensemble de mécanismes concurrents d'appel/réponse dans un réseau d'automates communicants à variables partagées ;
- une solution d'allocation est obtenue par recherche d'atteignabilité dans ce réseau.

Cette proposition est aussi comparée à une approche de programmation linéaire en nombres entiers en recherchant une solution faisable et une solution minimisant le nombre de contrôleurs. Cette comparaison permet de voir que l'approche par recherche d'atteignabilité est beaucoup plus efficace pour trouver rapidement une solution faisable et donc d'un ensemble de solutions faisables.

La proposition a ensuite été validée sur un ensemble de cas d'étude et a été complétée en termes de modélisation et de contraintes pour permettre une étude de cas industrielle prenant en compte un ensemble de 1200 fonctions avec des contraintes de capacité, des contraintes de sûreté et des contraintes techniques.

Au terme de ces travaux, nos contributions ouvrent sur plusieurs perspectives à court et moyen terme.

Notre contribution a été effectuée sur l'outil UPPAAL, il serait également intéressant de tester notre approche sur un outil de vérification formelle non temporisé. En effet,

le choix du logiciel UPPAAL a été fait car il possède une interface graphique très ergonomique et peut fournir une trace d'exécution en cas de preuve positive mais cet outil a été développé particulièrement pour analyser des systèmes représentés sous la forme d'automates temporisés.Il serait donc intéressant de reprendre l'étude à l'aide d'outils tels que SPIN ou NuSMV pour peut être améliorer les performances de la méthode.

Une possibilité de couplage entre la recherche d'atteignabilité et la programmation linéaire en nombres entiers pourrait donner des résultats intéressant en interfaçant ces deux approches dans le domaine où elles sont les meilleures. La recherche d'atteignabilité pourrait donc être utilisée pour trouver une première solution faisable qui serait ensuite traitée en programmation linéaire afin de trouver la meilleure solution possible. Quelques cas ont été traités et ont montré que le couplage permettait d'arriver plus rapidement à des solutions meilleures que celles trouvées par la programmation linéaire. Cette perspective est donc encourageante et devrait pouvoir être généralisée. Ce couplage peut aussi être comparé à d'autres méthodes de résolution de problème de satisfaction de contraintes comme les algorithmes génétiques ou les méta heuristiques, on notera tout de même que ces méthodes ont une difficulté pour la définition des populations initiales.

A plus long terme, les architectures fonctionnelle et organique devront être prises en compte dans l'optique d'intégrer, dans la mesure du possible, des contraintes de performances temporelles se basant sur des modèles temporisés et des analyses de fiabilité et de disponibilité se basant sur des modèles stochastiques. Cette amélioration sera possible uniquement si une description très fine de l'architecture organique existe.

Cette amélioration pourra peut être développée dans cadre du cluster CONNEXION, projet financé par le programme Investissements d'Avenir, volet Briques Génériques du Logiciel Embarqué (Contrôle-commande numérique nucléaire pour l'export et la rénovation) sur la base de la méthode développée dans cette thèse qui constitue une brique dénommée ATHENA. Le cluster CONNEXION a pour ambition de proposer et de valider une architecture innovante de plateformes de contrôle-commande adaptée aux centrales nucléaires en France et à l'International et plus particulièrement l'objectif de l'arc 2.2 qui est la validation d'une architecture de contrôle commande sous l'angle performance et sûreté de fonctionnement.

Bibliographie

[AADL, 2009]

AADL (2009). Architecture analysis & design language. *SAE International, Version 2.0*.

[Addad et Amari, 2008]

Addad, B. et Amari, S. (2008). Modeling and Response Time Evaluation of Ethernet-based control Architectures using Timed Event Graphs and Max-Plus Algebra. *In IEEE Conference on Automation Science and Engineering*, pages 418–423, United States.

[Auinger *et al.*, 2005]

Auinger, F., Brennan, R., Christensen, J., Lastra, L. et Vyatkin, V. (2005). Requirements and solutions to software encapsulation and engineering in next generation manufacturing systems : Oooneida approach. *International Journal of Computer Integrated Manufacturing*, 18:572 – 585.

[Beasley, 1985]

Beasley, J. E. (1985). An exact two-dimensional non-guillotine cutting tree search procedure. *Operations Research*, 33(1):49–64.

