Contribution à la validation opérationnelle de l'ordonnancement et de la génération de séquences de commande pour les systèmes SCADA industriels

Ouissem Mesli Mesli-Kesraoui

THESE

Pour l'obtention du Grade de

DOCTEUR DE L'ECOLE NATIONALE SUPERIEURE DE MECANIQUE ET D'AEROTECHNIQUE

(Diplôme National – Arrêté du 25 mai 2016 Modifié par l'Arrêté du 26 Août 2022)

Ecole Doctorale :

MIMME_Mathématiques, Informatique, Matériaux, Mécanique, Energétique

Secteur de Recherche : Informatique et applications

Présentée par :
Ouissem MESLI-KESRAOUI

****

Contribution à la validation opérationnelle de l'ordonnancement et de la génération de séquences de commande pour les systèmes SCADA industriels

****

Directeurs de thèse : Patrick GIRARD, Pascal BERRUET

Co-encadrant :

Emmanuel GROLLEAU, Yassine OUHAMMOU et Soraya KESRAOUI

****

Soutenue le 26/06/2024

devant la Commission d'Examen

****

JURY

Président :
GENIET Annie, professeure, Université de Poitiers, Poitiers

Rapporteurs :
Ameur SOUKHAL Professeur, Université de Tours.
Nasser Mebarki Professeur, Université de Nantes

Membres du jury :
Flavio Oquendo Professeur, université Bretagne-sud, Vannes.
Pascale MARANGE Maître de conférences, Université de Lorraine.
Patrick GIRARD Professeur, Université de Poitiers, Poitiers.
Pascal BERRUET Professeur, Université de Bretagne-Sud, Lorient.

Invité.e.s :
Emmanuel GROLLEAU , Yassine OUHAMMOU, Soraya KESRAOU

Contribution à la validation opérationnelle de l'ordonnancement et de la génération de séquences de commande pour les systèmes SCADA industriels

Résumé :

Les systèmes industriels sont généralement des systèmes complexes comportant plusieurs composants et offrant plusieurs fonctionnalités ou fonctions. Vu le nombre important des composants de ces systèmes, dans le monde industriel, des bancs de tests SCADA sont souvent développés pour les tester. Un banc de test SCADA (Supervisory Control and Data Acquisition) est un système SCADA qui a pour rôle de surveiller l'état du procédé et d'envoyer des commandes pour le manipuler. Pour répondre à un objectif de test sur ces bancs, les experts métiers se chargent du lancement des différentes fonctions de test. Pour cela, ils définissent un ordre précis de ces fonctions. Toutefois, certaines contraintes viennent compliquer la définition de cet ordre. En effet, les fonctions peuvent se réaliser en parallèle ou séquentiellement. D'autre part, certaines fonctions se partagent les mêmes composants et sont régies par les mêmes contraintes physiques comme la pression, le débit, etc. La définition manuelle de cet ordre est très fastidieuse et peut être une source d'erreur. Une fois l'ordre déterminé, les experts sont en charge d'identifier les séquences d'actions élémentaires permettant la réalisation de la fonction. Par exemple, afin de réaliser une fonction de transfert d'eau d'une soute à une autre, les experts sont obligés de réaliser la séquence d'actions élémentaire suivante : ouvrir les vannes, enclencher la pompe, choisir la soute de départ/d'arrivé, etc. Cependant, il faut s'assurer que les commandes envoyées tiennent compte des multiples contraintes de fonctionnement opérationnelles. De fait, la réalisation de ces actions peut devenir difficile et source d'erreur.

Pour répondre à ces problématiques, l'objectif des travaux de cette thèse consiste à proposer des solutions permettant l'optimisation des bancs de tests SCADA. Notre approche comporte trois contributions majeures.

1) Afin d'optimiser la réalisation des tests, notre première contribution porte sur la proportion une solution pour l'ordonnancement mathématique de deux machines parallèles dédiées avec une ressource partagée et des précédences locales. Des algorithmes et procédures ont été proposés afin d'ordonnancer l'exécution des fonctions d'un banc de tests SCADA.

2) Dans la deuxième contribution, nous proposons une approche pour la génération de séquences de commande permettant l'exécution opérationnelle de l'ordonnancement calculé. Basée sur les modes et états afin de spécifier, modéliser et déduire un modèle opérationnel pour l'exécution de l'ordonnancement sur un banc de test SCADA. Le modèle opérationnel est ensuite utilisé pour la génération automatique des séquences de commande permettant la réalisation de l'ordonnancement mathématique.

3) Dans la troisième contribution, nous proposons un xDSML qui permet d'outiller les deux premières contributions. Notre xDSML permet d'assister les experts non-informaticiens des systèmes SCADA dans le processus d'optimisation de leur système. Ce qui, permet d'une part de spécifier toutes les informations statiques du système SCADA telles que le P&ID, les fonctions et également la décomposition fonctionnelle. D'autre part, il permet la spécification des informations dynamiques comme les modes des composants et des fonctions.

Publications scientifiques

MESLI-KESRAOUI, Ouissem, OUHAMMOU, Yassine, GOUBALI, Olga, et al. Towards a Model-Based Approach to Support Physical Test Process of Aircraft Hydraulic Systems. In : Model and Data

Engineering: 10th International Conference, MEDI 2021, Tallinn, Estonia, June 21–23, 2021, Proceedings 10. Springer International Publishing, 2021. p. 33-40.

MESLI-KESRAOUI, Ouissem, LEDRECK, Loic, GROLLEAU, Emmanuel, et al. Resource constraint scheduling on two dedicated machines: application to avionics. EURO Journal on Computational Optimization, 2024, p. 100093.

Mots clés :

Acquisition automatique des données / Automatic data collection systems

Commande automatique / Automatic control

Optimisation mathématique / Mathematical optimization

Ordonnancement (informatique) / Computer scheduling

Temps réel (informatique) / Real-time data processing

SCADA (Système de contrôle et d'acquisition de données / Supervisory Control and Data Acquisition), Tests industriels, Minimisation de makespan, Méta-modélisation, Séquences de commande, Vérification formelle, XDSML, Gestion des modes.

Contribution to the operational validation of scheduling and generation of control sequences for industrial SCADA systems

Abstract :

Industrial systems are usually complex, multi-component systems offering a wide range of functions. Given the large number of components in these systems, SCADA test benches are often developed in the industrial world to test them. A SCADA (Supervisory Control and Data Acquisition) test bench is a SCADA system whose role is to monitor process status and send commands to manipulate it. To meet a test objective on these benches, business experts take charge of launching the various test functions. To do this, they define a precise order for these functions. However, certain constraints complicate the definition of this order. Functions can be run in parallel or sequentially. On the other hand, some functions share the same components and are governed by the same physical constraints, such as pressure, flow rate and so on. The manual definition of this order can be a source of error, and also very tedious. Once the order has been determined, the experts are responsible for identifying the elementary action sequences required to perform the function. For example, in order to carry out a function involving the transfer of water from one bunker to another, the experts have to carry out the following elementary sequence of actions: open the valves, switch on the pump, select the departure/arrival bunker, and so on. However, it must be ensured that the multiple commands sent take into account the multiple operating constraints. As a result, carrying out these actions can become difficult and error-prone.

To answer these problems, the objective of this thesis is to propose solutions allowing the optimization of SCADA test benches. Our approach has three major contributions.

1) In order to optimize the realization of the tests, our first contribution focuses on the proportion of a solution for the mathematical scheduling of two parallel dedicated machines with a shared resource and local precedents. Algorithms and procedures have been proposed to schedule the execution of the functions of a SCADA test bench.
2) In the second contribution, we propose an approach for the automatic generation of command sequences allowing the operational execution of the calculated scheduling. Based on modes and states in order to specify, model and deduce an operational model for executing scheduling on a SCADA test bench. The operational model is used for the automatic generation of control sequences allowing the realization of mathematical scheduling.

In the third contribution, we propose an xDSML that allows equiping the first two contributions. Our xDSML allows assisting the experts of SCADA systems in the optimization process of their system. This, on the one hand, allows to specify all the static information of the SCADA system such as the P&ID, the functions and also the functional decomposition. On the other hand, it allows the specification of dynamic information such as modes of components and functions

Scientific publications

MESLI-KESRAOUI, Ouissem, OUHAMMOU, Yassine, GOUBALI, Olga, et al. Towards a Model-Based Approach to Support Physical Test Process of Aircraft Hydraulic Systems. In : Model and Data Engineering: 10th International Conference, MEDI 2021, Tallinn, Estonia, June 21–23, 2021, Proceedings 10. Springer International Publishing, 2021. p. 33-40.

MESLI-KESRAOUI, Ouissem, LEDRECK, Loic, GROLLEAU, Emmanuel, et al. Resource constraint scheduling on two dedicated machines: application to avionics. EURO Journal on Computational Optimization, 2024, p. 100093.

Keywords:

Acquisition automatique des données / Automatic data collection systems

Commande automatique / Automatic control

Optimisation mathématique / Mathematical optimization

Ordonnancement (informatique) / Computer scheduling

Temps réel (informatique) / Real-time data processing

SCADA (Système de contrôle et d'acquisition de données / Supervisory Control and Data Acquisition), Tests industriels, Minimisation de makespan, Méta-modélisation, Séquences de commande, Vérification formelle, XDSML, Gestion des modes.

À ma fille LENA

Remerciements

Je commence par remercier ma famille pour son soutien permanant. Je remercie énormément mes parents, mon mari Salim, Djamal et Soraya de m'avoir soutenue dans les moments de doutes. Je remercie également mes beaux-parents d'avoir apporté un soutien considérable pendant le Covid.

Mes remerciements vont aussi à mon directeur de thèse Patrick GIRARD, mon co-directeur Pascal BERRUET et mes encadrants Emmanuel GROLLEAU, Yassine OUHAMMOU et Soraya KESRAOUI d'avoir proposé ce sujet de thèse et d'être toujours rendus disponibles quand j'en avais besoin. Je remercie également Olga BORDIER pour sa disponibilité et pour l'intérêt qu'elle a porté à la réalisation de ces travaux de thèse.

Je remercie également tous les membres du jury, qui ont eu l'amabilité de rapporter et d'examiner cette thèse.

Mes vifs remerciements vont également à tous les membres et au personnel du laboratoire **LIAS**. Un remerciement particulier à la secrétaire du laboratoire LIAS Bénédicte BOINOT et la chargée de la scolarité de l'école doctorale MIMME de l'ENSMA, Audrey VERON. Je remercie également tout le personnel du laboratoire **LAB-STICC**.

Je remercie tout le personnel de **SEGULA Technologies** de LANESTER, en particulier Perrine LE SENECHAL. Un grand merci à Loic, Robin et Baudouin à qui je dois énormément pour la concrétisation de ces travaux.

Cette thèse est le fruit d'une collaboration entre l'entreprise SEGULA Technologies, acteur majeur des métiers de l'ingénierie, le Lab-STICC, pôle de référence en recherche sur les systèmes communicants et le LIAS, pôle de référence en recherche sur l'ingénierie des données et modèles, les systèmes embarqués temps réels, l'automatique.

	SEGULA Technologies 56602 Lanester Cedex http://www.segula.fr/fr
	Laboratoire d'Informatique et d'Automatique pour les systèmes Université de Poitier ISAE/ENSMA 1 Avenue Clément Ader, 86961 Chasseneuil du Poitou. http://www.lias-lab.fr/
	Laboratoire des Sciences et Techniques de l'Information, de la Communication et de la Connaissance (Lab-STICC) - UMR n° 6285 Université de Bretagne-Sud Centre de recherche Christiaan Huygens, BP 92116 56321 Lorient cedex, France www.lab-sticc.fr

Mots clés : SCADA, Tests industriels, Ordonnancement, Minimisation de Makespan, Méta-modélisation, Séquences de commande, Modélisation, Vérification formelle, Contrôle-Commande, banc de test SCADA, XDSML, Gestion des modes.

Résumé en Français

Les systèmes industriels sont généralement des systèmes complexes comportant plusieurs composants et offrant plusieurs fonctionnalités ou fonctions. Vu le nombre important des composants de ces systèmes, dans le monde industriel, des bancs de tests SCADA sont souvent développés pour les tester. Un banc de test SCADA (Supervisory Control and Data Acquisition) est un système SCADA qui a pour rôle de surveiller l'état du procédé et d'envoyer des commandes pour le manipuler. Pour répondre à un objectif de test sur ces bancs, les experts métiers se chargent du lancement des différentes fonctions de test. Pour cela, ils définissent un ordre précis de ces fonctions. Toutefois, certaines contraintes viennent compliquer la définition de cet ordre. En effet, les fonctions peuvent se réaliser en parallèle ou séquentiellement. D'autre part, certaines fonctions se partagent les mêmes composants et sont régies par les mêmes contraintes physiques comme la pression, le débit, etc. La définition manuelle de cet ordre est très fastidieuse et peut être une source d'erreur. Une fois l'ordre déterminé, les experts ont la charge d'identifier les séquences d'actions élémentaires permettant la réalisation de la fonction. Par exemple, afin de réaliser une fonction de transfert d'eau d'une soute à une autre, les experts sont obligés de réaliser la séquence d'actions élémentaire suivante : ouvrir les vannes, enclencher la pompe, choisir la soute de départ/d'arrivé, etc. Cependant, il faut s'assurer que les commandes envoyées tiennent compte des multiples contraintes de fonctionnement opérationnelles. De fait, la réalisation de ces actions peut devenir difficile et source d'erreur.

Pour répondre à ces problématiques, l'objectif des travaux de cette thèse consiste à proposer des solutions permettant l'optimisation des bancs de tests SCADA. Notre approche comporte trois contributions majeures.

1) Afin d'optimiser la réalisation des tests, notre première contribution porte sur la proportion une solution pour l'ordonnancement mathématique de deux machines parallèles dédiées avec une ressource partagée et des précédences locales. Des algorithmes et procédures ont été proposés afin d'ordonnancer l'exécution des fonctions d'un banc de tests SCADA.
2) Dans la deuxième contribution, nous proposons une approche pour la génération de séquences de commande permettant l'exécution opérationnelle de l'ordonnancement calculé. Basée sur les modes et états afin de spécifier, modéliser et déduire un modèle opérationnel pour l'exécution de l'ordonnancement sur un banc de test SCADA. Le modèle opérationnel est ensuite utilisé pour la génération automatique des séquences de commande permettant la réalisation de l'ordonnancement mathématique.
3) Dans la troisième contribution, nous proposons un xDSML qui permet d'outiller les deux premières contributions. Notre xDSML permet d'assister les experts non-informaticiens des systèmes SCADA dans le processus d'optimisation de leur système. Ce qui, permet d'une part de spécifier toutes les informations statiques du système SCADA telles que le P&ID, les fonctions et également la décomposition fonctionnelle. D'autre part, il permet la spécification des informations dynamiques comme les modes des composants et des fonctions.

Résumé en Anglais

Industrial systems are usually complex, multi-component systems offering a wide range of functions. Given the large number of components in these systems, SCADA test benches are often developed in the industrial world to test them. A SCADA (Supervisory Control and Data Acquisition) test bench is a SCADA system whose role is to monitor process status and send commands to manipulate it. To meet a test objective on these benches, business experts take charge of launching the various test functions. To do this, they define a precise order for these functions. However, certain constraints complicate the definition of this order. Functions can be run in parallel or sequentially. On the other hand, some functions share the same components and are governed by the same physical constraints, such as pressure, flow rate and so on. The manual definition of this order can be a source of error, and also very tedious. Once the order has been determined, the experts are responsible for identifying the elementary action sequences required to

perform the function. For example, in order to carry out a function involving the transfer of water from one bunker to another, the experts have to carry out the following elementary sequence of actions: open the valves, switch on the pump, select the departure/arrival bunker, and so on. However, it must be ensured that the multiple commands sent take into account the multiple operating constraints. As a result, carrying out these actions can become difficult and error-prone.

To answer these problems, the objective of this thesis is to propose solutions allowing the optimization of SCADA test benches. Our approach has three major contributions.

3) In order to optimize the realization of the tests, our first contribution focuses on the proportion of a solution for the mathematical scheduling of two parallel dedicated machines with a shared resource and local precedents. Algorithms and procedures have been proposed to schedule the execution of the functions of a SCADA test bench.
4) In the second contribution, we propose an approach for the automatic generation of command sequences allowing the operational execution of the calculated scheduling. Based on modes and states in order to specify, model and deduce an operational model for executing scheduling on a SCADA test bench. The operational model is used for the automatic generation of control sequences allowing the realization of mathematical scheduling.
5) In the third contribution, we propose an xDSML that allows equiping the first two contributions. Our xDSML allows assisting the experts of SCADA systems in the optimization process of their system. This, on the one hand, allows to specify all the static information of the SCADA system such as the P&ID, the functions and also the functional decomposition. On the other hand, it allows the specification of dynamic information such as modes of components and functions

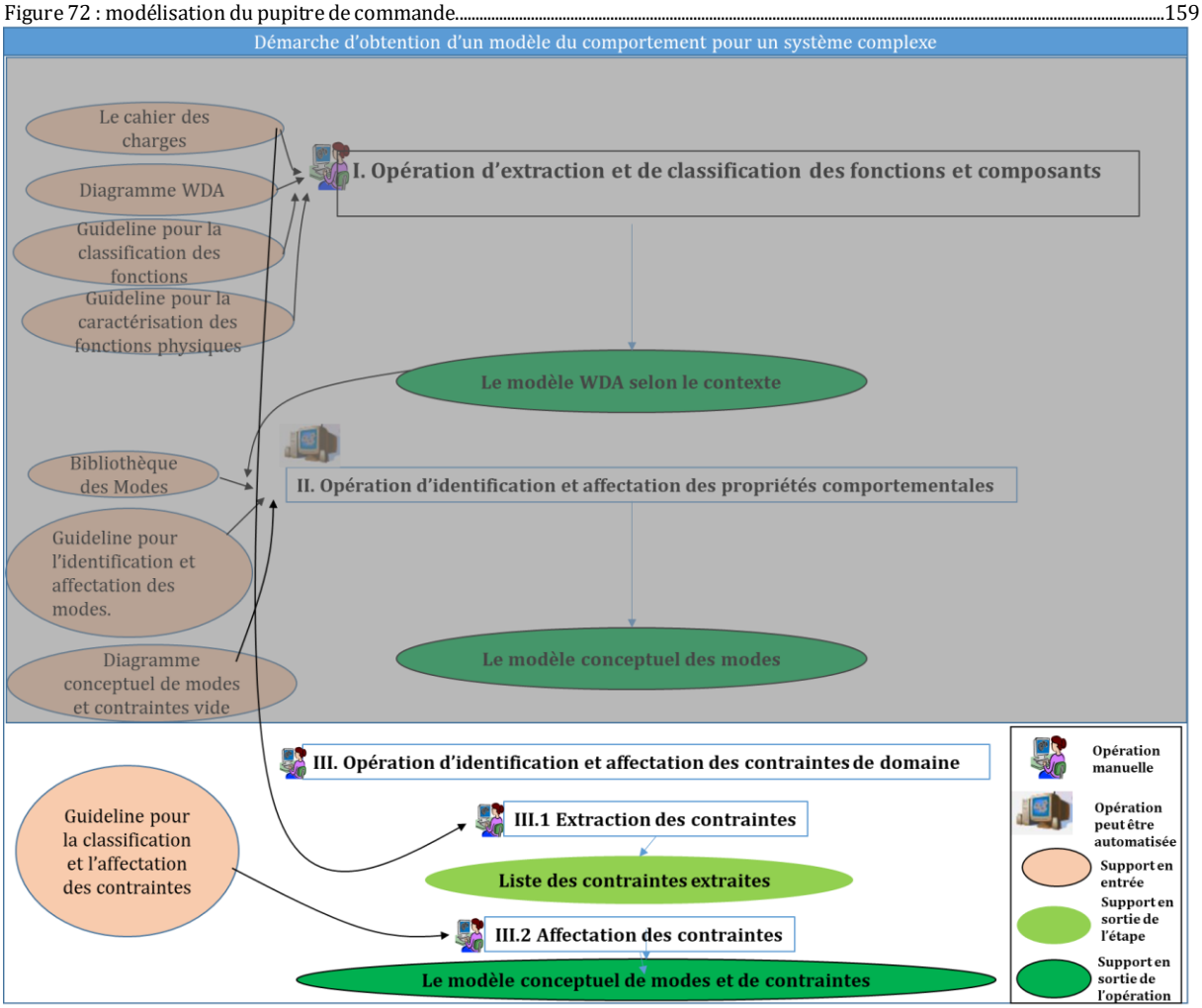
Publications scientifiques

MESLI-KESRAOUI, Ouissem, OUHAMMOU, Yassine, GOUBALI, Olga, et al. Towards a Model-Based Approach to Support Physical Test Process of Aircraft Hydraulic Systems. In : Model and Data Engineering: 10th International Conference, MEDI 2021, Tallinn, Estonia, June 21–23, 2021, Proceedings 10. Springer International Publishing, 2021. p. 33-40.

MESLI-KESRAOUI, Ouissem, Emmanuel GROLLEAU, Yassine OUHAMMOU, Pascal BERRUET, Soraya KESRAOUI, Patick GIRARD. Resource Constraint Scheduling on two dedicated machines: Application to avionics. EURO Journal on Computational Optimization : Manuscript number:EJCOMP-D-23-00049 **Under review**

Sommaire

Liste des Figures



Liste des tableaux

Introduction générale

Les systèmes industriels sont généralement des systèmes complexes comportant plusieurs composants et offrant plusieurs fonctionnalités ou fonctions. Vu le nombre important des composants de ces systèmes, dans le monde industriel, des bancs de tests SCADA sont souvent développés pour les tester. Un banc de test SCADA est un système SCADA qui a pour rôle de réaliser des tests sur un système.

Pour répondre à un objectif de test sur ces bancs, les experts métiers se chargent du lancement des différentes fonctions de test. Pour cela, ils définissent un ordre précis de ces fonctions. Toutefois, certaines contraintes viennent compliquer la définition de cet ordre. En effet, les fonctions peuvent se réaliser en parallèle ou en séquentiel. D'autre part, certaines fonctions se partagent les mêmes composants et sont régis par les mêmes contraintes physiques comme la pression, le débit, etc. La définition manuelle de cet ordre peut être source d'erreur et également très fastidieuse.

D'autre part, pour la réalisation de chaque fonction, les experts ont la charge d'identifier les séquences d'actions élémentaires permettant la réalisation de la fonction. Par exemple, afin de réaliser une fonction de transfert d'eau d'une soute à une autre, les experts se chargent de réaliser la séquence d'actions élémentaire suivante : ouvrir les vannes, enclencher la pompe, choisir la soute de départ/d'arrivé, etc. Ces actions sont des commandes qui permettent de manœuvrer les composants physiques du système. Toutefois, vu le nombre important des composants d'un système SCADA et les contraintes opérationnelles portant sur leur utilisation, la réalisation de ces actions peut devenir difficile et source d'erreur.

La dernière évolution industrielle dans le domaine (industrie 4.0) préconise l'utilisation des nouvelles technologies, telles que l'automatisation afin de gagner en efficacité et de réduire les erreurs. Par conséquent, l'optimisation des bancs de tests SCADA par l'utilisation de l'automatisation peut apporter des solutions efficaces pour ces problématiques. En effet, sur le plan du lancement des fonctions à réaliser, l'utilisation des algorithmes d'optimisation permet d'une part, de chercher automatiquement un ordonnancement des fonctions à réaliser et d'assister ainsi les experts. D'autre part, l'ordonnancement calculé permet une vraie optimisation de la production à travers une minimisation du temps de la production, l'utilisation de ressources ou également la maximisation du profit. Sur le plan identification des séquences de commande élémentaires, l'automatisation peut être une vraie aide aux experts. Plus du gain du temps considérable par la génération automatique, l'automatisation permet de prendre en compte plusieurs contraintes opérationnelles et de proposer des séquences de commande plus sûres.

Contexte

Segula Technologies et ses partenaires académiques travaillent depuis quelques années sur des projets pour faciliter la conception et l'exploitation des systèmes SCADA. Des travaux portant sur la génération automatique des applicatifs de contrôle commande (Bignon, 2012a), l'aide à la spécification fonctionnelle (Goubali, 2017), la vérification formelle des modèles métiers (Mesli 2017) et la validation par simulation des codes de commande générés (Prat, 2017) ont été proposés dans ce cadre. Ces travaux se placent dans les premières phases de conception et apportent des solutions intéressantes pour garantir la qualité des systèmes SCADA. Toutefois, lors de l'exploitation du système SCADA, les problématiques d'optimisation apparaissent.

Nos travaux, réalisés dans le cadre d'une thèse CIFRE entre Segula Technologies et les deux laboratoires : LabSTICC de Lorient (UMR 6285) et LIAS de Poitiers (EA 6315), ont pour objectif d'apporter des solutions pour l'optimisation des bans de tests SCADA.

Objectif de la thèse

Dans le but de proposer une solution complète permettant l'optimisation des bancs de tests SCADA, cette thèse a pour objectif de répondre à trois questions de recherches. La première porte sur la proposition d'un ordonnancement mathématique pour un banc de tests SCADA de l'entreprise. La deuxième tente de générer automatiquement les séquences de commande permettant la réalisation de cet ordonnancement et enfin la dernière , propose une première piste pour l'outillage de toute la solution à travers la définition d'un XDSML.

Structuration du manuscrit

Dans le cadre de cette thèse une solution pour l'optimisation et la génération de séquences opérationnelles des bancs de tests SCADA est proposée. Ce manuscrit est structuré en trois grandes parties et 8 chapitres.

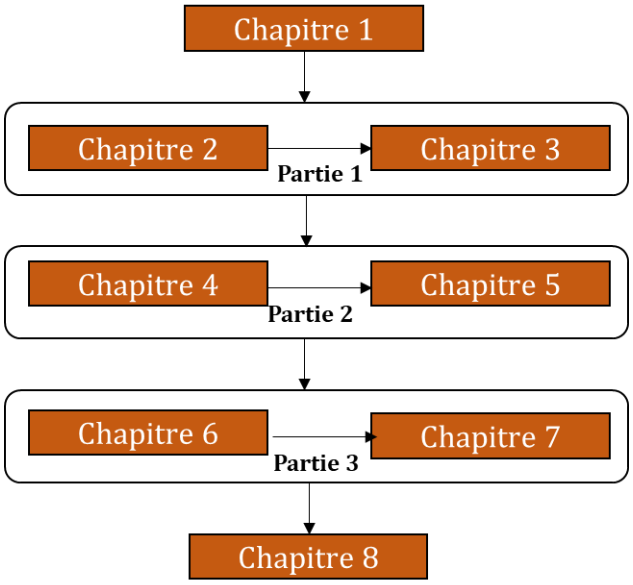


Figure 1 : Organisation du manuscrit.

Le premier chapitre présente le contexte industriel ainsi que les problématiques et verrous scientifiques de la thèse. Il finit par présenter la démarche générale proposée dans le cadre de cette thèse.

Dans **la première partie,** nous proposons des algorithmes et procédures pour ordonnancer l'exécution des fonctions d'un banc de tests SCADA. Cette partie comporte deux chapitres. Le chapitre 2 présente un état de l'art sur les algorithmes d'ordonnancement. Le chapitre 3, présente notre contribution qui porte sur l'ordonnancement de deux machines parallèles avec une ressource partagée et des précédences entre les tâches de chaque machine.

La deuxième partie porte sur la génération de séquences de commande permettant l'exécution opérationnelle de l'ordonnancement calculé. Deux chapitres composent cette partie. Le chapitre 4 présente un état de l'art sur la gestion des modes d'un système SCADA, les techniques d'obtention d'un modèle opérationnel d'un système SCADA et également les techniques de génération de séquences de commande. Le chapitre 5 présente notre approche, basée sur les modes afin de spécifier, de modéliser et de déduire un modèle opérationnel pour l'exécution de l'ordonnancement sur un banc de test SCADA. Ce modèle opérationnel est ensuite utilisé pour la génération automatique des séquences de commande permettant la réalisation de l'ordonnancement mathématique.

Dans **la troisième et dernière partie,** nous proposons une première solution vers l'outillage de notre approche à travers un XDSML. Le chapitre 6 dresse un état de l'art sur la composition des DSML et également les différents XDSML existants pour les SCADA. Dans le chapitre 7, nous présentons notre XDMSL, proposé pour assister les opérateurs des systèmes SCADA dans le processus d'optimisation de leur système. Ce XDSML, permet d'une part de spécifier toutes les informations statiques du système SCADA telles que le P&ID, les fonctions et également la décomposition fonctionnelle. Il permet également la spécification des informations dynamiques comme les modes des composants et des fonctions.

Enfin, le chapitre 8 revient sur les différentes contributions de cette thèse et présentent quelques perspectives pour les travaux réalisés.

Chapitre1 : Contexte industriel et problématique

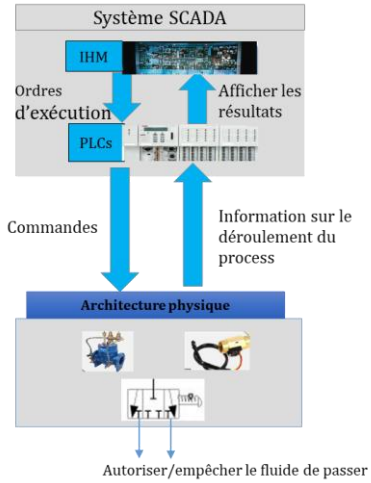
1 Introduction

Le processus d'intégration des systèmes industriels est une phase qui permet de rassembler et de connecter les sous-systèmes d'un système industriel. La phase d'intégration est toujours précédée d'une série de tests, appelés tests d'intégration. Dans le contexte industriel, les tests sont le seul moyen pour vérifier les performances d'un composant ou d'un système dans des conditions normales et sous contraintes (Ott, 2007). Toutefois, l'exécution de ces tests sur les systèmes industriels nécessite souvent le développement des bancs de tests. Un banc de test est un système qui est connecté au système industriel à tester afin de le tester par l'exécution automatique d'un ensemble de tests sur ce système. Dans le monde industriel, les bancs de tests SCADA sont les plus utilisés.

2 Les bancs de test SCADA

Un système SCADA signifie **Contrôle et acquisition de données des processus industriels.** Comme son nom l'indique, ces systèmes ne se résument pas juste au contrôle de données mais aussi à la supervision du procédé via des automates programmables. Le rôle principal d'un système SCADA est de surveiller l'état du procédé et d'envoyer des commandes pour le manipuler (Prat et al., 2015). Un système SCADA peut être divisé en deux parties (Kermani et al., 2021) : une partie matérielle pour l'acquisition des données, la communication et le contrôle et une partie logicielle pour l'enregistrement des données, la visualisation, l'optimisation, la gestion des alarmes, etc.

Concrètement un SCADA est composé d'une **IHM**[1] et des **PLC**[2]. L'IHM fournit des accès de contrôle et de supervision aux opérateurs de type *lancer une commande*, *contrôler l'état du système*, *suivi*, etc. Les PLCs reçoivent les actions des opérateurs sur l'IHM et les traduisent en ordres de commande exécutables permettant de manipuler les composants physiques avec **les actionneurs**. Ces manipulations physiques sont de type : *ouvrir vanne1, fermer vanne2, enclencher pompe,* etc. **Les capteurs** quant à eux, permettent de recenser des informations sur les composants physiques et les remonter à l'IHM en passant par les PLC.



[1] IHM : Interface Homme Machine
[2] PLC : Programmable Logique Controller

Figure 2 : Architecture du système SCADA

Ainsi, un banc de test SCADA comprend une interface de supervision permettant de contrôler le test, de connecter le procédé aux ressources, et des codes de commandes qui permettent l'exécution des ordres de test sur le procédé (Bussenot, 2018). Les tests peuvent être exécutés en différents modes : manuel, semi-automatique, automatique, etc. Ces modes opératoires permettent de spécifier si l'opérateur peut interagir avec le procédé pendant l'exécution de l'essai ou pas. Dans tous les cas, l'opérateur peut arrêter le test si une panne est constatée.

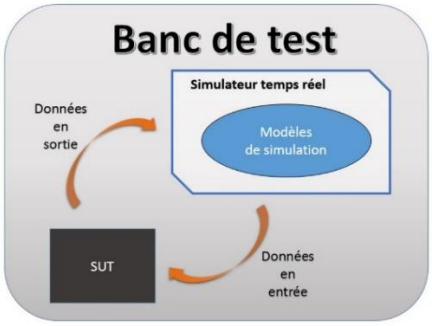


Figure 3 : Architecture d'un banc d'essai (Bussenot, 2018)

Ces bancs de test intègrent souvent un simulateur temps-réel qui permet de simuler les interactions entre les composants physiques et les commandes des opérateurs. Ceci permet de valider les spécifications de conception dans une étape antérieure par exemple.

3 Optimisation des systèmes SCADA

La littérature propose des solutions pour l'optimisation de la production des systèmes SCADA. La production dans un système SCADA consiste à réaliser des actions sur le système permettant d'ouvrir/fermer les composants du système afin de réaliser les objectifs de production. L'optimisation de la production permet de réaliser les objectifs de production d'une manière optimale. L'optimisation peut porter sur la maximisation des gains, ou bien la minimisation de l'utilisation de ressources ou du temps.

L'optimisation d'un système repose sur l'utilisation d'un processus complexe visant à optimiser les activités opérationnelles afin de répondre à des objectifs de production tout en respectant des contraintes du système (Parente et al. 2020). La mise en place d'un processus d'optimisation de production se fait en trois étapes (Figure 4): la planification, la définition de l'objectif de production (ou ordonnancement) et son exécution (Li et al. 2022).

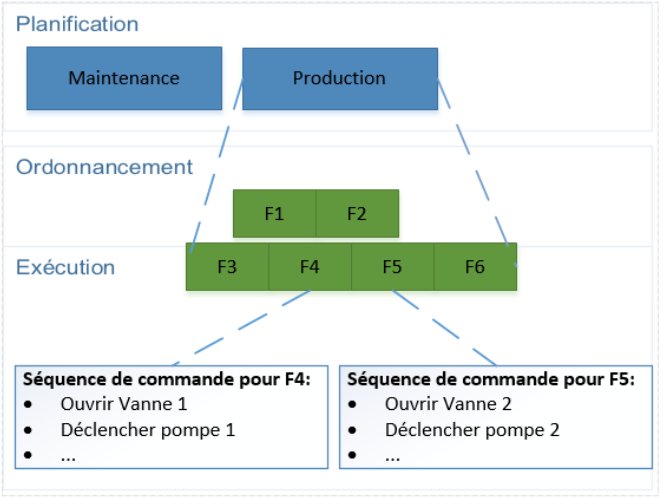


Figure 4 : Processus de production

3.1 La planification

L'étape de planification consiste à planifier les ordres de production à réaliser. Cette étape définit par exemple que les fonctions de production sont réalisées la journée alors que les fonctions de maintenance sont réalisées la nuit pour ne pas perturber la production.

3.2 Ordonnancement

La définition de l'objectif de production décompose chaque ordre de production en plusieurs tâches et les ordonne. Par exemple sur la Figure 4, l'ordre de réalisation de la production conduit à lancer 6 fonctions. Toutefois, plusieurs relations peuvent exister entre les fonctions. En effet, certaines fonctions utilisent des composants différents et peuvent par conséquent, se réalisées en parallèle. A contrario, les fonctions qui partagent des composants sont systématiquement réalisées séparément ou en séquentiel. D'autre part, sur certains systèmes, des relations dites de précédence peuvent exister entre les fonctions. Dans ce cas, une fonction précède toujours une autre fonction.

Le but de cette étape est de définir l'ordonnancement optimal par rapport à un objectif d'optimisation. Cet ordonnancement doit permettre la réalisation de toutes les fonctions du système tout en respectant les contraintes et relations qui existent entre ces fonctions (parallèle, séquentiel, précédence, etc.). Cet ordonnancement peut être définit manuellement par les opérateurs du système SCADA à travers des recettes (ISA-88, 2010) ou calculé mathématiquement par les algorithmes d'ordonnancement.

3.3 Exécution

L'étape d'exécution réalise l'ordonnancement calculé lors de l'étape d'ordonnancement. Les différentes opérations nécessaires à l'exécution des différentes fonctions sont lancées au travers des séquences de commandes élémentaires telles que : ouvrir/fermer les composants du système (vanne, pompe…). Cette exécution doit respecter les états des composants et leur disponibilité, et également des contraintes opérationnelles entre les composants, comme l'ouverture des pompes est lancée après l'ouverture des vannes. En effet, le respect des contraintes opérationnelles est indispensable pour assurer la sécurité des installations et des personnes lors de l'exécution des différentes fonctions.

Ainsi, l'étape de l'ordonnancements spécifie l'objectif de production d'une manière abstraite et indépendante des composants du système. Elle utilise des données abstraites comme le temps total de la réalisation des tâches ou le partage de certaines ressources (Köcher et al. 2022). A contrario, l'étape d'exécution consiste à transformer cet objectif abstrait en un objectif concret et dépendant des composants du système. Dans cette étape, des données plus concrètes comme les opérations utilisées, les composants réalisant ces opérations et les contraintes opérationnelles du système et de son environnement sont utilisées (Köcher et al. 2022).

4 Formalisation du problème industriel

Le cas d'application de cette thèse est un banc de tests SCADA pour le rinçage des systèmes hydrauliques.

4.1 Système hydraulique

Un système hydraulique est un assemblage complexe de composants hydrauliques, électroniques et mécaniques (Figure 5). Ce système utilise un fluide sous pression pour transférer de l'énergie d'un point à un autre (ISO 16431, 2012). Les équipements peuvent être élémentaires ou complexes. Un équipement complexe est composé de deux ou plusieurs équipements

élémentaires. Les lignes sont composées de tuyauterie (tuyaux et de jonctions). Les jonctions permettent de réaliser la liaison entre 2 tuyaux.

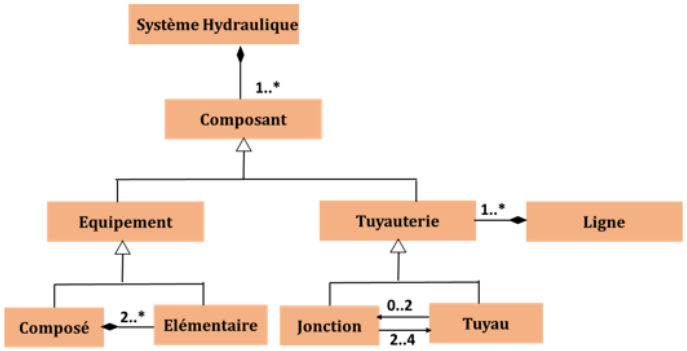


Figure 5 : Caractérisation du système hydraulique.

Chaque équipement a une fonction physique à réaliser. Par exemple : une génération hydraulique est un équipement qui permet de générer du fluide hydraulique, une vanne est un équipement qui permet de laisser/empêcher le fluide de passer. Le système hydraulique est composé de plusieurs connexions entre composants (composés et élémentaires) formant des circuits. Chaque circuit assure la circulation du fluide hydraulique afin de réaliser une fonction de haut niveau (Goubali & al, 2017). Exemple : Transfert de l'eau entre deux soutes. Le fluide hydraulique est guidé par des paramètres qui sont imposés à l'entrée : La pression P est générée par les résistances à l'écoulement du fluide ou par la résistance au déplacement rencontrée par les actionneurs ; le débit Q est généré par les pompes ou les accumulateurs.

4.2 L'objectif du banc de test

Le banc de test a pour objectif le rinçage des circuits hydrauliques. Lors de leur assemblage, les systèmes hydrauliques peuvent être contaminés. La présence de contaminants peut altérer leur fonctionnement et produire des dégâts matériels, humains et environnementaux. Pour réduire ces dommages, le système hydraulique et ses composants doivent être rincés (Xie et al., 2015)pour éliminer les contaminants qui pourraient avoir été créés lors de la fabrication, de l'assemblage et/ou de la maintenance des composants. Le rinçage est l'une des solutions les plus efficaces et rentables pour éliminer les contaminants des circuits hydrauliques. Pour un rinçage efficace, le système est décompressé en boucles (une partie du circuit constituée des équipements et de la tuyauterie) (Deuerlein et al., 2014).

La Figure 6 présente le système hydraulique d'un des clients de Segula Technologies pour lequel , un banc de test pour le rinçage de ce système a été développé par Segula. Le système comporte deux circuits, qui se partagent le même générateur de pression. Toutefois, chaque circuit est constitué d'un ou plusieurs composants.

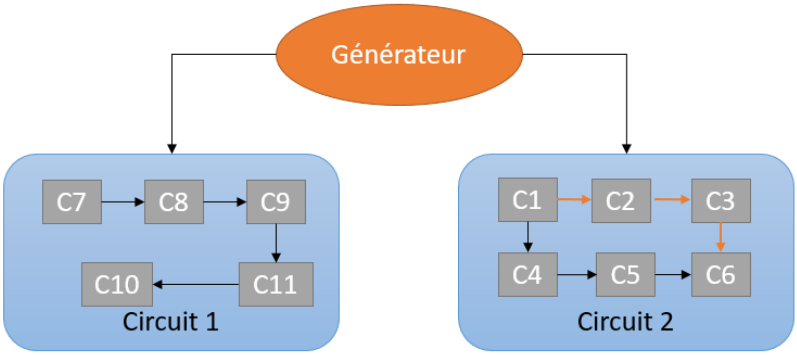


Figure 6. Le système hydraulique à rincer

4.3 Pratiques actuelles de conception du banc de tests SCADA

La conception du banc de tests SCADA pour le rinçage du système hydraulique est réalisée en plusieurs étapes, comme le montre la Figure 7. Ces étapes sont détaillées dans les sections suivantes.

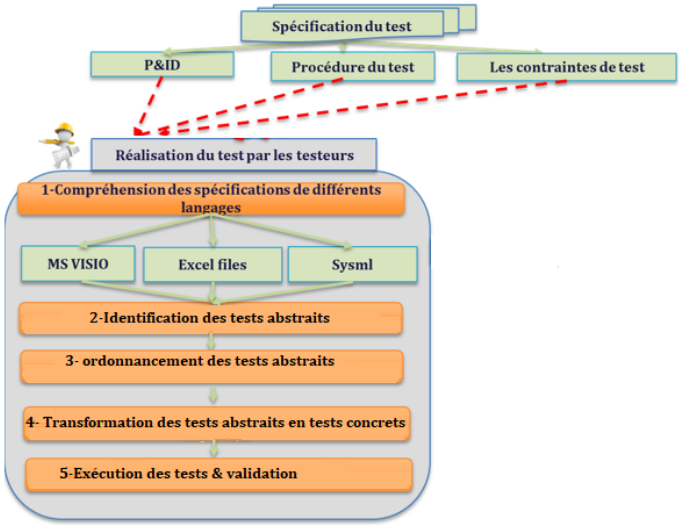


Figure 7 : Etapes de conception des bancs de tests SCADA

4.3.1 Compréhension des spécifications de différents langages

Dans cette étape les concepteurs doivent lire les spécifications fournies par le client. D'abord ils doivent comprendre l'architecture du système hydraulique à tester, ainsi que les procédures de tests et les différentes contraintes exigées par le client. Ensuite ils doivent se référer au P&ID[3] fourni sous format d'un dessin VISIO (Adawiyah et al., 2021). Dans ce document sont fournies les contraintes de longueurs de tuyaux, leurs types, leurs branchements ainsi que d'autres contraintes liées aux équipements tel que les ports d'entrées et sorties utilisées. La Figure 8 présente un exemple de schéma P&ID.

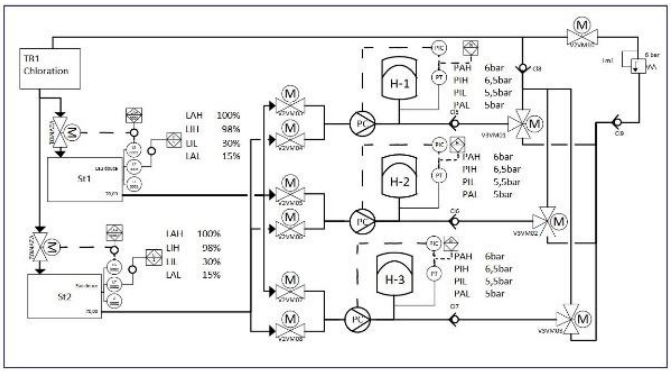


Figure 8 : Exemple d'un schéma P&ID

Une fois que l'architecture organique est comprise et bien spécifiée, les concepteurs définissent l'architecture logique qui représente les fonctions du banc de tests SCADA. Sur le banc de test, la fonction de rinçage est identifiée. Ces spécifications fonctionnelles sont généralement écrites en langage naturel. Des contraintes sont aussi fournies pour le respect des normes, liées au rinçage

[3] P&ID : Piping & Instrumentation Diagram

comme le type de rinçage, le fluide à utiliser, etc. Toutes ces informations issues de plusieurs documents doivent être combinées pour préparer l'étape suivante.