[Behrmann *et al.*, 2002]

Behrmann, G., Bengtsson, J., David, A., Larsen, K., Pettersson, P. et Yi, W. (2002). UPPAAL implementation secrets. *In Proc. of 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT)*, Oldenburg, Germany.

[Behrmann *et al.*, 2005]

Behrmann, G., Brinksma, E., Hendriks, M. et Mader, A. (2005). Production scheduling by reachability analysis - a case study. *Parallel and Distributed Processing Symposium, International*, 3:140–147.

[Behrmann *et al.*, 2006]

Behrmann, G., David, R. et Larsen, K. (2006). A tutorial on uppaal 4.0. http ://www.it.uu.se/research/group/d̃arts/papers/texts/new-tutorial.pdf.

[Bérard *et al.*, 2001]

Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L. et Schnoebelen, P. (2001). *Systems and Software Verification*. Springer.

[Berthomieu *et al.*, 2004]

Berthomieu, B., Ribet, P.-O. et Vernadat, F. (2004). The tool tina – construction of abstract state spaces for petri nets and time petri nets. *International Journal of Production Research*, 42(14):2741–2756.

[Boucheneb et Barkaoui, 2012]

Boucheneb, H. et Barkaoui, K. (2012). Reachability Analysis of P-Time Petri Nets with Parametric Markings. *In 12th International Conference on Application of Concurrency to System Design ACSD 2012 (to appear).*, pages 1–10, Hamburg, Germany.

[Campbell *et al.*, 2010]

Campbell, S. L., Chancelier, J.-P. et Nikoukhah, R. (2010). *Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4*. Springer.

[Cimatti *et al.*, 2002]

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. et Tacchella, A. (2002). NuSMV Version 2 : An OpenSource Tool for Symbolic Model Checking. *In Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 de *LNCS*, Copenhagen, Denmark. Springer.

[Clarke et Emerson, 1981]

Clarke, E. M. et Emerson, E. A. (1981). Design and synthesis of synchronization skeletons using branching-time temporal logic. *In* Springer-Verlag, éditeur : *Logic of Programs, Workshop*, pages 52–71, London, UK.

[Dantzig, 1990]

DANTZIG, G. (1990). *Origins of the simplex method.* A History of Scientific Computing.

[EDF, 2010]

EDF (2010). Rapport préliminaire de sûreté de flamanville 3. Rapport technique, Electricité De France.

[Emerson et Halpern, 1986]

EMERSON, E. A. et HALPERN, J. Y. (1986). "sometimes" and "not never" revisited : on branching versus linear time temporal logic. *ACM*, 33(1):151–178.

[Faure et Lesage, 2001]

FAURE, J.-M. et LESAGE, J.-J. (2001). Methods for safe control systems design and implementation. *In 10th IFAC Symposium on Information Control Problems in Manufacturing, INCOM'2001.*

[Fekete et Schepers, 2004]

FEKETE, S. P. et SCHEPERS, J. (2004). A combinatorial characterization of higher - dimensional orthogonal packing. *Mathematics of Operations Research*, 29:353–368.

[Ferrarini *et al.*, 2005]

FERRARINI, L., VEBER, C., SCHWAB, C., TANGERMANN, M. et PRAYATI, A. (2005). Control functions development for distributed automation systems using the torero approach. *In 16th IFAC World Congress*, Prague (Czech Republic).

[Garey et Johnson, 1979]

GAREY, M. R. et JOHNSON, D. S. (1979). *Computers and Intractability : A Guide to the Theory of NP-Completeness.* W. H. Freeman.

[Ghezal *et al.*, 1990]

GHEZAL, N., MATIATOS, S., PIOVESAN, P., SOREL, Y. et SORINE, M. (1990). SYNDEX un environnement de programmation pour multi-processeur de traitement du signal. Mecanismes de communication. Rapport de recherche RR-1236, INRIA. Projet SOSSO.

[Gilmore et Gomory, 1961]

GILMORE, P. et GOMORY, R. (1961). A linear programming approach to the cutting stock problem. *Ops. Res.*, 13:94–120.

[Gilmore et Gomory, 1963]

GILMORE, P. et GOMORY, R. (1963). A linear programming approach to the cutting stock problem - part ii. *Ops. Res.*, 11:863–888.

[Goldberg, 1989]

GOLDBERG, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1 édition.

[Gomory, 1960]

GOMORY, R. (1960). An algorithm for the mixed integer problem. Rapport technique, The Rand Corporation.

[Gourcuff *et al.*, 2008]

GOURCUFF, V., de SMET, O. et FAURE, J.-M. (2008). Improving large-sized PLC programs verification using abstractions. *In IFAC World congress*.

[Holzmann, 2004]

HOLZMANN, G. J. (2004). *The model SPIN checker*. Addison Wesley Professiona.

[INRIA, 2005]

INRIA (2005). Model driven engineering open-source platform for real-time and embedded systems. http://openembedd.org/home.html. ANR-05-RNTL-0031.

[Katzke et Vogel-Heuser, 2005]

KATZKE, U. et VOGEL-HEUSER, B. (2005). UML-PA as an Engineering Model for Distributed Process Automation. *In 16th IFAC world Conference*, Prague.

[Khanafer, 2010]

KHANAFER, A. (2010). *Algorithmes pour des problèmes de bin packing mono-et multi-objectif*. Thèse de doctorat, Université des Sciences et Technologies de Lille.

[Kobetski et Fabian, 2009]

KOBETSKI, A. et FABIAN, M. (2009). Time-optimal coordination of flexible manufac-

turing systems using deterministic finite automata and mixed integer linear programming. *Discrete Event Dynamic Systems*, 19:287–315. 10.1007/s10626-009-0064-9.

[Lawler et Wood, 1966]

Lawler, E. L. et Wood, D. E. (1966). Branch-and-bound methods : A survey. *Operations Research*, 14(4):699–719.

[Lemattre *et al.*, 2011a]

Lemattre, T., Denis, B., Faure, J.-M., Pétin, J.-F. et Salaün, P. (2011a). Aide à la conception d'architectures opérationnelles de commande de systèmes critiques par analyse d'atteignabilité. *In 4èmes Journées Doctorales / Journées Nationales MACS, JD-JN-MACS*, page CDROM, Marseille, France.

[Lemattre *et al.*, 2011b]

Lemattre, T., Denis, B., Faure, J.-M., Pétin, J.-F. et Salaün, P. (2011b). Designing operational control architectures of critical systems by reachability analysis. *In IEEE 7th International Conference on Automation Science and Engineering*, pages Pages 12–18, Trieste, Italy.

[Lemattre *et al.*, 2011c]

Lemattre, T., Denis, B., Faure, J.-M., Pétin, J.-F. et Salaün, P. (2011c). Using a meta-model to build operational architectures of automation systems for critical processes. *In 16th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'2011*, page CDROM, Toulouse, France.

[Lindahl *et al.*, 1998]

Lindahl, M., Pettersson, P. et Yi, W. (1998). Formal design and analysis of a gear controller : an industrial case study using uppaal. *LNCS, Proceedings of 4*th *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, 1384:281–297.

[Marangé *et al.*, 2011]

Marangé, P., Pétin, J.-F., Manceaux, A. et Gouyon, D. (2011). Contribution à la reconfiguration des systèmes de production : ordonnancement par recherche d'atteignabilité. *Journal Européen des Systèmes Automatisés*, 45(1/3):45–60.

[Marsal *et al.*, 2006]

MARSAL, G., DENIS, B., FAURE, J.-M. et FREY, G. (2006). Evaluation of Response Time in Ethernet-based Automation Systems. *In Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'06*, Prague, Czech Republic.

[Martin, 2004]

MARTIN, S. (2004). *Maîtrise de la dimension temporelle de la qualité de service dans els réseaux.* Thèse de doctorat, Université Paris XII.

[Martin et Minet, 2005]

MARTIN, S. et MINET, P. (2005). Improving the analysis of distributed non-preemptive fp/dp* with the trajectory approach. *In Telecommunication Systems, Springer*, 30:49–79.

[Muller *et al.*, 2005]

MULLER, P., FLEUREY, F. et marc JÉZÉQUEL, J. (2005). Weaving executability into object-oriented meta-languages. *In in : International Conference on Model Driven Engineering Languages and Systems (MoDELS), LNCS 3713 (2005*, pages 264–278. Springer.

[Perin et Faure, 2008]

PERIN, M. et FAURE, J.-M. (2008). Analyse prévisionnelle des fautes des systèmes embarqués discrets par vérification model-based. *In Actes de la Conférence Internationale Francophone d'Automatique*, page CDRom paper n ° 381, Bucarest, Roumanie.

[Pnueli, 1981]

PNUELI, A. (1981). The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45 – 60.