4.3.2 Identification des tests abstraits

Dans cette étape, pour chaque fonction identifiée, les concepteurs doivent définir ses sous-fonctions et également les composants qui permettent de la réaliser. Pour le banc de test par exemple, cela revient à décomposer la fonction du rinçage en plusieurs sous-fonctions : rinçage portion1, rinçage portion2. En effet, le rinçage de tout le système est impossible, le système est alors décomposé en portion ou en chemin. Un chemin ou portion est constitué d'un ensemble d'équipements (pompe, vanne, filtre, etc.) à rincer. On peut voir sur la Figure 6 que sur le circuit 1, un seul chemin est défini pour rincer ce circuit, alors que sur le deuxième circuit, deux chemins sont définis (une portion en couleur orange et une autre portion en couleur noir). Ces chemins sont considérés comme des tests abstraits, qui comportent juste les composants de chaque chemin.

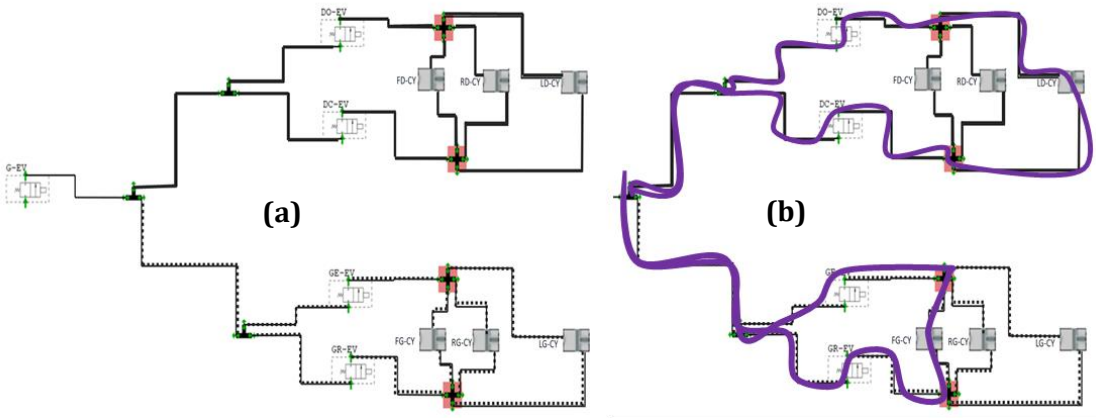


Figure 9 : (a) : Schéma P&ID d'un système hydraulique, (b) : Définition des tests abstraits en violet.

4.3.3 Ordonnancement des tests

Sur ce banc de tests, l'ensemble de portions identifiées est important. Pour gagner en efficacité, les testeurs définissent manuellement un ordre pour l'exécution du rinçage de ces portions. Lors de la définition de cet ordre, certaines contraintes de précédences pour garantir la sécurité de fonctionnement sont prises en compte. De plus, des contraintes de ressources partagées doivent être respectées dans le cas d'exécution de deux portions en parallèles. Cet ordre est défini manuellement, de fait, il n'est pas optimal.

4.3.4 Transformation des tests abstraits en séquences de commande

Une fois que les tests abstraits ont été définis et ordonnés. Les concepteurs doivent convertir ces tests abstraits en séquences de commande par l'ajout des informations opérationnelles de type (ouverture équipement 1 en état 1, ouverture équipement 2 en état 2, fermeture vanne 3, etc.). Ces tests concrets ajoutent de la dynamique en modifiant l'état de chaque composant permettant la réalisation du test abstrait. Ces changements d'états sont obligatoirement soumis à des contraintes opérationnelles pour assurer la sécurité des équipements et des personnes.

4.3.5 Exécution des tests & validation

Dans cette étape, tous les séquences de commande sont exécutées en respectant l'ordonnancement prédéfini. Sur le banc de test SCADA, l'exécution de ces séquences est réalisée

soit manuellement, semi-automatique ou automatiquement. Dans cette étape, la gestion des défauts dans le mode dégradé est aussi prise en compte. Ensuite, chaque résultat d'exécution est comparé à des valeurs permettant de valider l'exécution d'une portion ou pas.

5 Problématiques et verrous scientifiques

L'optimisation d'un banc de tests SCADA consiste à proposer des solutions permettant de calculer automatiquement les séquences de commandes des différentes fonctions du banc. Pour cela, des informations sur le système à tester, ses composants, les contraintes opérationnelles ainsi que l'objectif de test sont nécessaires. Dans la littérature, l'optimisation du processus de production sur un système SCADA repose sur les éléments suivants.

1. *Modèle de test.* Ce dernier modélise l'objectif de la production (ordonnancement) sous la forme d'une séquence abstraite de fonctions. Ce modèle est calculé automatiquement ou défini manuellement.
2. *Modèle opérationnel du système.* Ce deuxième modèle modélise toutes les informations opérationnelles du système, ses composants, ses fonctions ainsi que toutes les interactions et contraintes opérationnelles qui existent entre eux.
3. *Génération automatique de séquences. L*e déroulement du modèle de l'objectif sur le modèle du système permet de vérifier sa faisabilité et de générer les séquences de commandes nécessaires à l'exécution opérationnelle de cet objectif de production, tout en respectant les contraintes du système, décrites dans le modèle opérationnel.

L'objectif de cette thèse est de proposer une solution permettant d'optimiser un banc de test en automatisant le processus de production. Pour un calcul automatique des séquences de commandes, les verrous suivants sont à lever.

5.1.1 Modèle d'ordonnancement

Dans le cadre de cette thèse, l'ordonnancement des tests abstraits est calculé automatiquement. La définition d'un algorithme d'ordonnancement pour le calcul automatique de la séquence de test est une étape indispensable dans un processus d'optimisation. Cet ordonnancement a pour but de fournir la séquence de portions optimale pour d'une part, la réalisation de la fonction de rinçage et d'autre part, le respect de toutes les contraintes du système. Il est à noter que cet ordonnancement est vraiment dépendant du système à tester et des contraintes imposées par ce dernier. En effet, le système hydraulique pour lequel le banc de rinçage a été développé, comporte deux circuits qui se partagent une ressource commune (le générateur). L'ordonnancements doit respecter l'architecture de ce système et également les contraintes et les relations qui existent entre les fonctions du système, comme les contraintes de précédence, de parallélisme et de séquencement entre les fonctions. De plus, l'ordonnancement peut avoir plusieurs objectifs d'optimisation comme : la définition d'un algorithme d'ordonnancement pour le cas industriel traité dans cette thèse.

5.1.2 Intégration des informations opérationnelles

Une fois que l'ordonnancement des fonctions est calculé, il est nécessaire de le convertir en opérations de commande élémentaires telles que : ouvrir composant/fermer composant. Cela se fait par l'ajout des informations et différentes contraintes opérationnelles du système. Pour y parvenir, il est nécessaire de modéliser toutes les informations et les interactions des éléments constitutifs du système SCADA à savoir :

- Les contraintes physiques du système SCADA telles que les températures, les pressions, les débits, etc.

- Les informations liées aux fonctions comme leurs états et les transitions entre ces états. Il est nécessaire également de prendre en compte les configurations de chaque fonction qui spécifient la liste des composants utilisés pour la réalisation de la fonction. D'autre part, il faut prendre en compte les différentes relations qui existent entre les fonctions comme le parallélisme, le séquencement et également les relations de précédence.
- La disponibilité et les états des composants. En effet, un composant peut être utilisé par plusieurs fonctions, à condition que l'exécution simultanée de ces fonctions n'est pas possible. Il est alors nécessaire de vérifier la disponibilité du composant avant de lancer ces fonctions afin de s'assurer qu'il n'est pas utilisé par une autre fonction.

5.1.3 Vérification et validation

Le modèle opérationnel est utilisé pour la génération des séquences. Si ce modèle comporte des erreurs, ces dernières seront propagées dans les séquences générées. Il est alors important de vérifier sa qualité. Le verrou ici consiste à définir les propriétés à vérifier sur le modèle et de s'assurer que le modèle opérationnel obtenu est cohérent et ne comporte pas d'ambigüités.

5.1.4 Génération automatique des séquences de commande cohérentes

Le verrou ici est de générer des séquences de commandes cohérentes et faisables. Toutefois, si les contraintes opérationnelles du système ne permettent pas la réalisation de l'ordonnancement calculé, il est alors nécessaire de définir une stratégie de résolution du problème.

5.1.5 Outillage de la démarche

Afin de faciliter l'utilisation de notre solution par les ingénieurs industriels, son outillage est une étape indispensable.

6 Approche générale proposée

Dans cette thèse, nous proposons une approche qui permet de faciliter l'optimisation des bancs de tests de type SCADA.

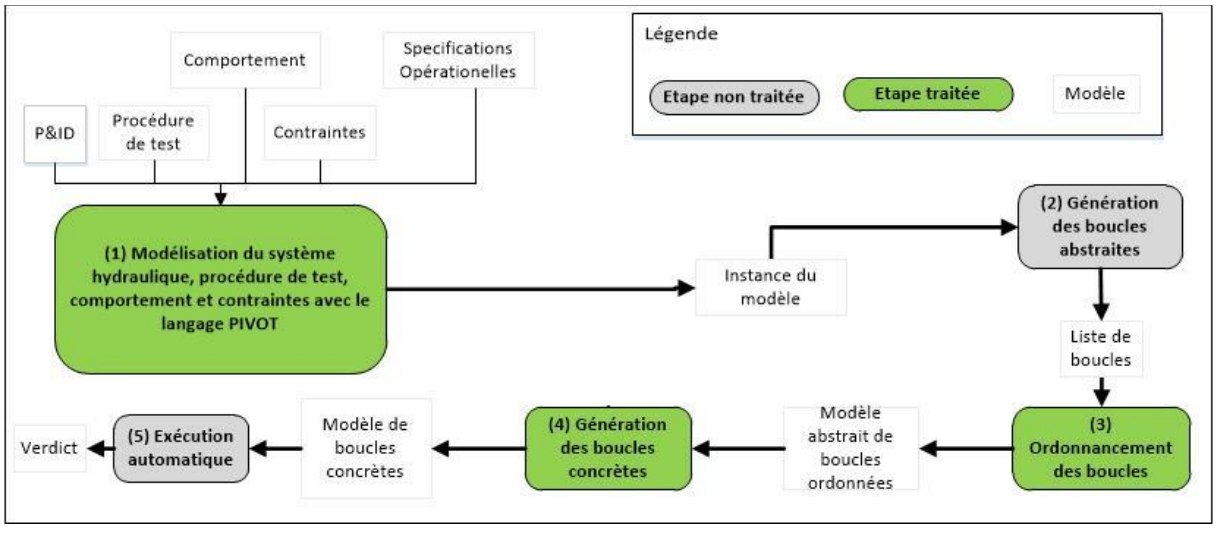


Figure 10 : Approche générale de la thèse.

Les étapes en verts sont des étapes traitées dans cette thèse. Nous proposons une approche qui automatise et optimise ces étapes.

6.1 Modélisation du système

Dans cette première étape, nous modélisons le système hydraulique. Cette modélisation concerne son architecture physique, son architecture logique, ses contraintes opérationnelles et son comportement dynamique. Le but est de pouvoir combiner toutes ces informations dans un seul modèle de domaine, homogène et cohérent. Nous modélisons également les informations opérationnelles qui permettent d'exécuter le modèle de domaine.

6.2 Génération des boucles abstraites

Cette étape permet de calculer les boucles de rinçage. Ce calcul est difficile vu le nombre de contraintes à prendre en considération tel que : la taille de la boucle, le débit nécessaire et les ressources partagées. Le but de cette étape est de calculer chaque boucle à travers ses composants, sa pression/débit et son objectif. Il est clair que si les boucles sont calculées automatiquement, le temps de rinçage est alors inconnu. Cette étape n'a pas été traitée dans ces travaux.

6.3 Ordonnancement des boucles

L'ordonnancement permet de déterminer le temps de début de rinçage de chaque boucle tout en respectant les contraintes de précédences et de ressources partagées. Ce calcul est fait par un algorithme d'ordonnancement de type deux machines parallèles avec contraintes de précédences et ressources partagées. Cet algorithme est déterministe car le temps de rinçage de chaque boucle est connu préalablement, si l'étape deux de notre approche a été faite automatiquement (calcul de boucles abstraites) dans ce cas le temps de rinçage de chaque boucle est inconnu et on aurait utilisé un algorithme d'ordonnancement stochastique.

6.4 Génération des boucles concrètes

Dans cette étape, nous complétons les boucles abstraites par des informations opérationnelles comme l'état de la vanne, l'état de la pompe, etc. Ces ajouts ne suffisent pas pour avoir un modèle cohérent de tout le système, il est nécessaire de respecter les contraintes de cohérence qui permettent de garantir une certaine sécurité et cohérence de notre modèle. Une fois le modèle construit, il peut être utilisé pour générer les séquences d'exécution de ces boucles concrètes.

6.5 Exécution

Dans cette étape, les séquences calculées sont exécutées en utilisant un banc de test SCADA.

7 Conclusion

Dans ce chapitre, nous avons introduit le contexte de la thèse ainsi que les verrous et problématiques rencontrées. Ces derniers portent essentiellement sur l'optimisation de la production d'un système SCADA.

Partie I : Ordonnancement de deux machines dédiées avec des contraintes de précédences locales et une ressource partagée.

Publication scientifique :

MESLI-KESRAOUI, Ouissem, LEDRECK, Loic, GROLLEAU, Emmanuel, et al. Resource constraint scheduling on two dedicated machines: application to avionics. EURO Journal on Computational Optimization, 2024, p. 100093.

Chapitre 2 : Etat de l'art sur l'ordonnancement

1 Introduction

L'architecture physique du système hydraulique traité dans cette thèse, se compose de deux circuits différents, partiellement redondants, et indépendants. Les deux circuits sont alimentés par une énergie hydraulique, générée par la génération hydraulique avec une pression maximale limitée. Afin de tester ces deux circuits, les testeurs découpent l'architecture physique de chaque circuit en portions, appelée boucles. De plus, les contraintes de test exigent que l'ordre des boucles du même circuit respecte des contraintes de précédence où certaines boucles doivent être rincées avant d'autres, afin d'assurer que le fluide du rinçage passe toujours par des circuits propres avant d'atteindre la partie à rincer. Une fois les boucles et leurs précédences spécifiées, ces boucles sont exécutées sur des bancs de test SCADA selon un ordonnancement défini manuellement. En effet, à travers des IHM, les testeurs lancent les ordres d'exécution des boucles en respectant un ordonnancement prédéfini à la main. Les ordres sont transmis aux automates programmables qui permettent d'ouvrir/fermer les vannes et les équipements hydrauliques afin de laisser passer le fluide dans la portion du circuit (boucle) visée par le rinçage. Le processus de rinçage de notre système hydraulique prend plusieurs heures à un jour ou deux.

2 Problématique

Une boucle est une séquence de tuyaux, configurée à l'aide de vannes, de telle sorte que le fluide sous pression puisse être injecté à partir d'un élément source, et collecté à un élément puits. Le rinçage d'une boucle nécessite du temps (en minutes) et de la pression en fonction des propriétés physiques de ses tuyaux. La source de pression est partagée par les deux circuits, il n'est donc pas possible de rincer en même temps deux boucles nécessitant une pression élevée. Les testeurs doivent préciser l'ordre d'exécution des boucles de rinçage en fonction de deux ensembles spécifiques de contraintes.

1) Premièrement, deux boucles concurrentes ne peuvent pas être rincés en même temps si la pression totale requise est supérieure à la pression du générateur.
2) Deuxièmement, pour s'assurer que le fluide hydraulique ne salit pas une conduite déjà propre, des relations de précédences entre les boucles d'un même circuit doivent être respectées. Puisque les circuits sont séparés, les contraintes de précédence sont appliquées aux boucles du même circuit. Les contraintes de précédence proviennent principalement de la topologie du système.

Les boucles peuvent être considérées comme des tâches, qui s'exécutent simultanément sur deux machines dédiées (les deux circuits séparés), mais qui partagent une ressource (la pression). Les tâches sont caractérisées par une durée fixe et une utilisation de la ressource (la pression d'entrée nécessaire pour rincer la boucle).

Un problème d'ordonnancement de ressource permet d'ordonner l'exécution d'un ensemble de tâches sur un ensemble de machines. Chaque tâche est caractérisée par sa durée d'exécution et par sa consommation de ressource. L'objectif de l'ordonnancement ici, est d'ordonner l'ensemble des tâches sur les machines de sorte à ce que la contrainte de ressource soit satisfaite (respectée) et que la durée d'exécution de toutes les tâches (Makespan) soit minimisée. Dans le cas de notre problème, nous avons deux machines dédiées et parallèles avec deux contraintes supplémentaires : la contrainte de la ressource partagée entre les deux machines et la contrainte de précédence locale entre les tâches de la même machine.

Dans la littérature les problèmes d'ordonnancement de ressource de machines parallèles RCPMSP[4] et des machines parallèles dédiées sont multiples et plusieurs formulations du problème RCPMSP ont été proposées. Toutefois, ces formulations ne tiennent pas compte des contraintes de précédence. Les contraintes de précédence n'ont pas été considérées pour RCPMSP sur des machines parallèles dédiées.

Notre problème pourrait être dénoté dans le domaine de l'ordonnancement par la notation

$PD2|res1\cdot\cdot,localprec|Cmax$ (Où PD2 signifie qu'il y a deux processeurs dédiés, c'est-à-dire que les tâches sont déjà placées sur leur processeur, une ressource additionnelle et il existe des précédences locales entre tâches s'exécutant sur la même machine et qu'on cherche à minimiser le makespan) n'a jamais été traité dans la littérature. De fait, il n'existe aucune formulation ni résolution de ce problème. Nous avons donc cherché dans la littérature les problèmes proches, afin de s'inspirer de leur résolution pour résoudre le problème qui nous intéresse.

3 Etude bibliographiques sur les problèmes d'ordonnancement

L'ordonnancement est une branche de la recherche opérationnelle qui vise à améliorer l'efficacité d'une entreprise (Aggoune, 2002). Les concepteurs de systèmes industriels complexes se penchent vers les problèmes d'ordonnancement afin de planifier le fonctionnement d'un système en vue de l'optimiser. L'optimisation des processus industriels permet de réduire le temps de fabrication et/ou les stocks tout en respectant les contraintes liées à la production, et donc de réduire le coût et/ou le *time-to-market* des produits. En raison de leur efficacité à planifier le fonctionnement des systèmes complexes, les problèmes d'ordonnancement ont été largement étudiés sous différents contextes (Aggoune 2002; Blażewicz et al. 2001; Campos Ciro 2015; Pinedo et Hadavi 1992). Les travaux de recherche portant sur la résolution des problèmes d'ordonnancement représentent un pan entier de la recherche opérationnelle, la littérature en est donc très riche. Nous trouvons d'ailleurs plusieurs ouvrages (Baker & Trietsch, 2013; Brucker, 1999; Kan, 2012; Lawler et al., 1993).

Dans cette section nous présentons un état de l'art des problèmes d'ordonnancement, liés à la problématique centrale de la thèse. D'abord nous présentons la démarche d'ordonnancement classique, où pour chaque étape, différents travaux existants sont cités. Ensuite nous abordons les problèmes d'ordonnancement à plusieurs machines parallèles, les problèmes de job shop, les problèmes de machines dédiées et enfin les problèmes avec des contraintes de ressources. Cette section nous permet de nous positionner par rapport aux travaux existants.

3.1 La démarche classique de l'ordonnancement

Un problème d'ordonnancement est caractérisé par n activités (ou *jobs*) $j_i(i = 1, .., n)$ à exécuter sur m machines $m_j(j = 1, .., m)$. Les activités nécessitent l ressources supplémentaires (i.e., en plus de la machine les exécutant) $r_k(k = 1, .., l)$ pour leurs exécutions. La planification des activités sur les machines doit respecter h contraintes $c_u(u = 1, .., h)$. L'ordonnancement répond d'une manière optimale à un critère d'optimisation $(C_{max}, L_{max}, etc.)$.

L'ordonnancement est le fait : de fixer les dates de début et de fin d'exécution de chaque activité j_i sur une machine m_j , d'allouer d'une manière optimale les ressources aux activités, d'assurer la satisfaction des contraintes. Et ce, tout en répondant à un critère d'optimisation (Lawler et al., 1993).

[4] Resource Constraint Parallel Machine Scheduling Problem

La résolution d'un problème d'ordonnancement suit une démarche séquentielle de quatre étapes : la caractérisation, la formulation, le calcul de complexité et la résolution. Notons que chaque étape est importante car elle affecte directement l'étape suivante.

3.1.1 Etape 1 : La caractérisation $\boldsymbol{\alpha|\beta|\gamma}$

(Graham et al., 1979) part du principe que la caractérisation (classification) d'un problème d'ordonnancement affecte sa formulation mathématique et donc sa méthode de résolution. La caractérisation est la classification du problème d'ordonnancement par une notation mathématique très utilisée dans la littérature.

La caractérisation du problème d'ordonnancement est basée sur trois principaux facteurs sous la forme mathématique $\boldsymbol{\alpha|\beta|\gamma}$.

3.1.1.1 Les machines ($\boldsymbol{\alpha}$)

La machine est un dispositif physique (système, sous système, etc.) qui permet d'exécuter une ou plusieurs activités. Le type de machine est le premier champ de caractérisation du problème d'optimisation, noté $\alpha = \alpha_1\alpha_2$.

- Chaque activité j_i est une seule opération et est exécutée sur une machine m_j, la durée d'exécution est ρ_{ij}.

Notation	Signification
$\alpha_1 = \emptyset$	Une seule machine. Chaque activité doit être exécutée sur une seule machine partagée pendant une durée p_j. La machine peut exécuter une seule activité à la fois.
$\alpha_1 = P$	Plusieurs machines parallèle identiques ayant la même vitesse de traitement ; chaque activité est exécutée sur une seule machine pendant une durée opératoire $p_{ij} = p_i\ (i = 1,..,n)$. A chaque instant chaque machine ne peut exécuter qu'une seule activité à la fois.
$\alpha_1 = Q$	Des machines parallèles uniformes (non-identiques), dont la durée opératoire de l'activité dépend de la machine choisie $p_{ij} = p_i / v_j (i = 1,..,n)$ avec : v_j est la vitesse de la machine M_j $(j = 1,..,m)$.
$\alpha_1 = R$	Des machines parallèles non liées. La durée opératoire dépend de toutes les machines $p_{ij} = p_i / v_{ij} (i = 1,..,n)$ avec v_{ij} la vitesse de la machine M_j $(j = 1,..,m)$. Chaque machine ne peut exécuter qu'une seule activité à la fois.

Tableau 1 : Classification des problèmes d'ordonnancement/chaque activité est une seule opération.

Les problèmes d'ordonnancement cités dans le **Tableau 1** permettent d'affecter les activités aux machines puis de décider de leurs moments d'exécution (en non préemptif, peut être caractérisé uniquement par la date de début ou de fin). Dans certains cas, les activités sont affectées au préalable aux machines, l'extension de la notation est faite par l'ajout de la lettre D au champs α_1 pour caractériser un problème de machines parallèles/uniformes/non liées et **dédiées**. L'ordonnancement permet dans ce cas de déterminer les moments d'exécution des activités seulement.

- Le tableau suivant présente un récapitulatif des classifications des problèmes d'ateliers. Chaque activité j_i est un ensemble d'opération (Brucker, 1999).

Notation	Signification

$\alpha_1 = G$	Dans un atelier général (*general model*), chaque activité j_i se compose de plusieurs opérations $\{O_{1i}, \dots, O_{mi}\}$. L'opération O_{ji} doit être exécutée sur la machine μ_{ij} pendant une durée p_{ij}. Chaque activité est exécutée sur une seule machine à un instant donné et chaque machine ne peut exécuter qu'une seule opération à la fois. Dans le cas d'un atelier général, il existe un ordre imposé pour l'exécution des opérations. En effet, chaque opération ne peut commencer que si l'opération précédente a terminé son exécution.
$\alpha_1 = J$	Un atelier de travail (*job shop*) où chaque activité j_i possède une chaine d'opérations $\{O_{1j}, \dots, O_{mj}\}$ à exécuter sur une machine μ_{ij} $(i = 1, .., m_j)$ pendant une durée p_{ij}. Une activité peut ne pas avoir des opérations à exécuter sur toutes les machines. Dans les problèmes d'atelier de travail, on possède d'une contrainte de précédence de la forme : $O_{i1} \to O_{i2} \to \cdots \to O_{i,ni}$ $i = 1, \dots, n$ .généralement $\mu_{i-1j} \neq \mu_{ij}$ $j = 1, \dots, n_i - 1$. On appelle *un job shop avec la répétition de machine* si $\mu_{i-1j} = \mu_{ij}$ $(i = 1, .., m_j)$.
$\alpha_1 = F$	Un atelier de flux (flow shop). Un cas particulier de problème d'atelier. Chaque activité j_i possède une chaine d'opérations $\{O_{1i}, \dots, O_{mi}\}$ à exécuter sur un ensemble de machines $n_i = m$, telle que $\mu_{ij} = M_j$, $i = 1, .., n$ *et* $j = 1, .., m$.
$\alpha_1 = X$	Un atelier mixe est une combinaison de job shop et open shop.
$\alpha_1 = O$	Un atelier ouvert (open shop) est un flow shop mais sans contraintes de précédences. L'ordre est fixé par l'algorithme. Contrairement au problème d'atelier (job shop), dans les ateliers ouverts chaque activité a exactement une opération à exécuter sur chaque machine.

Tableau 2 : Classification des problèmes d'ordonnancement/chaque activité est un ensemble d'opérations.

Les problèmes d'ordonnancement d'atelier sont caractérisés par le fait que chaque activité j_i possède une chaine d'opérations $\{O_{1j}, \dots, O_{mj}\}$. L'affectation des opérations aux machines est une entrée du problème. L'ordonnancement ici permet de déterminer les dates d'exécutions de chaque opération de chaque activité.

Notons que le problème central de la thèse, entre dans la catégorie des problèmes PD2 car nous avons deux machines dédiées (les deux circuits), et des activités qui sont les boucles dont le rinçage est à ordonnancer. Les activités sont physiquement sur un circuit et sont donc affectées à une machine, sans possibilité de changer.

3.1.1.2 Les activités et contraintes ($\boldsymbol{\beta}$)

Les activités sont caractérisées par un ensemble de propriétés et de contraintes $\beta = (\beta_1, \beta_2, \beta_3, \beta_4, \beta_5)$.

Champs $\boldsymbol{\beta_i}$	**Description**
$\beta_1 = pmtn$	Des activités préemptives dont l'exécution peut être découpée en morceaux. La reprise est autorisée sur une autre machine.
$\beta_2 = prec$	Il existe des contraintes de précédences entre des activités $j_i \to j_k$. Dans ce cas, l'activité j_k ne peut commencer que lorsque l'activité j_i a terminé son exécution. Lorsqu'un prédécesseur possède au plus d'un successeur, on écrit *chaines* au lieu de *prec*.

$\beta_3 = res\lambda\sigma\rho$	Cette notation est réservée aux problèmes d'ordonnancement avec ressources additionnelles. Les champs $\lambda\sigma\rho$ désignent respectivement le nombre de ressources, la valeur maximale des ressources et la consommation maximale des activités. Un point à la place d'une des trois lettres signifie « quelconque ».
$\beta_4 = d_i$	L'échéance (*deadline*) d'exécution est spécifiée. Il faut donc exécuter chaque activité avant son échéance d_i.
$\beta_5 = p_i$	Si p_i=1 ceci veut dire que chaque activité a un temps d'exécution égal à une unité d'exécution. Si toutes les activités ont le même temps d'exécution p, on écrit p_i=p. Si les durées d'exécution ne doivent pas dépasser une borne c alors on écrit $p_i \leq c$. Dans le cas où les durées appartiennent à un ensemble P de valeurs fixées, on écrit $p_i \in P$.

Tableau 3 : Classification des propriétés des activités et contraintes.

Dans le problème considéré, les activités considérées sont **non-préemptives**. L'exécution d'une activité requiert une durée de préparation (changement des vannes) qui n'est pas spécifiée par les experts. Ainsi, même s'il pourrait être possible de préempter une activité en changeant l'état des vannes, la durée des activités donnée intègre le changement d'état des vannes et le rinçage, sans distinction de l'un par rapport à l'autre. Nous n'avons donc pas les informations permettant de connaître le délai de préparation (*setup time*). Il existe des **précédences** entre certaines boucles de rinçage. Il existe une **ressource** (générateur de pression) utilisée par le rinçage des deux circuits en parallèle. Nous discuterons plus loin dans le manuscrit des valeurs $\sigma\rho$. Nos opérations ne sont pas munies d'échéances individuelles et les durées de rinçage peuvent être quelconques.

3.1.1.3 L'objectif de l'optimisation (γ)

Le troisième champ de caractérisation $\gamma \in \{f_{max}, \sum f_i\}$ définit les objectifs d'optimisation. Pour chaque activité j_i, on peut calculer :

- Le temps de fin : c_i
- Le retard: $L_i = c_i - d_i$
- La lenteur : $T_i = \max\{0, c_i - d_i\}$
- L'indice de pénalité : $u_i = \begin{cases} 1 \; si \; c_i \geq d_i \\ 0 \; sinon \end{cases}$
- Les critères d'optimisation les plus utilisés sont : $f_{max} \in \{C_{max}, L_{max}\}$

  Où : $f_{max} = max_i\{f_i(C_i)\}$ avec : $f_i(C_i) = c_i \; ou \; L_i$

Exemple : $J|premp|Cmax$ est la notation du problème d'ordonnancement d'ateliers (job shop) avec des activités préemptives où on cherche à minimiser le makespan (date de fin, c'est-à-dire durée totale d'exécution en supposant que l'on commence à l'instant 0).

Dans cette thèse, nous cherchons à minimiser la durée totale du rinçage, donc à minimiser le makespan.

3.1.2 Etape 2 : La formulation

La formulation est l'écriture mathématique du problème d'ordonnancement sous forme d'équations. Trois facteurs influencent la formulation : la caractérisation du problème, le type de données ; et le type des variables de décision.

3.1.2.1 Le facteur de la caractérisation

La caractérisation du problème détermine son modèle mathématique en termes de type d'équations. Afin de mieux expliquer ce facteur, nous dressons trois équations du calcul de Makespan pour trois problèmes différents (Unlu & Mason, 2010): le premier est le problème de minimisation de Makespan de machines parallèles (P), le deuxième est le problème de minimisation de Makespan de machines uniformes (Q) et le troisième est le cas de machines non liées (R) .

$$Minimise\ C_{max}$$

$$C_{max} \geq \max\{r_i\}\ \forall\ i = 1,..,n$$

Équation 1 : Calcul du Makespan dans le cas de machine parallèles identiques (P).

$$Minimise\ C_{max}$$

$$C_{max} \geq \max\left\{\frac{d_i}{v_j}\right\}\ \forall\ i = 1,..,n.\ j \in J$$

Équation 2 : Calcul du Makespan dans le cas de machines parallèles non-identiques (Q).

$$Minimise\ C_{max}$$

$$C_{max} \geq \max\left\{\frac{d_i}{v_{ij}}\right\}\ \forall\ i = 1,..,n \quad j = 1,..,m$$

Équation 3 : Calcul du Makespan dans le cas de machines parallèles non-liées (R)

Dans le cas de machines parallèles identiques (P), les activités ont des durées d'exécution $p_{ij} = p_i$. Le Makespan C_{max} est le maximum des dates de fin de la derrière tâche (r_i) de toutes les activités. Comme montré par l' Équation 1, la valeur de C_{max} n'est pas influencée par la vitesse de la machine. Contrairement à la formule du C_{max} dans l'Équation 2, les activités ont des dates de fin, qui dépendent de la machine choisie pour l'exécution. De fait, pour chaque activité i, sa date de fin r_i dépend de la vitesse v_j de la machine j, donc : $r_i = \frac{d_i}{v_j}$. Dans ce cas le Makespan C_{max} est le maximum des dates de fin r_i de toutes les activités, divisées par la vitesse $v_j$de la machine j.

Dans l'Équation 3, la date de fin r_i de chaque activité i est influencée par les vitesses de toutes les machines ($r_i = \frac{d_i}{v_{ij}}$). C'est à dire que la vitesse de chaque machine j à une vitesse v_{ij} différente d'une tache à une autre. Le Makespan est le maximum des dates de fin divisées par les vitesses de chaque machine.

Ces trois équations montrent que pour la même fonction objectif C_{max} (calcul du Makespan), il existe différentes formules d'équations. Ces équations sont directement influencées par la caractérisation du problème d'ordonnancement traité.

3.1.2.2 Le facteur de type de données

Le deuxième facteur qui influence la formulation des problèmes d'ordonnancement est le type de données. Il existe trois types de données. Les données linéaires, les données non linéaire et les données mixtes.

Un programme linéaire (PL) permet de maximiser/minimiser une fonction linéaire, sous des contraintes elles-mêmes linéaires. Si les variables de décision x_i sont limitées à des entiers, le programme est appelé *Un programme linéaire en nombres entiers* (*PLNE)*. Si les variables x_i ne peuvent prendre que les valeurs 0 ou 1, le programme linéaire entier correspondant est appelé *programme linéaire binaire*.

$$minimiser\ z(x) = c_1x_1 + \cdots + c_nx_n$$

$$s.c. \qquad a_{11}x_1 + \cdots + a_{1n}x_n \leq b_1$$

$$\vdots$$

$$a_{m1}x_1 + \cdots + a_{mn}x_m \leq b_n$$

$$x_i \geq 0, \forall\ i = 1,..,n$$

Algorithme 1 : Forme canonique d'un programme linéaire (Brucker, 1999)

Dans l'Algorithme 2, il existe m contraintes et n variables linéaires. La fonction objectif est aussi linéaire. Les sorties de la résolution de ce programme est l'ensemble des variables de décision x_i qui seront des variables continues. La résolution de ce type de formulation s'obtient en temps polynomial de la taille de la formulation.

Dans un programme non-linéaire, les données sont non-linéaires. Un programme linéaire contenant à la fois des variables continues et des variables discrètes est un programme mixte en nombres entiers (MILP).

$$\max \sum_{i=1}^{n} c_i x_i + \sum_{j=1}^{m} d_j z_j$$

$$s.c. \qquad \sum_{i=1}^{n} a_{ki}x_1 + \sum_{j=1}^{m} a'_{kj}z_j \leq b_k \quad k = 1,..,m$$

$$x_i \in \{0,1\}, \forall\ i = 1,..,n$$

$$z_j \in R_+ \quad , \forall\ j = 1,..,m$$

Algorithme 2 : Exemple d'un programme MILP (Maqrot, 2019).

Les variables de décision x_i sont des variables discrètes, contrairement aux variables de décision z_j qui sont des variables continues. D'une façon générale, la résolution d'un MILP peut utiliser une durée exponentielle en fonction du nombre de variables discrètes.

Les deux algorithmes Algorithme 2 et Algorithme 1 illustrent deux formulations différentes induites directement par le type de données utilisées pour représenter le problème (linéaires, mixtes).

3.1.2.3 Le facteur du type des variables de décision

Les problèmes d'ordonnancement peuvent être formulés de différentes manières selon le type des variables de décision nécessaires à leur formulation. En d'autres termes, les variables de décision peuvent permettre de représenter à chaque unité de temps t l'activité i qui occupe la machine j, ou bien plutôt les relations de précédences entre les activités ou le rang dans lequel l'activité est ordonnée dans chaque machine. Dans les problèmes d'ordonnancement avec ressources, les variables de décision permettent de formuler un programme MILP de trois manières différentes. La première est la formulation indexée sur le temps, la deuxième est la formulation disjonctive et la troisième est la formulation basée sur le rang.

3.1.2.3.1 Variables de décision temporelles

Dans cette formulation, le temps d'exécution est découpé en morceaux où chaque morceau est égal à une unité de temps. La variable de décision x_{ijt} est une variable booléenne qui est à *vrai* si l'activité i commence son exécution sur la machine j à l'instant t. Les variables de décisions x_{ijt} sont appelées aussi des *variables temporelles*.

En se basant sur la formulation basée sur le temps, donnée par (Kondili & Sargent, 1988), l'Équation 1 est écrite de la forme :

$$C_{max} \geq \sum_{t=1}^{H} t.x_{ijt} \quad \forall\, i = 1,..,n\,, \forall\, j = 1,..,m$$

$$H = \sum p_{ij},\ i = 1,..,n.\ \ j = 1,..,m$$

Équation 4 : Calcul du Makespan par des variables temporelles (Kondili & Sargent, 1988).

Minimiser l'Équation 4 permet de fournir en sortie la valeur du Makespan C_{max} ainsi que toutes les variables temporelles x_{ijt} pour chaque activité i pour tout instant $t \in H$ dans la machine j. H est une borne supérieure de temps égale à la somme de toutes les durées d'exécution des activités comme si elles étaient toutes exécutées séquentiellement (le pire des cas).

3.1.2.3.2 Variables de décision disjonctives

L'ordre linéaire des activités est exprimé par les variables de décisions de séquence $x_{ii'j}$. Les variables de décisions de séquence sont aussi appelées *variables de séquençage*. Une variable de séquençage est à *vrai* si l'activité i' commence après la fin de l'activité i dans la machine j. Toute variable $x_{ii'j}$ est à 1 si les deux activités i et i' se précédent et 0 sinon. Dans le cas où il y'a une précédence ($x_{ii'j} = 1$) les deux activités sont séquencées par leurs dates de début ($s_{i'j}$ *et* s_{ij} respectivement) et par leurs durées d'exécution (p_{ij} *et* $p_{i'j}$ respectivement).

$$x_{ii'j} = \begin{cases} 1\ alors\ s_{i'j} \geq s_{ij} + p_{ij} \\ 0 \end{cases} \qquad \forall\, i, i' \in n$$

Équation 5 : Formule de séquençage en utilisant des variables binaires (Šeda, 2007a).

D'une façon générale, il y a beaucoup moins de variables que dans le cas d'utilisation de variables temporelles. Ainsi, l'utilisation de la formulation disjonctive permet d'alléger la formule du calcul du Makespan comme suit :

$$C_{max} \geq s_{ij} + p_{ij} \quad \forall\, i = 1,..,n$$

Équation 6 : Calcul du Makespan par des variables entières dans la formulation disjonctive (Šeda, 2007a).

(Blazewicz et al., 1991) propose de formuler les variables de séquençage par de simples variables entières s_{ij} (date de début de l'activité i dans la machine j) pour exprimer le séquençage entre activités sans avoir à utiliser les variables binaires $x_{ii'j}$. Dans l'Équation 7, qui est en général linéarisée en utilisant un *big M*, l'activité i' ne peut commencer que si l'activité i a terminé son exécution ou bien l'activité i ne peut commencer qu'après l'activité i'.

$$s_{i'j} \geq s_{ij} + p_{ij}\ \ \boldsymbol{ou}\ s_{ij} \geq s_{i'j} + p_{i'j}\,,\ \ \forall\, i, i' \in n$$

Équation 7 : Formule de séquençage en utilisant des variables entières..

3.1.2.3.3 Variables de décision de rang

Dans cette formulation, le raisonnement est concentré sur la machine. En effet, chaque machine possède des positions selon le nombre d'activités à planifier. L'ordonnancement dans ce cas, permet de planifier les activités dans la machine et de définir dans quelle position, chaque activité a été planifiée. La variable binaire x_{ijk} est à 1 si l'activité i est ordonnancée dans la machine j à la position k et 0 sinon. De fait la formule du calcul du Makespan devient comme suit :

$$C_{max} \geq \sum_{i \in n} s_{ij} + p_{ij}\, x_{ijk} \quad \forall\, j = 1,..,m, k = 1,..,n.$$

Équation 8 : Calcul du Makespan par des variables binaires de rang (Wagner, 1959).

Dans leurs travaux (Ku et Beck 2016), les auteurs ont prouvé pour un problème d'ordonnancement d'atelier que la formulation disjonctive est plus légère en terme du nombre de variables binaires et contraintes que les deux autres formulations (formulation par des variables temporelles et formulation par des variables de rang). Les travaux de (Roselli et al., 2018) prouvent que la formulation disjonctive est la plus facile à résoudre (en terme de temps de résolution) par rapport à la formulation temporelle et la formulation par rang.

3.1.3 Etape 3 : La complexité

Dans la littérature, il existe plusieurs problèmes d'ordonnancement. Certains sont considérés comme *simples* à résoudre et d'autres sont considérés comme *difficiles*. Le calcul de complexité permet de déterminer la simplicité ou la difficulté de résolution d'un problème d'ordonnancement (Brucker, 1999). Dans la théorie de la complexité de base, il existe différentes catégories de problèmes d'ordonnancement. Les problèmes polynomiaux P nécessitent un nombre d'étapes de résolution polynomial en fonction de la taille du problème en entrée. Les problèmes Non déterministes Polynomiaux NP sont des problèmes dont une solution connue peut se vérifier en temps polynomial, et parmi ces problèmes on identifie les problèmes NP-complet, classe qui définit une équivalence entre plusieurs problèmes réputés les plus durs de la classe NP. Les problèmes dont le complémentaire est NP ou NP-complet sont dits respectivement co-NP et co-NP-complets. On distingue aussi les problèmes au moins aussi durs que les problèmes NP-complets (resp. co-NP-complets) mais dont on n'est pas sûr qu'ils soient dans NP en les classant en problèmes NP-difficiles (resp. co-NP-difficiles). La classification d'un problème d'ordonnancement dans l'une de ces catégories nécessite la classification de son *problème de décision.*

Le problème de décision associé à un problème d'ordonnancement permet de répondre par « *oui* » ou « *non* » à la question « *pour une instance donnée, peut-on trouver une solution S telle que la valeur de la fonction objectif f ne dépasse pas la borne k ? ($f(S) \leq k$)*(Brucker, 1999). Tandis que le problème d'ordonnancement exige une réponse à la question « *trouver une solution qui répond à une fonction objectif f*». De fait, il est possible d'analyser la difficulté d'un problème d'ordonnancement en analysant son problème de décision (Oulamara, 2009).

- Un problème de décision est P s'il existe un algorithme déterministe[5] qui peut le résoudre en un temps polynomial.
- Un problème de décision est NP est un problème pour lequel la réponse « *oui* » peut être décidée par un algorithme non-déterministe en un temps polynomial par rapport à la taille d'instance. De façon équivalente, on peut le voir comme un problème pour lequel une solution donnée se vérifie en temps polynomial.
- La réduction d'un problème A_1 en un problème A_2 est polynomiale, s'il existe une fonction g qui transforme une instance I_1 du problème A_1 à une instance I_2 du problème A_2 telle que la réponse de décidabilité x est « oui » pour I_1 si et seulement si la réponse est « oui » pour $g(I_1)=I_2$ (Brucker, 1999). De fait, un problème de décision A est NP-Complet s'il est de la classe NP (Oulamara, 2009) et tous les problèmes NP peuvent être réduits en A.

La théorie de complexité (Garey & Johnson, 1979) permet de déterminer si un problème d'ordonnancement est solvable en temps polynomial ou non. De fait, si un problème d'ordonnancement est polynomial alors l'algorithme qui permet de le résoudre devra être d'une complexité algorithmique polynomiale. Dans le cas contraire, où le problème est difficile alors sa

[5] Voir dans (Garey & Johnson, 1979) pour la définition d'algorithme déterministe et non déterministe.

complexité algorithmique accroit exponentiellement avec le nombre d'activités à planifier et le nombre de machines (Aggoune, 2002) sauf si P=NP.

L'étude de complexité d'un problème d'ordonnancement permet de décider de la méthode de résolution. En effet si un problème est difficile, l'utilisation des méthodes approchées comme les heuristiques est envisageable pour éviter l'explosion combinatoire.

3.1.4 Etape 4 : la résolution

La quatrième étape de la démarche d'ordonnancement est la résolution. En effet, il existe deux méthodes de résolution des problèmes d'ordonnancement. Les méthodes exactes et les méthodes approchées.

3.1.4.1 Les méthodes exactes

Les méthodes exactes permettent de donner la solution optimale à un problème d'ordonnancement en parcourant l'espace des solutions d'une manière implicite. Parmi les méthodes exactes les plus connues dans la littérature, nous citons *Les méthodes arborescentes*, ces méthodes consistent à créer des arbres de recherche de solution, comme la méthode de *séparation et d'évaluation (*Branch&Bound) (Land et Doig 1960). Cette méthode consiste à construire un arbre de recherche en partitionnant l'espace de recherche de solutions en un ensemble de sous-problèmes. Chaque sous-problème est traité de manière élémentaire.

Nous considérons le problème d'optimisation P suivant :

$$Min\ f(x)$$

Sous contraintes : $x \in F$

Tel que : F est l'ensemble des solutions admissibles.

L'application d'un algorithme de Branch&Bound nécessite plusieurs étapes.

- *Etape 1 Initialisation* : Calculer une borne supérieure U.

$$U = +\infty$$

Le calcul d'une borne supérieure du problème peut être fait par une heuristique.

- *Etape 2 Branch :* décomposer le problème P en K sous problèmes :

Pour chaque sous-problème P_k :

$$\min f(x), x \in F_K$$

Décomposer l'espace de solutions :

$$F = F_1 + F_2 + .. + F_K$$

Soit x_k^* la solution optimale du problème P_k. De fait, x_k^* est une solution admissible pour le problème P.

Si une solution x_i^*, tel que : $f(x_i^*) \leq f(x_k^*), \forall\, k \in 1..K$. alors x_i^* est la solution optimale pour le problème P.

- *Etape 3 Bound :* calcul des bornes inférieures

On calcule une borne inférieure b_k en relâchant certaines contraintes du problème, telle que :

$b_k \leq f(x), \forall x \in F_k$.

- Etape 4 arrêt/suppression : suppression des sous-problèmes inadmissibles
  - Si un sous-problème P_k est inadmissible, alors le supprimer et supprimer ses fils.
  - Si $U \leq b_k$, alors éliminer le sous-problèmes P_k.
  - Soit y une solution admissible du problème P. Si $f(y) \leq b_k \leq f(x), \forall x \in F_k$ alors éliminer le sous-problème P_k.
  - Si tous les sommets ont été visités : alors arrêter.

La méthode du Branch&Bound est efficace mais généralement non polynomiale, de plus un sous-problème peut être aussi difficile à résoudre que le problème initial. De fait, cette méthode a été améliorée par (Mitchell, 2002) en proposant une méthode appelée Branch&Cut qui consiste à ajouter en plus de la méthode de séparation et évaluation, des contraintes au programme linéaire afin de le rapprocher des solutions.

Une autre méthode de résolution exacte des problèmes d'optimisation sont les méthodes dynamiques. Le principe des méthodes de résolution dynamique ou *la programmation dynamique* (Bellman & Kalaba, 1957) a été proposé afin de faciliter la recherche d'un chemin dans un graphe. En effet, elle vient pour répondre au problème de lenteur des méthodes existantes car on a tendance à visiter des nœuds plusieurs fois. La programmation dynamique permet d'éviter de passer par des nœuds plusieurs fois en mémorisant l'information de chaque nœud et ceci permet de l'utiliser à chaque fois qu'on a besoin pour calculer une solution d'un sous-problème. Un problème d'optimisation P est découpé en K sous-problèmes $P_1 \dots P_K$. La solution optimale du problème P est obtenue par l'ensemble de décisions et solutions des sous-problème $P_1 \dots P_K$.

La dernière méthode exacte que nous présentons est la *méthode linéaire* ou la *programmation linéaire.* Cette méthode permet de modéliser le problème d'optimisation sous forme d'inégalités linéaires. Chaque solution x de l'ensemble de solutions S permet de minimiser ou maximiser un certain cirière linéaire f. Une instance d'un problème linéaire peut s'écrire de la façon suivante :

$$\min cx$$

$$Sujet \text{ à:}$$

$$Ax \leq b$$

$$x \in R_+^n$$

$$c \in R^n , b \in R^m \ et \ A \ \in R^{mm}$$

Si les variables de décisions x_i sont limitées aux entiers, le problème est appelé problème linéaire en nombre entiers. Si les variables sont binaires (0 ou 1) alors le problème est appelé problème linéaire binaire. Dans de nombreux cas, le problème nécessite des variables binaires entières et des variables réelles. Ces problèmes sont appelés des problèmes linéaires mixtes. En général, les méthodes de résolution exactes sont moins performantes que les problèmes d'optimisation complexes qui intègrent des variables binaires ou entières (Aggoune, 2002).

3.1.4.2 Les méthodes approchées

Les méthodes de résolution approchées sont utilisées pour résoudre les problèmes d'optimisation de grandes tailles en diminuant le temps de calcul (Maqrot, 2019). Ces méthodes permettent de donner des solutions, si possible de bonne qualité, mais avec une complexité inférieure à la complexité du problème traité. Certaines méthodes approchées permettent de résoudre les problèmes de façon déterministe, elles sont appelées *heuristiques.* Dans le cas où on utilise des méthodes stochastiques elles sont appelées *méta-heuristiques*.

Les méta-heuristiques peuvent être basées sur la recherche locale (Soriano & Gendreau, 1997) telles que *la méthode tabou* (Glover, 1986). Elles consistent en un ensemble de règles et de mécanismes (Soriano & Gendreau, 1997) permettant de poursuivre la recherche même lorsqu'un optimum local a été trouvé. L'idée de base consiste à utiliser des *mémoires* pour sauvegarder les traces du cheminement passé du processus de recherche afin de guider le déroulement futur.

D'autres méta-heuristiques sont basées sur des populations de solution comme *les algorithmes génétiques* (Coello, 2005). Le principe de cette méthode repose sur le fait que les individus doivent adapter leur comportement pour pouvoir survivre à *une évolution* (Coello, 2005). En effet, chaque individu d'une population représente une solution x, telle que son degré d'adaptation à l'environnement est calculé par la fonction de fitness $f(x)$. Cela permet de faire évoluer une population à une autre population plus forte. Le principe est simple et repose sur les principes de l'évolution. D'abords il faut choisir un codage et générer une population initiale de N individus. Chaque chromosome de cette population est évalué afin de créer une population parent (Maqrot, 2019). Les individus de cette population subiront une opération d'évolution. En effet, un opérateur de croisement est appliqué pour obtenir une population enfant. Puis une opération de mutation est appliquée sur les individus de la population enfant. Cette dernière population est évaluée et permet de remplacer tout ou une partie de la population initiale (Aggoune, 2002).

3.2 Problème d'ordonnancement de machines parallèles identiques

Dans cette section, nous présentons les problèmes d'ordonnancement de machine parallèles identiques. Nous mettons l'accent sur les problèmes avec des contraintes de précédences et de ressources. Pour cela, nous décrivons d'abord ces problèmes puis nous dressons quelques formulations qui permettent de résoudre ces problèmes i) sans contraintes particulières, ii) avec contraintes de ressources puis iii) avec les contraintes de précédences. Nous présentons ensuite quelques résultats de complexité de ces problèmes trouvés dans la littérature. Nous terminons par les méthodes de résolution proposées dans la littérature. Nous utiliserons ici indifféremment les termes job ou tâche pour désigner les travaux à effectuer sur les machines.

3.2.1 Description

Un problème d'ordonnancement de machine parallèles identiques se définit de la manière suivante. Un ensemble de tâches $j_1, j_2, .., j_n$ à exécuter sur m machines parallèles $(M_1, M_2, .., M_m)$. Chaque tâche *j* nécessite une machine à la fois et une durée de p_j unités de temps. Chaque machine M_i ne peut traiter qu'une seule tâche à la fois. La préemption n'est pas autorisée. Résoudre ce problème revient à affecter les tâches aux machines pour assurer leur exécution sous forme d'une séquence d'exécution dans le but de minimiser le Makespan (Cheng & Sin, 1990).

Le problème d'ordonnancement de machines parallèles est particulièrement présent dans les systèmes industriels. Dans le domaine de production, les industriels préfèrent souvent faire fonctionner plusieurs machines parallèles en même temps pour : 1) gagner du temps et 2) si l'une rencontre un problème, une alternative est directement prise par le basculement du fonctionnement à une autre machine. Dans les stations pétrolières, les industriels utilisent plusieurs pompes connectées au même réservoir pour augmenter le débit. Dans notre cas industriel, le rinçage des circuits hydrauliques est réalisé par des bancs de test. Ces derniers sont capables de rincer deux circuits à la fois. Les deux circuits sont liés à une source d'énergie. Le rinçage de deux circuits en même temps permet de réduire le temps de test par rapport à un rinçage séquentiel.

A travers les exemples d'utilisation de machines parallèles identiques, nous pouvons dire que la nature du problème d'ordonnancement de machines parallèles identiques est directement liée au contexte industriel dans lequel ces machines sont utilisées. En effet, chaque domaine d'utilisation peut nécessiter des contraintes et/ou des ressources spécifiques ou pas.

3.2.2 Caractérisation

Les problèmes d'ordonnancement de plusieurs machines parallèles identiques $(M_1, M_2, .., M_m)$ sont désignés par la notation P_m dans le champ α (Graham et al., 1979). Les champs β et γ sont à spécifier selon le cas traité (voir section 3.1.1).

3.2.3 Formulation

La formulation d'un problème d'ordonnancement de deux machine parallèles et identiques peut être réalisée à travers des variables de décisions x_{ij} qui permettent de spécifier si une tâche j est affectée à une machine i ou pas. La formulation est réalisée par un programme linéaire mixte (MILP). Nous présentons trois formulations pour les problèmes de machines parallèles identiques selon le type de contraintes :

- Formulation pour minimiser le Makespan ;
- Formulation pour minimiser le Makespan avec contrainte de précédence ;
- Formulation pour minimiser le Makespan avec contraintes de ressources.

3.2.3.1 Formulation pour minimiser le Makespan

Afin de discuter la formulation des problèmes de machines parallèles pour minimiser le Makespan, nous dressons le problème $P||C_{max}$. Il est formulé comme suit, pour n jobs indicés par j, et m machines indicées par i :

$$minimiser\ C_{max}\ (1)$$

$$\sum_i x_{ij} = 1, \forall j = 1, .., n \quad (2)$$

$$\sum_j x_{ij} p_j £\ C_{max}, \forall i = 1, .., m \quad (3)$$

$$xij \in \begin{cases} 1\ si\ la\ tâche\ j\ est\ affectée\ à\ la\ machine\ i \\ 0\ sinon \end{cases} \quad (4)$$

La contrainte (1) est la fonction objectif qui permet de minimiser le Makespan. La contrainte (2) force chaque tâche à être affectée à une et une seule machine. La contrainte (3) permet de spécifier que le makespan est la durée la plus longue des jobs exécutés par chaque machine. La contrainte (4) permet d'affecter les tâches aux machines si elle est à 1 ou non si elle est à 0.

La variable de décision x_{ij} permet de spécifier si une tâche est affectée à une machine i ou pas. Dans ce problème d'ordonnancement, on cherche en effet à affecter les jobs aux machines et calculer le Makespan.

Les données d'entrées.

- La liste des jobs $j_j, j = 1, .., n$, leur durée d'exécution p_j et l'ensemble des machines $(M_1, M_2, .., M_m)$.

Les données de sorties.

- Les variables binaires $x_{ij}, \forall i = 1,..,m, \forall j = 1,..,n$ et la valeur du Makespan C_{max}. Notons que le makespan étant indépendant dans ce cas de l'ordre individuel des jobs sur les machines, cet ordre n'est pas calculé, puisque toute permutation des jobs donne le même makespan.

3.2.3.2 La formulation avec contraintes de précédence : $P_m|prec|C_{max}$

Les contraintes de précédences sont des contraintes qui permettent de dire que la tache *j* précède une autre tâche *j'*. Dans ce cas, une donnée $y_{jj'}$ est utilisée pour décrire la contrainte de précédences entre les deux tâches *j* et *j'*.

Le problème $P_m|prec|C_{max}$ est formulé comme suit :

$$minimiser\ C_{max} = \max\{s_j + p_j\} \quad (1)$$

$$\sum_i x_{ij} = 1, \forall j = 1,..,n \quad (2)$$

$$\sum_j x_{ij} p_j \leq C_{max}, \forall i = 1,..,m \quad (3)$$

$$xij \in \begin{cases} 1\ si\ la\ tâche\ j\ est\ affectée\ à\ la\ machine\ i \\ 0\ sinon \end{cases} \quad (4)$$

$$\begin{aligned} s_{j'} &\geq s_j + p_j - T(1 - y_{jj'}) \\ s_j &\geq s_{j'} + p_{j'}(1 - y_{jj'}) - T y_{jj'} \end{aligned} \quad (5)$$

Cette formulation présente une contrainte en plus par rapport à la formulation basique du calcul du Makespan (section 3.2.3.1). En effet, la contrainte (5) (Šeda, 2007a), permet d'exprimer le séquençage entre deux tâches j et j'. Si la donnée $y_{jj'} = 1$ alors la tâche *j'* ne peut commencer que si la tache *j* termine son exécution. Elle est égale à 0 sinon.

Les données d'entrées.

- La liste des tâches $j_j, j = 1,..,n$, leur durée d'exécution p_j et l'ensemble des machines $(M_1, M_2, .., M_m)$.
- Les contraintes de précédences entre deux tâches : $y_{jj'}$

Les données de sorties.

- Les variables binaires $x_{ij}, \forall i = 1,..,m, \forall j = 1,..,n$ et la valeur du Makespan C_{max}.
- Les variables s_j de toutes les tâches.
- La valeur du Makespan C_{max}.

3.2.3.3 La formulation avec contraintes de ressources : $P_m|res...|C_{max}$

Dans certains problèmes d'ordonnancement de machines parallèles identiques, les tâches ont besoin de K ressources additionnelles (i.e., en plus des machines les exécutant) pouvant être partagées par plusieurs tâches s'exécutant sur des machines différentes. On considère que chaque ressource a une valeur disponible maximale R_k à ne pas dépasser. Chaque tâche consomme une valeur de r_{jk} de la ressource k. Les contraintes de ressources sont ajoutées au programme MILP

pour garantir qu'à tout moment la consommation des tâches (r_j) ne peut excéder la valeur maximale R_k de la ressource k.

La prise en charge des contraintes de ressources (Özdamar & Ulusoy, 1995) nécessite l'ajout d'une contrainte pour pouvoir affecter les ressources aux tâches sans dépasser leur valeur maximale. Pour expliquer mieux cette contrainte, nous présentons le problème $P_m|res\ldots|C_{max}$ qui est formulé comme suit :

$$C_{max} = \max\{r_j\} \quad (1)$$

$$\sum_j xij = 1, \forall i = 1,..,n \quad (2)$$

$$xij \in \begin{cases} 1\ si\ la\ tâche\ j\ est\ affectée\ à\ la\ machine\ i \\ 0\ sinon \end{cases} \quad (3)$$

$$\sum_{j\in A_t} r_{jk} \leq R_k\ ,\ \forall\ k \in R, \forall\ t \in H \quad (4)$$

Avec : $A_t = \{j \in J | s_j \leq t < s_j + p_{ij}$ représente toutes les tâches en exécution à l'instant t.

$H = \{0,1,..,T\}$ est l'horizon de l'ordonnancement et T=$\sum_j s_j + p_{ij}$.

La contrainte (4) permet de vérifier à chaque instant t, la consommation de toutes les tâches en cours d'exécution.

Les données d'entrées

- La liste des tâches$j_j, j = 1,..,n$, leur durée d'exécution p_j et l'ensemble des machines $(M_1, M_2, .., M_m)$.
- La table des consommations de ressources.
- La valeur maximale de chaque ressource.
- Le nombre de ressources.

Les données de sorties

- Les variables binaires $x_{ij}, \forall\ i = 1,..,m, \forall\ j = 1,..,n$.
- La valeur du Makespan C_{max}.

3.2.4 La complexité

La complexité des problèmes d'ordonnancement de machines parallèles dépend du nombre de machines m, des contraintes de précédences et/ou ressources, ou encore de la disparité ou non des durées d'exécution des tâches. En effet, si l'ordonnancement des tâches de chaque machine est traité séparément, telle que : pour chaque tâche est affectée à une machine$f_i x_j$ alors le problème $(P2|f_i x_j|Cmax)$ est résolu en temps polynomial (Hoogeveen et al., 1994). Toutefois, sur trois machines parallèles, le problème $P3|f_i x_j|Cmax$ devient fortement NP-hard, autrement dit, impossible à résoudre de manière exacte par un ordinateur pour les cas non triviaux (Blazewicz et al., 1992). D'autre part, si on permettait aux tâches d'être exécutées sur n'importe quelle machine, le problème $P2||C$ (Lenstra et al., 1977), le problème $P||C_{max}$ (Bruno et al., 1974; Garey & Johnson, 1978) et le problème $P_2||C_{max}$ (Brucker, 1999) sont NP-hard.

Par ailleurs, le problème d'ordonnancement des tâches sur deux machines parallèles avec des contraintes de précédence $(P2|p_j = p, prec|Cmax)$ est NP-hard même si le temps de traitement

des tâches est égal ($p_j = p \; \forall \; j = 1..n$) (Coffman & Graham, 1972; Gabow, 1982; Hoogeveen et al., 1994; Sethi, 1976).

Les contraintes de ressources augmentent généralement la complexité du problème ($P2||Cmax$). Dans (Blazewicz et al., 1983), le problème d'ordonnancement de machines parallèles sous contraintes de ressources ($P2|res111, pj = 1|C$) est solvable en temps polynomial (Blazewicz et al., 1983). Toutefois, ce problème ne tient pas compte des contraintes de précédences.

3.2.5 La résolution

Dans la littérature, plusieurs travaux ont proposé des algorithmes, heuristiques ou méta-heuristiques pour la résolution des problèmes d'ordonnancement de machines parallèles.

Les travaux de (Gacias et al., 2010) ont permis de proposer deux méthodes pour la résolution du problème $Pm|prec, s_{ij}, \; r_j|\sum c_j$. La première méthode est la méthode de divergence limitée, combinée à la méthode de recherche locale. La deuxième méthode repose sur les conditions de dominances et la proposition de borne inférieures.

Les auteurs (Vincent et al., 2016) ont proposé un algorithme heuristique qui s'appuie sur un algorithme de recuit simulé (Kirkpatrick et al., 1983). L'heuristique utilise certaines propriétés de dominance issues de la littérature. La performance de cette heuristique a été testée sur des instances de la littérature et les résultats montrent que les solutions retournées sont proches de l'optimal.

Dans (Li et al., 2011) un algorithme basé sur le recuit simulé (Kirkpatrick et al., 1983) est proposé pour minimiser le Makespan. Le cas traité est un problème d'ordonnancement de machines parallèles où les temps d'exécution des tâches sont contrôlables avec des consommations de ressources.

3.3 Problème d'ordonnancement de machines parallèles dédiées

Dans cette section, nous présentons les problèmes d'ordonnancement de machine parallèles dédiées. Nous commençons par faire une description formelle de ces problèmes. Ensuite nous dressons plusieurs formulations de ces derniers. Nous terminons par les classes de complexité proposées dans la littérature et les méthodes de résolution proposées.

3.3.1 Description

Il existe un ensemble de tâches $j_1, .., j_n$ à exécuter sur un ensemble de machines $M_1, .., M_m$ *parallèles* et *dédiées*. Chaque tâche doit être exécutée sur une et une seule machine pendant une durée p_j. Les machines ne peuvent exécuter qu'une tâche à la fois. La préemption des tâches n'est pas autorisée. Le but est de trouver un ordonnancement qui minimisant le Makespan seulement car l'affectation des tâches aux machines est une entrée de ces problèmes, contrairement aux problèmes de machines parallèles où l'affectation est une sortie.

Les problèmes d'ordonnancement de machines parallèles dédiées sont très utilisés dans la gestion des équipes de travail sur des projets (Kellerer & Strusevich, 2003b). En effet, chaque équipe est vue comme une machine qui peut traiter des projets (tâches) en consommant des ressources. Une solution pour ces problèmes permet de minimiser le temps total de traitement de tous les projets, en s'assurant que les ressources ont bien été affectées aux équipes. Dans notre cas, les tuyauteries des deux circuits sont rincées en même temps. Pour ce faire, chaque circuit est décomposé en boucles. Les circuits sont considérés comme des machines qui sont dédiées à des boucles (tâches). Cependant, les deux circuits partagent une ressource commune qui est l'énergie hydraulique et est limitée à une valeur maximale (Mesli-Kesraoui et al., 2021). La solution de ce problème permet

de planifier l'exécution des boucles de deux circuits différents en même temps sans dépassement de la valeur maximale de l'énergie hydraulique. Il est à noter qu'entre tâches de même machine, il existe des contraintes de précédences.

Dans la littérature, les problèmes de machines parallèles dédiées sont souvent traités sous contraintes de ressources et/ou contraintes de précédences (Kellerer & Strusevich, 2003b, 2008).

3.3.2 Caractérisation

Les problèmes d'ordonnancement de machines parallèles dédiées $(M_1, M_2, .., M_m)$ ont été étudiés dans (Kellerer & Strusevich, 2003b, 2003a). La caractérisation de ces problèmes est notée par PD_m. L'utilisation des ressources dans ces problèmes nécessite d'ajouter la notation $res\lambda\sigma\rho$ (Blażewicz et al. 2007; Slowiński 1980) au deuxième champ comme suit : $PD_m|res\lambda\sigma\rho$. Ce qui implique qu'il existe un nombre λ de ressources, la valeur maximale de chaque ressource est égale à σ. Chaque tâche ne doit pas consommer plus que ρ unités d'une ressource.

Dans la littérature des cas particuliers de machines dédiées ont été étudiés. Dans (Kellerer & Strusevich, 2003b, 2003a), les auteurs ont étudié le problème de machines parallèles dédiées avec un seule ressource. La valeur maximale de la ressource est égale à σ et chaque tâche ne peut consommer plus que ρ unités de cette ressource à un instant donné. Ce problème est noté par : $PD_m|res1\sigma\rho|C_{max}$. Un cas particulier dérivé de ce problème est $PD_m|res111|C_{max}$ lorsque la taille de la ressource est limitée à 1 et qu'il existe des tâches qui peuvent consommer 1 unité de la ressource pendant leur exécution. Dans les travaux de (Kellerer & Strusevich, 2008) une autre notation pour les problèmes de machines dédiées avec une accélération de la ressource est proposée. Ce problème est caractérisé par $PD_m|res1\sigma\rho, B_i|C_{max}$

. Il est à noter que ce problème particulier a nécessité d'utiliser la notion de *ressource binaire*. C'est-à-dire qu'*une tâche à ressource* (une tâche qui nécessite la ressource pour fonctionner) consomme une unité de cette ressource. Dans le cas contraire *une tâche non-ressource* (qui ne demande pas de ressource pour fonctionner) consomme 0 unité de la ressource. La notation $PD_m|res1\sigma\rho, Int|C_{max}$ et $PD_m|res1\sigma\rho, Lin|C_{max}$ peuvent être utilisées pour désigner des problèmes de machines dédiées et parallèles (Edis et al., 2013; Kellerer & Strusevich, 2008).

3.3.2.1 La formulation sous contraintes de ressources : $\boldsymbol{PD_m|res111|C_{max}}$

Le problème de machines dédiées parallèles est combiné avec les contraintes de ressources. Afin de mieux expliquer ce cas particulier, nous présentons le problème (Kellerer & Strusevich, 2003a) caractérisé par $PD_m|res111|C_{max}$ où m définit le nombre de machines et $res111$ signifie qu'il existe une seule ressource. La valeur maximale de cette ressource est 1 et chaque tâche ne peut consommer plus que 1 unité de cette ressource. Il est à noter que dans ce problème la ressource est partagée. C'est-à-dire que deux *tâches à ressource* ne peuvent pas être exécutées en même temps. La préemption des tâches n'est pas autorisée. Une solution de ce problème permet de proposer un ordonnancement qui minimise le Makespan, c'est-à-dire la date de fin de la dernière tâche ordonnancée.

Ce problème est formulé comme suit :

Soit $N = \bigcup_{i=1}^{m} N_i$ l'ensemble des tâches. Chaque ensemble N_i de tâches affectées à la machine i, est composée de deux sous-ensemble Q_i et R_i. Tels que : Q_i est l'ensemble des tâches non-ressources (elles ne demandent pas de ressource) et R_i est l'ensemble de tâches ressources (elles demandent 1 unité de la ressource).

Les durées de tâches sont modélisées comme suit :

Pour chaque ensemble $N'_i \subseteq N_i$ on a : $P(N'_i) = \sum_{J_j \in N'_i} P_j$. La date de fin de toutes les tâches assignées à la machine i est notée par : $C_i(S)$. Puisque le Makespan total est la date de fin de la dernière tâche alors : $C_{max}(S) = \max\{C_i(S), \text{i=1,..,m}\}$. Donc, $C_i(S) \geq p(N_i)$.

Puisque : $p(N_i) = p(Q_i) + p(R_i)$ alors : $C_{max}(S) \geq \max\{p(Q_i) + p(R_i), i = 1, .., m\}$ ici la valeur du $C_{max}(S)$ est appelée *borne inférieure basée sur la machine*.

La formule totale du Makespan est : $C_{max}(S) \geq \sum_{i=1}^{m} p(R_i)$ et est appelée *borne inférieure basée sur la ressource*.

3.3.2.2 La formulation sous contraintes de précédences : $PD|res1\cdot\cdot, Int|C_{max}$

Les travaux de (Daniels et al., 1996) permettent de traiter le problème de machines parallèles dédiées avec contraintes de ressources et précédences comme un sous problème des problèmes machines parallèles avec des ressources flexibles (MPRF). Il est à noter que dans les problèmes MPRF l'allocation des ressources aux tâches est *dynamique*, c'est-à-dire que la ressource peut basculer d'une machine à l'autre pendant le traitement (Edis et al., 2013).

Les contraintes de précédences sont désignées par la variable de décision $y_{jj'}$, tel que :

$$y_{jj'} = \begin{cases} 1, si\ la\ tâche\ j\ précède\ la\ tâche\ j'\ dans\ la\ machine\ i \\ 0, sinon \end{cases}$$

Puisque dans le problème $PD|res1\cdot\cdot, Int|C_{max}$, la durée d'exécution d'une tâche j dépend de la ressource allouée. La variable de décision x_{jkt} tel que :

$$x_{jkt} = \begin{cases} 1, si\ la\ tâche\ j\ finit\ son\ exécution\ avec\ k\ unités\ de\ ressoure\ à\ l'instant\ t. \\ 0, sinon \end{cases}$$

Et elle permet de calculer la durée d'exécution et la date de fin de la tâche j exécutée dans le mode $k \in K_j$ comme suivant :

$$\sum_{k \in K_j} \hat{p}_{jk} \sum_{t=1}^{T} x_{jkt} = p_j, \forall\, j = 1, .., n$$

$$\sum_{t=1}^{T} t \sum_{k \in K_j} x_{jkt} = C_j\ , \forall\, j = 1, .., n$$

Les contraintes de précédences sont formulées en utilisant la variable $y_{jj'}$.

$$C_{j'} - C_j + T\left(1 - y_{jj'}\right) \geq p_{j'}, j' \neq jet\ j, j' \in N_i, i = 1, .., m$$

$$y_{jj'} + y_{j'j} = 1, , j \in N_i, j' \in N_i\ , j' \neq \{1,2, .., j\}, i = 1, .., m$$

La contrainte de valeur maximale de la ressources est formulée en utilisant la variable x_{jkt}, avec b, est la valeur maximale de la ressource.

$$\sum_{j=1}^{n} \sum_{k \in K_j} \sum_{l=t}^{t+\hat{p}_{jk}-1} x_{jkt} k \leq b\ , t = 1, .., T$$

3.3.3 La complexité

Le problème de machines dédiées parallèles avec une ressource non partagée a été prouvé NP-complet dans (Kellerer & Strusevich, 2003a). Les problème $PD_2|res111|C_{max}$ et $PD_2|res211|C_{max}$ (Edis et al., 2013; Kellerer & Strusevich, 2003a) sont polynomiaux en $O(n)$.

Le problème $PD_2|res1\cdot\cdot|C_{max}$ est $O(nlogn)$ (Kellerer & Strusevich, 2003b).

Les problèmes $PD_2|res222|C_{max}$ et $PD_2|res311|C_{max}$ (Kellerer & Strusevich, 2003b) sont NP-hard.

3.3.4 La résolution

(Daniels et al., 1996) ont résolu le problème $PD2|res1\cdot\cdot|C_{max}$. L'algorithme détermine la séquence des tâches de chaque machine. Sur chaque machine les tâches sont ordonnées par leur consommation en ressource : en ordre croissant sur une machine et décroissant sur l'autre. Les tâches de la machine décroissante commencent à t=0. Par contre les tâches de la machine croissante sont ordonnées au dernier instant t, tel que la ressource est disponible. L'algorithme est d'une complexité de $O(nlogn)$.

Les travaux de (Kellerer & Strusevich, 2003a) ont permis de résoudre de façon optimale le problème $PD2|res111|C_{max}$. Les tâches de chaque machine sont catégorisées en deux ensemble. Les tâches non-ressources $(Q_i)\ et$ les tâches ressources (R_i) (section 3.3.2.1). De fait, toutes les tâches de ces sous-ensembles sont considérées comme des batchs. Les tâches non-ressources peuvent être ordonnées avec n'importe quel autre batch. Tandis que les tâches ressources ne peuvent pas être traitées simultanément car chacune consomme 1 unité de la ressource et sa valeur maximale est égale à 1 $(res111)$. L'algorithme affecte le lot R_1 à la première machine au temps t=0, puis le lot Q_1 une fois que le lot R_1 a terminé son exécution. Sur la deuxième machine, le lot Q_2 est démarré au temps t=0, et ensuite le lot R_2 le plus tôt possible.

3.4 Les problèmes PMRCSP (Parallel Machines Resource Constraint Scheduling Problem)

Dans cette section, nous présentons les problèmes d'ordonnancement sous contraintes de ressources. D'abord nous décrivons ces problèmes et leurs différences avec les problèmes RCSP (Resource Constraint Scheduling Problem) et PMFRCSP (Parallel Machines Flexible Resource Scheduling Problem). Ensuite nous dressons plusieurs formulations de ces derniers. Nous terminons par les classes de complexité calculées dans la littérature et les méthodes de résolution proposées.

3.4.1 Description

Un problème de machines parallèles avec des contraintes de ressources PMRCSP est un problème qui permet d'ordonner un ensemble de tâches $j_1,..,j_n$ sur un ensemble de machines $M_1,..,M_m$ *parallèles*. Chaque tâche doit être exécutée sur une et une seule machine pendant une durée p_j et utilise pour ce faire r_{jk} unités de la ressource R_k. Les machines ne peuvent exécuter qu'une tâche à la fois. La préemption des tâches n'est pas autorisée. En général, il existe des relations de précédence entre les tâches. Ces relations sont représentées par des ensembles de prédécesseurs immédiats $Pred_j$, indiquant qu'une activité j ne peut pas être commencée avant l'achèvement de chacun de ses prédécesseurs $(h \in Pred_j)$. Une solution permet de minimiser le Makespan telle que toutes les tâches soient exécutées sans dépassement de la valeur maximale de la ressource à tout instant t et sans violer les contraintes de précédences si elles existent.

Les ressources sont de plusieurs types. La distinction est faite soit par la nature de la ressource, le temps d'utilisation de la ressource, la manière d'utiliser la ressource et le type de valeurs que peut prendre la ressource (Edis et al., 2013).

La nature de la ressource

Ressource renouvelable. Est une ressource consommable. Elle est renouvelée après chaque fin d'exécution d'un job qui l'avait utilisée. Une ressource renouvelable est *sous contraintes de disponibilité* (Edis et al., 2013; Habibi et al., 2018).

Ressource non-renouvelable. Une ressource non-renouvelable est une ressource utilisée une seule fois. Son utilisation totale dans un projet est sous contraintes (Edis et al., 2013). *Elle est contrainte par sa consommation* (Geurtsen et al., 2023).

Ressource doublement contrainte. Une ressource renouvelable et non-renouvelable en même temps. C'est-à-dire qu'elle est sous contraintes *à chaque instant de son utilisation* et en même temps *sa consommation totale* dans un projet est sous contraintes. Cette ressource peut être considérée dans un problème comme renouvelable ou non renouvelable (Habibi et al., 2018).

Le temps d'utilisation de la ressource (Blazewicz et al., 2004). *Les ressources de traitement*. sont des ressources nécessaires à l'exécution de la tâche. *Les ressources input-output* sont des ressources nécessaires avant ou après l'exécution de la tâche

La manière d'utiliser la ressource. (Edis et al., 2013) *Une ressource statique* est utilisée par une machine d'une valeur fixée et statique. La ressource ne peut basculer à une autre machine durant l'ordonnancement. *Une ressource dynamique* peut basculer d'une machine à une autre pendant l'ordonnancement.

Le type de valeur que peut prendre une ressource. (Blażewicz et al. 2007; Edis, Oguz, et Ozkarahan 2013) Une ressource peut prendre une valeur *discrète* parmi un ensemble de valeurs possibles, ou bien une valeur continue à l'intérieur d'un intervalle.

Les problèmes PMRCSP sont utilisés dans plusieurs domaines. Les ressources telles que la monnaie et l'énergie sont utilisées sous contraintes dans plusieurs projets. Une utilisation concrète est faite par notre banc de test, une énergie hydraulique est nécessaire pour rincer chaque portion du circuit ainsi elle est partagée entre deux circuits parallèles.

3.4.2 Comment différencier RCPSP, PMRCSP et PMFRSP ?

La proximité relative entre les problèmes RCPSP, PMRCSP et PMFRSP peut mener à des incompréhensions pour les lecteurs de la littérature. Afin de mieux montrer cette incompréhension, nous reprenons la citation suivante :

> *"(Daniels et al., 1996) classify RCPMSP as a special case of the dynamic PMFRS problem in which the processing times and the resource requirements are given for each job. However, the PMFRS problem in its original form assumes that the machines are dedicated and there is only one additional resource type in limited supply. Therefore, the RCPMSP, in our opinion, may cover a more general field since it may also consider the job-machine assignment sub-problem and may allow more than one additional resource type. It only excludes the resource allocation problem. In the light of this discussion, the RCPMSP seems to be closer to the dynamic UPMFRS problem, which incorporates the job-machine assignment sub-problem." (Edis et al., 2013)*

En effet, les auteurs (Edis et al., 2013) considèrent que les problèmes RCPMSP sont plus larges (en terme du nombre de problème contenus) que les problèmes PMFRS. Alors que (Daniels et al., 1996) classifient les problèmes RCPMSP comme des cas particuliers des problèmes PMFRS car ces derniers peuvent résoudre les problèmes contenant le sous-problème d'allocation ressource-machine. Tandis que les problèmes RCPMSP résolvent les problèmes d'affectation tâche-machine seulement (les consommations en ressources sont des données fixes en entrée). C'est-à-dire qu'au niveau ressource, les RCPMSP ne résolvent pas l'affectation des ressources aux tâches. Cependant, pour les problèmes PMFRS dans leur forme initiale, les machines sont dédiées, donc il n'existe pas de sous-problème tâche-machine, ils permettent d'inclure le sous-problème d'affectation d'une et une seule ressource aux tâches. D'un autre côté, les RCPMSP peuvent inclure le sous-problème d'affectation des tâches au machines avec une ou plusieurs ressources. Cette différence est née du fait que les problèmes RCPMSP et PMFRS peuvent être caractérisés par la même notation et peuvent prendre en considération le même type de ressource (renouvelable) (EDİS, 2009).

Le but de cette comparaison n'est pas de décider quel problème est contenu dans l'autre. Nous cherchons à faciliter aux lecteurs le positionnement de leurs problèmes parmi ces larges notations. Pour cela, on propose dans la Figure 11, un guide qui permet de connaitre la caractérisation et les caractéristiques de chaque problème en se basant sur les sous-problèmes inclus dans chaque problème et le type de variables de décision (voir 3.1.2.3) utilisées pour formuler le problème.

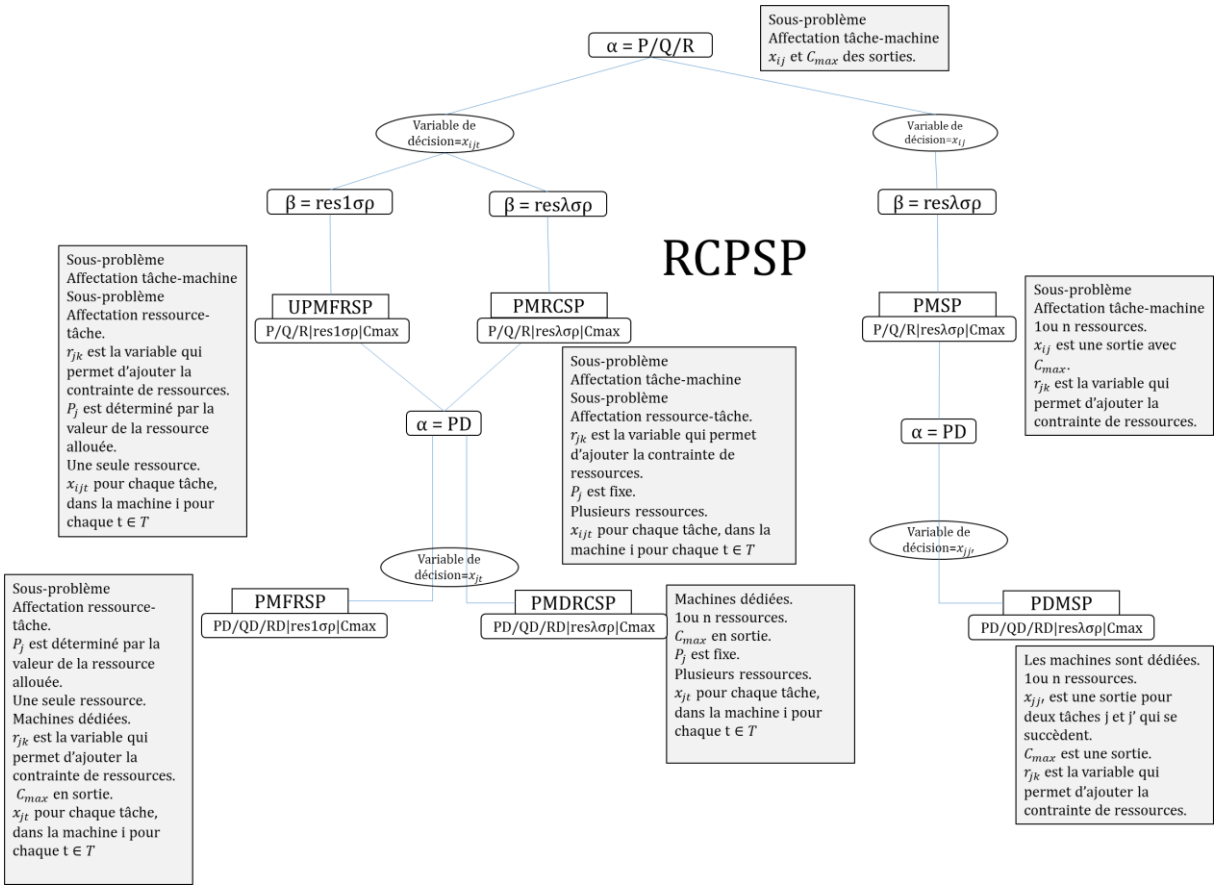


Figure 11 : caractéristiques du RCPSP, RCPMSP, PMFRS et UMFRS.

Les problèmes de machines parallèles avec contraintes de ressources (*P*) sont des problèmes qui traitent le sous-problème d'affectation des tâches aux machines. De fait, si le nombre de ressource est inclus dans {1, n} et la consommation de chaque tâche en ressources est connue ; alors le problème est un problème de PMRCSP. Dans le cas contraire où il y'a qu'une seule ressource avec

la consommation restant à déterminer (inclut le sous-problème d'affectation ressource-tâche) tel que le p_j de chaque tâche dépend de la valeur de la ressource allouée, alors le problème est alors UPMFRSP[6].

Dans certains problèmes d'ordonnancement, l'affectation des tâches aux machines est prédéfinie. De fait, le problème UPMFRSP devient un problème de PMFRSP, le problème PMRCSP devient PMDRCSP et le problème PMSP devient PDMSP. La caractérisation d'un problème appartenant à ces trois catégories peut être faite de la même manière. Par contre, la formulation diffère selon les variables de décision de chaque catégorie comme le montre la Figure 11.

Selon nos connaissances, on considère que le RCPSP est le fait de gérer les contraintes de ressources dans n'importe quel problème d'ordonnancement.

3.4.3 La formulation

3.4.3.1 La formulation PMRCS : $\boldsymbol{Pm|res\lambda\varrho\rho|Cmax}$

La formulation du problème PMRCSP avec des machines identiques avec u, le nombre de ressources a été proposé par (EDİS, 2009). La consommation de la tâche j en ressource k est désignée par la variable $res_{i,k}$ telle que la valeur maximale de la ressource k est égale à b_k.

$$x_{jt} = \begin{cases} 1, si\ la\ tâche\ j\ commence\ son\ exécution\ à\ temps\ t \\ 0, sinon \end{cases}$$

Le but est de minimiser le Makespan C_{max} :

$$\sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} x_{jt}(t + p_j) \leq C_{max} \quad , j = 1, .., n$$

$$\sum_{t=1}^{T-p_j+1} x_{jt} = 1 \quad , j = 1, .., n$$

$$\sum_{s=T-p_j+1}^{t} x_{js} \leq 1 \quad , t = 1, .., T$$

Sous la contrainte de ressources qui est modélisée comme suivant :

$$\sum_{j=1}^{n} \sum_{s=\max\{0, t-p_j\}}^{t-1} res_{j,k} x_{jt} \leq b_k \quad , k = 1, .., u; t = 1, .., T$$

Il est à préciser que cette formulation considère les ressources comme renouvelables et statiques (EDİS, 2009). Les machines dans ce cas sont aussi considérées comme des ressources. De fait, ce problème inclut aussi le sous-problème d'affectation tâche-machine. Selon la classification donnée dans la Figure 11, on peut dire que cette formulation est une formulation pour un problème *PMRCSP caractérisé selon les champs $\alpha|\beta|\gamma$ comme suit : $Pm|res\lambda\varrho\rho|Cmax$.*

3.4.3.2 La formulation PMRCSP pour le problème : $\boldsymbol{Pm|res\lambda\varrho\rho, prec|Cmax}$

Les contraintes de précédences en RCPSP peuvent être formulées de deux façons : la DT (temps discret) et la DDT (temps discret désagrégé) (Koné et al., 2013).

[6] UPMFRSP : Unspecified Parallel Machines Flexible Resource Scheduling Problem

Dans la formulation DT, une variable temporelle x_{jt} indexée par le temps t et la tâche j est utilisée. Cette variable binaire prend ses valeurs 1 ou 0 comme suit :

$$x_{jt} = \begin{cases} 1, si\ la\ tâche\ j\ commence\ son\ exécution\ à\ temps\ t \\ 0, sinon \end{cases}$$

Pour chaque tâche j deux dates sont calculées. Une date de début au plus tôt ES_j (Earliest Starting time), telle que la tâche j ne peut commencer avant cette date. La date de début au plus tard LS_j (Latest Starting time) pour laquelle, la tâche j ne peut commencer après.

L'utilisation de ces deux variables (ES_j et LS_j) pour chaque tâche j, permet de formuler les contraintes de précédences entre la tâche j et la tâche j' de la façon suivante :

$$\sum_{t=ES_{j'}}^{t=LS_{j'}} t x_{j't} \geq \sum_{t=ES_j}^{t=LS_j} t x_{jt} + p_j \quad \forall (j, j') \in E$$

L'ensemble E est l'ensemble des couples (j, j') tel que : j précède j'.

Dans la formulation DT, une contrainte est formulée pour chaque paire de tâche (j, j') et pour chaque précédence (Koné et al., 2013).

Une autre formulation pour les contraintes de précédences en RCPSP est la formulation temps discret désagrégé ou DDT (Christofides et al., 1987). Dans DDT, une contrainte est formulée pour chaque paire de tâches, pour chaque précédence et pour chaque instant t (Koné et al., 2013). Les contraintes de précédences sont formulées par la contrainte suivante :

$$\sum_{\tau=t}^{LS_j} x_{j\tau} + \sum_{\tau=ES_{j'}}^{\min(LS_{j'}, t+p_j-1)} x_{j'\tau} \leq 1 \quad \forall (j, j') \in E, \forall\ t \in \{ES_j, LS_j\}$$

3.4.4 Complexité

La complexité des problèmes PMRCSP augmente avec le nombre de ressources (EDİS, 2009). Dans les travaux de (Blazewicz et al., 1983) les problèmes $P2|res1\cdot\cdot, tree, p_j = 1|Cmax$, $P2|res111, prec, p_j = 1|Cmax$, $P2|res111, chain, p_j = 1|Cmax$, $P3|res1\cdot\cdot, p_j = 1|Cmax$ sont NP-Hard au sens fort.