[Rossi et Schnoebelen, 2000]

ROSSI, O. et SCHNOEBELEN, P. (2000). Formal modeling of timed function blocks for the automatic verification of ladder diagram programs. *In Proc. of the 4*th *International Conference Automation of Mixed Processes (ADPM'00)*, pages pp. 177–182.

[Ruel, 2009]

RUEL, S. (2009). *Évaluation des bornes des performances temporelles des Architectures*

d'Automatisation en Réseau par preuves itératives de propriétés logiques. Thèse de doctorat, École Doctorale Sciences Pratiques, ENS Cachan, F 94230 CACHAN.

[Ruel *et al.*, 2009]

Ruel, S., De Smet, O. et Faure, J.-M. (2009). Finding the bounds of response time of networked automation systems by iterative proofs. *In Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2009)*, pages 1365–1370, Moscow, Russia.

[Subbiah et Engell, 2010]

Subbiah, S. et Engell, S. (2010). Short-Term Scheduling of Multi-Product Batch Plants with Sequence-Dependent Changeovers Using Timed Automata Models. *In 20th European Symposium on Computer Aided Process Engineering*, volume 28, pages 1201–1206.

[SysML, 2008]

SysML (2008). Systems modeling language. *Object Management Group, Version 1.1.*

[TOPCASED, 2010]

TOPCASED (2010). The open-source toolkit for critical systems. http ://www.topcased.org/.

[Tranoris et Thramboulidis, 2006]

Tranoris, C. et Thramboulidis, K. (2006). A tool supported engineering process for developing control applications. *Computers in Industry*, 57(5):p 462–472.

[UML, 2011]

UML (2011). Unified modeling language. *Object Management Group, Version 2.4.1.*

[UML MARTE, 2009]

UML MARTE (2009). A uml profile for marte : Modeling and analysis of real-time embedded systems. *Object Management Group, Version 1.0.*

[Willard, 2007]

Willard, B. (2007). Uml for systems engineering. *Computer Standards and Interfaces*, 29:69 – 81.

[Wonham et Ramadge, 1987]

WONHAM, W. et RAMADGE, P. (1987). On the supremal controllable sub-language of a given language. *SIAM J. Control and optimisation*, 25 (3):637–659.

[Wäscher *et al.*, 2007]

WÄSCHER, G., HAUSSNER, H. et SCHUMANN, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183:1109 – 1130.