3.4.5 Résolution

Vue la complexité des problèmes PMRCSP, peu de travaux ont pu résoudre ces problèmes à l'optimal. Parmi les méthodes exactes, on peut citer le B&B, utilisée par (Blazewicz et al., 1993), pour résoudre le problème $P2|res\cdots, p_j = 1|Lmax$ et les travaux de (Daniels et al., 1996) pour résoudre le problème $PD|res1\cdot\cdot, Int|Cmax$. La programmation dynamique a été utilisée par (Kellerer & Strusevich, 2008) pour résoudre le problème $PD2|res111, Bi|Cmax$.

Le problème $P2|res\cdots, p_j = 1|Cmax$ a été résolu en temps polynomial par (Garey & Johnson, 1975a). Il est à noter que dans ce problème, les tâches ont une consommation égale à 1 ou 0.

4 Bilan critique de l'état de l'art

Les problèmes à ressources sont des problèmes très connus dans le domaine de l'ordonnancement déterministe (Blazewicz et al., 1993). De fait plusieurs modèles ont été proposés pour résoudre les problèmes qui utilisent plusieurs machines sous contraintes de ressources (Edis et al., 2013).

Le problème de machines parallèles avec contraintes de ressources et contraintes de précédences a été étudié et formulé (Blażewicz et al. 2007; Daniels, Hoopes, et Mazzola 1996; Edis, Oguz, et Ozkarahan 2013; Kellerer et Strusevich 2003a, 2003b, 2008). En effet, les auteurs commencent souvent par caractériser le problème selon les champs $\alpha|\beta|\gamma$ de (Graham et al., 1979). Cette caractérisation permet de classifier un problème d'ordonnancement et de faciliter sa formulation par rapprochement aux problèmes existants dans la littérature. Le calcul de complexité est une étape qui permet de déterminer si un problème peut être résolu à l'optimal par des approches exactes. Dans le cas contraire, il est nécessaire d'utiliser des méthodes approchées pour sa résolution. Nous pouvons aussi noter qu'il est complexe, à partir de la seule notation à trois champs des problèmes avec ressource, de différencier, par exemple sur une formulation res111, si les ressources sont binaires, ou bien continues. Dans le cas continu, on pourrait par exemple représenter de façon normalisée, la quantité maximale de ressource (1 représente 100%) et avoir une tâche qui consomme un pourcentage de cette quantité maximale, pourcentage qui ne peut pas dépasser 100% et par conséquent est au maximum à 1. Dans le cas discret, les limites étant fixées à 1, chaque tâche ne peut que consommer 0 ou 1 de la ressource. Certains articles de la littérature ne précisent pas explicitement l'interprétation de ce terme, il est parfois laissé au lecteur le soin de déterminer si les auteurs considèrent des valeurs continues ou discrètes.

A travers l'état de l'art, nous avons étudié les problèmes de machines parallèles identiques avec ressources et contraintes de précédences. Cependant, ces problèmes incluent le sous-problème d'affectation des tâches aux machines. Les machines parallèles dédiées, sont des problèmes de machines parallèles identiques où les tâches sont affectées au préalable aux machines. L'ordonnancement dans ce cas permet de minimiser le Makespan tout en respectant les contraintes de ressources et de précédences. De fait, il y a un petit allégement par rapport aux problèmes de machines parallèles identiques car les machines dédiées n'incluent pas le sous-problème d'affectation des tâches aux machines (Edis et al., 2013).

Les problèmes PMRCSP sont des problèmes qui peuvent prendre en considération plusieurs ressources. En effet, dans certains cas, les machines sont considérées comme des ressources nécessaires au traitement des tâches. De fait, l'utilisation des variables temporelles et des variables de ressources permet de vérifier les contraintes à chaque instant t.

Le problème soulevé par (Edis et al., 2013) sur l'appartenance de PMRCSP aux PMFRP est très souvent rencontré et laisse les lecteurs se questionner par rapport à la formulation adéquate de leurs problèmes. Cette ambiguïté est due au fait que les PMRCSP et PMFRP peuvent formuler le même problème différemment en utilisant des variables de décision différentes, sachant que ce problème est caractérisé dans les deux cas de la même manière.

Cependant, pour un problème de machines parallèles identiques et dédiées avec contraintes de ressources et précédences, il y a deux manières de le formuler : par les PDRCSP ou par les PDSP. La différence réside dans les variables de décision utilisées. Dans la littérature, il n'y a pas de formulation pour le problème de deux machines parallèles dédiées avec une ressource et contraintes de précédences. Les seules qui ont été proposées concernent les problèmes PMFRP. Par conséquent le problème de rinçage central dans cette thèse n'a pas, à notre connaissance, été traité directement dans la littérature.

Chapitre 3 : Ordonnancement de deux machines parallèles dédiées sous contraintes de ressources et précédences locales

1 Introduction

Notre objectif est de résoudre un problème d'ordonnancement de deux machines dédiées et parallèles avec deux contraintes : la contrainte de la ressource partagée entre les deux machines et les contraintes de précédences locales entre des tâches de la même machine.

Dans la littérature, les problèmes de ressource partagée (PMRCSP) et les contraintes de précédences sont traités mais la notion de machines parallèles et dédiées n'est pas prise en compte dans ces problèmes. D'autre part, les problèmes d'ordonnancement de ressources de machines parallèles PMSP et DPMSP sont multiples et plusieurs formulations ont été proposées. Toutefois, les contraintes de précédence n'ont pas été considérées pour RCPMSP sur des machines parallèles dédiées. Par conséquent, le problème d'ordonnancement de deux machines dédiées et parallèles avec des contraintes de ressource et de précédences locales ($PD2|res1\cdots, localchain|Cmax$) n'a jamais, à notre connaissance, été traité dans la littérature. De fait, il n'existe aucune formulation ni résolution accessible de ce problème. De plus, la complexité est ouverte, sachant que la complexité du même problème sans contraintes de précédences est polynomiale (voir section 3.2.4)

2 Aléas et verrous

Afin de résoudre le problème d'ordonnancement des tâches de deux machines dédiées et parallèles avec des contraintes de précédences, les verrous suivants sont à lever.

Le premier verrou porte sur **la caractérisation** de notre problème d'ordonnancement. En effet, une simple ambiguïté dans sa caractérisation peut entrainer une mauvaise formulation du problème et de sa résolution. Le problème traité ici concerne le partage d'une ressource entre des tâches de deux machines dédiées et parallèles. C'est un cas particulier de RCPMSP avec des contraintes de précédences. De plus, il existe deux caractérisations possibles en RCPMSP pour ce problème. Le défi ici consiste à définir la caractérisation exacte du problème.

Le deuxième verrou porte sur **l'étude de la complexité** de notre problème d'ordonnancement. Avant de résoudre un problème d'ordonnancement, il est nécessaire de calculer sa complexité, car les problèmes NP-difficiles présentent des performances limitées en termes de temps de calcul face aux grandes instances.

Le troisième verrou porte sur **la formulation mathématique** du problème d'ordonnancement. Les différentes formulations du problème RCPMSP prennent en considération la contrainte de ressource en utilisant des variables temporelles. Cependant, pour l'expression des contraintes de précédences, les contraintes temporelles deviennent assez lourdes. Il est nécessaire de proposer une nouvelle formulation simplifiée du problème afin d'améliorer ses performances

Si notre problème est NP-difficile, il serait alors nécessaire de proposer **des heuristiques alternatives** de résolution permettant de résoudre le problème (quatrième verrou). Contrairement à la formulation mathématique qui est optimale (elle retourne la solution optimale), les heuristiques sont capables de retourner des solutions même pour les instances de grande taille ; cependant la solution alors retournée n'est pas optimale. C'est une solution parmi d'autres. Le défi est de proposer des heuristiques capables de retourner des solutions proches de l'optimal.

3 Etude du problème d'ordonnancement du rinçage

3.1 Caractérisation du problème

Initialement, nous avons caractérisé notre problème comme un problème d'ordonnancement de deux machines parallèles ($\boldsymbol{P2|prec|Cmax}$). Les tâches peuvent s'exécuter sur n'importe quelle machine. Pour résoudre le problème d'affectation des tâches aux machines, nous avons alors caractérisé notre problème par des machines dédiées. Ainsi les tâches sont affectées aux machines au préalable ($\boldsymbol{D2|prec|Cmax}$). Ensuite, nous avons utilisé la notation standard RCPSP, augmentée avec la notation $res\lambda\sigma\varrho$ pour modéliser la ressource partagée. Ainsi, notre problème d'ordonnancement peut être exprimé à la fois par les notations $\boldsymbol{PD2|res1\cdot\cdot, localprec|Cmax}$ ou $\boldsymbol{PD2|res111, localprec|Cmax}$. Cependant, la deuxième notation pourrait interpréter la ressource res comme une ressource discrète qui peut être consommée entièrement (1) ou pas du tout consommée (0) par une tache (Kellerer & Strusevich, 2003c). Toutefois, dans notre cas, les tâches peuvent consommer une partie de la ressource (la demande de ressources n'est pas discrète). Nous choisissons donc d'exprimer notre problème d'ordonnancement sous la forme $\boldsymbol{PD2|res1\cdot\cdot, localprec|Cmax}$. Le problème est un problème d'ordonnancement de deux machines parallèles et dédiées ($PD2$) partageant une ressource ($\lambda = 1$). La valeur maximale de la ressource ainsi que la consommation maximale d'une tâche sont des valeurs continues entre 0 et 1 ($\sigma =\cdot$ et $\varrho =\cdot$).

3.2 Etude de la complexité de notre problème d'ordonnancement

Pour prouver que notre problème est NP-Difficile, nous opérons une réduction polynomiale de 3-partition (Garey & Johnson, 1975b) au problème de décision $\boldsymbol{PD2|res111, localchain|Cmax£ k}$. Les chaînes de précédence locales étant des cas particuliers des graphes de précédences, et les problèmes avec ressource additionnelle $\boldsymbol{res111}$ étant des cas particuliers des problèmes $\boldsymbol{res1\cdot\cdot}$, et étant donné un ordonnancement trouvé, il est polynomial de montrer que son makespan est inférieur ou égal à k. Le problème de décision qui est un cas particulier de notre problème d'ordonnancement sera ainsi montré NP-Complet au sens fort, le problème de recherche d'ordonnancement étant donc NP-Difficile au sens fort.

Théorème 1. $PD2|res111, \text{localchain}|Cmax$ est NP-Difficile au sens fort. On considère que la taille de la ressource est normalisée à 1 ainsi que les consommations de ressource, c'est-à-dire : $\rho_{i,j} \in \{0,1\}$.

Preuve. La preuve consiste en une réduction de n'importe quelle instance arbitraire de 3-Partition à un problème de décision $PD2|res111, \text{localchain}|Cmax \leq k$. Le problème de 3-partition d'entrée est donné par :

- B, un entier donné.
- $S = (i_1, i_2, .., i_{3m})$ un ensemble de 3m entiers positifs, tel que : $\forall\, j \in \{1, .., 3m\}: {}^{B}/_{4} < i_j < {}^{B}/_{2}$ *et* $\sum i_j = mB$

Le problème 3-Partition, qui est NP-complet au sens fort, consiste à décider s'il existe une partition de S en m sous-ensembles $S = \{S1 \cup S2 \cup .. \cup Sm\}$ telle que la somme de chaque ensemble soit égale à B, c'est-à-dire que :

$$\sum_{i_j \in S_1} i_j = \sum_{i_j \in S_2} i_j = \cdots = \sum_{i_j \in S_m} i_j = B$$
$$|S_i| = 3$$

Etant donnée une instance de 3-Partition, nous créons le problème d'ordonnancement suivant :

- Un ensemble de $3m$ tâches est assigné à la machine $M1$, donc $N1 = \{J_{1,1}, \dots, J_{1,3m}\}$, tel que chaque tâche est caractérisée par : 1) un temps d'exécution $(\rho_{1,i} = i_j)$; et 2) la ressource additionnelle est utilisée $\rho_{1,i} = 1$.
- Sur la machine $M2$, nous affectons $2m$ tâches $N2 = \{J_{2,1}, J_{2,2}, \dots, J_{2,2m-1}, J_{2,2m}\}$, avec une chaîne de précédence de la première à la dernière tâche $(J_{2,i} \prec j_{2,i+1}, \forall\, i \in \{1, .., 2m - 1\})$. Pour les tâches impaires de la machine M2 $(p_{2,2k+1}, k \in \{0, \dots, m - 1\})$, le temps d'exécution est $p_{2,2k+1} = B$, et la ressource n'est pas utilisée $\rho_{2,ék+1} = 0$. Pour les tâches paires de la machine M2 $(j_{2,2k}, k \in \{0, \dots, m\})$, le temps de d'exécution est $p_{2,2k} = 1$ et la ressource est utilisée, i.e., $\rho_{2,2k} = 1$.

Ceci oblige les tâches paires $j_{2,2k}$ à être exécutées seules puisque chaque tâche sur la machine $M1$ utilise la ressource $(\rho_{1,i} = 1\ et\ \rho_{2,2k} = 1)$. Compte tenu de la chaîne de précédence, ces tâches sont séparées les unes des autres pendant au moins une durée B, car chaque tâche est exécutée pendant une durée égale à 1. Le modèle créé sur la machine $M2$ est visible sur la Figure 12-a.

Si l'entrée du problème 3-Partition peut être 3-partitionnée (Figure 12-a), alors il est possible d'ordonnancer, pendant chaque tâche impaire de durée B, 3 tâches sur la machine $M1$ correspondant à une partition, avec la somme du temps d'exécution de ces trois tâches égale exactement à B. Donc, la tâche paire suivante peut être exécutée sur $M2$ sans délai après la tâche précédente. Cela permet à l'ordonnancement d'avoir un Makespan de exactement $mB + m$. Si le problème de 3 partitions en entrée ne peut pas être 3-partitionné, alors il existe au moins une partition dont la somme est strictement supérieure à B (la troisième sur la Figure 12-b). En conséquence, les tâches correspondantes sur $M1$ ont une durée supérieure à B, ce qui interdit l'exécution suivante sur M2 des tâches paires juste après les tâches impaires. Cela repousse tout le reste de l'ordonnancement et augmente le Makespan à une valeur strictement supérieure à $mB + m$. A noter que les contraintes de précédences ont été utilisées seulement sur la machine M2 pour prouver le résultat.

La Figure 12(a) (resp. Figure 12(b)) présente l'application de la réduction sur notre problème d'ordonnancement d'un problème 3-partitionnable (resp. non 3-partitionnable). Sur ces figures, la ligne du haut représente l'ordonnancement sur M1, la ligne du bas l'ordonnancement sur M2, et la ligne du milieu l'utilisation de la ressource additionnelle. Les flèches entre les tâches exécutées sur M2 représentent des contraintes de précédence. La construction faite par les tâches d'indice paire sur M2 est bien visible : elles utilisent la ressource donc empêchent toute exécution concurrente sur M1 où toutes les tâches utilisent la ressource additionnelle. Ce faisant, elles créent m espaces de taille B. Sur l'ordonnancement (a), l'exécution des tâches est possible sur la machine M1 dans les espaces de taille B laissés entre les tâches d'indice paire de M2, car le système est 3-partitionnable. Cependant, sur l'ordonnancement (b), le système n'étant pas 3-partitionnable, le regroupement de 3 tâches telles que la somme des durées vaut B est impossible. Par conséquent, au moins l'un des intervalles d'exécution de 3 tâches sur M1 dépasse B, ce qui repousse l'exécution de la prochaine tâche d'indice paire sur M2, repoussant ainsi le makespan au-delà de mB+m.

Corollaire 1. $PD2|res111, \text{localchain}|Cmax \leq k$ est NP-Complet au sens fort.
Preuve. Etant donné un ordonnancement du problème, il est polynomial de vérifier que son makespan est plus petit ou égal à k.

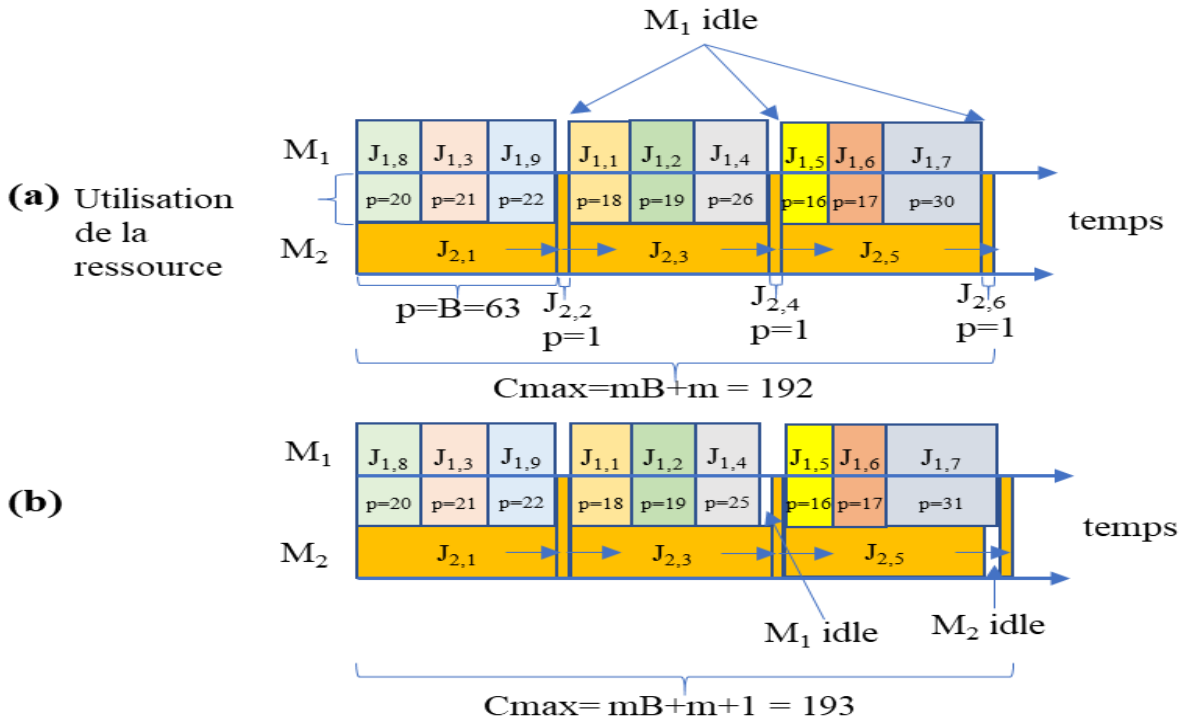


Figure 12. Ordonnancement correspondant à la réduction de deux instances du problème 3-Partition, avec B = 63, m = 3, et pour (a) S = (18, 19, 21, 26, 16, 17, 30, 20, 22) et pour (b) S={16,17,18, 19,20,21,22,25,31}.

Corollaire 2. $PD2|res\lambda\sigma\varrho$, localchain$|Cmax$ est NP-Difficile au sens fort pour toute valeur de λ, σ, ϱ.

Preuve. Pour toute combinaison possible de valeurs de λ, σ, ϱ, le cas res111, où la consommation de ressources est discrète (c.-à-d. 0 ou 1) est un cas particulier possible, qui est NP-Difficile au sens fort du théorème 1.

3.3 Formulation du problème

Dans cette section, nous présentons les deux formulations que nous proposons, à but de comparaison, pour notre problème d'ordonnancement.

3.3.1 Formulation Discret-Time (DT) avec des variables temporelles

Pour ordonnancer les n tâches sur les 2 machines, on utilise une variable temporelle $x_{j,t}$ qui est égale à 1, si la tâche j de la machine i est démarrée à l'instant t , et à 0 sinon. La ressource r a une valeur maximale de B_r. Chaque tâche a une durée d'exécution p_i et une consommation de la ressource b_{ir}. Les contraintes de précédence sont formalisées par un ensemble E de paires de tâches $(j, j') \in E$, qui spécifie que l'exécution de la tâche j doit précéder celle de la tâche j'. La contrainte de ressource est formulée de telle sorte à ce que à chaque instant t, la consommation de la tâche j' est comparée à la consommation de la tâche j, avec j et j' deux tâches, exécutées en parallèles sur deux machines différentes. La formulation temporelle proposée est détaillée par le modèle mathématique suivant :

$$Minimize: \left(max_j \sum_{t=0}^{T} t x_{j,t} \right) (1)$$

$$subject\ to:$$

$$\forall j \in J, \sum_{t=0}^{T} x_{j,t} = 1 \quad (2)$$

$$(j,j') \in E, \sum_{t=0}^{T} t(x_{j',t} - x_{j,t}) \geq p_j \quad (3)$$

$$\sum_{(j,j')\in J} r_j \sum_{t=\max(0,t-p_j+1)} x_{j,t} \leq R_k \quad (4)$$

$$\forall j \in J, \forall i \in M, \forall t \in [0,..,T-1], \quad x_{j,t} \in \{0;1\} \quad (5)$$

Avec :

$M = \{1,2\}$, l'ensemble des machines.

J_i, l'ensemble des tâches de la machine $i \in M$.

$J = \{(i;\ j)\ /\ i \in M, j \in J_i$ L'ensemble des tâches.

p_j , la durée de la tâche j.

$T = \sum_{(i,j)\in J} p_j$, un majorant du Makespan optimal.

r_j , la consommation de ressource k de la tâche j de la machine i.

$E = \{(j,j')/\ \ (j,j') \in J_i^2, tel\ que: j\ précéde\ j'\}$ l'ensemble des contraintes de précédences sur chaque machine i.

La contrainte (1) minimise le Makespan. La contrainte (2) assure qu'une tâche est démarrée une et une seule fois. La contrainte (3) garantit le respect des contraintes de précédences locales, c'est-à-dire entre deux tâches de même machine. La contrainte (4) garantie le respect de la contrainte de la ressource partagée entre deux tâches de machines différentes, exécutées en même temps et le respect d'affectation des machines aux tâches. La contrainte (5) force la variable de décision $x_{j,t}$ pour qu'elle ait des valeurs binaires.

Bien que cette formulation exprime efficacement les contraintes de ressources, l'utilisation des variables temporelles pour l'expression des contraintes de précédences la rend très consommatrice en termes de temps. En effet, la variable de décision $x_{j,t}$ est vérifiée pour chaque tâche j de la machine i à chaque instant t.

3.3.2 Formulation disjonctive

Selon la littérature, les contraintes de précédence sont facilement formulées à l'aide des variables binaires d'exclusion (Šeda, 2007b). Dans notre cas, la formulation disjonctive peut simplement être traduite en utilisant des variables binaires d'exclusion pour chaque paire de tâches sur des machines différentes si la demande totale en ressources des deux taches est supérieure à la valeur maximale de la ressource partagée. Notre formulation disjonctive est présentée ci-après.

$$minimise\ max_{i\in\{1,2\},j\in\{1,..,N_m\}}(s_{i,j} + p_{i,j}) \quad (1)$$

$$\forall i \in \{1,2\}, \forall j \in N_i : s_{i,j} \geq 0 \quad (2)$$

$$\forall i \in \{1,2\}, \forall j,j' \in N_i, j \neq j' : s_{i,j'} \geq s_{i,j} + p_{i,j} \quad si\ (j,j') \in E \quad (3)$$

$$\forall i \in \{1,2\}, \forall j,j' \in N_i\ j \neq j' : \begin{cases} x_{j,j'} \in \{0,1\} \\ s_{i,j'} \geq s_{i,j} + p_{i,j} - T(1 - x_{j,j'}) \\ s_{i,j} \geq s_{i,j'} + p_{i,j'} - T x_{j,j'} \end{cases} \quad (4)$$

$$\forall j \in N_1, \forall j' \in N_2, \rho_j + \rho_{j'} > 1 : \begin{cases} x_{j,j'} \in \{0,1\} \\ s_{i,j'} \geq s_{i,j} + p_{i,j} - T(1 - x_{j,j'}) \\ s_{i,j} \geq s_{i,j'} + p_{i,j'} - T x_{j,j'} \end{cases} \quad (5)$$

Avec :

$M = \{1,2\}$ Est l'ensemble des machines.

N_i est l'ensemble des tâches de la machine $i \in M$.
$p_{i,j}$ est la durée de la tâche j de la machine i.
$T = \sum_{(i,j)\in J} p_{i,j}$, un majorant du Makespan optimal.
$R_1 = 1, la$ valeur maximale de la ressource.
$E = \{(j,j')/ \ (j,j') \in J_i^2, tel\ que: j\ précéde\ j'\}$ Est l'ensemble des contraintes de précédences entre j et j'.

La contrainte (1) est la fonction objectif qui minimise le Makespan. La contrainte (2) assure que toutes les tâches commencent au temps 0 ou après. La contrainte (3) exprime les contraintes de précédence entre les tâches d'une même machine. La contrainte (4) permet de garantir que la machine ne peut exécuter qu'une seule tâche à la fois. La contrainte (5) garantit que pour les deux tâches exécutées simultanément sur une machine différente, la somme de leur consommation de ressource ne doit pas dépasser 1.

3.4 Evaluation du modèle mathématique

Evaluer un algorithme d'ordonnancement consiste à le tester sur plusieurs critères. Dans notre cas, on compare sa performances face à un algorithme DTD pour les problèmes d'ordonnancement de machines parallèles avec contraintes de ressources.

Nous évaluons les performances en fonction de la taille de l'instance et tester également le modèle à l'aide de différents solveurs. Toutes les expériences ont été menées sur un serveur équipé d'un processeur Intel Xeon™ E5-2630 v3 avec 6 cœurs double thread à 2,40 GHz et 32 Go ou RAM, exécutant Ubuntu 20.04.4. Les algorithmes ont été implémentés dans Python 3.10 et les modèles mathématiques ont été résolus à l'aide de CPLEX 22.1, comprenant à la fois le solveur CPLEX et Constraint Programming Optimizer (CPO), Gurobi 11.0 et Z3 4.13.

La formulation disjonctive du problème pour CPO est simple : chaque tâche est allouée à sa machine. Pour chaque contrainte de précédence, la contrainte exige que la fin du prédécesseur ait lieu avant le début du successeur. Nous utilisons une contrainte « no_overlap » entre chaque paire de tâches s'exécutant sur la même machine, et entre chaque paire de tâches dont la consommation totale de ressources est plus importante que PMAX.

3.4.1 Comparaison entre la formulation DT et la forme disjonctive proposée

Les instances de problèmes ont été générées selon un schéma spécifique. Le nombre de machines a été fixé à 2. Le temps de traitement de chaque tâche a été généré aléatoirement en utilisant une distribution uniforme. Les relations de précédences entre les tâches ont été attribuées aléatoirement. Le nombre de contraintes de précédences par machine est choisi , en moyenne, un facteur de densité multiplié par le nombre de tâches sur la machine. Nous avons choisi un facteur de densité de (une moyenne d'une contrainte de précédences par tâche), car cela correspond à ce que nous avons observé dans des instances réelles du problème de rinçage considéré. La taille de l'instance est indiquée par la notation $N1 \times N2$, où $N1$ est le nombre de tâches sur la machine 1 et $N2$ est le nombre de tâches sur la machine 2. Chaque point dans les résultats présentés est la moyenne de 200 instances randomisées de même taille.

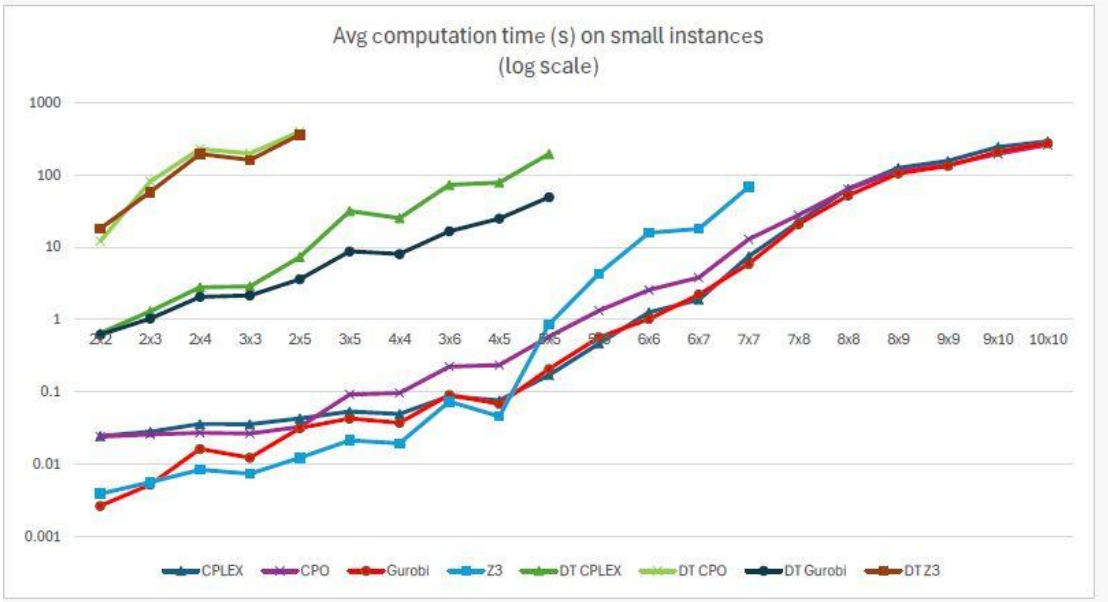


Figure 13. Performance de la formulation disjonctive et la formulation à temps discret (DT) pour le problème d'ordonnancement $\boldsymbol{PD2|res111, localchain|Cmax}$ *dans les solveurs : a) SMT Z3 ; b) Cplex ; et c) Gurobi*

Les résultats montrent un écart important entre les performances du modèle disjonctif et le modèle à temps discret (DT). Le temps de calcul des quatre solveurs traitant de la formulation DT augmente rapidement, en raison de l'augmentation de le nombre de variables binaires. Même si CPLEX et Gurobi fonctionnent mieux que Z3 et CPO pour la formulation DT, il n'est pas surprenant que la formulation disjonctive permet à tous les solveurs testés de gérer des instances plus grandes.

Ces expériences mettent en évidence un écart de performance significatif entre le modèle disjonctif proposé et le modèle DT pour le problème $\boldsymbol{PD2|res1\cdot\cdot, localprec|Cmax}$. Pour les instances plus grandes que 2x5 pour CPO et Z3, et 5x5 pour CPLEX et Gurobi, nous avons abandonné la formulation DT pour les instances plus grandes.

3.4.2 Etude des performances du modèle disjonctif face à la taille d'instance

Pour évaluer les performances de notre formulation disjonctive, nous l'avons testé sur des instances allant de 3x3 à 30x30. Pour les instances allant de 3x3 à 10x10, nous avons utilisé les quatre solveurs, puis exclusivement le CPO solveur pour les instances allant jusqu'à 30x30.

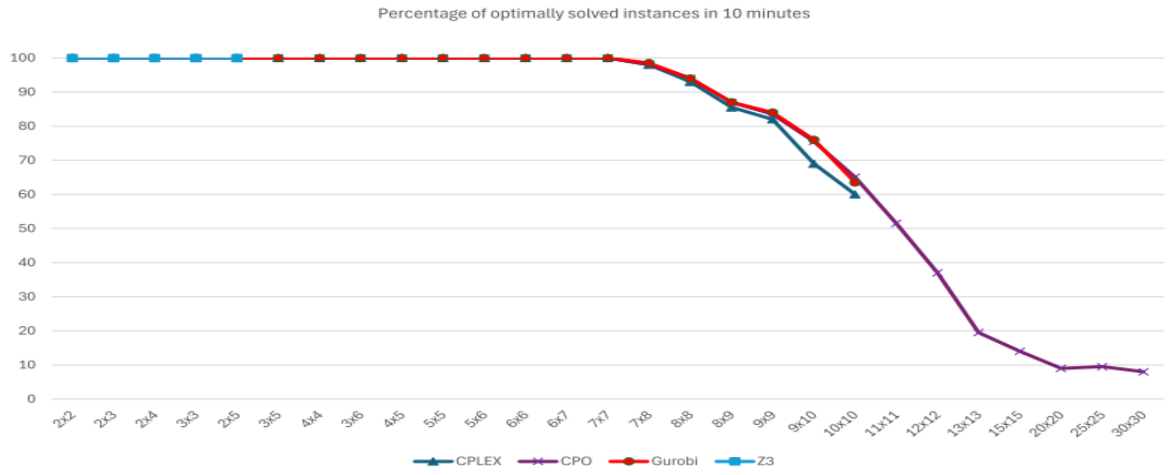


Figure 14. Évaluation du modèle disjonctif face à la taille des instances.

Les résultats indiquent que le modèle disjonctif est capable de trouver des solutions optimales pour des instances de taille 3x3 à 7x7. Pour les instances de taille 8x8 à 13x13, le modèle disjonctif peut renvoyer une solution optimale dans le délai de dix minutes pour certains cas. Cependant, à mesure que la taille de l'instance augmente jusqu'à 30x30, le nombre d'instances pouvant être résolues dans le délai imparti diminue considérablement, jusqu'à moins de 10 % pour le meilleur solveur, CPO.

Le modèle disjonctif proposé pour le problème PD2|$res1 \cdot \cdot$, localprec|$Cmax$ a montré de bonnes performances par rapport au modèle standard à temps discret (DT). Cependant, le modèle disjonctif peut rencontrer des limites dans la résolution d'instances plus grandes au-delà de la taille de 13x13. Il s'agit d'une limitation importante étant donné que les problèmes réels de rinçage peuvent être beaucoup plus importants, atteignant environ taille 30x30. Par conséquent, pour résoudre de tels cas, des approches heuristiques sont explorées comme solutions alternatives.

3.5 Proposition des heuristiques

Pour pallier le problème de la complexité NP-Hard de notre problème d'ordonnancement, il était nécessaire de proposer des heuristiques capables de retourner des solutions même pour des instances de grande taille. Toutefois, les solutions retournées par les heuristiques ne sont pas optimales.

3.5.1 Heuristique croissant/décroissant

L'heuristique croissant/décroissant trie les tâches d'une machine par ordre décroissant et les tâches de l'autre machine par ordre croissant par rapport à la consommation de la ressource ρ_j. Sans les relations de précédence, cet algorithme en temps polynomial est optimal pour la minimisation du makespan (Kellerer & Strusevich, 2003c). Pour introduire les relations de précédence à cette heuristique, nous ajoutons la notion de *tâche disponible*. Une tâche est disponible si tous ses prédécesseurs au sens des relations de précédence ont déjà été exécutés. L'ordonnancement produit est donc le schéma classique d'incrémentation/décrémentation qui prend en charge la disponibilité des tâches. Cette heuristique est présentée dans la Figure 15.

Algorithme 1 : Pseudo-code pour l'heuristique croissant/décroissant

Entrées : $N = (1, .., n)$ ensemble de tâches. Chaque tâche est affectée à une machine $i \in 1, 2$, sur laquelle une tache j sera traitée pendant $p_{i,j}$ unités de temps, en utilisant ρ_j de la ressource. Il existe des contraintes de précédence entre les tâches programmées sur la même machine ;

Sorties : Un ordonnancement des taches N sur la machine 1 et la machine 2 ;

```
ranger les tâches de la machine 1 dans l'ordre croissant;
ranger les tâches de la machine 2 dans l'ordre décroissant;
tant que il y a des tâches disponibles (au sens des précédences) faire
    si il y a au moins une tâche disponible sur la machine 2 alors
        Planifier la première tâche disponible sur la machine 2 au plus tôt;
    fin
    si il y a au moins une tâche disponible sur la machine 1 alors
        si (la somme des besoins en ressource de la tâche planifiée sur la machine 1 et de
          la première tâche disponible pour la machine 2 est inférieure à la limite
          maximale) ou (il ne reste plus de tâches à planifier sur la machine 2) alors
            Planifier la première tâche disponible de la machine 1 au plus tôt;
        fin
    fin
fin
```

Figure 15. Proposition d'une heuristique croissant/décroissant

Concrètement, cette heuristique commence par trier les tâches en fonction de leurs besoins en ressources ρ_j. Les tâches de la machine M1 sont triées en ordre croissant et celles de la machine M2 en ordre décroissant. Ensuite, la première tâche disponible $J_{2,1}$ de la machine M2 est démarrée. Pour chaque tâche disponible $J_{1,i}$ de la machine M1, l'algorithme évalue la compatibilité de son besoin en ressources avec la tâche $J_{2,1}$, déjà démarrée sur la machine M2. Si cette tâche ($J_{1,i}$,) est compatible avec la tâche démarrée, alors la tâche $J_{1,i}$, de la machine M1 est démarrée, sinon, elle est reportée jusqu'à ce qu'elle puisse l'être. Il est à noter que l'algorithme n'est pas symétrique et que l'ordonnancement généré en considérant d'abord M1 est différent de celui généré en considérant d'abord M2.

Cette heuristique est optimale dans le cas où il n'y a aucune contrainte de précédence, ou bien qu'elles vont dans le même sens que le tri par consommation de ressource. Cependant, l'optimalité de cette heuristique se dégrade en fonction du nombre de contraintes de précédence n'allant pas dans ce sens. En effet, le Makespan de cette heuristique est très mauvais pour des instances ayant par exemple des chaînes de précédences inverses par rapport au tri effectué.

3.5.2 Heuristique sens variable

Afin de résoudre le problème de la contrainte de précédence, nous proposons l'heuristique sens variable (Figure 16). L'algorithme trie les tâches de la machine M1 par ordre décroissant et les tâches de l'autre machine M2 par ordre croissant par rapport à leur consommation de la ressource. L'algorithme sélectionne la première tâche disponible $J_{1,1}$ de la machine décroissante et l'ordonnance. Ensuite, il sélectionne la première tâche disponible $J_{2,1}$ de la machine croissante et évalue sa compatibilité avec la tâche $J_{1,1}$. Si la consommation de ressources des deux tâches ne dépasse pas la valeur maximale de la ressource partagée, alors la tâche $J_{2,1}$est planifiée. Dans le cas contraire, un temps d'arrêt est généré sur la machine croissante. A ce moment, les rôles des deux machines sont inversés, c'est-à-dire que les tâches de la machine décroissante M1 sont reclassées dans un ordre croissant et les tâches de la machine M2 en ordre décroissant.

Algorithme 1 : Pseudo-code pour l'heuristique sens variable

Entrées : $N = (1, .., n)$ ensemble de tâches. Chaque tâche est affectée à une machine $i \in 1, 2$, sur laquelle une tache j sera traitée pendant $p_{i,j}$ unités de temps, en utilisant ρ_j de la ressource. Il existe des contraintes de précédence entre les tâches programmées sur la même machine ;

Sorties : Un ordonnancement des taches N sur la machine 1 et la machine 2 ;

```
ranger les tâches de la machine 1 dans l'ordre décroissant;
ranger les tâches de la machine 2 dans l'ordre croissant;
tant que il y a des tâches disponibles (au sens des précédences) faire
    si il y a au moins une tâche disponible sur la machine 1 alors
        Planifier la première tâche disponible sur la machine 2 au plus tôt;
    fin
    si il y a au moins une tâche disponible sur la machine 2 alors
        si (la somme des besoins en ressource de la tâche planifiée sur la machine 1 et de
          la première tâche disponible pour la machine 2 est inférieure à la limite
          maximale) ou (il ne reste plus de tâches à planifier sur la machine 1) alors
            Planifier la première tâche disponible de la machine 2 au plus tôt;
        sinon
            Ordonner les tâches de la machine 1 dans l'ordre décroissant;
            Ordonner les tâches de la machine 2 dans l'ordre croissant;
            Inverser les rôles de la machine 1 et de la machine 2;
        fin
    fin
fin
```

Figure 16. Proposition d'une heuristique à sens variable

Bien que cette heuristique se comporte mieux en particulier sur les cas où les précédences vont dans le sens opposé du tri par consommation de ressources, elle se comporte assez mal notamment lorqu'une tâche possède plusieurs prédécesseurs.

3.5.3 Heuristique priorité à la tâche avec plus de successeurs

Les deux heuristiques précédentes n'exploitent pas les contraintes de précédence pour améliorer le Makespan. Cette heuristique (Figure 17) vise à exploiter simultanément la contrainte de la ressource partagée et de précédence pour obtenir de meilleurs résultats. L'algorithme commence par appliquer le même principe que les heuristiques précédentes. C'est-à-dire que les tâches de la machine M1 sont ordonnées par ordre décroissant et les tâches de la machine M2 par ordre croissant. Ensuite, il sélectionne la première tâche disponible $J_{1,1}$ et la planifie sur la machine M1. Puis, il sélectionne la première tâche disponible $J_{2,1}$ de la machine M2. Si $(\rho_1 + \rho_2 > 1)$ alors, il sélectionne la tâche $J_{2,i}$ ayant le plus de successeurs directs et il la planifie au temps t le plus petit possible.

Algorithme 1 : Pseudo-code pour l'heuristique priotité à la tâche ayant plus de succes-seurs

Entrées : $N = (1, .., n)$ ensemble de tâches. Chaque tâche est affectée à une machine $i \in 1, 2$, sur laquelle une tache j sera traitée pendant $p_{i,j}$ unités de temps, en utilisant ρ_j de la ressource. Il existe des contraintes de précédence entre les tâches programmées sur la même machine ;

Sorties : Un ordonnancement des taches N sur la machine 1 et la machine 2 ;

```
ranger les tâches de la machine 1 dans l'ordre décroissant;
ranger les tâches de la machine 2 dans l'ordre croissant;
tant que il y a des tâches disponibles (au sens des précédences) faire
    si il y a au moins une tâche disponible sur la machine 1 alors
        tant que Cmaxmachine1 ⩽ Cmaxmachine2 et que toutes les tâches n'ont pas
         été testées faire
            Planifier au plus tôt la première tâche disponible sur la machine 1 qui permet
             de conserver Cmaxmachine1 ⩽ Cmaxmachine2
        fin
    fin
    si il y a au moins une tâche disponible sur la machine 2 alors
        si (la somme des besoins en ressource de la tâche planifiée sur la machine 1 et de
         la première tâche disponible pour la machine 2 est inférieure à la limite
         maximale) ou (il ne reste plus de tâches à planifier sur la machine 1) alors
            Planifier la première tâche disponible de la machine 2 au plus tôt;
        sinon
            Planifier au plus tôt la première tâche disponible de la machine 2 ayant le plus
             de fils (graphe de précédences);
        fin
    fin
fin
```

Figure 17. Proposition d'une heuristique priorité à la tâche ayant plus de successeurs

3.5.4 Evaluation des heuristiques

Pour illustrer l'efficacité des heuristiques proposées précédemment, cette section présente des résultats qui comparent la solution retournée par le modèle disjonctif proposé et la meilleure valeur obtenue par toutes les heuristiques proposées. L'écart moyen des temps de calcul entre le modèle disjonctif et la meilleure solution des heuristiques est calculé sur 200 instances de même taille pour lesquels les heuristiques en question ont pu obtenir un ordonnancement réalisable. Cet écart est calculé avec la formule suivante :

$$Ecart\ moyen\ = \frac{(SolutionHeuristiques - SolutionOptimale)}{SolutionOptimale}$$

Les résultats (Figure 18) illustrent la variation des performances entre différentes tailles d'instance à l'aide d'une barre à moustaches, où chaque tracé encapsule les variations de 200 instances. Par exemple, dans les 200 instances de taille 13x13, le premier quartile, la clôture inférieure et la valeur minimale s'élève à 0 %, tandis que l'écart médian, représenté par une ligne horizontale dans la case, est de 1,45 %. Le troisième quartile, formant la limite supérieure, est évaluée à 4,5 %. La limite supérieure, calculée comme le troisième quartile plus 1,5 fois l'intervalle interquartile, atteint 10,5 %, avec 10 valeurs aberrantes s'étendant jusqu'à un écart maximum de 19,3 %. L'écart moyen est indiqué par une ligne horizontale pointillée et s'élève à 2,9%. Dans des instances plus petites, allant de 2 × 2 à 6 × 6, l'écart moyen s'approche si près de zéro que les barres apparaissent réduits en un point, marquant toute variation non nulle comme une valeur aberrante. Un exemple notable à 3x4 a montré une déviation jusqu'à 29,95 %, ce qui indique une performance heuristique sous-optimale. Cependant, parmi les 4 800 cas représentés, seulement 8 cas aberrants ont dépasser un écart de 20 %, affectant principalement les petites instances où un faux pas de planification unique a un impact significatif sur l'écart global. On peut également observer que pour les instances supérieures à 20x20, l'écart moyen se stabilise en dessous de 5%, tandis que l'écart supérieur se stabilise autour de 15%.

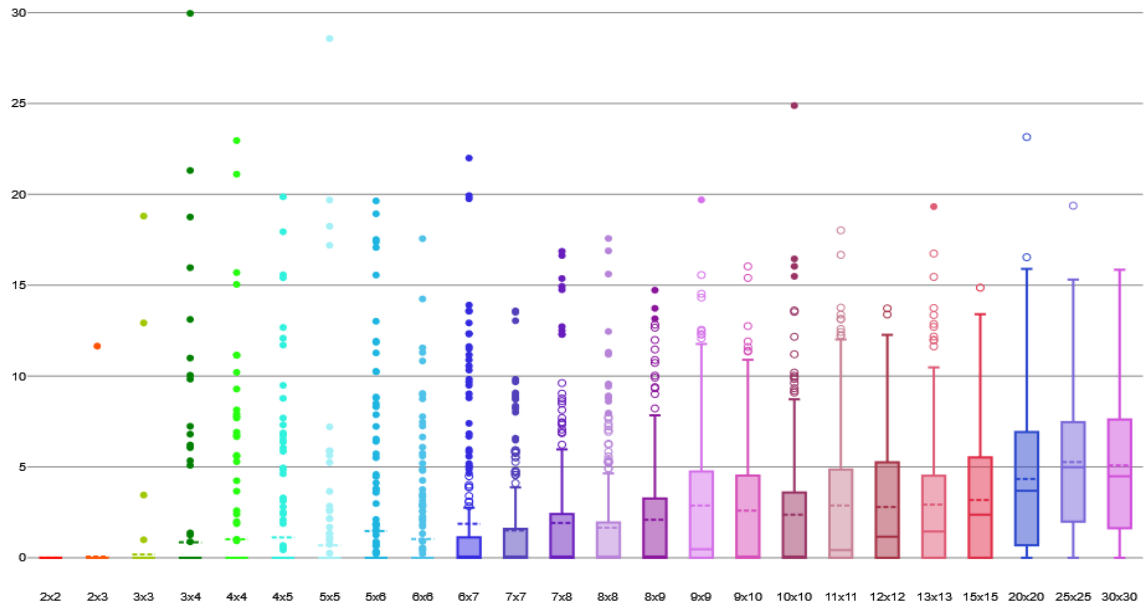


Figure 18. Déviation de la meilleure heuristique/meilleur solveur.

D'autre part, afin d'identifier la meilleure heuristique, qui retourne des solutions les plus proches de l'optimal, la Figure 19 compare les trois heuristiques, incorporant toutes les variantes, (1) en termes de pourcentage d'instances pour lesquelles une variante fourni une solution optimale ou, dans les cas où le(s) solveur(s) a expiré, la meilleure solution obtenue dans les 10 minutes. (2) les trois heuristiques sont comparées à l'aide des trois courbes marquées d'un « x ». Pour les instances de taille 10x10, les trois familles d'heuristiques, en utilisant une de leurs variantes, produit une solution optimale pour respectivement 32%, 38% et 38% des instances, tandis qu'une des heuristiques (courbe noire en pointillés) a produit un ordonnancement optimal pour 53,5 % des instances. Indépendamment de l'optimal, mais seulement d'un point de vue du makespan, la famille Heuristics 2 donne le meilleur makespan parmi les heuristiques pour 61,5 % des instances, suivie par la famille 3 avec 58,5% et famille 1 pour 48%. L'heuristique 2, qui se distingue par sa stratégie d'alternance de la machine principale, démontre les meilleures performances dans

environ 70 % des instances pour les instances supérieures à 7x7. Notez que les pourcentages ne totalisent pas 100 % en raison des liens. Il est à noter que, puisque pour toutes les instances, le temps d'exécution cumulé de toutes les heuristiques et de leurs variantes est moins de 63 millisecondes, l'intégration de stratégies supplémentaires est réalisable.

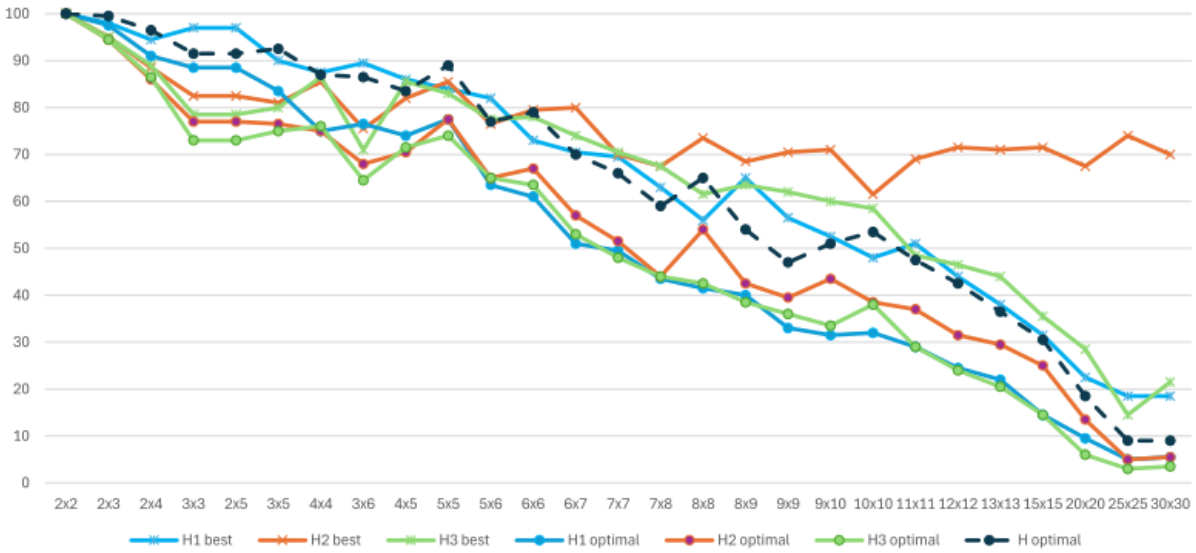


Figure 19 : Comparaison des familles des heuristiques.

4 Conclusion

Nous avons proposé un nouveau modèle mathématique pour le problème d'ordonnancement $(PD2|res1\cdot\cdot,localprec|Cmax)$ avec ressource sur deux machines parallèles et dédiées avec la prise en compte des contraintes de précédences locales.

Initialement et afin d'éviter toute ambiguïté, nous avons posé une caractérisation du problème selon les champs $\alpha|\beta|\gamma$ et selon la caractérisation des problèmes RCPSP, basée sur la notation $res\lambda\sigma\rho$.

Une fois que la caractérisation a été fixée, nous avons proposé une nouvelle formulation du problème allégée et simplifiée (formulation disjonctive) par rapport aux formulations classiques du RCPSP telles que la Discrete Time DT. Cette formulation prend en compte la contrainte de ressource et les contraintes de précédences sans avoir à utiliser des variables binaires temporelles. En effet, les évaluations menées sur ces deux formulations montrent que la formulation disjonctive permet de traiter des instances plus grandes que celles traitées par la formulation temporelle. Les expérimentations réalisées sur trois solveurs différents (Cplex, SMT Z3 et Gurobi) montrent que le solveur Cplex reste le solveur le plus performant pour cette formulation disjonctive.

L'étude de complexité de notre problème d'ordonnancement nous a permis de prouver qu'il est NP-difficile au sens fort (NP-hard). Ces résultats sont aussi validés expérimentalement. En effet, notre algorithme (formulation disjonctive) est incapable d'ordonner certaines instances comportant plus de 9 tâches sur chaque machine en moins de 10 minutes.

Pour pallier ce problème, plusieurs heuristiques, qui permettent de résoudre efficacement le problème RCPMSP, ont été proposées. Ces dernières ont été évaluées par rapport à la solution optimale. Les résultats montrent que ces heuristiques sont efficaces en moyenne dans la résolution des instances très grandes, mais que certains cas particuliers pouvaient voir le makespan obtenu très au-delà (jusqu'à 45% observé sur des instances 13x13) de l'optimal.

Partie II : Génération de séquences de commande SCADA opérationnelles

Chapitre 4 : Etat de l'art sur la génération de séquence

1 Introduction

L'ordonnancement permet l'affectation temporelle d'un ensemble de tâches tout en respectant certaines contraintes (de précédences, ressource, etc.). A cet effet, l'ordonnancement tente de transformer une demande de production en une commande déterministe prête à être appliquée sur le système (J.-C.Gentina, 2000). Dans notre cas, l'ordonnancement permet de calculer une solution qui minimise le temps total d'exécution des fonctions d'un banc de test SCADA. Cette solution (sous forme de séquence d'exécution de fonctions) est calculée par des modèles mathématiques qui prennent en considération des contraintes de précédences, de ressource afin de les exécuter sur deux machines parallèles initialement dédiées. Ces modèles mathématiques ont été évalués par calcul de performance face à la taille d'instance ou le temps de calcul.

La mise en œuvre opérationnelle de l'ordonnancement nécessite de mettre le système ainsi que ses éléments (fonctions et composants) dans des états précis leurs permettant de répondre aux objectifs de l'ordonnancement. Les changements d'états des composants sont soumis à des contraintes d'ordre opérationnel que l'algorithme d'ordonnancement ne peut prendre en compte. En effet, au niveau des fonctions par exemple, des sous-fonctions sont exécutées soit en séquentiel ou en parallèle. Egalement, lors de l'exécution d'une fonction, d'autres contraintes de sécurité portant sur les composants sont également respectées, comme par exemple, une pompe ne peut être enclenchée que si toutes les vannes en aval sont en état ouvert. D'autre part, au niveau du système complet, des contraintes de compatibilité entre les différentes éléments du système (fonctions, composants) ainsi que des contraintes les modes de fonctionnement (automatique, manuel, dégradé, etc.) sont ajoutées, aux contraintes des modes de fonctions (exécution, abandon, etc.) et de composants (ouvert, fermé) qui sont liés par leurs contraintes de comportement (Berruet, 1998). Toutes ces contraintes opérationnelles ne peuvent pas être prises en compte dans l'algorithme d'ordonnancement.

2 Problématique

Bien que les modèles d'ordonnancement donnent une *séquence abstraite*, la faisabilité opérationnelle de la séquence abstraite est possible à travers *des modèles concrets*. En effet, la validité opérationnelle de la séquence abstraite n'est connue que lorsque l'actionnement des composants physiques nécessaires à la mise en œuvre de cette séquence est possible. Cette validité opérationnelle est ainsi conditionnée d'une part par la disponibilité des composants et d'autre part par l'adéquation des actions d'actionnement et de manœuvrabilité de ces composants par rapport aux contraintes opérationnelles. Lors de l'exécution opérationnelle de la séquence d'ordonnancement, les opérateurs peuvent rencontrer des problèmes, qui les empêchent de mener à bien cette exécution. Parmi ces problèmes, on peut citer : :

1. *Actions opérationnelles non-connues*. La séquence d'ordonnancement est une séquence d'exécution de fonctions. Toutefois, pour la mise en œuvre d'une seule fonction de la séquence, les opérateurs doivent mettent les composants physiques dans des états précis (ouvert, fermé) leur permettent d'accomplir la fonction. L'enchaînement de ces activation d'états est soumis à la connaissance préalable de l'état du composant nécessaire à la réalisation de la fonction et l'action qui permet de le mettre dans l'état souhaité. Ces actions d'activations sont enchaînées en construisant une séquence opérationnelle (ou une configuration). Il est à noter que pour mettre en œuvre une fonction, plusieurs configurations sont possibles. Sur les systèmes SCADA complexes, la gestion de toutes les

configurations des différentes fonctions peut s'avérer difficile et également source d'erreur lorsqu'elle est réalisée manuellement par les opérateurs (Goubali, 2017).

2. *Non-connaissance des disponibilités des composants et leurs états*. Sur les système SCADA, un composant peut être utilisé par plusieurs fonctions. Mais lorsqu'un composant est utilisé par une fonction, les autres fonctions ne peuvent plus l'utiliser. Elles doivent attendre qu'il soit de nouveau disponible. Pour un système SCADA, le nombre de composants est généralement très conséquent et la gestion de la disponibilité des composants devient très compliquée pour un opérateur humain et peut conduire à des incohérences sur le comportement du système. D'autre part, mettre en œuvre l'ordonnancement calculé peut être infaisable opérationnellement à cause de la non-disponibilité des composants. En effet, si l'ordonnancement préconise d'exécuter deux fonctions en parallèle et que ces fonctions se partagent certains composants, alors cette exécution parallèle n'est pas réalisable opérationnellement.
3. *Contraintes opérationnelles non prises en compte*. L'algorithme d'ordonnancement ne peut prendre en entrées que des contraintes techniques formulées mathématiquement. Les contraintes opérationnelles telles que : une pompe ne peut être ouverte que si toutes les vannes, qui lui sont en aval, sont déjà ouvertes ne peuvent être prises en compte par l'algorithme d'ordonnancement. En plus, dès qu'une entité change de comportement (passage d'un état à un autre), tout le comportement du système change ce qui peut conduire à des incohérences entre les états. L'absence des contraintes opérationnelles peut conduire à des blocages.
4. *Dysfonctionnement du système*. L'ordonnancement calculé ne prend pas en compte la non disponibilité des composants ou des fonctions à cause de certains dysfonctionnements qui peuvent surgir sur le système. Dans le cas d'un dysfonctionnement, les opérateurs doivent décider de sélectionner un nouvel état du système lui permettant de continuer de fonctionner d'une manière dégradée (reconfiguration), ou d'arrêter complètement le système pour des questions de sûreté. Toutefois, la reconfiguration d'une fonction peut remettre tout l'ordonnancement en question. En effet, il se peut que la reconfiguration choisie par les opérateurs ne respecte plus les contraintes de ressources partagées et de précédence, utilisées dans l'algorithme d'ordonnancements.

Dans le but d'étudier la faisabilité opérationnelle de la séquence de commande abstraite, calculée par notre algorithme d'ordonnancement, nous proposons dans cette partie une solution permettant de calculer automatiquement la séquence opérationnelle à partir de l'ordonnancement abstrait et d'étudier sa faisabilité en utilisant la gestion des modes.

Pour l'obtention d'un modèle opérationnel complet d'un système SCADA, nous dressons dans les sections suivantes, un état sur les approches et solutions proposées dans la littérature pour la gestion des modes des : composants, fonctions et système. L'état de l'art présente également les techniques de vérification et de validation généralement utilisées pour s'assurer de la qualité du modèle opérationnel.

3 La gestion de modes pour la modélisation opérationnelle

Les travaux sur la supervision des systèmes sociotechniques industriels proposent de combiner l'ordonnancement à un module de *gestion des modes* (Berruet, 1998; Dangoumau, 2000a). La sollicitation de la gestion des modes est nécessaire pour vérifier la faisabilité opérationnelle de l'ordonnancement. En effet, si l'ordonnancement agit sur le plan décisionnel, la gestion des modes agit sur le plan opérationnel (actions). La gestion des modes permet de fournir les informations opérationnelles afin de mettre un système dans un état qui lui permet de réaliser ses fonctions. En fonctionnement dit normal, la gestion des modes fournit les disponibilités des composants

ainsi que les actions à effectuer sur les composants physiques selon le plan de l'ordonnancement. En fonctionnement dégradé (une panne existe), la gestion des modes fournit les différentes configurations possibles et les actions de transition vers un nouvel état sans panne. Elle permet d'informer sur les séquences alternatives possibles et donne les informations opérationnelles pour actionner les composants selon le fonctionnement dégradé.

3.1 Définition

La gestion des modes ou approche modale est le fait d'étudier le comportement du système et celui de ses entités à travers les modes (Kamach & Piétrac, 2006). Le mode est un ensemble d'états caractérisant une ressource ou un ensemble de ressources selon un point de vue. A un instant donné, dans un mode, un seul état est actif (Dangoumau, 2000a). L'enchaînement ou la commutation d'un mode à un autre permet de décrire un fonctionnement continu du système.

La mise en place d'une gestion des modes nécessite la connaissance de tous les modes du système (Faraut, 2010). Pour un système complexe, un mode n'est pas suffisant pour décrire un fonctionnement donné. Par exemple : l'opérateur veut mettre le système en marche. Alors il doit préciser si la marche désirée est la marche automatique ou la marche manuelle. En revanche, marche tout seul ne signifie rien. Ce niveau de détail est décrit par les états du mode. En effet, si un mode décrit un fonctionnement non permanent durant lequel le système réalise une fonction, l'état décrit la manière par laquelle le système réalise cette fonction dans cet intervalle de temps.

Charles S. Wasson a mentionné dans ses travaux (Wasson, 2011) que modes et états du système sont souvent les concepts les plus controversés de l'ingénierie des systèmes en raison du manque de normes de mise en œuvre. En effet, certains utilisent « état » comme ensemble de modes et d'autres modes comme ensemble d'états. Ce manque de clarification laisse les concepteurs utiliser ces concepts directement dans le code en ignorant que mode et état permettent d'autoriser ou d'interdire des actions afin de respecter des contraintes de sécurité. Pourtant, ce sujet est l'un des principes directeurs les plus importants de l'ingénierie des systèmes. En effet, utiliser les modes et les états pour décrire un système complexe permet de le décomposer en niveaux et entités gérables avec un risque acceptable conduisant à des solutions de conception du système optimales. Le mode est une option à sélectionner par l'utilisateur, l'état est la manière qui permet de réaliser ce mode.

3.2 Difficulté de mise en œuvre de la gestion des modes

Généralement, pour mettre en œuvre une gestion des modes, les concepteurs doivent recueillir toutes les informations relatives au comportement dynamique, nécessaires pour l'implémentation et l'exploitation des fonctions complexes du système. Ces informations sont divisées en trois niveaux : composant, fonction et système.

Au niveau des composants, il s'agit de définir l'état des composants (ouvert ou fermé, etc.) qui détermine leur positionnement ou leur disponibilité ainsi que toutes les contraintes opérationnelles liées à ces composants. Au niveau des fonctions, les informations d'exploitation concernent les états de la fonction ainsi que la manière dont une fonction interagit avec les autres fonctions du système et également comment elle met en place les composants qui entre dans sa configuration. Enfin, au niveau système, il s'agit de recenser les informations utiles pour répondre aux objectifs permettant d'assurer le bon fonctionnement du système (fonctionnement normal, fonctionnement dégradé, etc.).

La prise en compte de toutes ces informations pour les systèmes SCADA est indispensable pour obtenir des systèmes sûrs de fonctionnement (Dangoumau, 2000a). Cependant, ce comportement à une gestion des modes est difficile à déterminer pour un système SCADA. Par ailleurs, la non-existence d'une gestion des modes peut provoquer : un problème de sécurité lors du changement

de comportement, un ordonnancement pas faisable opérationnellement, des blocages inexpliqués dont les opérateurs ignorent les causes.

4 Mise en œuvre d'une gestion des modes dans la littérature

Dans cette section, nous dressons un état de l'art sur les approches et solutions qui ont été proposées dans la littérature pour la gestion des modes au niveau : composant, fonction et système.

4.1.1 Gestion des modes d'un composant

Les travaux (Dangoumau, 2000a) proposent un ensemble de 4 modes pour un composant et spécifient également pour chaque mode ses états. La Figure 20 illustre ce modèle.

Dans la famille de mode production, un composant est caractérisé par 4 modes : Arrêt, Marche, Fonctionnement et Production. Puis, dans chaque mode, un ensemble des états est défini. Par exemple, le mode Arrêt comporte 5 états : arrêt non-significatif (PA-ns), arrêt déterminé (PA-d), arrêt initial (PA-i), arrêt de fin de cycle (PA-c) et arrêt hors service (PA-hs). Les transitions entre ces états sont également spécifiées.

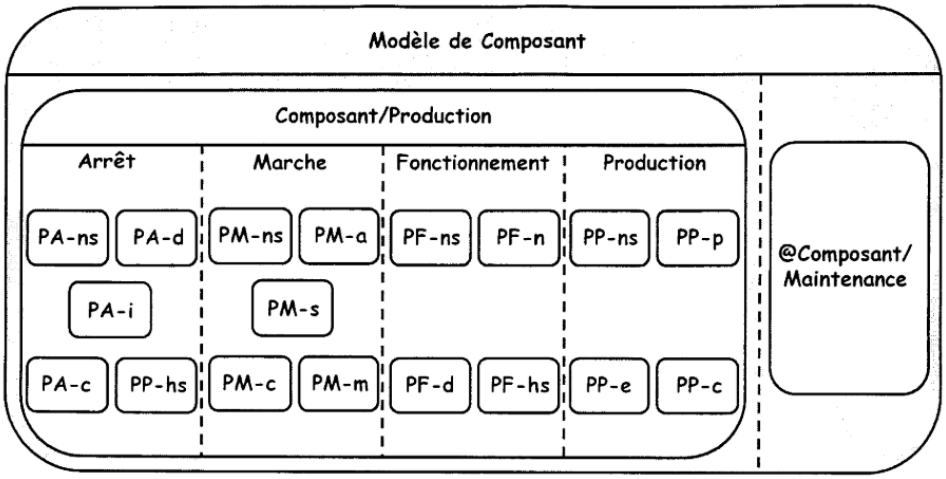


Figure 20. Gestion des modes pour un composant (Dangoumau 2000)

D'autre part, plusieurs travaux dans la littérature proposent la modélisation du comportement des composants d'un système SCADA. Ces travaux modélisent un composant à travers ses états et les transitions entre ces états (Mesli-kesraoui, 2016 ; Mesli 2017 ; Cochard, 2017). Toutefois, ces solutions ne spécifient pas les modes de ces composants.

4.1.2 Gestion des modes d'une fonction

La gestion des modes d'une fonction consiste à identifier ses modes et les différents états de ses modes. La Figure 21 illustre le modèle de fonction proposé dans (Dangoumau, 2000a). Dans ce modèle, une fonction est caractérisée par les mêmes modes que les composants et deux autres modes qui lui sont spécifiques : le mode de disponibilité et le mode des alternatives. Le mode disponibilité comporte deux états qui décrivent si la fonction est disponible ou non disponible dans un instant donné. Le mode des alternatives décrit quant à lui, les différentes configurations et versions de la fonction.

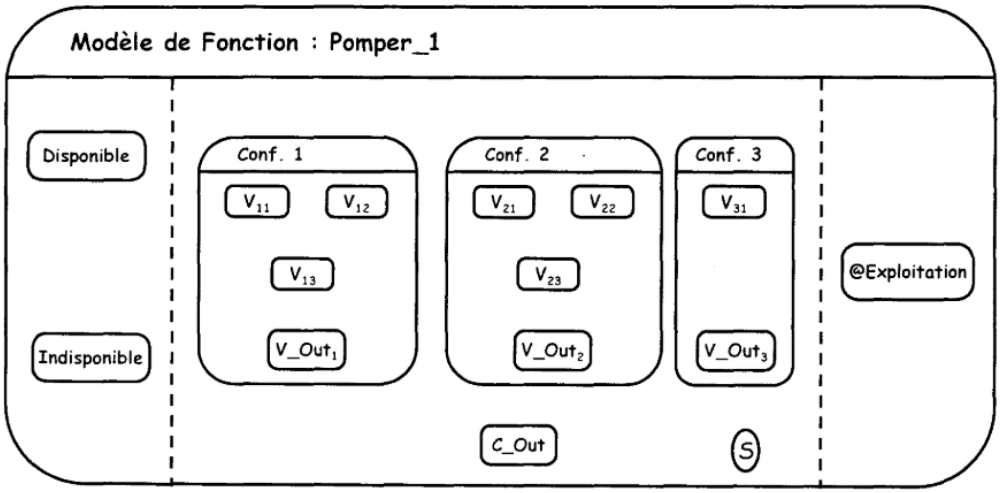


Figure 21. Gestion des modes d'une fonction (Dangoumau, 2000a)

D'autre part le standard PackML (Barbieri et al., 2015) définit trois modes qui gèrent les machines de production ou les services : Production, maintenance et manuel. Le mode production (Figure 22) comporte 7 états fonctionnels et 10 états intermédiaires. Il définit également les différentes transitions permettant le passage d'un état à un autre.

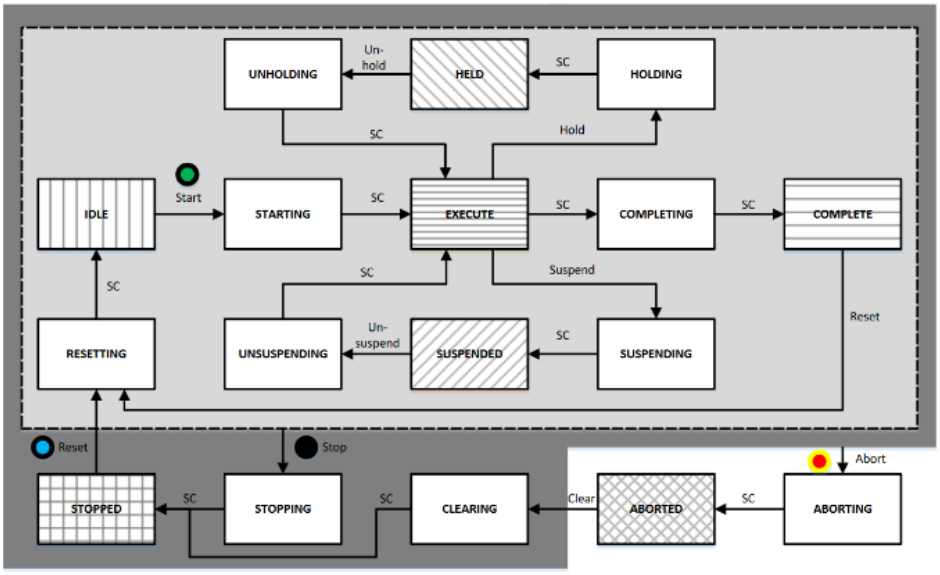


Figure 22. Le modèle d'état de l'interface PackML

Dans la Figure 23, un modèle décrivant les états d'une fonction dans le mode production. Dans ce mode, la fonction est en état attente, configuration ou en exécution. Nous pouvons constater sur ce modèle que l'état configuration par exemple permet de configurer les composants de la fonction.

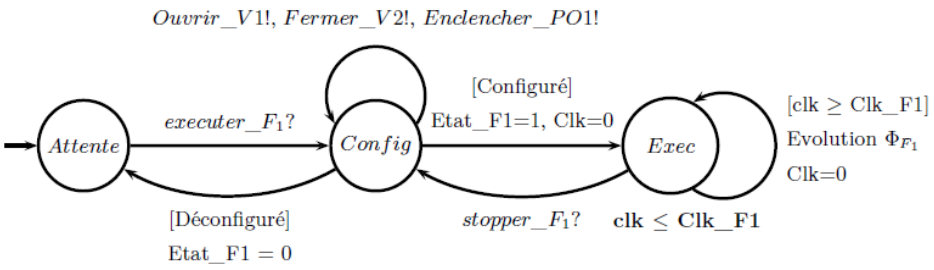


Figure 23. Modèle d'une fonction (Cochard, 2017)

4.1.3 Gestion des modes du système

Gérer les modes d'un système revient à gérer le comportement de tous ses modes, résultants des modes de ses fonctions et de ses composants ainsi que les différentes interactions et commutations entre tous ces modes.

Les premiers travaux sur la gestion des modes des systèmes complexes ont permis de proposer un guide sous forme d'une représentation graphique appelée GEMMA (Des et al., s. d.). Dans le GEMMA, les modes sont caractérisés par des rectangles contenant des sous-rectangles représentant les états de chaque mode. Les conditions de passage entre les états sont modélisées par des arcs. De ce fait, la modélisation du système se fait par la représentation de tous les états pour décrire son fonctionnement complet, qu'il soit normal, dégradé ou défaillant. Bien que le GEMMA soit un guide pour spécifier les modes de marche et d'arrêt d'un système complexe, il ne peut pas être utilisé pour des systèmes complexes regroupant plusieurs composantes ayant différents comportements. Pour la définition des modes d'un système, trois approches existent : l'approche structurelle, l'approche fonctionnelle et l'approche structuro-fonctionnelle.

4.1.3.1 Décomposition structurelle

La gestion des modes structurelle permet de décrire le comportement du système en se basant seulement sur le comportement de ses composants. La mise en œuvre d'une gestion des modes d'un point de vue structurel, permet de gérer les commutations de modes entre composants. En effet, la mise à l'arrêt ou marche d'un composant engendre la mise à l'arrêt d'un autre composant fortement lié avec le précédent (Kermad, 1996).

Dans une gestion des modes structurelle les contraintes opérationnelles physiques telles que la coopération, l'exclusion (Kermad, 1996) sont fortement prises en compte. L'approche proposée par (Stéphane, 1991) est nettement structurelle et permet de construire un modèle structurel qui spécifie les composants d'un système ainsi que leurs liens structurels. Toutefois, elle ne permet pas de représenter toutes la flexibilité du système. Par exemple : un composant est lié avec d'autres composants formant une cellule (Stéphane, 1991). Chaque changement d'état d'un des composants de la cellule affecte automatiquement les autres composants, de fait si un composant est en *Hors-service*, toute la cellule est en *Hors-service*. Dans certains cas, chaque composant peut être utilisé séparément.

Bien qu'elle prenne en compte les contraintes opérationnelles entre composants, l'approche structurelle ne prend en compte les contraintes opérationnelles des fonctions. La non liaison entre les états du composant et sa fonction restreint la visibilité du système.

4.1.3.2 Décomposition fonctionnelle

Dans l'approche fonctionnelle, le système est découpé en fonctions. Chaque fonction est ensuite découpée en états. La gestion des modes fonctionnelle suppose que le système soit apte à remplir sa mission en respectant les contraintes techniques d'ordre fonctionnel et non-fonctionnel issus d'un référentiel d'exigence (Chapurlat et al., 2013). Cette approche vise à s'assurer que le

comportement du système à faire aboutisse à la réalisation de sa mission d'une manière crédible. Pour cela, les commutations entre modes sont gérées par des matrices de compatibilités. En effet, le but est de vérifier que le système évolue pour remplir sa mission sans blocages. Pour couvrir ces besoins, plusieurs approches ont proposé des modèles graphiques qui permettent de décrire les fonctions d'un système complexe et les commutations entre ces fonctions (ADEPA, 1981; Chapurlat et al., 2013).

Dans cette approche fonctionnelle, le système évolue pour remplir ses missions sans blocage. Cependant, les fonctions du système ne sont pas reliées aux composants physiques qui sont eux-mêmes mis sous contraintes de comportement. Si un composant est en *Hors-service,* sa fonction physique n'est pas disponible, donc la fonction du système n'est pas possible. De plus, la fonction est directement mise en œuvre en actionnant les composants.

4.1.3.3 Décomposition structuro-fonctionnelle

La gestion des modes au niveau système repose sur la gestion des relations entre les fonctions (décomposition fonctionnelle) et également les liens entre les fonctions et les composants (configuration). Elle consiste également à gérer la cohérence des modes des fonctions et composants ce qui permet d'assurer la réalisation de la fonction.

Le gestion des modes du système dans (Dangoumau, 2000a) permet de spécifier comment les fonctions du système sont obtenues. Le résultat est un graphe dont la racine représente la fonction pour laquelle le système a été conçu. Cette fonction est ensuite liée par des liens (ET/OU) à d'autres fonctions de niveau intermédiaire. Ces dernières sont, quant à elles, reliées aux fonctions physiques (réalisées par les composants) par des liens (ET/OU). Le modèle de fonction comporte également les différentes configurations d'une fonction et spécifient ainsi les liens entre les fonctions et les composants. Toutefois, le niveau de décomposition fonctionnelle dans cette approche n'est pas fini et n'est pas cadré.

4.1.3.4 La WDA pour un cadre de décomposition structuro-fonctionnel

La WDA est une représentation sur plusieurs niveaux d'abstraction, utilisée pour décrire les systèmes complexes (Rasmussen, 1984). Les niveaux hauts (Tableau 4) décrivent le système en terme d'objectifs et fonctions, tandis que les niveaux bas décrivent le système en terme de fonctions physiques (composants). L'atteinte des objectifs est une décomposition et liaison structuro-fonctionnelle « de haut en bas », par ailleurs la non réalisation des objectifs est liée aux changements de configurations des entités physiques (niveau le plus bas) qui est déduite facilement par une lecture « de bas en haut ».

La WDA est largement utilisée dans la littérature dans la représentation de différents types d'informations des systèmes industriels (Anokhin et al., 2018; King et al., 2022; Wang et al., 2018). Ainsi la WDA peut être utilisée comme cadre pour représenter les fonctions des systèmes complexes et peut réduire la complexité du système (seulement 5 niveau de décomposition), tout en ayant une liaison avec les composants physiques. Elle peut ainsi être utilisée comme une représentation structuro-fonctionnelle d'un système SCADA dans le cadre de la gestion de ses modes.

Tableau 4. Définition des niveaux d'abstractions de la WDA

Fonctions WDA	
Objectif fonctionnel	Représente les objectifs pour lesquels le système a été conçu. Autrement dit, ce sont les services délivrés par le système.
Fonction abstraite	C'est la décomposition des services du système en sous services ou sous objectifs. Chaque sous-système réalise un service différent de l'autre. Une fonction abstraite est mise en œuvre par plusieurs fonctions génériques.

Fonction générique	C'est une fonction réalisée par plusieurs ressources. Une fonction générique est mise en œuvre par une ou plusieurs *opérations* (fonction physique).
Fonction physique	C'est une fonction réalisée par une ressource. Ce type de fonction que nous préférons le nommer *opération* en se basant sur la définition suivante *« Une opération c'est une fonction mise en œuvre par une ressource lorsque la ressource est considérée comme un composant du système »* (Berruet, P., Toguyeni, A. K. A., Elkhattabi, S., and Craye, 1999). Les fonctions physiques ou opérations réalisées par les ressources sont statiques, c'est-à-dire pour chaque type de ressource la fonction physique de cette dernière est toujours la même et ne dépend pas du contexte.

5 Vérification et validation du modèle

5.1 Vérification & validation

Les méthodes de vérification et de validation sont utilisées pour garantir la qualité d'un système ou d'un produit. La vérification a pour objectif de vérifier que les étapes de conception ou de modélisation ont été bien faites et que des erreurs n'ont pas été introduites dans ces étapes. Selon l'ISO (ISO, 2015), la vérification est « la confirmation par des preuves tangibles que les exigences spécifiées ont été satisfaites ». En effet, vérifier c'est répondre à la question : « Faisons-nous le produit correctement ? ». A contrario, la validation permet de s'assurer que les schémas et codes produits répondent aux exigences des clients. Selon l'ISO (ISO, 2015), la validation est la « Confirmation par des preuves tangibles que les exigences pour une utilisation spécifique ou une application prévue ont été satisfaites ». En effet, valider c'est répondre à la question : « Faisons-nous le bon produit ? ».

La vérification est assurée par des méthodes formelles. Ces dernières sont des techniques basées sur les mathématiques qui permettent de vérifier formellement des spécifications écrites en langages formels (Førner & Havelund, 2014). Le caractère mathématique de ces méthodes, les rend plus précises et exhaustives comparées aux tests (Bennion & Habli, 2014). Ainsi, ces techniques sont de plus en plus utilisées pour la vérification des systèmes actuels. Le model checking est la méthode formelle la plus utilisée dans l'industrie. Le model checking (Schnoebelen, 1999) consiste à vérifier automatiquement les propriétés souhaitées sur une modélisation formelle du système. Le système est modélisé par des machines à états et les propriétés à vérifier sur le système sont écrites dans une logique temporelle comme CTL (Clarke & Emerson, 1981). Le model checker, par exemple UPPAAL (Larsen et al., 1997), explore et vérifie dans chaque état la satisfaction des propriétés désirées. Si la propriété n'est pas vérifiée, il retourne un contre-exemple illustrant la trace de l'erreur (Schnoebelen, 1999). Toutefois, pour les systèmes de taille importante, cette exploration exhaustive d'états peut dépasser la capacité des machines (ordinateurs), ce qui fait que la vérification échoue. Ce problème est connu sous le nom de l'explosion combinatoire de l'espace des états.

La validation est assurée souvent par des tests. Le test est le processus d'exécution d'un programme ou d'une partie d'un programme avec l'intention de trouver des erreurs (Wallace & Fujii, 1989). Le test permet de détecter les erreurs d'un système par rapport à un ensemble de scénarios définis antérieurement (Beizer, 2003). De ce fait, cette méthode est loin d'être exhaustive. Le test est pratiqué généralement après l'intégration du système, une fois que ce dernier est opérationnel. La correction des erreurs détectées à ce stade de développement (intégration), est plus coûteuse (Boehm, 1984). Néanmoins, cette méthode de vérification reste la plus utilisée dans l'industrie (Beizer, 2003), car elle peut être réalisée sur des systèmes de taille importante. Elle permet aussi de vérifier le système réel et pas seulement une abstraction de celui-ci, comme dans le cas des méthodes formelles.

La vérification formelle est utilisée dans les premières phases de conception où le niveau d'abstraction est très élevé. Elle permet de démontrer que la conception est bien faite et qu'elle ne contient pas d'erreurs. Toutefois, elle souffre du problème d'explosion combinatoire. Les tests quant à eux, sont appliqués sur les systèmes finis et de grande taille car ils ne souffrent pas du problème d'explosion combinatoire. Toutefois, les tests ne garantissent pas la non présence des erreurs car la complétude des scénarios des tests ne peut pas être garantie.

5.2 Vérification & validation des modes de composants et fonctions

Dans la littérature, les propriétés souvent vérifiées sur les modèles de modes sont de deux types: les propriétés basiques et les propriétés spécifiques (Dangoumau, 2000; Hamani et al., 2009a). Les propriétés basiques permettent de vérifier la validité des modèles élémentaires de chaque mode (intra-mode) telles que la non existence d'état puit, l'atteignabilité, etc. ainsi que les conditions de compatibilité entre modes (inter-modes). Les propriétés spécifiques sont des propriétés spécifiques au système traité et son contexte.

Dans la littérature, des travaux ont tenté de vérifier les propriétés basiques à travers des calculs matriciels (Dangoumau, 2000). Les lignes et les colonnes de la matrice représentent tous les états du système et les cases de la matrice sont remplis par des 1 et des 0. Si les 1 représentent les états compatibles (peuvent être actifs en même temps), les 0 représentent les états non-compatibles (ne peuvent pas être actifs en même temps). Toutefois, les calculs matriciels permettent simplement de vérifier qu'il existe des transitions sortantes et entrantes de chaque état (éliminer les états puits et les états non-atteignables). Les calculs matriciels ne permettent pas de garantir la satisfaction des contraintes techniques ajoutées sur les arcs des transitions. Une vérification formelle est alors nécessaire pour garantir le respect de ces contraintes techniques.

Les travaux de Hamani et al. (Hamani et al., 2009b) proposent de vérifier formellement les propriétés basiques des modes tout en respectant les contraintes techniques. Ils proposent aussi de décomposer les modes selon plusieurs niveaux de décomposition. Cependant les propriétés spécifiques ne sont pas vérifiées. Les travaux de Kadri et al. (Kadri et al., 2013) proposent un modèle formel pour la vérification formelle des propriétés inter-modes (entre deux modes) et ne vérifient pas les autres propriétés. Dans (Faraut et al., 2009), les auteurs proposent une modélisation formelle des modes et leurs commutations. Le but de cette modélisation est d'assurer mathématiquement que les commutations entre modes (inter-modes) respectent les contraintes techniques.

Cependant, les modèles formels proposés dans les approches de vérification formelle existantes ne permettent pas de vérifier les propriétés spécifiques. En effet, contrairement à un modèle du système qui modélise tout le système et ses modes, les modèles utilisés dans ces approches sont élémentaires. Les propriétés spécifiques ne peuvent se vérifier que sur un modèle de comportement complet, comme celui du système.

Etant donné que la WDA permet une décomposition structuro-fonctionnelle des fonctions et des composants, la vérification et la validation d'un modèle du système, modélisé avec la WDA, revient à vérifier et à valider les informations de la WDA. Nous présentons dans ce qui suit les travaux qui ont traité la vérification et la validation d'une matrice WDA.

5.3 Vérification et validation de la WDA

La WDA a été largement utilisée dans la représentation de différents types d'informations des systèmes industriels (Anokhin et al., 2018; King et al., 2022; Wang et al., 2018). Toutefois, un petit nombre de travaux qui se sont intéressés à la vérification formelle de la WDA. La WDA a été essentiellement validée par des tests. En effet, le but est de dérouler des scénarios de tests sur la WDA afin de valider la cohérence et la complétude des informations qu'elle contient. Dans (Burns

et al., 2001; Miller & Vicente, 1998), des scénarios ont été joués sur la WDA afin de valider sa complétude et sa cohérence. Les travaux dans (Rechard et al., 2015) ont permis de vérifier et de valider les informations de la WDA à travers une simulation des scénarios de test par machine de Turing.

Toutes ces approches utilisent les scénarios de tests pour valider expérimentalement les informations de la WDA. Toutefois, les résultats de validation ne peuvent pas être généralisés car un sous-ensemble de scénarios est validé sur la matrice WDA. En effet, il n'est pas possible de jouer tous les scénarios sur la matrice pour la valider. D'autre part, aucune approche formelle n'a été proposée pour la vérification formelle de la WDA.

6 Génération automatique de séquences de commande

La génération de séquences de commande repose sur une modélisation du système à base de machine à états et sur des séquences recherchées sous forme de trace. Le modèle du système permet de modéliser les composants et les fonctions du système avec leurs états et comportement (transitions). L'ordonnancement est modélisé par une séquence de tâches. L'objectif est de vérifier formellement une propriété d'atteignabilité pour la réalisation de l'ordonnancement sur le modèle du procédé.

6.1 Modélisation du système

Le système est composé de composants dont chacun réalise une fonction élémentaire. Les composants sont mis dans des états opérationnels pour la mise en œuvre de fonctions du système. Chaque fonction a différents états qui la spécifient pendant son fonctionnement (exécution, complétion, etc.) en adéquation avec l'état du système (automatique, manuel, etc.)

6.1.1 Modélisation des composants et leurs comportements

Dans les travaux de (Cochard, 2017; Cochard et al., 2015), l'ensemble des composants $C = \{c_i, .., c_n\}$, manipulé par le contrôle-commande ou les opérateurs comme des vannes, pompes, etc. Chaque composant c_i est caractérisé par un couple $\{status, état\}$ tel que : état peut être ouvert, fermé, enclenché….etc. La variable interne status indique la disponibilité du composant. De fait, la modélisation du composant est réalisée en général par les modèles à états. Chaque état du composant est un nœud et les transitions entre états sont les actions sur le composant (ouverture de la vanne par exemple). Les transitions sont cependant conditionnées par des contraintes opérationnelles de sécurité (une pompe n'est enclenchée que si toutes les vannes sont ouvertes).

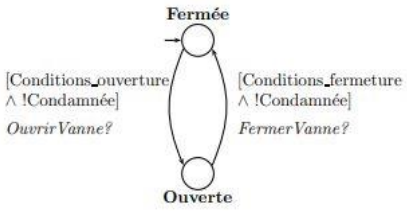


Figure 24 : Modèle à états d'une vanne (Cochard et al., 2015)

6.1.2 Modélisation des fonctions et leurs configurations

Dans les système de contrôle de batch (International Society of Automation & American National Standards Institute, 2010), les fonctions et équipements sont modélisés en spécifiant leurs états et transitions entre ces états. Pour une entité procédurale (fonction) les états sont illustrés dans la (Figure 25):

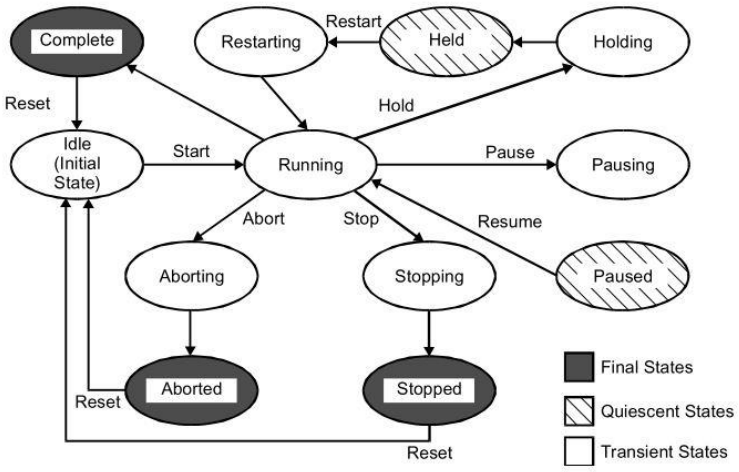


Figure 25 : modèle d'une fonction et son comportement (International Society of Automation & American National Standards Institute, 2010)

Le passage d'un état à un autre est assuré par des transitions. Si une condition est satisfaite alors la transition peut se faire.