Annexe

Données du 1^{er} cas d'étude de 200 fonctions

f^i	CF	NIl	NIa	NOl	NOa
f^{1}	1	2	4	4	7
f^{2}	1	7	0	3	6
f^{3}	1	1	4	0	6
f^{4}	1	2	9	5	6
f^{5}	1	7	7	8	8
f^{6}	1	3	0	8	7
f^{7}	1	8	8	0	4
f^{8}	1	6	6	1	2
f^{9}	1	2	9	7	0
f^{10}	1	3	6	8	5
f^{11}	1	4	2	1	2
f^{12}	1	0	8	8	9
f^{13}	1	8	5	5	5
f^{14}	1	4	9	6	3
f^{15}	1	4	7	0	8
f^{16}	1	0	6	2	7
f^{17}	1	3	6	4	4
f^{18}	1	5	1	5	8
f^{19}	1	8	2	8	3
f^{20}	1	3	8	6	6
f^{21}	1	4	4	0	1
f^{22}	1	3	3	9	4
f^{23}	1	2	4	1	5
f^{24}	1	2	7	3	2
f^{25}	1	1	3	1	0
f^{26}	1	4	9	3	4
f^{27}	1	6	6	5	1
f^{28}	1	3	5	9	5
f^{29}	1	8	7	0	9
f^{30}	1	8	8	3	7
f^{31}	1	6	6	5	5
f^{32}	1	7	9	5	8
f^{33}	1	6	7	9	4
f^{34}	1	1	2	6	7
f^{35}	1	0	4	9	2
f^{36}	1	4	6	5	9
f^{37}	1	8	7	8	7
f^{38}	1	7	0	1	8
f^{39}	1	6	9	6	3
f^{40}	1	4	0	7	5
f^{41}	1	4	3	3	3
f^{42}	1	1	1	0	9
f^{43}	1	5	6	0	7
f^{44}	1	6	0	0	6
f^{45}	1	2	8	5	3
f^{46}	1	4	3	1	3
f^{47}	1	5	6	5	8
f^{48}	1	6	9	2	2
f^{49}	1	6	3	1	4
f^{50}	1	0	5	8	2
f^{51}	2	2	9	6	2
f^{52}	2	5	3	1	7
f^{53}	2	4	1	2	8
f^{54}	2	7	0	9	3
f^{55}	2	8	2	5	4
f^{56}	2	7	3	4	4
f^{57}	2	9	1	0	6
f^{58}	2	4	8	9	8
f^{59}	2	9	6	5	9
f^{60}	2	0	0	8	2
f^{61}	2	8	3	1	9
f^{62}	2	1	0	2	5
f^{63}	2	7	4	2	2
f^{64}	2	5	9	0	4
f^{65}	2	0	1	1	5
f^{66}	2	8	8	9	2
f^{67}	2	6	3	6	0
f^{68}	2	3	4	5	7
f^{69}	2	5	4	4	7
f^{70}	2	6	7	8	3
f^{71}	2	8	6	2	3
f^{72}	2	3	7	5	6
f^{73}	2	0	1	2	4
f^{74}	2	2	8	7	0
f^{75}	2	4	8	2	6
f^{76}	2	3	6	6	4
f^{77}	2	2	4	6	6
f^{78}	2	3	7	2	3
f^{79}	2	8	3	3	1
f^{80}	2	6	7	4	2
f^{81}	2	5	0	0	6
f^{82}	2	3	3	5	6
f^{83}	2	8	9	2	0
f^{84}	2	6	9	1	9
f^{85}	2	9	3	6	1
f^{86}	2	5	6	2	0
f^{87}	2	2	3	5	9
f^{88}	2	2	6	4	6
f^{89}	2	3	3	7	6
f^{90}	2	3	9	7	9
f^{91}	2	3	4	6	7
f^{92}	2	9	7	6	9
f^{93}	2	5	6	1	7
f^{94}	2	8	8	9	8
f^{95}	2	9	8	0	6
f^{96}	2	2	3	3	6
f^{97}	2	1	4	0	2
f^{98}	2	7	2	8	0
f^{99}	2	0	4	8	8
f^{100}	2	8	2	9	4
f^{101}	3	2	4	4	1
f^{102}	3	2	6	1	5
f^{103}	3	9	6	7	5
f^{104}	3	1	5	6	6
f^{105}	3	0	6	1	3
f^{106}	3	2	1	1	1
f^{107}	3	1	3	1	5
f^{108}	3	6	4	1	7
f^{109}	3	6	2	6	6
f^{110}	3	8	9	5	2
f^{111}	3	1	8	3	0
f^{112}	3	9	3	7	6
f^{113}	3	3	8	6	2
f^{114}	3	6	1	5	8
f^{115}	3	6	8	3	0
f^{116}	3	1	8	7	1
f^{117}	3	4	1	0	2
f^{118}	3	2	5	5	1
f^{119}	3	8	6	2	7
f^{120}	3	0	5	5	0
f^{121}	3	6	2	2	1
f^{122}	3	0	4	7	8
f^{123}	3	2	2	4	9
f^{124}	3	3	6	8	6
f^{125}	3	1	9	8	8
f^{126}	3	8	3	0	1
f^{127}	3	5	2	7	9
f^{128}	3	6	9	0	5
f^{129}	3	1	5	8	2
f^{130}	3	4	4	8	1
f^{131}	3	5	4	0	6
f^{132}	3	8	7	3	9
f^{133}	3	3	5	2	5
f^{134}	3	0	8	7	0
f^{135}	3	3	1	1	9
f^{136}	3	0	1	5	8
f^{137}	3	1	5	3	1
f^{138}	3	1	8	1	7
f^{139}	3	7	6	8	6
f^{140}	3	2	9	3	8
f^{141}	3	6	8	4	6
f^{142}	3	2	1	8	6
f^{143}	3	1	7	6	3
f^{144}	3	2	2	9	8
f^{145}	3	4	8	7	5
f^{146}	3	1	0	7	8
f^{147}	3	2	0	8	0
f^{148}	3	3	9	0	0
f^{149}	3	9	5	5	5
f^{150}	3	9	2	7	6
f^{151}	4	8	9	6	3
f^{152}	4	8	5	1	7
f^{153}	4	7	1	0	3
f^{154}	4	9	1	0	1
f^{155}	4	0	9	9	1
f^{156}	4	9	7	5	5
f^{157}	4	8	9	7	5
f^{158}	4	5	4	5	9
f^{159}	4	5	1	8	1
f^{160}	4	2	2	9	1
f^{161}	4	9	6	6	4
f^{162}	4	6	4	4	0
f^{163}	4	5	0	4	4
f^{164}	4	9	5	2	8
f^{165}	4	6	4	0	7
f^{166}	4	3	3	3	6
f^{167}	4	6	3	9	3
f^{168}	4	7	8	0	0
f^{169}	4	8	3	7	9
f^{170}	4	6	4	5	8
f^{171}	4	9	5	6	6
f^{172}	4	3	3	5	7
f^{173}	4	5	0	0	7
f^{174}	4	7	4	8	6
f^{175}	4	5	8	4	7
f^{176}	4	7	9	6	5
f^{177}	4	2	3	3	2
f^{178}	4	3	4	8	9
f^{179}	4	1	2	2	9
f^{180}	4	7	9	3	0
f^{181}	4	9	6	7	2
f^{182}	4	0	1	6	7
f^{183}	4	8	8	7	4
f^{184}	4	9	9	2	6
f^{185}	4	2	3	1	1
f^{186}	4	2	2	5	7
f^{187}	4	2	7	0	0
f^{188}	4	2	6	4	1
f^{189}	4	1	2	1	8
f^{190}	4	2	8	8	0
f^{191}	4	3	4	7	8
f^{192}	4	1	8	7	0
f^{193}	4	9	2	1	8
f^{194}	4	2	2	4	0
f^{195}	4	9	4	6	1
f^{196}	4	4	9	0	5
f^{197}	4	3	5	5	7
f^{198}	4	0	0	0	5
f^{199}	4	6	9	2	8
f^{200}	4	4	0	2	9