6.1.3 Modélisation du système et son comportement

6.2 Modélisation de l'ordonnancement

Pour modéliser l'ordonnancement les travaux se dirigent pour la plupart d'entre eux en méthode ASTRID. En effet, l'ordonnancement est modélisé sous forme de séquence de fonctions à réaliser appelé *Séquenceur* ou *Recette*.

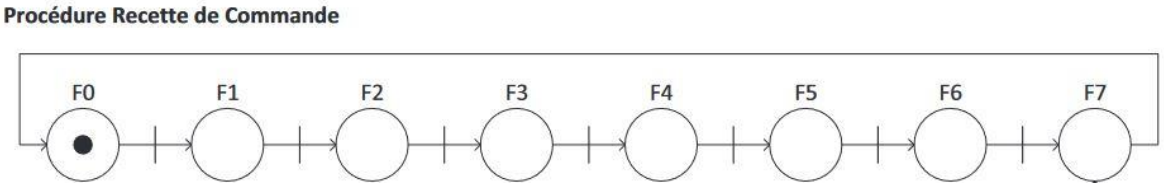


Figure 26 : Modèle de Recette d'une fonction par la méthode ASTRID (Cochard, 2017)

D'autres travaux modélisent l'ordonnancement sous forme formelle (Marangé et al., 2011) dans le but de le calculer. Un ordonnancement consiste à affecter des opérations d'une gamme à des machines et à définir la date de début de chaque opération. Ces travaux se basent sur des formalismes semi-formels et ne permettent pas de décrire l'aspect dynamique du comportement.

6.3 Génération de séquences

6.3.1 Méthode formelle

Les méthodes formelles utilisées pour la génération de séquences suivent le processus d'atteignabilité (Cochard et al., 2015). Le model checking peut être utilisé par exemple pour chercher la séquence de commande permettant la réalisation de l'ordonnancement sur un modèle du système.

6.3.2 EUD (End User Devlopement)

Les travaux de (Goubali, 2017) ont permis d'enregistrer les actions opérateurs sous forme de commandes SCADA par les techniques de EUD[7]. En effet, un prototype basé sur le P&ID du système permet aux opérateurs de préciser les chemins (ensemble de composants) nécessaires pour chaque composant. Ce prototype permet ensuite de généraliser ces actions et de générer les codes de commandes.

6.3.3 IDM (Ingénierie Dirigée par les Modèles)

Les travaux de (Bignon, 2012b) ont permis de générer conjointement l'interface de supervision et les codes de commandes d'un système SCADA. Le système est modélisé sous forme de synoptique en utilisant une librairie de composants. La réalisation de la fonction *Transfert d'un point A à un point B* du P&ID est possible par plusieurs chemins. En effet, l'obtention des codes de commande est générée par des opérations de transformations de modèles avec ATL (ATLAS Transformation Language) (Jouault et al., 2008).

7 Synthèse

Dans ce chapitre, nous avons présenté les travaux portant sur la gestion des modes afin de définir les modes ainsi que les états des composants, des fonctions et du système complet. Le but étant l'obtention d'un modèle opérationnel pour les système SCADA. A la lumière de cet état de l'art, nous pouvons faire les deux constats suivants.

7.1 Structuration du modèle

Au niveau des composant, la modélisation doit spécifier les différents états et les transitions entre ces états comme illustré dans les travaux de (Cochard, 2017). A contrario, au niveau des fonctions, la modélisation doit définir les informations suivantes :

- Les états de la fonction ainsi que les transitions entre ces états (Barbieri et al., 2015).
- La configuration de la fonction, autrement dit, comment et à quel moment la fonction active et/ou désactive les composant pour assurer son fonctionnement.

Au niveau du système, il est nécessaire de définir :

- Le mode de décomposition du système : structurelle, fonctionnelle ou structuro-fonctionnelle.
- Les différentes relations (Et, OU, etc.) qui peuvent exister entre les fonctions.
- Gérer la disponibilité des composants. En effet, un composant peut être utilisé par plusieurs fonctions, à condition que l'exécution simultanée de ces fonctions ne soit pas possible. Il est alors nécessaire de vérifier la disponibilité du composant avant de lancer une fonction.
- Gérer les commutations et transitions interdites entre tous les états du système, qui résultent de ses composants et ses fonctions (Dangoumau, 2000b).

7.2 Vérification et validation du modèle

Au niveau de la vérification& validation du modèle du système, plusieurs travaux existent. Toutefois, ces travaux se limitent à vérifier les propriétés basiques sur des modèles de modes

[7] EUD : End User Devlopement

élémentaires. Ils sont incapables de vérifier les propriétés spécifiques, car le comportement complet du système n'est pas modélisé à travers ses modes.

Pour une vérification et validation complète du modèle, il est nécessaire de la réaliser sur les trois niveaux : composants, fonctions et système. Pour y parvenir, il est nécessaire de définir les propriétés à vérifier à chaque niveau. Si certaines propriétés sur les niveaux basiques comme les composant existent, ceux des niveaux fonctions et systèmes sont à spécifier.

Chapitre 5 : Génération automatique des séquences de commande

1 Introduction

La génération de séquences de commande repose sur la définition d'un modèle opérationnel du système. La réalisation des modèles opérationnels d'un système SCADA est une étape très fastidieuse souvent délaissée par les experts au profit de l'implémentation réelle du système. Cependant, la non-réalisation de cette étape peut engendrer des erreurs de conception dont les corrections peuvent retarder la livraison des projets. Faciliter l'obtention de ces modèles pour permettre leur utilisation lors de la génération automatique des séquences de commande est l'objectif de ce chapitre.

Afin de proposer une approche complète pour l'obtention d'un modèle opérationnel des systèmes SCADA, nous divisons ce chapitre en trois grandes sections. La première section présente l'approche utilisée pour l'obtention du modèle. La deuxième vérifie et valide le modèle opérationnel résultant afin de garantir sa qualité. La troisième section génère des séquences d'actions pour mettre en œuvre l'ordonnancement.

2 Aléas et verrous

La définition d'un modèle opérationnel qui décrit le comportement complet d'un banc de test SCADA engendre souvent les problèmes suivants.

2.1 Obtention du modèle

La première tâche lors de la construction d'un modèle de comportement consiste à définir sa structure. Les travaux de (Cochard, 2018) proposent une solution complète pour la génération de séquences. Cette solution souffre de certaines limites. La première limite est que la décomposition fonctionnelle n'est pas prise en compte. Le modèle regroupe seulement un modèle de composants et un modèle de fonctions. D'autre part, le modèle de fonction est très simpliste et ne modélise pas les états de repos et d'arrêt d'urgence que peut avoir un système SCADA. Les travaux de (Dangoumau 2000) quant à eux, proposent un modèle pour les composants, les fonctions et également le système. La décomposition fonctionnelle est prise en compte à travers des liens de ET/OU. Toutefois, les modèles proposés sont très détaillés (comportent plusieurs états et transitions). Leur utilisation pour la génération de séquences peut être difficile. D'autre part, la vérification des commutations et des interactions entre les modèles est réalisée à travers des matrices de compatibilité, ce qui est fastidieux et surtout source d'erreur. De plus, la décomposition fonctionnelle utilisée n'est pas cadrée, on peut avoir plusieurs niveaux de décomposition. Cela peut compliquer davantage la modélisation.

Il est alors nécessaire de définir le type de décomposition du système à choisir : structurelle, fonctionnelle ou structuro-fonctionnelle. Il est important de proposer des lignes directrices pour faciliter cette structuration. Le défi ici est de trouver une solution permettant de faciliter la décomposition du système et de gérer d'une manière efficace et sans ambiguïtés toutes les interactions et les commutations entre tous les éléments du modèle complet du système. En effet, le système, pendant son cycle de vie bascule d'un état à un autre. Ces transitions sont parfois inattendues et peuvent mener le système à un blocage. Afin d'éviter les blocages du système engendré par le changement d'états, il faut définir d'une manière précise toutes les conditions de passage et d'incompatibilités du système.

D'autre part, il est nécessaire de définir les modèles des composants et des fonctions. Le choix du langage mathématique utilisé pour la modélisation est primordial car il doit aussi faciliter la génération de séquences plus tard. Le langage choisi doit également permettre de modéliser les

trois niveaux : composant, fonction et système. Le degré de granularité à adopter sur ces modèles est également important. En effet, la littérature montre que la majorité des approches de génération automatique des séquences se fait par l'utilisation des méthodes formelles comme le Model checking. Toutefois, ces méthodes souffrent d'explosion combinatoire. Le détail de la modélisation est donc primordial pour en assurer la pertinence tout en évitant l'explosion combinatoire.

2.2 Vérification et validation

La vérification et la validation du modèle de comportement des systèmes SCADA nécessitent de définir la solution la plus appropriée à utiliser : vérification formelle ou tests. Au niveau de la vérification formelle des modes, plusieurs travaux existent. Toutefois, ces travaux se limitent à vérifier les propriétés basiques des modes élémentaires (Hamani et al., 2009). Ils sont incapables de vérifier les propriétés spécifiques (globale du système), car le comportement complet du système n'est pas modélisé. L'approche de structuration du modèle du système peut aussi influencer la vérification. En effet, si les modèles sont structurés dans une matrice WDA, sa vérification et sa validation peuvent être réalisées en utilisant les informations de WDA. Les approches existantes proposent souvent des validations de la WDA à travers des scénarios de tests. Aucune vérification formelle de la WDA n'a été proposée dans la littérature.

Notre solution doit prendre en compte l'objectif final du modèle qui est la génération de séquences et également sa robustesse et les limites de sa mise en œuvre. Si les tests sont faciles à mettre en œuvre et permettent la génération de séquences, ils ne permettent pas de statuer formellement sur la robustesse et la qualité du modèle. D'autre part, les méthodes formelles ont démontré leur efficacité dans la génération de séquences et également dans la vérification du comportement mais souffrent du problème d'explosion combinatoire.

D'autre part, il est nécessaire de définir les propriétés à vérifier. Si certaines de ces propriétés ont été identifiées dans la littérature, d'autres nécessitent d'être définies. Par exemple les propriétés liées à la décomposition fonctionnelle et également celles qui sont spécifiques au système étudié.

3 Approche globale

Pour la génération automatique des séquences de commande permettant la réalisation opérationnelle de l'ordonnancement mathématique, calculé précédemment, nous avons proposé l'approche suivante (Figure 27). Notre approche comprend 3 grandes phases et 10 étapes.

La première phase a pour but la construction d'une bibliothèque de modes et comprend 4 étapes. Dans la première étape, les modèles à états des différents modes, sont représentés par des automates temporisés (AT). Lors de la deuxième étape, les propriétés basiques inter et intra-modes, qui seront vérifiées sur les automates temporisés, sont identifiées. Les propriétés basiques portent essentiellement sur la compatibilité entre les états des différents modes. Une fois les propriétés identifiées, elles sont réécrites en logique CTL. La troisième étape permet de lancer la vérification formelle de ces propriétés sur les AT. Si une propriété est vérifiée alors le modèle est valide, sinon un contre-exemple est retourné. Le contre-exemple comporte la trace d'exécution (la séquence des états) qui a mené à la violation de la propriété. Ce contre-exemple est alors étudié et analysé afin de corriger le modèle (quatrième étape). Une nouvelle itération est alors exécutée jusqu'à ce que toutes les propriétés soient vérifiées et que la bibliothèque des modes soit vérifiée et validée.

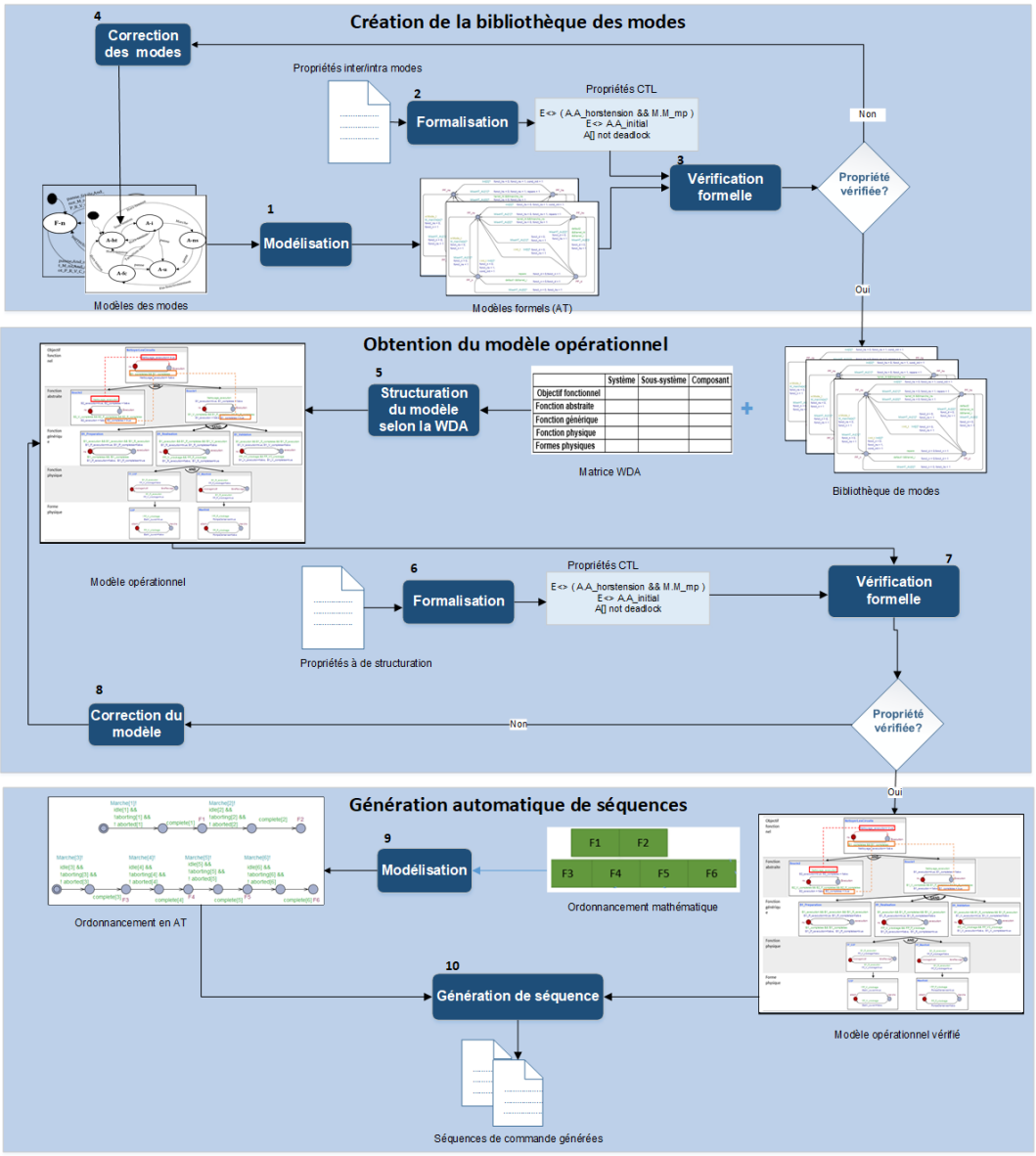


Figure 27. Approche générale proposée pour la génération automatique de séquences de commande

La deuxième phase de notre approche a pour but l'obtention du modèle opérationnel. Pour la décomposition du système, nous avons choisi d'utiliser la WDA, car cette décomposition est structuro-fonctionnelle et de plus, elle est très utilisée dans le cadre des systèmes industriels SCADA. A partir de la bibliothèque de modes obtenus et la matrice WDA, structurant les fonctions et les composants du système, un modèle opérationnel est obtenu. Les règles de structuration et de synchronisation entre les différents modèles de la matrice sont définies dans l'étape 5. La sixième étape consiste à identifier les propriétés spécifiques permettant de s'assurer de la cohérence entre les liaisons créées entre les différents modèles de la WDA. Les propriétés spécifiques s'assurent de la cohérence entre les liaisons créées entre les différents modèles de la WDA. Si les propriétés basiques sont décrites dans la littérature, l'identification des propriétés spécifiques nécessite une analyse supplémentaire afin de définir un ensemble de propriétés complet et cohérent. Les étapes 7 et 8 sont similaires aux étapes 3 et 4 et permettent de réaliser des vérifications formelles et des itérations de corrections jusqu'à l'obtention d'un modèle opérationnel correct.

La dernière phase de notre approche permet la génération automatique des séquences. Dans la neuvième étape, l'ordonnancement mathématique est modélisé en AT pour pouvoir l'exécuter sur

le modèle opérationnel obtenu. Enfin, la dernière étape, permet de dérouler l'ordonnancement sur le modèle opérationnel et de générer automatiquement les séquences de commande permettant la réalisation opérationnelle de cet ordonnancement.

4 Bibliothèque de modes

4.1 Identification des modes

Pour l'identification des modes des systèmes SCADA, nous avons pris en considération tous les états et modes proposés dans (Dangoumau, 2000c). Ensuite, nous avons ajouté les états présents dans le GEMMA et qui ne sont pas pris en compte dans (Dangoumau, 2000). Nous avons alors identifié 4 modes : le mode arrêt, le mode marche, le mode de production et le mode de fonctionnement. La Figure 28 présente tous les modes et les états des modes de la famille exploitation, ceux présentés par (Dangoumau, 2000), et ceux que nous avons ajoutés (en rouge) dans notre proposition.

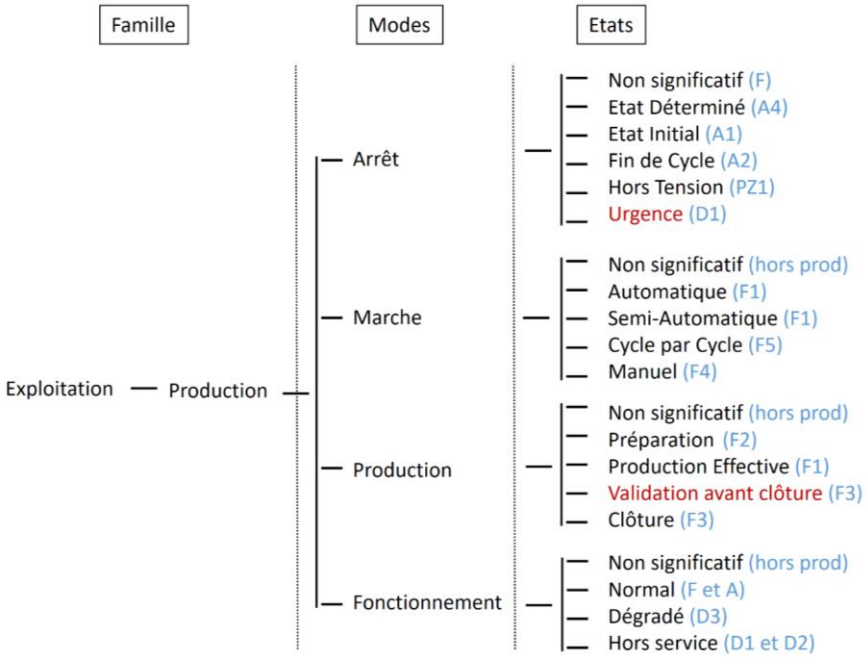


Figure 28 : Liste des modes retenus

Les modes de marche et d'arrêt présentent la manière dont s'exécute et s'arrête la production, tandis que les modes production et fonctionnement caractérisent le comportement attendu et l'avancement de la production.

- ***Le mode arrêt*** correspond à l'arrêt du système pour des raisons extérieures, ce sont : l'arrêt hors tension (PA-ht), initial (PA-i), déterminé (PA-d), final (PA-f) et l'arrêt d'urgence (PA-u). Nous avons ajouté l'état « arrêt d'urgence (PA-u) » à la liste des états du mode arrêt définie par (Dangoumau, 2000c) et nous faisons la différence entre l'arrêt hors tension et l'arrêt d'urgence. Contrairement à l'arrêt hors tension où les actionneurs et les capteurs sont mis hors énergie, l'arrêt d'urgence provoque une mise hors énergie des actionneurs seulement (« Arrêt d'urgence (automatique) », 2020).
- ***Le mode marche*** correspond à l'autonomie du système, les états qui existent dans le GEMMA sont automatique (PM-a), semi-automatique (PM-s), cycle par cycle (PM-c) et manuel (PM-m).

- ***Le mode Fonctionnement*** correspond au type de fonctionnement : comment se comporte le système. En industrie, nous trouvons un fonctionnement normal (PF-n), dégradé (PF-d) et hors service (PF-hs).
- ***Le mode Production*** correspond aux états où le système est en réalisation des objectifs pour lesquels il a été conçu. Ces objectifs de production sont liés à la sécurité et la qualité. Nous trouvons la préparation pour la production (PP-p), la production effective (PP-e), et la clôture de production (PP-c).

L'état « non significatif » est un état supplémentaire ajouté dans chaque mode. Cet état caractérise l'état d'un mode lorsque les autres états de ce mode ne le permettront pas (Dangoumau, 2000). Exemple : l'état A-ns (Arrêt non-significatif) est actif lorsque le système est dans l'état M-a (Marche automatique). Dans ce cas aucun des états du mode Arrêt ne peut être actif en même temps que le mode Marche, excepté l'état A-ns.

4.2 Modélisation des modes en AT

Dans cette section, nous présentons la modélisation formelle de tous les modes présentés précédemment à savoir : mode marche, arrêt, fonctionnement et production.

4.2.1 Modélisation du Pupitre de commande

Tous les systèmes s'opèrent à partir du pupitre ou du terminal (console) d'exploitation. Il concerne : la mise en route, les modes de marche et d'arrêts. Afin de synchroniser tous les modes aux différentes actions de l'utilisateur, nous avons déclaré toutes les actions de l'utilisateur sur le pupitre de commande comme un canal de diffusion. C'est le cas des boutons poussoirs du pupitre de commande et le défaut 2. Les commentaires donnent la signification de chaque variable dans la Figure 29 ci-dessous.

```
broadcast chan MiseST, Reprise, rearmer, arretDet, AU, Dcy, def2;
//MiseST : Mise Sous Tension
//Reprise : Ordre de reprise après un arrêt déterminé
//rearmer : Réarmement
//arretDet : Demander l'Arrêt dans un état déterminé
//AU : Arret d'urgence
//Dcy : Départ cycle
//def2 : Synchoniser la détection du défaut 2

broadcast chan Init[I], M_marche[N], Fcy[F]; //cannaux Init, Modes de marche et fin de cycle
//Init[0]  -->  Initialiser le système par l'opérateur
//Init[1]  -->  Retour à l'état initial à la fin du cycle du système (Ordre par le dernier évènement)

//M_Marche[0] -->  M_Manu : Marche Manuel (F4)
//M_Marche[1] -->  M_Auto : Marche Automatique (F1)
//M_Marche[2] -->  M_Cyclecycle : Marche cycle par cycle (F5)
//M_Marche[3] -->  M_SemiAuto : Marche Semi Automatique (F5)
//M_Marche[4] -->  M_Etape : Marche étape par étape (F6)

//Fcy : Demander l'arret en fin de cycle
```

Figure 29 : Extrait de la déclaration des variables sur UPPAAL

Notre pupitre de commande est présenté dans la Figure 30 ci-après.

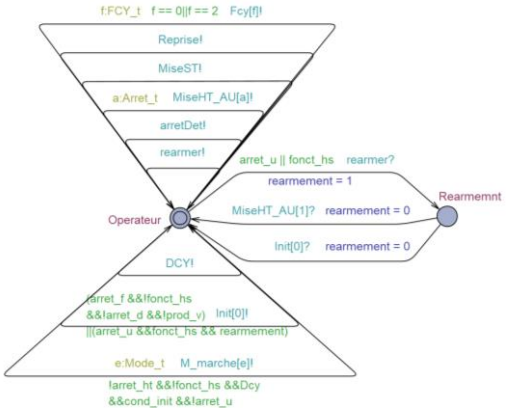


Figure 30 : Pupitre de commande

4.2.2 Modélisation du Mode Arrêt

Dans le GEMMA, à l'état initial, le mode est en arrêt hors tension, modélisé par (PA-ht). Lors de la mise en énergie de l'automatisme, il peut se produire qu'il ne soit pas dans son état initial (Figure 31). Auquel cas, la machine ne peut pas aller directement en "A1" lors de la mise en énergie. Le rectangle-état "A6" est prévu pour déclarer la façon dont se fera l'initialisation de la machine. Dans notre modèle nous considérons un passage directement vers "A1" en exigeant la présence des conditions initiales pour un objectif d'optimisation de modèle. Le système a donc une seule et unique action pour démarrer. L'opérateur doit mettre sous tension le système, il appuie dans le pupitre sur (MiseST) et le système se met en état d'arrêt initial. Les conditions initiales qui doivent être satisfaites pour pouvoir démarrer en mode marche peuvent être les vérins en positions initiales. Nous définissons ces conditions en (cond_init) dans le modèle. Le système a donc changé d'état de (PA-ht) à (PA-i), et par conséquent les conditions initiales sont devenues vraies. La variable d'état (arret_ht) passe à 0, la varibale (arret-i) passe 1. Ce passage correspond exactement à la transition « Mise en énergie → A6 → A1 » dans le GEMMA.

Maintenant que le système est en arrêt initial, il n'a qu'à attendre un démarrage. Lorsque l'opérateur choisie un mode de marche (auto, semi-auto, manuel), le système passe à l'état non-significatif (PA-ns). Les mise à jour dans cette transition sont (arret_i = 0, arret_ns = 1, cond_init = 0) et le passage correspond exactement à la transition « A1 → F1 ou F4 ou F5 » dans le GEMMA.

En arrêt non significatif, le système peut s'arrêter dans un état déterminé (PA-d) ou dans un état final (PA-f) selon le GEMMA. Si l'opérateur demande un arrêt au cours de la production, le mode arrêt passe à (PA-d). Les mises à jour seront (arret_ns = 0, arret_d = 1) et ce passage correspondent exactement à la transition « F1 → A3 → A4 » dans le GEMMA. La case A3 est considérée comme transitoire pour arriver à un arrêt déterminé. Ce n'est pas un arrêt système mais un arrêt temporaire qui prend fin si l'opérateur appuie sur le bouton (reprise). Les mises à jour seront l'inverse (arret_d = 0, arret_ns = 1) et ce passage correspond exactement à la transition « A4 → F1 » dans le GEMMA.

Dans le cas où l'opérateur souhaite arrêter le système à l'issu de la fin de production, il y a deux possibilités pour l'arrêt final. Soit l'opérateur appuie sur le bouton poussoir de fin de cycle (Fcy), ou bien la fin de la production est atteinte. Dans le premier cas, nous parlons d'un ordre demandé sur le pupitre et écouté par le système, et donc nous mettons sur un premier arc (Fcy_M ?). Dans le deuxième cas de fin de cycle, un ordre est envoyé du mode arrêt aux autres modes pour synchroniser la clôture de marche du système. Cet ordre est (Fcy [1] !). Les mises à jour sont (arret_ns = 0, arret_f = 1) pour les deux actions de fin de cycle et correspondent exactement à la transition « F1 → A2 » dans le GEMMA.

Pour certains systèmes, l'arrêt en fin de cycle mène tout le système à l'arrêt initial, comme il peut s'arrêter en arrêt final à la fin du dernier cycle de marche. Pour l'arrêt initial, nous avons mis un ordre (Init [1] !) qui sera écouté par les autres modes, et passerons à l'état « non significatif » qui est compatible avec (arret_i). Pour l'arrêt final, le système s'arrête en (arret_f) tout en restant dans le mode marche déjà actif. L'opérateur initialise le système par le (Init [0] !) et donc nous mettons un deuxième arc de (PA-f) vers (PA-i) avec une synchronisation (Init [0] ?). Les mises à jour seront l'inverse (arret_f = 0, arret_i = 1, cond_init = 1) et ce passage correspond exactement à la transition « A2 → A1 » dans le GEMMA.

Nous avons ajouté un état d'arrêt d'urgence (PA-u) qui est indépendant de l'arrêt hors tension. Tout appui sur le bouton poussoir d'arrêt d'urgence arrête tous les actionneurs par figeage. D'un point de vue modes, nous mettons une transition de chaque état sauf l'état hors tension vers l'état (PA-u) avec une écoute du canal « MiseHT_AU[0] ? ». Si le système est en arrêt hors tension il ne peut pas passer en urgence et c'est tout à fait normal. Ce passage correspond exactement à la transition « Depuis tous les états → D1 » dans le GEMMA.

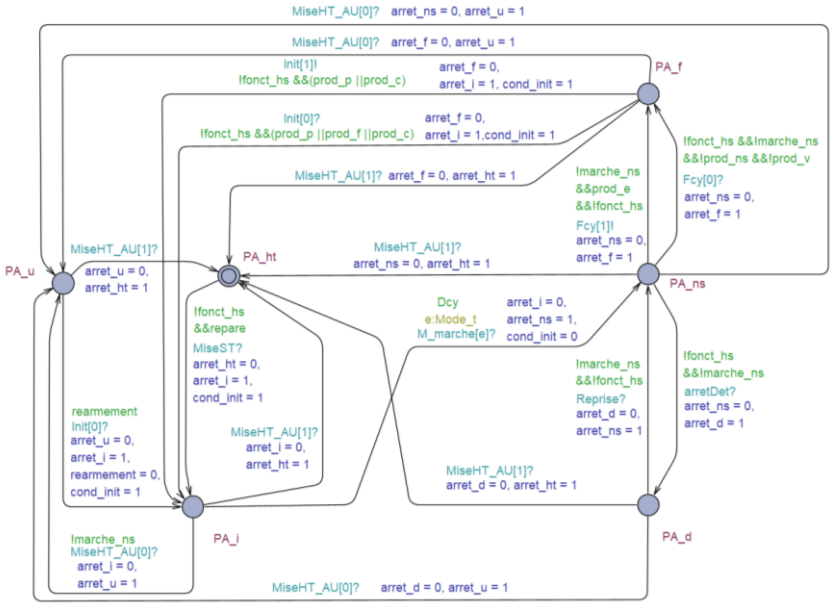


Figure 31 : Modèle du mode arrêt

Tout appui sur l'arrêt d'urgence ramène l'arrêt à « arrêt d'urgence (PA-u) ». Le système ne produit qu'après un réarmement, initialisation et un démarrage. (Dans le GEMMA : D1 ou D2 → A5 → A6 → A1 → F1 ou F4 ou F5).

Toute coupure d'électricité ramène l'arrêt à « arrêt Hors tension (PA-ht) ». Le système ne produit qu'après mise sous tension, initialisation et démarrage. (Dans le GEMMA : Mise sous tension → A5 → A6 → A1 → F1 ou F4 ou F5).

4.2.3 Modélisation du Mode Marche

Le mode marche est composé de quatre états de marche qui sont : marche automatique, semi-automatique, marche par cycle et marche manuelle (Figure 32). Initialement, aucun état de marche n'est sélectionné, le système n'est pas en mode de marche et donc en « non significatif ». L'opérateur doit choisir un état en appuyant sur le bouton poussoir ou orienter le sélecteur vers la position appropriée du pupitre de commande. Cette information sera écoutée par l'état correspondant. Dans le modèle nous utilisons l'instanciation par l'indice de l'état mis entre crochets (M_marche[e] ?) sur un arc allant de (PM-ns) vers (PM_état_de_marche). L'expression

« état_de_marche » remplace les états de marche (m, a, s, c) pour faciliter la compréhension des transitions et n'a aucune signification technique.

Nous mettons les images d'états à jour sur les arcs. Dès que l'action de synchronisation est satisfaite, ils s'actualisent. Pour aller de (PM-ns) à (PM_ état_de_marche), l'état non significatif se désactive et l'état de marche s'active, les variables (marche_ns = 0, marche_ état_de_marche = 1) sont mises à jours. Ce passage correspond exactement à la transition « A1 → (F1 ou F4 ou F5) » dans le GEMMA. Tout appui sur l'arrêt d'urgence (MiseHt_AU) ou une réinitialisation du système (Init[i]) ramène le modèle à l'état « non significatif ».

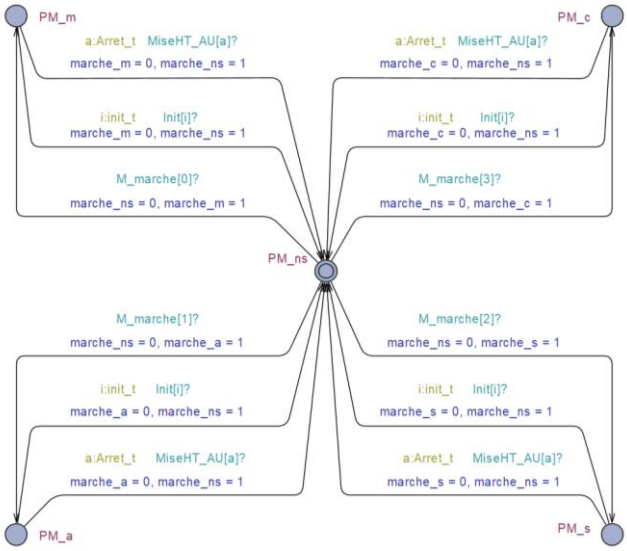


Figure 32 : Modèle du mode marche

4.2.4 Modélisation du Mode Production

Le mode de production (Figure 33) est en « non significatif » lorsque le système est en arrêt initial (PA-i). Dès que les conditions de démarrage du système (choix du mode, conditions initiales, départ cycle) sont vraies, la production commence soit par la préparation ou en production effective. Pour tous les états de mode, le passage direct à la production effective (PP-e) est possible. La production passe de l'état (PP-ns) vers (PP-e) si une demande de marche M_marche [e] ?) est réalisée par l'opérateur. Les mises à jour sont (prod_ns = 0, prod_e = 1) et ce passage correspond exactement au démarrage sans préparation « A1 → F1 ou F4 ou F5 » dans le GEMMA. Les mises à jour sont (prod_ns = 0, prod_p = 1) et ce passage correspond exactement à la transition « A1 → F2 → F1 » dans le GEMMA.

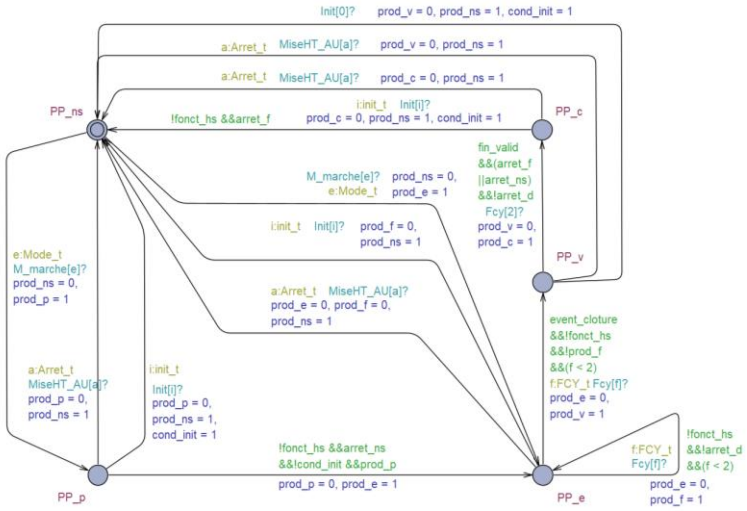


Figure 33 : Modèle du mode production

A l'état préparation (PP_p), l'opérateur a, par exemple, des mises en routes diverses à réaliser comme le préchauffage des outillages. Une fois ces préparations sont atteintes (température désirée par exemple), la production passe à effective (PP_e). Les mises à jour sont (prod_p = 0, prod_e = 1) et ce passage correspond exactement au démarrage sans préparation « F2 → F1 » dans le GEMMA.

Si une demande d'arrêt est émise lorsque la production est en effective, deux cas se présentent : un arrêt de fin de cycle et initialisation ; ou bien fin de cycle, clôture et initialisation. Dans le premier cas, nous avons ajouté un arc autour de la production effective où une nouvelle variable de fin de production (prod_f) est mise à 1. La raison est que les deux rectangles F1 et A2 à l'intérieur des pointillés "Production" du GEMMA, correspondent à des états pour lesquels la machine produit. Nous restons toujours dans l'état (PP-e) de production mais avec ces nouvelles mises à jour pour éviter d'ajouter un état de fin de production et de confondre entre l'état de production effective et l'état fini. Dès que l'arrêt final est obtenu, l'initialisation du système ramène le mode production en non significatif (PP-ns) car il ne produit plus. Les mises à jour sont (prod_f = 0, prod_ns = 1) et ce passage correspond exactement à la transition « F1 → A2 → A1 » dans le GEMMA.

Dans le deuxième cas, le système nécessite une étape de clôture à la fin de cycle, nous avons ajouté une étape intermédiaire de validation que nous avons appelé « *(PP-v) : Validation avant clôture* ». Pour notre projet, après la fin du cycle et à la fin de la production, l'utilisateur doit valider l'arrêt avant de clôturer tout le système. Ici un état de validation est nécessaire avant la clôture dans le modèle. Il est placé entre « PP-e » et « PP-c » et n'existe ni dans (Dangoumau, 2000), ni dans le GEMMA mais leurs conditions sont toujours respectées. Les mises à jour sont (prod_e = 0, prod_v = 1). Si la condition de validation (fin_valid) est vraie, le système passe à l'état de clôture et les mises à jour seront (prod_v = 0, prod_c = 1). Ce passage correspond exactement à la transition « F1 → F3 » dans le GEMMA.

Ensuite dès que les évènements de clôture s'exécutent, le système s'initialise soit par la dernière action de clôture ou bien par l'appui de l'opérateur. Nous relions (PP-c) au (PP-ns) avec la synchronisation (Init [i] ?) et les mises à jour sont (prod_c = 0, prod_ns = 1). Ce passage correspond exactement à la transition « F3 → A1 » dans le GEMMA.

Tout appui sur l'arrêt d'urgence ou une coupure de courant ramène la production à « non significatif ». Le système ne produit qu'après un réarmement et une reprise.

4.2.5 Modélisation du Mode Fonctionnement

A partir de l'état PF-ns, le modèle passe à l'état de fonctionnement normal (PF-n) si une demande de marche est réalisée par l'opérateur (Figure 34). A ce niveau, si un défaut mineur (modélisé par défaut 1) (joint d'étanchéité mal serré, des vibrations…) surgit, alors le modèle passe du fonctionnement normal au fonctionnement dégradé (PF-d). C'est un défaut mineur et ne change quasiment rien dans les autres modes, marche, arrêt et production en respectant le GEMMA. Nous restons toujours en production mais en régime non optimal. Le défaut 1 n'est pas une action demandée par l'opérateur et nous n'avons pas à le synchroniser avec les autres modes. Nous le modélisons sous la forme d'une condition (defaut1). Ce passage correspond exactement à la transition « F1 → D3 » dans le GEMMA. Si l'opérateur peut effectuer une réparation du défaut sans arrêter le système et en toute sécurité (serrage d'une vice…), le retour à la normale est automatiquement maintenu. Ce passage correspond exactement à la transition « D3 → D2 » dans le GEMMA.

Contrairement au premier défaut, un défaut grave va basculer le système en fonctionnement hors service, c'est-à-dire il impacte tous les modes. Le système passe de l'état (PF-d) à (PF-hs). Ce passage correspond exactement à la transition « F1 → D3 » dans le GEMMA.

Dans l'état hors service (PF_hs), l'opérateur peut soit lancer un arrêt d'urgence ou juste une réinitialisation du système. Dans le premier cas, à la réception du message (MiseHT_AU [1] ?), le modèle revient à l'état non significatif. Ce passage correspond exactement à la transition « D1 → A5 » dans le GEMMA. Si l'opérateur choisit d'initialiser le système, le message (Init[0] ?) est reçu, alors le fonctionnement n'est pas actif, et passe en (PF-ns). Ce passage correspond exactement à la transition « A5 → A6 → A1 » dans le GEMMA.

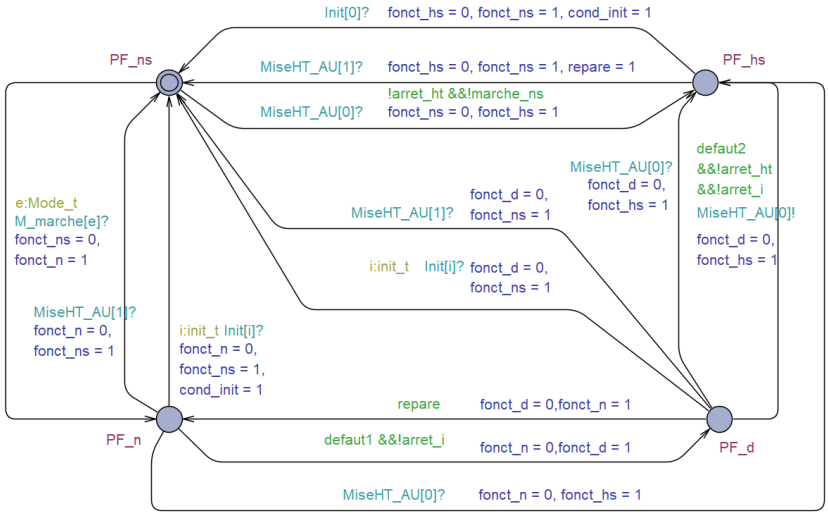


Figure 34 : Modèle du mode fonctionnement

4.3 Vérification formelle

Nous avons vérifié 313 propriétés intra et inter-modes dans le modèle (Tableau 5). Les propriétés intra-mode ont été définies dans (Dangoumau, 2000). Pour la propriétés inter-modes, nous avons utilisé la matrice de compatibilité (Tableau 7). La matrice de compatibilité, inspirée de (Dangoumau, 2000), comporte autant de lignes que de colonnes. Chaque ligne (ou colonne) représente un état des différents modes. Les 1 dans la matrices correspondent aux états compatibilités et les 0 aux états non-compatibles. Les lignes et colonnes en orange représentent les états que nous avons ajouté dans le cadre de cette thèse.

Tableau 5. Liste des propriétés intra et iner-modes

Niveau	Type	Propriété	Description	CTL	Nb de propriétés
		Absence de blocage	Le mode ne comporte pas d'état puit ou d'état de blocage	A[] not deadlock	1
		Atteignabilité	Tous les états du mode sont atteignables	E<> Mode.Etat	20
		Atteinte mutuelle	Chaque état de chaque mode est atteignable de n'importe quel autre état de ce même mode	E<> Mode.Etat1 imply Mode.Etat2	82
		Etats Compatibles (toutes les case à 1 dans Tableau 3)	Les états compatibles entre deux modes doivent être actifs en même temps	E<> (Mode1.Etat && Mode2.Etat)	108
		Etats non compatibles (toutes les case à 0 dans Tableau 3)	Les états incompatibles entre deux modes ne doivent pas être actifs en même temps	A[] not (Mode1.Etat && Mode2.Etat)	102
				Total	313

La vérification formelle de toutes les propriétés CTL définies a permis de vérifier nos modèles et également de constater des contradictions dans la matrice de compatibilité proposée par (Dangoumau, 2000). En effet, les cellules en verts dans le Tableau 3 correspondent aux cellules corrigées. En effet, le système n'est pas en en fonctionnement (PF-ns) dans seulement deux états d'arrêt : PA-ht et PA-i. C'est le même cas entre le mode Marche, Production et le mode Arrêt où PM-ns et PP-ns ne sont compatibles qu'avec PA-ht et PA-i. Cela implique que PF-ns et PP-ns ne doivent avoir une compatibilité qu'avec PM-ns. Une fois un mode soit sélectionné par l'opérateur, les modes fonctionnement et production doivent sortir de l'état « non significatif ».