Programme de recherche d'un ensemble de solution

```
import TL_module_v4
import os
import os.path
import csv
import time
import random

skip_exp = (1, 1, 1, 1, 0)

# Parametres generaux des experiences
data_dir = r"./data"      # emplacement des fichiers de donnees (format csv)
uppaal_dir = r"./uppaal" # emplacement des fichiers generes (xml, q, xtr, txt, dot, gif, html...)

print('\n\n')
    print('================================================================================')
    print('================================================================================')
    print('====    exp3 : generation de plusieurs solutions d''allocation pour')
    print('====           un meme cas d''etude')
    print('================================================================================')
    print('================================================================================')
    print('\n')

    data_file = "alloc_F020D1_000.csv"
    root_fn = "alloc_F020D1_000"
    # Initialisation de l'objet "uppal_alloc_model" incluant la lecture de du fichier csv
    UAL = TL_module_v4.uppal_alloc_model(data_file, data_dir)

    UAL.set_section_in_html_report('intro')
    UAL.add_title_to_html_report('Experiment #3')
    UAL.add_paragraph_to_html_report('Generation of several solutions allocation for the same case study.')
    UAL.add_function_param_to_html_report()
    UAL.set_section_in_html_report('chapter')

    solutions = []
    solution_number = 0
    NUMBER_OF_CONTROLLER = 20
    NUMBER_OF_FUNCTION = 20
    while solution_number < 10:
        file_name_complement = '_C020_S%02d' % solution_number
        # generation aleatoire de contrainte d'exclusion
        func_exclusions = []
        while len(func_exclusions) < (NUMBER_OF_FUNCTION // 2):
            f1 = random.randint(0, NUMBER_OF_FUNCTION - 1)
            f2 = random.randint(0, NUMBER_OF_FUNCTION - 1)
            if (f1 != f2) and ((f1, f2) not in func_exclusions):
                func_exclusions.append((f1, f2))
        #
        UAL.uppal_file_generator(NUMBER_OF_CONTROLLER, uppaal_dir,
```

```
                            file_name_complement, func_exclusions)
    (prop, ncu, alloc) = UAL.uppal_check()
    s = alloc.__repr__()
    if s not in solutions:
        solutions.append(s)
        solution_number = solution_number + 1
        print(alloc)
        UAL.add_title_to_html_report('Allocation #%d' % solution_number)
        UAL.add_alloc_to_html_report()
        UAL.add_graph_to_html_report()

#   generation du fichier html
UAL.html_report_generation(root_fn)

print('\n')
print('Ouvrir le fichier "%s.html"' % root_fn)
```