Tableau 6. Matrice de compatibilité inter-modes

		Arrêt						**Marche**					**Fonctionnement**				**Pruduction**				
		PA-ns (F)	**PA-ht** (PZ1)	**PA-i** (A1)	**PA-f** (A2)	**PA-d** (A4)	**PA_u** (D1)	**PM-ns** (Hors prod)	**PM-a** (F1)	**PM-s** (F1)	**PM-c** (F5)	**PM-m** (F4)	**PF-ns** (Hors prod)	**PF-n** (F et A)	**PF-d** (D3)	**PF-hs** (D2)	**PP-ns** (Hors prod)	**PP-p** (F2)	**PP-e** (F1)	**PP-v** (F3)	**PP-c** (F3)
	PA-ns	1	0	0	0	0	0	0	1	1	1	1	0	1	1	1	0	1	1	1	1
	PA-ht (PZ1)		1	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0
	PA-i (A1)			1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0
	PA-f (A2)				1	0	0	0	1	1	1	1	0	1	1	1	0	1	1	1	1
	PA-d (A4)					1	0	0	1	1	1	1	0	1	1	1	0	1	1	1	1
	PA_u (D1)						1	0	1	1	1	1	0	0	0	1	1	0	0	0	0
	PM-ns							1	0	0	0	0	1	0	0	0	1	0	0	0	0
	PM-a (F1)								1	0	0	0	0	1	1	1	0	1	1	1	1
	PM-s (F1)									1	0	0	0	1	1	1	0	1	1	1	1
	PM-c (F5)										1	0	0	1	1	1	0	1	1	1	1
	PM-m (F4)											1	0	1	1	1	0	1	1	1	1
	PF-ns												1	0	0	0	1	0	0	0	0
	PF-n (F et A)													1	0	0	0	1	1	1	1
	PF-d (D3)														1	0	0	1	1	1	1
	PF-hs (D2)															1	0	1	1	1	1
	PP-ns																1	0	0	0	0
	PP-p (F2)																	1	0	0	0
	PP-e (F1)																		1	0	0
	PP-v (F3)																			1	0

5 Obtention du modèle opérationnel

Nous présentons dans cette section les différentes étapes de modélisation qui nous ont permis d'obtenir un modèle opérationnel complet pour un système SCADA.

5.1 Utilisation de la WDA pour la décomposition du système

La Figure 35 présente un exemple de décomposition structuro-fonctionnelle d'un système en utilisant la WDA. Sur cette décomposition, nous avons ajouté les opérateurs de décomposition AND, OR et SAND qui peuvent exister entre les fonctions. La réalisation de la fonction globale nécessite la réalisation parallèle de deux sous fonctions. La réalisation de la sous-fonction2 nécessite également la réalisation de deux autre sous-fonctions, qui sont à réaliser séquentiellement. Ces sous fonctions utilisent les composants vanne1, vanne2, pompe1 et pompe2 pour la réalisation des fonctions.

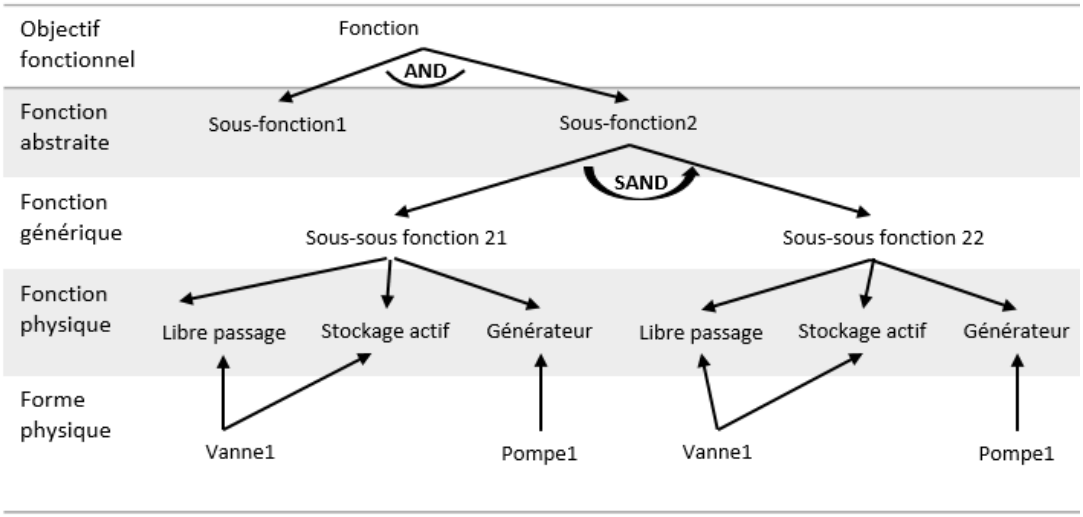


Figure 35. Décomposition structuro-fonctionnelle d'un système en utilisant la WDA

Le but étant de proposer maintenant des modèles pour les différents niveaux de la WDA et les interactions entre ces niveaux. Pour ce faire, nous avons proposé un modèle pour les trois opérateurs AND, OR, SAND utilisés pour la décomposition fonctionnelle. Ces modèles sont utilisés dans les deux niveaux haut de la WDA. Nous avons également proposé un modèle pour les fonctions génériques. Ce modèle intègre également la gestion de configuration de la fonction qui lui permet d'activer les composants nécessaires pour sa réalisation. Nous avons également un modèle pour le niveau 4 de la WDA (Fonction physique) qui modélise le comportement et les états d'un composant. Le dernier niveau de la WDA représente les composants eux même, aucun modèle n'a été proposé à ce niveau.

5.2 Modélisation des composants

Dans le but d'éviter le problème d'explosion combinatoire lors de la génération de séquences, nous avons choisi de proposer un modèle très réduit du composant. Ce modèle comporte seulement les actions importantes pour l'activation et la désactivation du composant.

Les composants ont été modélisés par un automate temporisé générique comportant deux états stables (fermé/ouvert) et deux états intermédiaires (Figure 36). Initialement le composant est à l'état fermé. Si un ordre d'ouverture est reçu, alors l'automate passe à l'état ouvert et inversement. La variable *Position_eq* est mise à 1, indiquant que le composant est à l'état ouvert. Elle est remise à 0 pour l'état fermé.

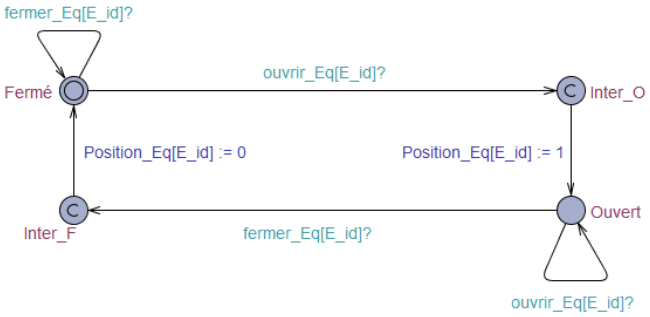


Figure 36 : Automate temporisé pour modéliser les composants

5.3 Modélisation générique des fonctions

Pour les fonctions, nous proposons deux modèles : un modèle générique, qui reproduit les états de la fonction et les transitions entre ces états, et un modèle de pupitre permettant de lancer les commandes sur le modèle générique.

5.3.1 Modèle réduit et générique de fonction

L'automate temporisé, illustré en Figure 37, modélise les différents états d'une fonction. Ce modèle a été inspiré du PackML et il est générique pour toutes les fonctions. Dans ce modèle chaque fonction est caractérisée par un identifiant (id).

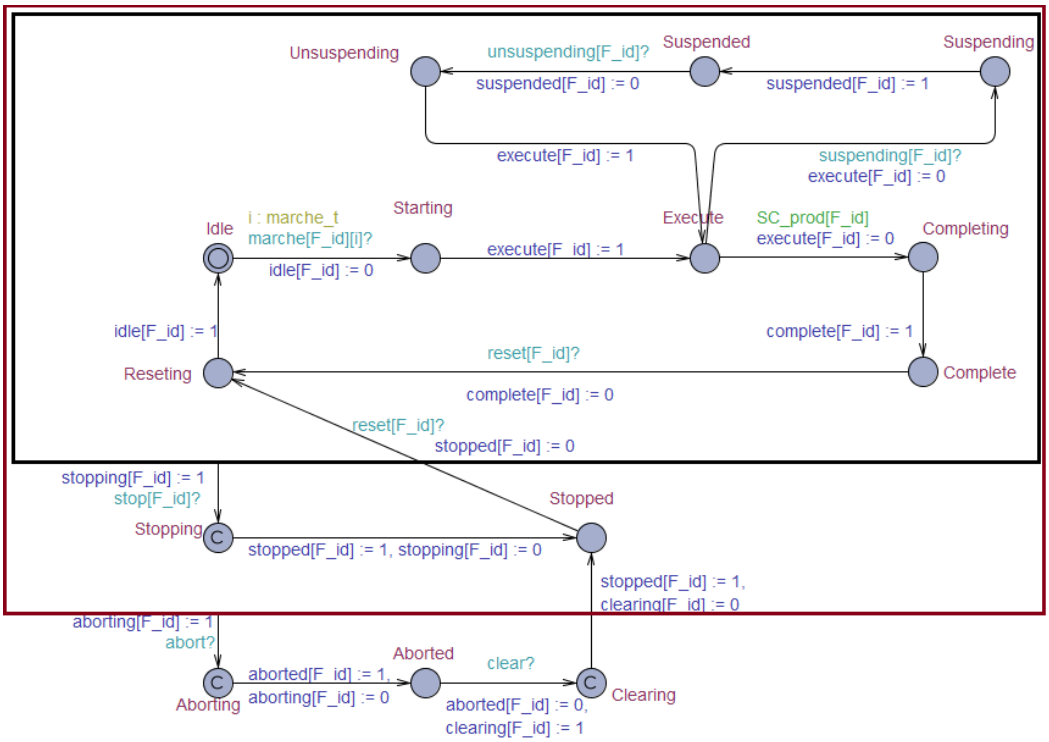


Figure 37 : Automate temporisé proposé pour la modélisation des modes marche et arrêt d'une fonction

Une fonction est caractérisée par 11 états simples et 3 états urgents (marqués par le symbole C). A l'état initial, la fonction est dans l'état *Idle*, la fonction est prête à être lancée. A la réception d'une demande de marche (*M_marche_F[m] ?*), spécifiant un des modes de marche (automatique, semi-automatique et manuel), la fonction atteint l'état transitoire *Starting* et ensuite l'état d'exécution. Les modes de marche sont spécifiés par l'entier *m* passé en paramètre dans le message *M_marche_F[m]*. Quand la fonction est en exécution, elle peut être interrompue (*suspended*) à tout moment. D'autre part, si les conditions physiques nécessaires à la fin de la fonction sont atteintes, alors la fonction passe à l'état complétée (*Complete*), puis réinitialisée (Idle). À tout moment et

dans tous les états sauf l'état *Aborted* (tous les états entourés par le rectangle noir), la fonction peut être arrêtée (*Stopped*) par l'émission du message *Stop*. Après un ordre d'initialisation (*Reset*?), elle passe par l'état transition (*Reseting*) afin d'atteindre l'état *Idle* où toutes les informations liées à cette fonction sont initialisées. Elle peut être mise en arrêt d'urgence (*Aborted*) à tout moment également (tous les états entourés par le rectangle marron). A partir de cet état, une action de réarmement (M_*Clear*?) est requise pour passer à l'état stoppé.

Pour chaque état, une variable booléenne est ajoutée. Par exemple la variable *idle_F[id]* est associée à l'état *Idle*. Elle est mise à 1 quand l'automate rentre dans l'état *Idle* et elle est à 0, dès qu'il en sort. Ces variables sont utilisées pour gérer l'interaction entre les différents modèles du système.

5.3.2 Modèle du pupitre de commande

Les messages permettant le passage d'un état à un autre sont envoyés à partir d'un pupitre de commande ou d'une interface graphique. Nous avons proposé le modèle, présenté en Figure 38, pour l'envoi de ces messages. Il comporte un seul état et autant d'arcs qu'il y a des messages à envoyer. Toutefois, l'envoi de chaque message est régi par un ensemble de contraintes. Par exemple, l'envoi du message (*Reset!)*, permettant la réinitialisation d'une fonction, n'est possible que si la fonction est dans un état d'arrêt (*stopped_F*) ou dans un état complète (*complete_F*). Ce modèle est générique et instancié pour chaque fonction du système.

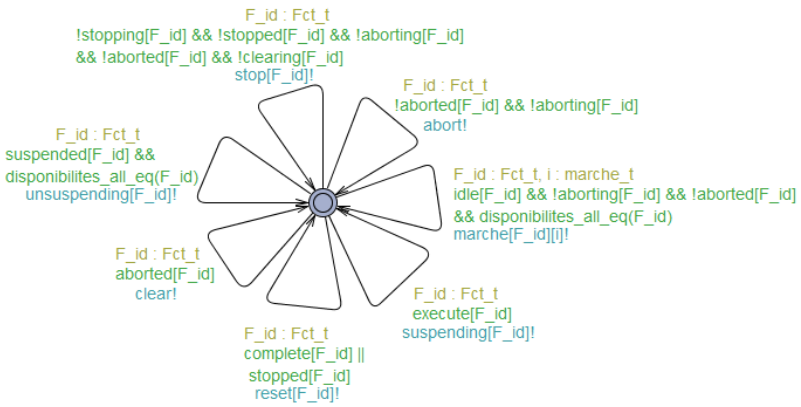


Figure 38 : Modèle générique correspondant au pupitre de commande

5.4 Modélisation des fonctions génériques de la WDA

Les fonctions génériques de la WDA (troisième niveau dans la matrice) ne sont pas décomposées en sous-fonctions et interagissent directement avec les composants du système à travers les configurations. La configuration d'une fonction décrit l'ensemble des composants nécessaires à la réalisation de la fonction et leurs états. Par exemple : vanne ouverte, pompe fermée....

Pour la gestion des configurations des fonctions, nous avons utilisé 4 matrices et 6 fonctions développées en C dans UPPAAL. Les matrices comportent autant de lignes qu'il y a de fonctions et autant de colonnes qu'il y a de composants dans le système. La première matrice, nommée *influence_fonction_equipements*, définit les composants nécessaires à la fonction. Ainsi, les cases de cette matrice comportant des 1 correspondent aux composants nécessaires à la fonction. La deuxième matrice, nommée *configurations_fonctions*, comporte également des 0 et des 1 mais ces derniers modélisent l'état ouvert (1) ou fermé (0) de chaque composant. La troisième matrice de*configurations_fonctions* spécifie les états des composants une fois que la fonction est terminée. Enfin, la matrice *comp* permet d'enregistrer quel composant est utilisé actuellement par quelle

fonction. Si la case *[F_id,E_id]* est à 1 dans cette matrice, cela signifie que la fonction *F_id* utilise le composant *E_id*.

D'autre part, la fonction, nommée *influence_f(F_id,E_id)*, retourne vrai si le composant *E_id* entre dans la configuration de la fonction *F_id*. La fonction *disponibilites_eq(E_id)* vérifie la disponibilité du composant en question avant de le manipuler. La fonction *disponibilites_all_eq(F_id)* retourne vrai si tous les composants nécessaires pour la fonction sont disponibles. Les deux dernières fonctions *configuration(F_id)* et *deconfiguration(F_id)* retournent vrai si tous les composants de la fonction sont dans l'état souhaité pour la configuration, respectivement pour la déconfiguration.

```
const int NB_EQ = 3;
const int NB_Fct = 2;
typedef int[0, NB_EQ-1] EQ_t;
typedef int[0, NB_Fct-1] Fct_t;

// Déclaration des cônes d'influence pour les fonctions
bool influence_fonction_equipements[NB_Fct][NB_EQ] = {{1,1,1},{1,0,1}};

//Déclaration des configurations des fonctions
bool configurations_fonctions[NB_Fct][NB_EQ] = {{1,0,1},{0,1,0}};

//Déclaration des déconfigurations des fonctions
bool deconfigurations_fonctions[NB_Fct][NB_EQ] = {{0,0,0},{0,0,0}};

// Matrice qui nous permet de vérifier par quelle fonction un composant est utilisé
bool comp[NB_Fct][NB_EQ] = {{0,0,0},{0,0,0}};

//Fonction influence_f
bool influence_f(int f_id, int e_id) {
return cone_influence_fonction_equipements[f_id][e_id];
}

//Fonction vérifie la disponibilité de l'équipement
bool disponibilites_eq(int e_id){
int i = 0;
for(i = 0; i < NB_Fct; i++)
   if(comp[i][e_id])
      return false;
return true;
}
```

```
//Fonction qui vérifie la disponibilité de tous les équipements d'une fonction avant de l'exécuter
bool disponibilites_all_eq(int f_id){
int i = 0;
int j = 0;
for(i = 0; i < NB_EQ; i++)
   if(cone_influence_fonction_equipements[f_id][i])
      for(j = 0; j < NB_Fct; j++)
         if( comp[j][i]!=0 && j!=f_id)
            return false;
return true; }

//Fonction configuration des équipements pour la fonction
bool configuration(int id) {
int i = 0;
for (i = 0; i < NB_EQ; i++)
   if(cone_influence_fonction_equipements[id][i])
      if(configurations_fonctions[id][i] != (Position_Eq[i] && comp[id][i] ))
         return false;
   return true;
}

//Fonction deconfiguration des équipements pour la fonction
bool deconfiguration(int id) {
int i = 0;
for (i = 0; i < NB_EQ; i++)
   if(cone_influence_fonction_equipements[id][i])
      if(deconfigurations_fonctions[id][i] != Position_Eq[i])
         return false;
   return true;
}
```

Ensuite, nous avons adapté le modèle générique de fonction proposé en Figure 37 afin de gérer essentiellement la configuration des fonctions et leurs interaction avec les composants. La gestion de la disponibilité des composants est également gérée à ce niveau. L'interaction du modèle de fonction avec les composant porte sur deux axes : la configuration (la sollicitation des composants) et la configuration (la libération des composants). Nous avons ensuite défini les états de la fonction qui peuvent déclencher une configuration, respectivement une configuration. Les états *Starting* et *Unsuspending* déclenchent essentiellement de la configuration, car c'est à ce niveau que les composants sont alloués pour la réalisation de la fonction. D'autre part, les états : *Completing, Stopping, Aborting et Suspending* déclenchent une déconfiguration et une libération des composants. Le modèle des fonctions interagissant avec les composants est illustré par la Figure 39.

Le modèle actionne des composants selon des configurations prédéfinies dans les deux matrices. A l'état *Starting* par exemple, les ordres d'ouverture et de fermeture de tous les composants correspondant à la configuration de la fonction et contenus dans les deux matrices sont lancés. Le modèle envoie un ordre d'ouverture, par exemple, si le composant est inclus dans la configuration de la fonction (*influence_f*), qu'il est disponible et que la matrice de configuration indique qu'il est

à mettre à 1. La fonction passe à l'état *Exécution* si la configuration est réalisée (tous les composants sont selon l'état souhaité pour la réalisation de la fonction). Il est à noter ici que lorsqu'un ordre d'ouverture est lancé pour un composant, la case de matrice *Comp(F_id,E_id)* est mise à 1 indiquant que le composant est actuellement utilisé par la fonction *F_id* et qu'il est indisponible pour les autres fonctions. Le même principe est utilisé pour la déconfiguration.

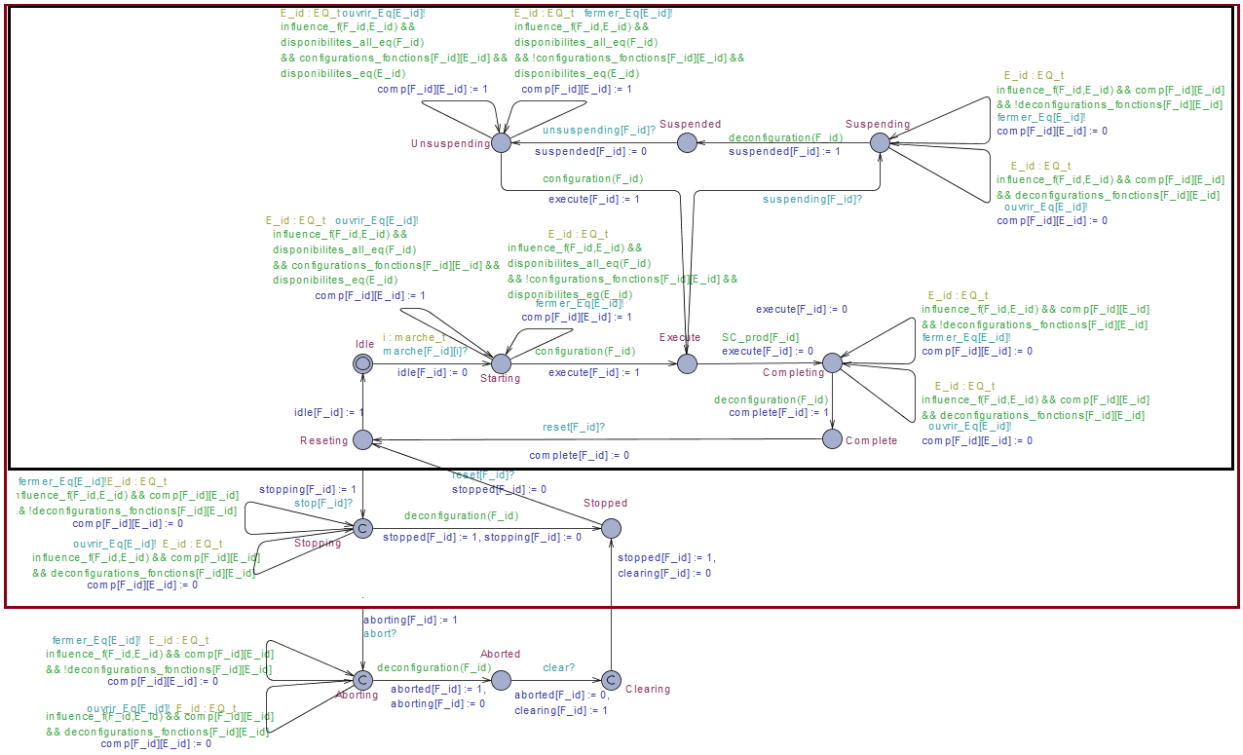


Figure 39 : Automate temporisé pour la modélisation des fonctions filles et l'interaction avec leur composants

5.5 Modélisation des fonctions de haut niveau

Des opérateurs sont utilisés pour décomposer une fonction de haut niveau en plusieurs sous-fonctions. L'opérateur OR est utilisé pour représenter des sous-fonctions alternatives. L'opérateur AND représente la décomposition conjonctive. Finalement l'opérateur SAND, qui est un cas spécifique du AND, représente une décomposition séquentielle des sous-fonctions (celles-ci sont réalisées d'une façon séquentielle de gauche vers la droite).

5.5.1 Modélisation de la décomposition AND

Le modèle générique en Figure 37 est alors enrichi pour modéliser chaque opérateur. La Figure 40 illustre l'automate temporisé modélisant une fonction, comportant deux sous-fonctions, reliées par un opérateur AND. On peut constater que certaines contraintes ont été ajoutées sur le modèle. Si une demande de marche est lancée pour cette fonction à partir de l'état *Idle*, elle passe à l'état *Starting* et met en marche également ses sous-fonctions, en envoyant le message *Marche_F[F_id][i] !,* spécifiant la sous-fonction par son *F_id* et le mode de marche souhaité avec l'entier *i*. La fonction atteint l'état exécution si une de ses sous-fonctions est à l'état exécution (*execute[0] || execute [1]*). Toutefois, elle n'atteint l'état *Completing* seulement si ses deux sous-fonctions sont en *Completing (completing[0] && completing [1]).* La même condition est également utilisée pour les états : *Complete, Idle, Stopped, Aborted, Suspendend*.

Le même principe est utilisé pour modéliser les autres opérateurs OR et SAND. Les changements entre les modèles de ces trois opérateurs portent sur les contraintes et également sur l'ordre

d'envoi de messages aux sous-fonctions. En effet, dans l'opérateur SAND par exemple, les sous-fonctions étant exécutées séquentiellement, l'envoi des messages de marche et d'arrêt pour ces sous-fonctions doit également être séquentiel.

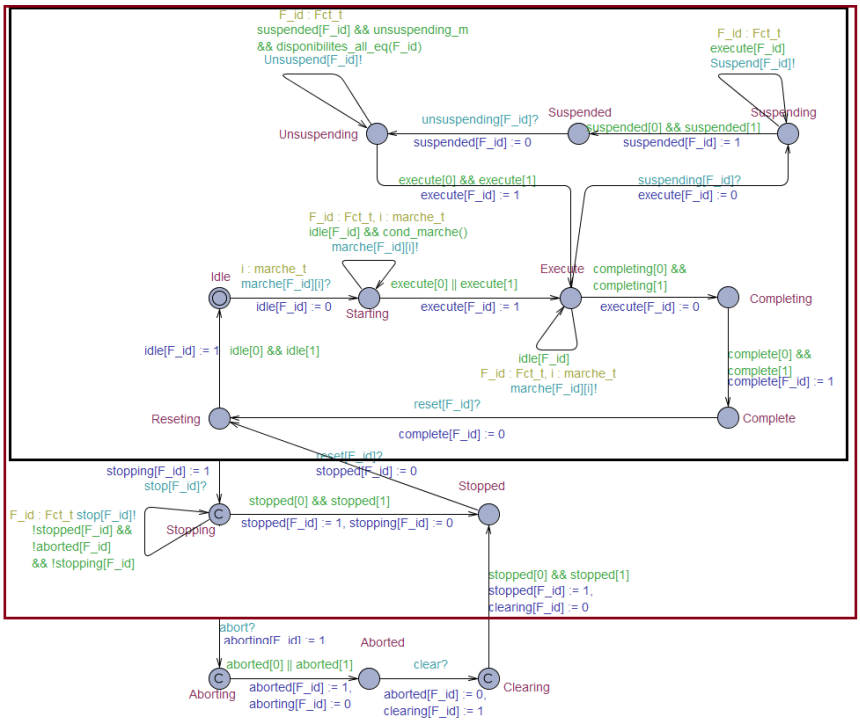


Figure 40 : Automate temporisé proposé pour la modélisation d'une fonction mère décomposée par l'opérateur AND

5.6 Obtention du modèle complet du système

Afin de respecter la décomposition de la WDA, les différents AT communiquent à travers les interactions définies dans les modèles (Figure 41). En effet, pour chaque case de la WDA, un automate temporisé modélisant les fonctions ou les composants est instancié et les variables booléennes sont mises à jour. Par exemple, les fonctions de haut niveau activent leur sous-fonctions selon l'opérateur utilisé (AND, OR, SAND). Lorsque la sous-fonction est activée, elle active également ses sous-fonctions. Enfin, les sous-sous-fonction enclenchent ou déclenchent les composants selon la configuration prédéfinie. D'autre part, lorsque la sous-sous-fonction est terminée (elle atteint l'état *complete*), cette information est remontée à la sous-fonction qui l'a déclenchée. Si la sous-fonction est de type AND, elle se termine une fois que toutes ses sous-fonctions sont terminée. Le même principe est utilisé pour les fonctions du haut niveau.

Le Tableau 7, présente les différentes caractéristiques de nos modèles. On peut constater que le modèle des fonctions de haut niveau de type AND par exemple, comporte 14 états, 21 conditions ou gardes, 8 canaux de communication et 49 variables.

Tableau 7. Taille des modèles

	États	**Gardes**	**Canal de synchronisation**	**Variables**	**Nombre de propriétés vérifiées**
Composant	4	0	2	2	-
Fonction bas niveau	14	10	8	50	210
Fonction haut niveau OR	14	13	8	43	392
Fonction haut niveau AND	14	21	8	49	392

Fonction haut niveau SAND	14	24	8	53	392
Pupitres de commande	1	8	8	0	-
Total	**61**	**76**	**42**	**197**	**1386**

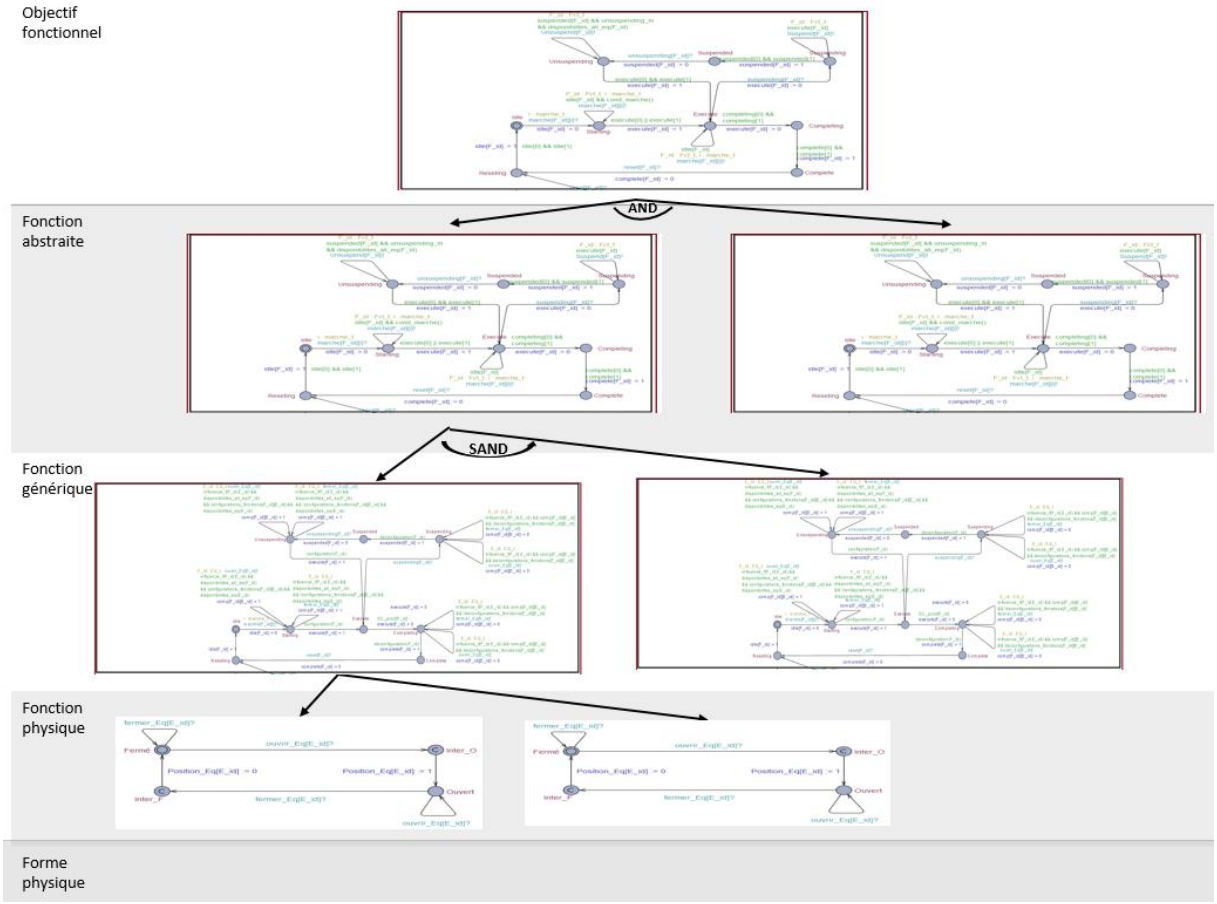


Figure 41. Modélisation de la décomposition des modes opératoires sur la WDA

5.7 Vérification formelle du modèle

5.7.1 Vérification des interactions entre les fonctions de haut niveau

Afin de s'assurer de la qualité de nos modèles de haut niveau, qui modélisent les opérateurs AND, OR et SAND, nous avons construit deux tables de cohérence. Dans la première table, la cohérence entre les états de la fonction de haut niveau et ses sous-fonctions est évaluée. La deuxième table évalue la cohérence entre les sous-fonctions appartenant à la même fonction de haut niveau. Le Tableau 9 présente les deux tables de cohérences pour une fonction de haut niveau de type AND. Les lignes et les colonnes correspondent aux différents états d'une fonction. 3 couleurs sont utilisées.

Tableau 8. Propriété CTL

Case dans la table	Propriété en CTL
La couleur rouge indique que l'état de la fonction de haut niveau est incompatible avec l'état de toutes ses sous-fonctions.	A[] not ((Sous-fonction(0).state && Sous-fonction(1).state) && Fonction.state)
La couleur verte foncé indique que l'état de la fonction de haut niveau est compatible avec l'état de toutes ses sous-fonctions.	E<> ((Sous-fonction(0).state && Sous-fonction(1).state) && Fonction.state)

La couleur vert clair indique que l'état de la fonction de haut niveau est compatible avec une de ses sous-fonctions.	E<> ((Sous-fonction(0).state || Sous-fonction(1).state) && Fonction.state)

Ensuite pour chaque couleur, nous avons écrit une propriété en CTL afin de la vérifier formellement avec UPPAAL (Tableau 9). Un ensemble de 392 propriétés, correspondant à chaque case des deux tables de cohérence, a été vérifié pour un chaque fonction de haut niveau.

Tableau 9. Table de cohérence entre fonction AND et ses sous-fonctions

Mere \ Fille	Idle	Starting	Execute	Suspending	Suspended	Unsuspendin	Completing	Completed	Aborting	Aborted	Clearing	Stopping	Stopped	Reseting
Idle														
Starting														
Execute														
Suspending														
Suspended														
Unsuspending														
Completing														
Completed														
Aborting														
Aborted														
Clearing														
Stopping														
Stopped														
Reseting														

Fille \ Fille	Idle	Starting	Execute	Suspending	Suspended	Unsuspendin	Completing	Completed	Aborting	Aborted	Clearing	Stopping	Stopped	Reseting
Idle														
Starting														
Execute														
Suspending														
Suspended														
Unsuspending														
Completing														
Completed														
Aborting														
Aborted														
Clearing														
Stopping														
Stopped														
Reseting														

5.7.2 Vérification des interactions entre les fonctions de bas niveau

Nous avons ensuite créé deux tables de cohérence (Tableau 10) pour vérifier la configuration des fonctions de bas niveau ainsi que la gestion de la disponibilité des composants. On peut constater que la différence entre les deux tables réside dans les états *Execute et Starting*. En effet, lors de du partage de composants, si une fonction a déjà configuré le composant partagé, l'autre fonction ne peut plus être lancée (Starting) et par conséquent, elle ne peut être exécutée en même temps. Toutefois, pour deux fonctions qui sont indépendantes et qui ne partagent pas de composants, ces états sont compatibles (table à droite). Pour chaque case colorée des deux tables, une propriété CTL est écrite selon le Tableau 8. Un total de 210 propriétés CTL ont été vérifiées sur les modèles de bas niveau.

Au total, un ensemble de 1386 propriétés CTL a été vérifiées et validées sur nos modèles.

Tableau 10. Table de cohérence entre deux fonctions de bas niveau : à droite) sans partage de composants ; à gauche) avec partage de composants

Fonctions de bas niveau avec partage de composants	Idle	Starting	Execute	Suspending	Suspended	Unsuspending	Completing	Completed	Aborting	Aborted	Clearing	Stopping	Stopped	Reseting
Idle														
Starting														
Execute														
Suspending														
Suspended														
Unsuspending														
Completing														
Completed														
Aborting														
Aborted														
Clearing														
Stopping														
Stopped														
Reseting														

Fonctions de bas niveau sans partage de composants	Idle	Starting	Execute	Suspending	Suspended	Unsuspending	Completing	Completed	Aborting	Aborted	Clearing	Stopping	Stopped	Reseting
Idle														
Starting														
Execute														
Suspending														
Suspended														
Unsuspending														
Completing														
Completed														
Aborting														
Aborted														
Clearing														
Stopping														
Stopped														
Reseting														

6 Génération de séquences

Pour la génération de séquence, nous avons modélisé l'ordonnancement en automate temporisé. Le but ici est de créer un modèle qui modélise l'ordonnancement mathématique, obtenu dans les chapitres précédents, en un automate temporisé pouvant interagir avec le modèle opérationnel du système. La synchronisation entre ces deux modèles est également prise en compte.

6.1.1 Modélisation de l'ordonnancement

L'ordonnancement mathématique calculé correspond à deux machines exécutant chacune des fonctions séquentiellement. Nous avons transcrit cet ordonnancement mathématique en automates temporisés (Figure 42). Deux automates, modélisant chacun le séquencement sur une seule machine, ont été proposés. Dans chaque automate, le message *Marche[F_id] !* est lancé afin de lancer la fonction en question. Le passage d'un état à un autre est possible lorsque la fonction arrive à la fin de son exécution (la variable *complete[F_id]* est égale à 1).

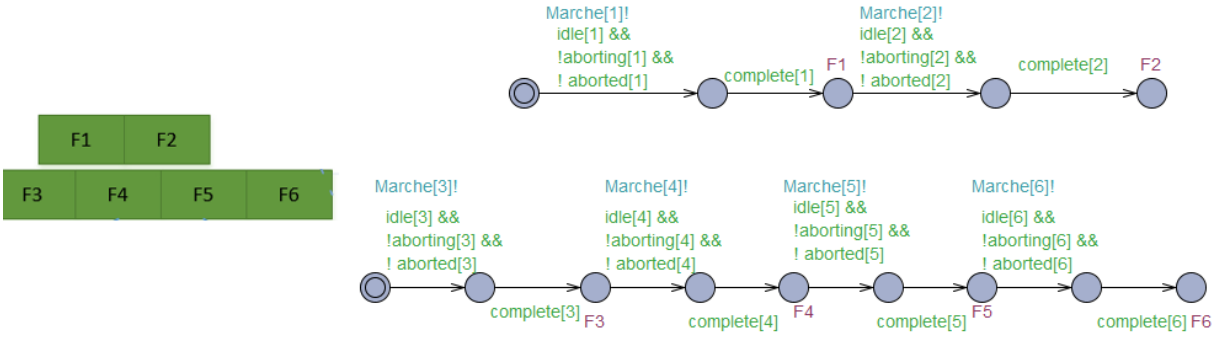


Figure 42 : Automates temporisés (à droite) pour modéliser le séquencement des fonctions sur les deux machines parallèles (à gauche)

6.1.2 Génération automatique de séquence

Ensuite, nous avons écrit une propriété d'atteignabilité dans Uppaal. Cette propriété vérifie que les derniers états de chaque machine, autrement dit, les dernières fonctions de chaque machine sont atteintes sur le système. Elle est libellée comme suit :

$$\boldsymbol{E} <> (Machine1.F2 \;\&\&\; Machine2.F6)$$

Si la propriété est vérifiée alors cela signifie que l'ordonnancement mathématique calculé est réalisable sur le système. Le déroulement de cet ordonnancement permet également de générer les séquences de commande nécessaires à son exécution. Les séquences obtenues sont de la forme suivante.

1. Marche automatique AND
    1.1. Marche automatique sous-fonction SAND1
    1.2. Marche automatique sous-fonction SAND2
        1.2.1. Marche automatique sous-sous-fonction 1
            1.2.1.1. Ouvrir vanne1
            1.2.1.2. Ouvrir pompe1
            1.2.1.3. Fin production sous-sous-fonction1
            1.2.1.4. Fermer vanne1
            1.2.1.5. Fermer pompe1
        1.2.2. Marche automatique sous-sous-fonction 2
            1.2.2.1. Ouvrir vanne1
            1.2.2.2. Ouvrir pompe1
            1.2.2.3. Fin production sous-sous-fonction2
            1.2.2.4. Fermer vanne1
            1.2.2.5. Fermer pompe1
            1.2.2.6.
    1.3. Fin production SAND1
        1.3.1. Marche automatique sous-sous-fonction 1
            1.3.1.1. Ouvrir vanne1
            1.3.1.2. Ouvrir pompe1
            1.3.1.3. Fin production sous-sous-fonction1
            1.3.1.4. Fermer vanne1
            1.3.1.5. Fermer pompe1
        1.3.2. Marche automatique sous-sous-fonction 2
            1.3.2.1. Ouvrir vanne1
            1.3.2.2. Ouvrir pompe1
            1.3.2.3. Fin production sous-sous-fonction2
            1.3.2.4. Fermer vanne1
            1.3.2.5. Fermer pompe1
            1.3.2.6.
    1.4. Fin production SAND2
2. Fin production AND

6.1.3 Résultats & discussion

6.1.3.1 Caractéristiques des séquences générées

La séquence générée montre que les opérateurs de décomposition (OR, SAND et AND) entre les différentes fonctions ont été respectés. Au niveau des fonctions de type AND, l'opérateur AND a été respecté, car toutes les sous-fonctions ont été lancées en même temps (SAND1 et SAND2). Mais comme la fonction SAND2 utilisent les mêmes sous-fonctions que SAND1, elle a été lancée une fois que ces deux sous-fonctions ont été libérées. Egalement pour le lancement séquentiel des sous-fonctions liées par un SAND. La séquence générée montre également que les fonctions qui partagent les mêmes composants sont exécutées séquentiellement. L'exclusion mutuelle de ces composants est également respectée.

Nous constatons également que les séquences générées par notre approche sont plus fines car elles illustrent clairement la décomposition fonctionnelle du système, contrairement aux approches existantes qui se limitent à un seul niveau de fonction (la décomposition fonctionnelle n'est pas prise en compte).

6.1.3.2 Problème d'explosion combinatoire

Toutefois, le problème de l'explosion combinatoire peut empêcher ce processus. En effet, quand la taille du système est important et le nombre de fonctions et des composants du système est élevé, la génération de séquence échoue.

Les travaux de (Cochard, 2018) proposent une approche itérative pour contrer ce problème et permettant la génération de séquences sur des systèmes complexes de tailles importantes. Ainsi, l'application de cette approche par itérations sur nos travaux par exemple, permet une exploration itérative des niveaux d'abstractions de la WDA. On peut alors imaginer trois

itérations. La première itération est réalisée entre les fonctions de haut niveau et leurs sous-fonctions. La deuxième itération est réalisée entre les sous-fonctions et leurs sous-sous-fonctions. Enfin, la dernière itération se réalise entre les sous-sous-fonctions et les composants du système. Ainsi, la taille du modèle est réduite pour chaque itération et permet d'éviter l'explosion combinatoire.

7 Conclusion

Ce chapitre présente une approche de génération automatique des séquences de commande. Cette approche permet le passage d'un ordonnancement abstrait, calculé mathématiquement, à un ensemble de commandes élémentaires exécutables.

Notre approche permet de faciliter l'obtention du modèle opérationnel du système. Initialement, une bibliothèque de mode a été proposée et vérifiée formellement. Cette bibliothèque regroupe les différents modes de fonctionnement, ainsi que leurs états. Sur ces modèles, un ensemble de propriétés a été identifié, formalisé en CTL, puis vérifié sur les automates temporisés. Ces vérifications formelles permettent de s'assurer de la qualité des modèles.

Ensuite, la décomposition WDA a été utilisée conjointement avec la bibliothèque de mode afin d'obtenir le modèle opérationnel du système. Des informations sur les fonctions du système, leur décomposition ainsi que leurs configurations ont été ajoutées. Quatre modèles de fonctions ont été proposés : 3 modèles pour la décomposition fonctionnelle suivant : OR, AND, SAND et un modèle pour les fonctions de bas niveau et leur interaction (configuration) avec les composants du système. Un autre modèle, représentant les composants du système ainsi que leur disponibilité, a été également proposé. Ces modèles sont conçus d'une manière générique afin de faciliter leur adaptabilité à d'autres contextes. Des boucles de vérification formelle ont été réalisées afin de s'assurer de la qualité du modèle global avant son utilisation et qu'il respecte la décomposition fonctionnelle de la WDA. En effet, l'activation des modes des niveaux supérieurs entraine l'activation des modes des niveaux inférieurs. De plus, la désactivation des modes des niveaux supérieurs est possible si les modes des niveaux inférieurs sont désactivés. D'autre part, les opérateurs de décomposition (OR, AND, SAND) sont également respectés. Des boucles de vérification formelle ont permis également de garantir la qualité de ce modèle.

D'autre part, l'ordonnancement mathématique , a été modélisé par des automates temporisés. Une démarche facilitant le passage du modèle mathématique vers l'automate temporisé a été également proposée. Le déroulement du modèle de l'ordonnancement sur le modèle du système a permis la génération des séquences de commande correctes par model checking. A la suite de cette approche, nous avons pu générer des séquences plus fines que celles de la littérature. En effet, notre modèle du comportement intègre plus d'informations opérationnelles en comparaison avec les approches existantes. Toutefois, toutes ces informations rendent le modèle complexe et introduisent un problème d'explosion combinatoire. La génération de séquence par itérations (Cochard,2018) peut apporter une solution efficace à ce problème.

Partie III : Approche outillée et proposition d'un xDSML.

Publication scientifique :

MESLI-KESRAOUI, Ouissem, OUHAMMOU, Yassine, GOUBALI, Olga, et al. Towards a Model-Based Approach to Support Physical Test Process of Aircraft Hydraulic Systems. In : Model and Data Engineering: 10th International Conference, MEDI 2021, Tallinn, Estonia, June 21–23, 2021, Proceedings 10. Springer International Publishing, 2021. p. 33-40.

Chapitre 6 : Etat de l'art sur les DSMLs et xDSML

1 Introduction

La spécification des différentes facettes des systèmes industriels de contrôle-commande est complexe car elle exige la combinaison de plusieurs langages. La littérature est riche en termes de langages dédiés au domaine. Cependant, ces langages ne permettent de spécifier qu'une seule facette à la fois. De plus, l'utilisation de ces langages par des experts du domaine nécessite des efforts d'apprentissage qui vont alourdir le processus de spécification. Le but de cette partie de la thèse est de proposer un langage dédié exécutable (xDSML), qui va en premier temps aider les experts dans la spécification des différentes facettes d'un système SCADA. Dans un second temps, notre xDSML intègre les informations opérationnelles nécessaires pour la réalisation de l'ordonnancement et l'exécution des séquences de commandes générées.

2 Problématique

La réalisation de cet objectif nécessite de faire face à certaines difficultés, et plus particulièrement l'analyse détaillée du domaine. En effet, comprendre un domaine nécessite beaucoup d'efforts d'analyse. De plus il faudrait pouvoir combiner et homogénéiser toutes les données des différentes facettes en un seul langage pivot.

La complexité de conception des systèmes SCADA ne cesse de croître. Les concepteurs font toujours face aux problèmes liés à l'intégration de programmes dépendants de plusieurs logiciels. Les efforts de correction en cas d'erreur sont alors démultipliés car il faut faire intervenir plusieurs concepteurs afin d'en comprendre la source et apporter des corrections qui sont fastidieuses. Pour résoudre ces problèmes, plusieurs travaux ont utilisé l'Ingénierie Dirigée par les Modèles (IDM).

L'IDM et le développement des langages dédiés (*Domain Specific Languages* - DSL) sont des méthodes qui font face aux challenges liés au développement des systèmes complexes (Roy Chaudhuri et al., 2019). En effet, lors du développement, le point d'intérêt des concepteurs est leur corps de métier qui leur permet d'extraire les notions liées à la spécification des différentes facettes du système d'une manière proche de la réalité et de concentrer leurs efforts sur la spécification plutôt que sur les détails d'implémentation. L'utilisation des concepts et termes du domaine offre aux experts la possibilité de comprendre, de valider, de modifier, ou même de créer des DSL (Bussenot, 2018). Un DSL est un langage conçu pour répondre aux besoins d'un domaine particulier. Il permet de créer des modèles de systèmes complexes, plus proches de la réalité, en utilisant des vocabulaires propres au domaine, pour faciliter la compréhension auprès des différents concepteurs.

Lors de la conception des systèmes SCADA, les différentes étapes de conception peuvent enchaîner l'utilisation de différents DSL adressant différentes phases de conception, ou même combiner des DSL adressant différents domaines. En effet, la conception complète d'un système SCADA nécessite une réflexion sur plusieurs parties complexes : son P&ID, ses fonctions, son comportement dynamique, ses exigences et son interaction avec les utilisateurs finaux.

- Le P&ID représente l'architecture physique du système à concevoir. Il est souvent décrit par un des logiciels dédiés, comme MS Visio par exemple.
- Le comportement dynamique représente le fonctionnement du système. Il est souvent décrit avec des langages permettant d'exprimer des changements d'états tels que les automates et les diagrammes Etat/Transition.

- Les exigences, quant à elles, concernent toutes les contraintes du domaine. Elles peuvent par exemple être spécifiées avec Sysml (Wolny et al., 2020).
- Les spécifications fonctionnelles décrivent les fonctions réalisées par le système et la façon dont il les réalise. Elles peuvent être dans l'industrie simplement décrites dans des fichiers tableur.

3 Les DSLs

3.1 Définitions

MDE (Model Driven Engineering) est une approche de développement basée sur les modèles (Almonte et al., 2022). De fait, le *modèle* est une notion très essentielle dans ce processus de développement car il est utilisé dans la spécification, la validation, la simulation et la génération de code du système considéré. Définir un système par des modèles revient à utiliser des langages qui permettent de spécifier un ensemble de concepts et leurs relations (Kanso, 2010). Chaque langage possède une *syntaxe abstraite* appelée *méta-modèle*, c'est un modèle sous forme d'entités et de relations entre ces entités. Généralement, la syntaxe abstraite ne contient pas assez d'informations pour définir le sens de construction et la cohérence des modèles alors elle est complétée par une *sémantique statique* qui se concentre sur la signification du modèle avant son exécution (Temate Ngaffo, 2012). La sémantique statique peut être exprimée sur le méta-modèle soit en la complétant par un langage comme OCL (Mohanan, 2020). Dans certains travaux une sémantique dynamique est nécessaire pour décrire le comportement du modèle afin de l'animer (Combemale et al., 2012).

Un DSL, petit langage ou langage dédié, est un langage spécifique à un domaine, conçu pour répondre un besoin particulier (Zdun et al., 2009). Il permet d'élever le niveau d'abstraction de programmation, il permet des constructions haut niveau avec des notations dédiées au domaine (Consel et al., 2005). L'objectif étant de fournir aux utilisateurs des outils dédiés à leurs métiers ou à la tâche (Temate Ngaffo, 2012) fidèles aux domaines et aux habitudes. Ceci permet de concentrer les efforts des utilisateurs sur le « quoi » (fonctions logiques) faire plutôt que sur e « comment » (implantation). Les exemples de langages dédiés sont nombreux, nous pouvons citer le langage de de requêtes de base de données SQL (Audibert, 2007), le langage MS VISIO (Hervo & Le Frapper, 2007), le langage Latex (Kopka & Daly, 2003), etc. Le cycle de développement d'un langage dédié est décrit par les étapes suivantes.

3.2 Etapes de développement de DSL

3.2.1 Analyse du domaine de travail

Le développement d'un DSL passe par l'analyse du domaine (Kosar et al., 2016), qui permet de conceptualiser le DSL à créer. L'objectif est de faire identifier par un expert du domaine tous les concepts, les terminologies, les liaisons entre ces concepts ainsi que toutes les informations du domaine nécessaires pour la création du DSL L'analyse du domaine est une étape très importante pour garantir la couverture et la complétude du langage. Elle permet d'extraire les différentes facettes du domaine sous étude, les différents concepts et règles du domaine.

3.2.2 Création de la syntaxe abstraite du DSL

La deuxième étape consiste à développer la syntaxe abstraite du DSL, sous la forme d'un méta-modèle. Dans cette étape, les données récoltés issues de l'étape précédente sont représentées sous forme de classes et de liaisons entre les classes. Le méta-modèle est une représentation graphique contenant les concepts d'un domaine et les liaisons entre ces concepts. La représentation sous forme graphique limite la complexité de l'expression des règles du domaine sur les concepts et les

liaisons entre eux. *Une sémantique statique* est ajoutées d'abords sur les classes pour ajouter des règles du domaine et par des contraintes OCL si nécessaire.

3.2.3 Création de la syntaxe concrète du DSL

A partir de syntaxe abstraite, des instances concrètes peuvent être générées. Les instances ne sont pas compréhensibles par des experts du domaine car leur manipulation nécessite des compétences en méta-modélisation. Ces langages peuvent être optimisés pour les experts du domaine avec peu ou pas d'expérience en programmation grâce à une syntaxe textuelle ou graphique simplifiée nommée une syntaxe concrète permettant la représentation de ces modèles(Greiner et al., 2023). La syntaxe concrète (celle utilisée par les utilisateurs finaux) peut être graphique ou textuelle. La manipulation du DSL est faite à travers la création d'un modèle (instance) conforme au méta-modèle, utilisant généralement sa syntaxe concrète. Le choix de la syntaxe concrète est basé sur les préférences des utilisateurs finaux. Dans la littérature, il existe plusieurs DSL supportés par des outils graphiques qui permettent de modéliser les besoins d'un domaine précis.

Un modèle issu d'un DSL est fidèlement conforme au méta-modèle qui a permis sa construction. De la même manière qu'un méta-modèle est conforme au méta-méta-modèle du langage. Un méta-méta-modèle est une description du langage dans lequel le méta-modèle est écrit (Kanso, 2010).

3.3 Les DSLs industriels

Dans cette partie, nous présentons des DSLs industriels dédiés pour chaque facette du système SCADA à savoir : physique, logique, contraintes et comportement des systèmes industriels.

3.3.1 DSLs pour la facette « architecture physique »

Les DSLs architecture physique permettent de créer des modèles pour représenter le système à travers ses composants physiques et liaisons entre eux (Lebeaupin, 2017). Les DSLs, les plus répondus, sont ceux qui permettent la conception assistée par ordinateur tel que : MS Visio (Hervo & Le Frapper, 2007) et Autocad (Le Frapper, 2006). En effet, ces langages permettent de modéliser l'architecture physique des systèmes industriels, basés sur des méta-modèles simples qui permettent la description des composants physiques à travers une syntaxe concrète graphique. La liste de ces composants est proposée sous forme d'une bibliothèque enrichie par des règles de domaine propres à chaque type de système (mécanique, hydraulique, etc.). Cependant, ces logiciels permettent de modéliser uniquement l'architecture physique (P&ID) des systèmes complexes et ont été créés dans le but de faciliter une tâche bien précise des concepteurs dans le processus de développement. Ces derniers ne permettent pas de modéliser l'architecture logique du système SCADA et il est impossible de les étendre à d'autres facettes, car ils sont privés. Les concepteurs sont alors obligés d'utiliser d'autres DSLs dont la combinaison est couteuse (Mesli-Kesraoui et al., 2021).

Le langage dédié structure (S2ML) (BATTEUX et al., s. d.) est un langage proposé dans le cadre du projet Altarica (Batteux et al., 2018b). Ce langage est très utilisé dans le monde industriel pour créer des modèles d'architecture organique des systèmes complexes. S2ML est basé sur deux types de représentation : la première est ascendante orientée objet à classe, la deuxième descendante orientée objet à prototype (Batteux et al., 2020). La représentation orientée classe permet de représenter un composant comme une construction structurelle générique puis utilisé par instanciation. A contrario dans la représentation orientée prototype, un composant est représenté par une construction structurelle par bloc ayant une occurrence unique. Le langage S2ML a deux objectifs : le premier est la construction de modèle de structure et le deuxième est de pouvoir réaliser une transformation de modèle de structure de différents langages vers des

modèles Altarica (Batteux et al., 2019). Le modèle de structure a deux représentations : la première est graphique, la deuxième est textuelle.

La présentation graphique permet de représenter chaque composant Figure 43 avec ses ports de connections, port de commande et ports d'entrées/sorties.

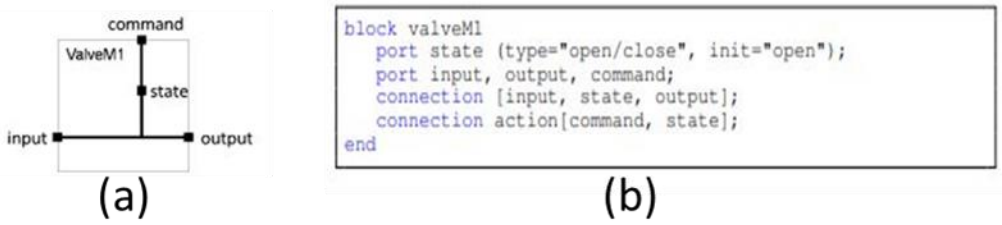


Figure 43 : (a) Vue graphique du composant ValveM1, (b) Vue textuelle du composant ValveM1 (Batteux et al., 2018a) .

Ce type de DSLs est très expressif en terme d'architecture physique. Cependant, au niveau des connections entre deux composants, cette dernière est considérée comme la relation entre eux. A contrario, dans le cas des systèmes hydrauliques, une connexion (tuyau) est considérée comme un composant à part entière : il a une référence, un nom, une longueur et d'autres caractéristiques.

3.3.2 DSLs pour la facette « architecture logique »

L'architecture logique ne manque pas de difficulté par rapport à l'architecture physique. En effet, la réalisation des objectifs du système est réalisée suivant un processus fonctionnel complexe. Cette complexité réside dans la manière d'ordonner les priorités d'exécution de ces fonctions, les contraintes métier à prendre en compte et les ressources partagées entre ces fonctions. La difficulté rencontrée ici réside dans l'harmonisation des différentes représentations des fonctions tout en les ordonnant, en les groupant, en les classifiant et en les caractérisant.

Par exemple, un DSL formel a été développé afin de faciliter la spécification de l'architecture logique des systèmes ferroviaires (Issad et al., 2015) à travers des scénarios. Un ensemble de composants est défini permettant de réaliser des actions suivant un ensemble de scénarios. Les relations entre scénarios sont définies par des opérateurs logiques de haut niveau, telles que : *Precedence* pour décrire la relation de précédence, *Parallelism* pour décrire une relation de parallélisme, *Preemption* pour décrire que deux composants peuvent réaliser une fonction mais pas les deux à la fois, *Refinement* entre deux niveaux d'abstraction, *Assignment* pour assigner un composant à une fonction, *Cooperation* deux composants nécessaires pour une fonction, *Access* pour l'accès aux sous-composants.

Le DSL ARCADIA (ARChitecture Analysis & Design Integrated Approach) (Roques, 2017) propose une approche de modélisation basée sur les modèles et destinée à plusieurs contextes industriels (aéronautique, transport, etc.). Supportée par l'outil CAPELLA, elle permet aux concepteurs systèmes de modéliser l'architecture physique (limitée aux calculateurs, réseaux ou composants physiques abstraits) et logique des systèmes complexes à travers une représentation graphique.

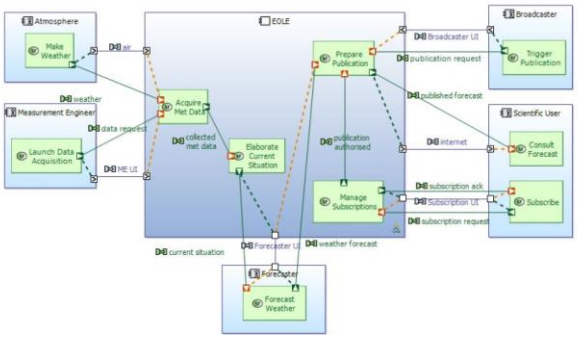


Figure 44 : Exemple d'architecture du système avec ARCADIA (Roques, 2016).

La modélisation se fait par la création des blocs pour représenter les différents composants ou fonctions du système. Cependant, la prise en compte des contraintes est réalisée au travers du langage SysML (Wolny et al., 2020). Toutefois, ce DSL ne permet pas la modélisation des composants à travers leurs ports et caractéristiques ainsi que les niveaux d'abstraction de fonctions. De par sa nature généraliste, l'adaptation du DSL ARCADIA pour les systèmes SCADA nécessite de grands efforts de modification.

3.3.3 DSLs pour la facette « Contraintes »

Dans le domaine de l'ingénierie des exigences, le DSL SysML (Wolny et al., 2020) permet de faciliter la spécification et la modélisation des exigences. SysML a été largement utilisé pour la spécification des contraintes des systèmes complexes sous forme textuelle. La force de Sysml réside dans la notion de relation établie entre les exigences. SysML est un langage et non une méthode (Belloir et al., 2014), il peut être vu comme une boite à outils pour des besoins particuliers. Sysml propose plusieurs façons de lier les données entre les classes, en utilisant par exemple des attributs de bloc. Malheureusement, la liaison n'est pas vraiment explicite (Pedroza et al., 2011).

3.3.4 DSLs pour la facette « Comportement dynamique »

Le comportement dynamique des systèmes complexes est souvent modélisé sous forme d'états et de transitions (Dangoumau, 2000a). Le langage Guarded Transition Systems GTS (Prosvirnova, 2014) est proposé pour des analyses de sécurité à travers un comportement dynamique, modélisé sous forme des machine à états. Le GTS est un langage dédié proposé dans le projet Altarica (Batteux et al., 2019). Un modèle GTS est composé d'états, de transitions et de conditions. Les états sont des variables assignées et les transitions sont enclenchées par des évènements si les conditions sont satisfaites. Avec GTS, il est possible de transformer une machine à états à un arbre de défaillance (Prosvirnova & Rauzy, 2013). Les auteurs de ces travaux assurent que les modèles de haut niveau améliorent considérablement la conception, le partage et la maintenance de modèles.

3.3.5 Bilan

La plupart des DSL qui existent dans la littérature ne permettent de modéliser qu'une ou au mieux deux facettes d'un système complexe. En effet, ces langages sont souvent utilisés séparément. Certains langages, plus généralement graphiques, permettent de spécifier l'architecture physique d'un système complexe, tandis que d'autres permettent de spécifier soit l'architecture logique, soit les contraintes. De fait, la modélisation d'un système dans sa globalité nécessite l'utilisation de plusieurs DSL, ce qui alourdit la conception des systèmes industriels complexes et engendre plusieurs problèmes de compatibilité entre ces langages.

Bien qu'il existe dans la littérature des DSL comme ARCADIA et Sysml qui permettent à la fois de modéliser l'architecture physique et l'architecture logique et d'intégrer la modélisation des contraintes, ces langages ne permettent pas d'intégrer certaines informations opérationnelles nécessaires à l'exécution correcte de l'ordonnancement.

L'intégration de plusieurs DSL développés avec des concepts et des outils différents est une tâche plus complexe comparée au développement d'un nouveau DSL pivot unifiant tous les concepts (Zdun et al., 2009).

3.4 DSL pour les systèmes SCADA

La conception d'un système SCADA nécessite plusieurs langages (Mesli-Kesraoui et al., 2021). Cette multidisciplinarité les met souvent face aux problèmes liés à l'intégration des programmes dépendants de plusieurs logiciels. Les systèmes SCADA sont orientés à deux niveaux. Au premier niveau, les automates programmables gèrent eux-mêmes leurs codes de commande et communiquent avec le procédé physique. Au niveau supérieur, la supervision utilise les données des automates pour animer des synoptiques, des tableaux, etc.

L'utilisation de l'IDM (Albeaik et al., 2022) pour la conception d'un système SCADAs revient donc à générer l'interface de supervision (IHM) et les codes de commandes correspondant à cette interface (Bignon, 2012b).

Dans le domaine de l'IHM, l'utilisation de l'IDM est une solution adaptée aux difficultés rencontrées lors de la migration d'un système de contrôle commande, d'une plateforme à une autre. Le développement des IHM s'est penché aussi sur l'automatisation basée sur le MBUID[8] (Sanctorum & Signer, 2019) afin de produire des IHM de qualité répondant aux différents challenges : hétérogénéité des utilisateurs finaux, différentes plateformes, différents langages de programmation et différents environnements de développement. C'est dans ce contexte que (Calvary et al., 2003) proposent un cadre de référence nommé CAMELEON, pour les IHM multi-cibles ou multi-contextes. La démarche proposée part des modèles abstraits décrivant les tâches et leurs contextes pour aboutir à des modèles finaux décrivant les composants exécutables. Le Framework comprend plusieurs couches : le modèle de tâches et de concepts, l'interface abstraite, l'interface concrète et l'interface utilisateur finale. Le modèle de tâches et concepts représente une hiérarchie des différentes tâches utilisateurs en termes d'objectifs et de procédures dans le but de répondre aux attentes utilisateurs du domaine concerné. L'interface abstraite est définie indépendamment de toute plateforme et permet de structurer l'IHM en espaces de travail, constitués d'un ensemble de dialogues et de présentations. Cet espace de travail doit offrir les éléments nécessaires à la réalisation d'une ou de plusieurs tâches définies dans le modèle de tâches et de concepts. L'interface concrète quant à elle, transforme les espaces de travail précédemment définis en fenêtres. Elle décrit également les relations entre les fenêtres et en matérialise les tâches élémentaires en widgets (interacteurs). C'est donc à ce niveau qu'est fait le choix des interacteurs, pour avoir une maquette permettant de visualiser la façon dont l'IHM pourra être perçue par l'utilisateur final. Enfin, l'interface utilisateur finale permet de représenter l'IHM dans un langage de programmation et d'exécution. C'est cette interface qui sera utilisée par les utilisateurs. CAMELEON est devenu un modèle de référence largement utilisé par la communauté IHM (Abrahão et al., 2021) pour générer automatiquement des interfaces multi-plateformes, multimodales et multi-langages pour différents contextes d'utilisation.

[8] Model Based User Interface Development

Dans le domaine de l'automatisme, la proposition d'approches pour la conception de la commande ne cesse de croître (Capawa Fotsoh et al., 2020). En effet, les langages dédiés sont utilisés pour représenter l'architecture organique (physique) du système industriel et ses différentes configurations (Capawa Fotsoh et al., 2020; Kanso, 2010). Ces DSLs sont construits dans le but de générer des modèles de configurations à partir de modèle d'architecture de haut niveau (De Lamotte, 2006). Pour le faire, une description logique et physique du système sont crèees. La description logique concerne les produits du SAP (Système Automatisé de Production) et les séquences de traitements qu'il subit. Ces traitements sont réalisés en organisant un ensemble de fonctions associées à chaque produit. La description physique décrit l'organisation du système en terme de ressources. Chaque ressource réalise des opérations pour implémenter une fonction de la partie logique. Dans les travaux de (Prat et al., 2015) une typologie de ces opérations est donnée, comme par exemple, *l'opération de transfert* qui permet de déplacer un fluide d'une soute à une autre, *l'opération de stockage* qui permet le stockage du fluide dans un réservoir en attendant son utilisation, etc.

Les opérations décrites au niveau de l'architecture sont potentielles, c'est-à-dire, elles ne sont pas encore utilisées et indiquent qu'une ressource donnée peut réaliser l'opération qui implémente une fonction de la partie logique (De Lamotte, 2006). Au niveau de la configuration, une partie logique reprend les fonctions utilisées en les instanciant et une partie physique qui décrit les ressources utilisées. Dans ces travaux, la configuration a deux objectifs : le premier au niveau de la configuration physique qui permet de décrire les chemins que peuvent emprunter les produits appelés *séquences de transferts*. Le deuxième au niveau de la configuration logique qui permet de décrire l'enchainement des opérations pour réaliser les séquences de fonctions. Cet enchainement appelé *séquence d'opérations* peut être rapproché aux gammes opératoires des problèmes d'ordonnancement.

Les travaux de (Bignon, 2012b; Goubali, 2017; Kesraoui, 2017; Prat et al., 2015) ont permis la construction d'un langage dédié permettant la génération conjointe de la supervision et code de commandes d'un système SCADA. Le méta-modèle représentant l'architecture physique est décrit sous forme de librairie d'éléments dont chaque composant possède une vue synoptique, une vue commande et une vue supervision. Le synoptique est aussi représenté par un méta-modèle conforme à la norme ANSI/ISA 5.1 (Symbols, 2009). Le modèle de synoptique est ainsi obtenu par transformation du modèle synoptique en VISIO vers le modèle synoptique décrit par le méta-modèle proposé. Les modèles IHM et commandes sont générés conjointement par transformations de modèles.

La génération conjointe des codes de commande et des interfaces de supervision permet d'assurer la cohérence entre les différentes parties du système industriel (Bignon, 2012a). Bien que les approches proposées dans (Goubali, 2017) (Goubali et al., 2014) permettent de garantir cette cohérence, de réduire le temps et les erreurs de conception, elles possèdent quelques lacunes. En effet, l'outil SCADA offre une interface pour créer un modèle d'entrée avec un DSL qui n'est habituellement pas utilisé dans la conception des systèmes SCADA. Ce DSL n'est donc pas compréhensible par des experts métier et il leur est impossible de le manipuler. Ces travaux permettent de générer automatiquement et conjointement des interfaces de supervision et des codes de commande qui sont spécifiques à des plateformes d'exécution de systèmes SCADA telles que Panorama E2 et Straton (Goubali, 2017; Kesraoui, 2017; Prat et al., 2015). En effet, le processus de génération proposé s'appuie, à ce jour, sur un choix de logiciels de contrôle-commande effectué par (Bignon, 2012a). Ces logiciels servent d'une part à décrire les composants du système à concevoir, puis à réaliser la fenêtre qui va accueillir ces composants au niveau de l'interface de supervision. D'autre part, ces logiciels servent à interpréter les interfaces et les programmes de commande globaux générés. De fait, l'utilisation de nouveaux logiciels de

contrôle-commande conduira à la modification des transformations de modèle. La modification de toutes ces transformations peut demander un effort considérable de développement. Il serait alors intéressant d'explorer des pistes pouvant permettre de partir de modèles de type PIM (Plateform Independant Model) pour ensuite les adapter à différents logiciels afin d'obtenir des PSM (Plateform Specific Model) (Melouk et al., 2020), sans trop d'efforts de modification des transformations.

4 DSL Exécutable (xDSML)

Le concepteur d'un modèle décrivant le comportement dynamique d'un système doit généralement le simuler et l'animer pour vérifier son comportement (Combemale et al., 2010). Malheureusement, au niveau méta-modèle seules les informations liées au domaine, contraintes, fonctions, états/transitions sont définies. Les informations portant sur comment ces informations interagissent entre elles pendant l'exécution ne sont pas définies. En effet, pour pouvoir modéliser la séquence d'un ordonnancement, nous avons besoin d'un ordonnancement de fonctions et les codes de commandes (ouverture, fermeture, exécution, etc.) qui permettent sa mise en œuvre d'une manière concrète. Le méta-modèle (syntaxe abstraite) permet juste la définition de ces informations de domaine sous forme de classes et propriétés. Cependant, pour l'exécution, il est nécessaire de capturer les informations dynamiques telles que : la séquence d'évènements (séquence de code de commandes) qui animent les éléments du domaine définis dans la syntaxe abstraite (méta-modèle) ainsi de décrire comment ces animations changent les états des éléments du domaine. La *sémantique opérationnelle* (Leroy et al., 2020) vient pour ajouter les règles d'exécution à la syntaxe abstraite du langage considéré (Nastov et al., 2015). Elle considère donc comme domaine sémantique une extension du domaine syntaxique (Jézéquel et al., 2012).

Dans la pratique, il existe deux manières (Jézéquel et al., 2012; Nastov et al., 2015) pour implanter la sémantique opérationnelle directement dans le méta-modèle (syntaxe abstraite). La première est l'utilisation d'un langage de méta-modélisation offrant un langage d'action tel que : Kermeta (Jézéquel et al., 2011), xOCL (Ramalho et al., 2003),etc. La deuxième est d'exprimer une transformation endogène sur la syntaxe abstraite à l'aide d'un langage de réécriture tel que QVT (Rouhi & Lano, 2020) ou ATL (Götz & Tichy, 2020). Cependant, ces deux techniques sont dépendantes des langages de programmation, ce qui est dans certain cas difficile pour les concepteurs systèmes (Nastov et al., 2015).

Une autre approche proposée par (Combemale et al., 2010) permet de crée un xDSML en suivant un design pattern composé de cinq blocs (Figure 45) de méta-modèles. Le premier , est le *méta-modèle de domaine (DDMM)*, ce méta modèle regroupe toutes les informations concernant le système (architecture physique et logique), ce qui est représenté généralement par la syntaxe abstraite statique. Le deuxième, u*n méta-modèle d'évènements EDMM*, qui définit l'ensemble de stimuli internes et externes qui peuvent changer le comportement des éléments du DDMM. Les informations de comportement sont définies dans le troisième *méta-modèle d'états/transitions (SDMM) qui* regroupe tous les états possibles que peuvent avoir les éléments du DDMM. Tous ces méta-modèles (DDMM, EDMM, SDMM) sont les éléments clés pour la syntaxe abstraite du XDSML (Combemale et al., 2010).

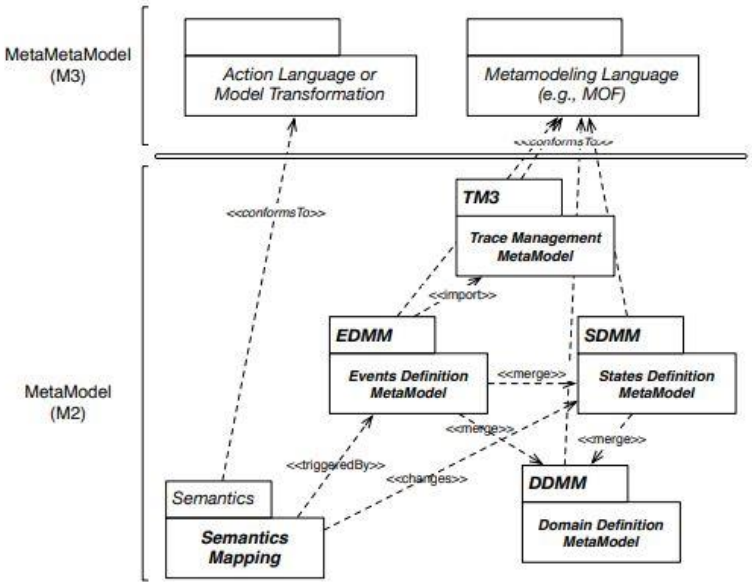


Figure 45 : Design pattern pour les XDSML (Combemale et al., 2010)

Pour la partie exécution, le quatrième méta-modèle :*TDMM, un méta-modèle de trace*, définit la séquence d'ordonnancement par exemple (Nastov et al., 2015). Il permet de gérer l'exécution du modèle en capturant les stimuli sous forme d'évènements du EDMM. La dernière partie (la cinquième) est la *sémantique exécutable* composée de règles d'exécution. Concrètement, elle définit comment le modèle SDMM évolue à travers les éléments DDMM (changement de leurs états) en adéquation avec le stimulus du EDMM.

Les DSMLs exécutables permettent de générer des modèles exécutables. Les règles d'exécution sont illustrées à travers une sémantique dynamique. La définition d'un XDSML est nécessaire pour la traduction des séquences de commandes (réf. Génération de séquence) et la séquence d'ordonnancement calculée (réf. Formulation du problème).

5 Conclusion

Lors de la conception des systèmes complexes de type SCADA, différents langages sont utilisés. Chaque langage correspond à une facette du système. Par exemple, les langages utilisés pour décrire l'architecture d'un SCADA ne sont pas adaptés pour décrire les contraintes, ni les spécifications fonctionnelles. Des problèmes d'intégration et de compréhension de langages, dont les structures et les spécificités sont différentes, émergent lors de la conception ; la génération des systèmes SCADA s'en trouve également impactée.

Le proposition d'un langage dédié pour la conception d'un système SCDA semble difficile vu qu'il faut combiner, homogénéiser et structurer les données issues de plusieurs facettes.

Afin de permettre l'exécution du DSML, les sémantiques opérationnelles (dynamiques) viennent pour ajouter des règles d'exécution à la syntaxe abstraite. Cette sémantique est exprimée soit à l'aide de langage d'action comme kermeta (Jézéquel et al., 2011) ou par des transformations de modèles (Rouhi & Lano, 2020). Ces deux approches semblent difficiles à appliquer vu qu'elles nécessitent beaucoup d'efforts de programmation. Une autre approche a été proposée par (Combemale et al., 2010) et permet de définir cette sémantique à travers un design pattern composé de cinq méta-modèle est la plus adaptée à notre cas.

Chapitre 7 : Proposition d'un Langage dédié exécutable (DSML) pour les bancs de tests SCADA

1 Introduction

L'objectif est de proposer un langage dédié permettant la spécification et la modélisation de toutes les facettes d'un banc de tests SCADA.

Lors de la conception d'un banc de tests SCADA, les différentes facettes (architecture physique, architecture logique, comportement dynamique, contraintes, utilisateurs et domaine) sont spécifiées avec des outils, et des langages, différents les uns des autres pour représenter des données dans des formats bien adaptés au type de spécifications réalisées. Homogénéiser toutes les informations des différentes spécifications dans un seul langage afin de centraliser les efforts de spécification n'est pas chose aisée. En effet, il faut pourvoir définir un langage pivot riche en sémantique et homogène qui tient compte de toutes les facettes : l'architecture physique, l'architecture logique, les contraintes, le comportement dynamique.

2 Aléas et verrous

Pour le développement d'un XDML pour les systèmes SCADA, nous devons répondre aux verrous suivants :

- Définition des concepts du domaine à prendre en considération et les combiner et homogénéiser.
- Structuration de toutes les données dans un seul méta-modèle.
- Assurer l'exécution du modèle en suivant la séquence d'ordonnancement calculée et les séquences générées.
- Afin de rendre facile l'utilisation de ce XDSML par des experts non-informaticiens, il est nécessaire de le présenter dans un format proche du domaine.

3 Approche globale

Pour le développement d'un xDSML dédié aux systèmes SCADA, nous avons proposé une approche composée de deux étapes.

La première étape est la proposition d'un modèle de référence pour la conception des systèmes SCADA. Ce modèle de référence nous permet de poser les facettes de bases composant un système SCADA et les liaisons entre ces facettes.

Dans la deuxième étape, nous proposons un xDSML dont la syntaxe abstraite statique et la sémantique statique se basent sur le modèle de référence de la première étape. La syntaxe abstraite dynamique ainsi que sa sémantique dynamique (opérationnelle) viennent ajouter du dynamisme au DSML. Enfin, la syntaxe concrète est créée pour faciliter l'utilisation du xDSML par des experts de domaine non-informaticiens.

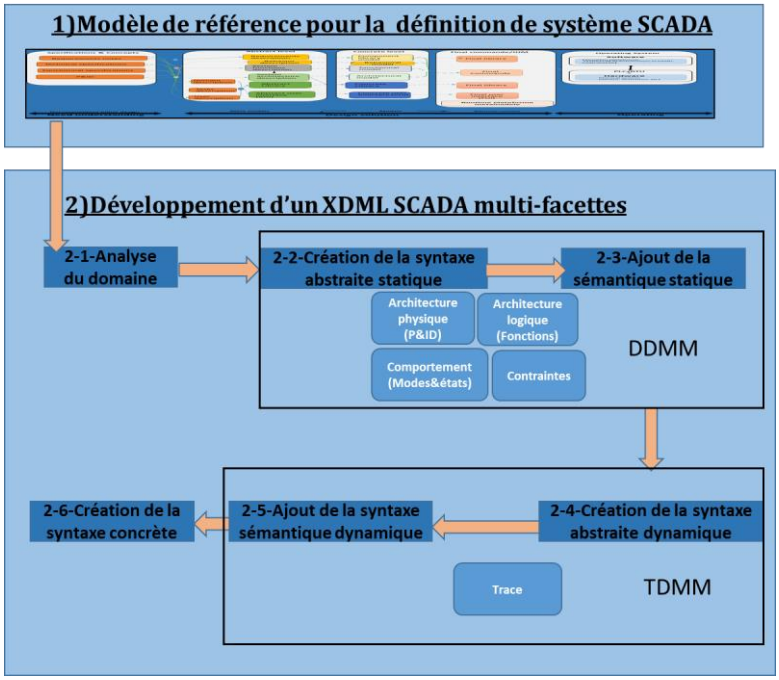


Figure 46 : Approche utilisée pour créer le xDSML SCADA.

4 Un modèle de référence pour la conception des systèmes SCADA

Pour rendre la démarche de définition d'un langage pivot dédié aux systèmes SCADA complétement générique, il est nécessaire de d'abord s'intéresser à la définition d'un cadre de référence comme CAMELEON (Collignon et al., 2008), adapté à la conception de systèmes SCADA.

La définition d'un modèle de référence, dédié aux système SCADA, nécessite la connaissance a priori des modèles de référence pour la définition de toutes les parties de ce type de système. Cependant, dans la littérature, à notre connaissance, il n'existe à ce jour aucun modèle de référence complet pour la partie commande et pour les interfaces de supervision des systèmes SCADA. Il est donc nécessaire de définir un modèle de référence pour chaque partie (supervision et commande) du système.

4.1.1 Proposition d'un modèle de référence pour la partie commande

Le modèle proposé pour la partie commande commence par le recueil des spécifications extraites du cahier des charges et les différentes exigences fournies par les experts métier sous forme d'exigences métier. Ces spécifications sont ensuite utilisées pour la définition de la commande d'une manière abstraite, c'est-à-dire sous forme d'entités et de relations entre ces entités (méta-modèle). Ensuite, un modèle concret de la commande devra être créé, tout en assurant la conformité avec le méta-modèle du niveau abstrait. Le modèle concret de la commande ainsi défini devra être proche de la réalité sans décrire toutes les facettes du code final. Après que le modèle concret ait été défini, un passage au modèle final permettra d'intégrer les informations d'implémentation pour la génération des codes de la commande (exploitable par les logiciels de contrôle-commande).

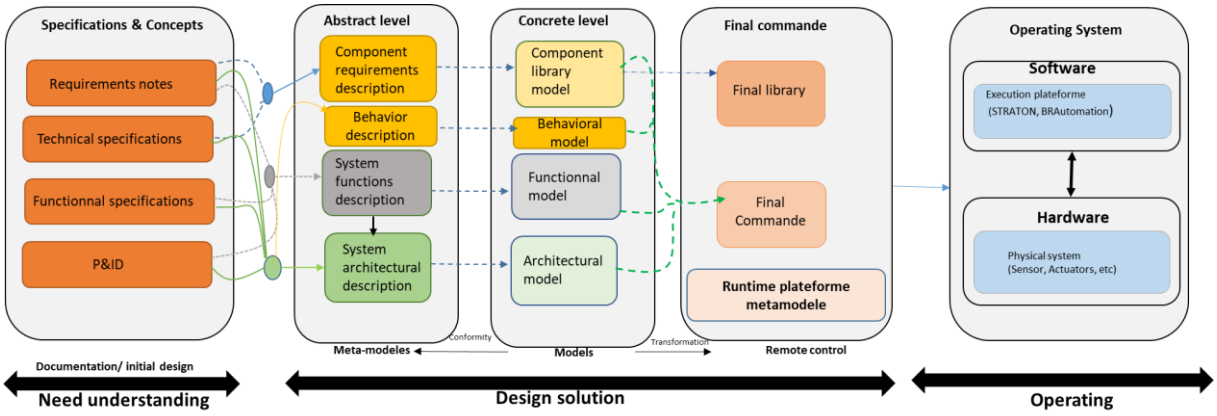


Figure 47 : Modèle de référence pour la génération de la commande multiplateforme

La Figure 47 présente le modèle de référence proposé pour la définition de la commande. Il est structuré en quatre niveaux d'abstraction: *Spécifications & Concepts*, *Abstract level*, *Concrete level* et *Final commande*.

- ✓ *Spécifications & concepts*. C'est un ensemble de spécifications de domaine et du contexte qui permettent une bonne compréhension du système à concevoir. On y retrouve généralement trois documents ainsi que des schémas industriels :
  - Cahier des charges. Un document client qui contient les informations sur le système à concevoir, le contexte, les besoins clients.
  - Spécifications techniques. Ce sont des exigences techniques à respecter et à suivre lors de la définition du système. Elles sont décrites dans des fichiers au format Word, ou obtenues par entretien avec les experts métier.
  - Spécifications fonctionnelles. C'est la description de ce que le client attend du système en termes de fonctions. Ici, le système à concevoir devra pouvoir réaliser des fonctions attendues.
  - P&ID. C'est le schéma synoptique du système fourni sous un format bien défini (comme VISIO, PDF, AUTOCAD, etc.). Ce schéma permet d'avoir une vision de l'architecture du système ainsi que de ses composants.
- ✓ *Abstract level*. A ce niveau, trois descriptions essentielles du système sous différentes facettes, sont créées. Ces trois descriptions sont exprimées sous forme de méta-modèle et s'appuient sur les différents modèles définis dans la méthode:
  - *Components requirements description*. Un méta-modèle construit à partir du cahier des charges et des spécifications techniques. Dans ce méta-modèle, on regroupe toutes les exigences métier nécessaires au fonctionnement du système.
  - *System functions description*. Dans cette étape, les différentes fonctions que le système doit accomplir sont définies. Ces fonctions sont regroupées et ordonnées logiquement sous forme d'un méta-modèle de fonctions.
  - *Behavior description*. Dans cette étape, tous les états et transitions entre composants, fonctions sont définis. Nous recueillons aussi les informations concernant les modes du système (normal, dégradé, etc.)
  - *System architecture description*. Un méta-modèle représentant l'architecture physique du système. L'ensemble des composants du système et les connexions entre eux, sont extraits à partir du P&ID. Dans ce méta-modèle, on fera aussi une allocation des fonctions aux composants afin d'avoir une idée sur comment le système doit fonctionner pour répondre aux attentes.
- ✓ *Concrete level*. A partir des méta-modèles obtenus de la phase précédente, la partie commande sera représentée sous une forme plus concrète en définissant les

entrées/sorties des composants ainsi que les opérations nécessaires à leur fonctionnement.

- ✓ *Final command*. Arrivé à cette étape, le logiciel d'exécution des codes de commandes, est identifié ainsi que son méta-modèle. Les modèles construits à l'issu des étapes précédentes seront utilisés pour générer les codes de commande spécifique au logiciel identifié.

4.1.2 Proposition d'un modèle de référence pour la partie supervision

Le modèle de référence pour la supervision débute par la spécification de toutes les informations nécessaires pour la construction d'une interface de supervision (spécifications du domaine, informations sur les utilisateurs et leurs tâches). Ces spécifications et concepts seront utilisés pour être représentés sous forme d'entités (méta-modèle) : c'est l'interface abstraite. En ajoutant les différents acteurs, interactions et liens entre ces entités, l'interface concrète sous forme d'un ou de plusieurs modèles est obtenue. L'interface finale est l'interface opérationnelle implémentée dans un langage de programmation et une plateforme d'exécution spécifique.

La Figure 48 présente le modèle de référence pour la définition de la supervision, structuré sur quatre niveaux d'abstraction : *Spécifications & concepts, Abstract level, Concrete level et Final user interface.*

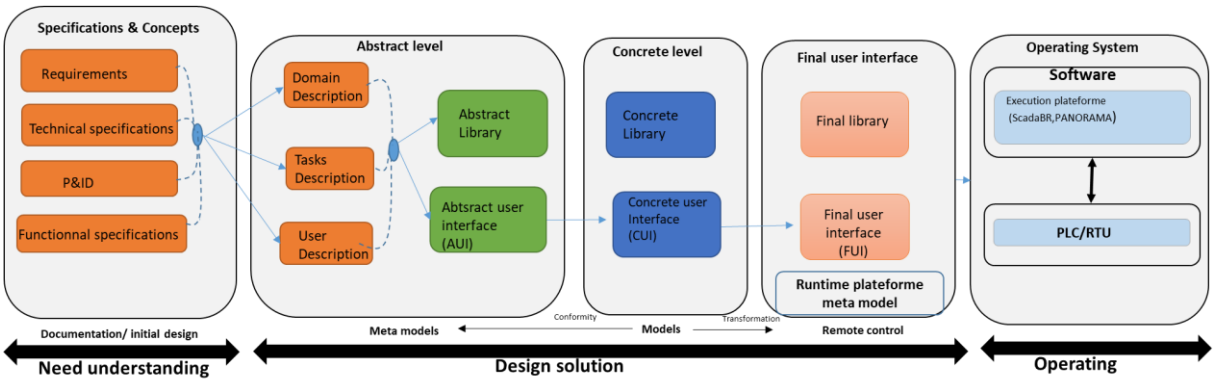


Figure 48 : Modèle de référence pour la génération de la supervision multi-destination

- ✓ *Spécifications & concepts*. Les mêmes spécifications utilisées dans le modèle de référence pour la commande (voir section 4.1.1).
- ✓ *Abstract level*. A ce niveau, on va d'abord procéder à la création de trois méta-modèles à partir des Spécifications & concepts. Ces méta-modèles sont d'une abstraction très élevée et importante pour la création des interfaces car ils contiennent des informations sur le domaine, les caractéristiques des utilisateurs et les différentes tâches que l'utilisateur va effectuer sur cette interface. Cela permettra de créer des interfaces multiplateformes. Ils sont définis comme suit :
  - *Domain Description*. Contient les différentes caractéristiques et propriétés du domaine, c'est-à-dire les différentes fonctions qui caractérisent les interactions entre l'utilisateur et l'interface.
  - *Task Description*. Représente les actions de l'utilisateur sous forme de tâches. Il exprime comment l'utilisateur devra interagir avec le système pour atteindre un but.
  - *User Description*. Permet d'identifier et de définir le profil des utilisateurs. Ce modèle est utilisé pour classifier les utilisateurs. Cette classification mènera à la création des profils utilisateurs selon leur préférence et permettra de justifier les variations entre différentes phases de modélisation du comportement du système.

A partir de ces trois descriptions, une interface utilisateur abstraite sera créée en prenant en compte des différentes facettes de cette dernière. Cette interface est représentée par des entités et des liens entre ces entités sous forme d'un méta-modèle complet.

- ✓ *Concrete level*. C'est la représentation de l'interface abstraite obtenue lors de la phase précédente, sous une forme plus concrète en utilisant les composants graphiques de la bibliothèque concrète (widgets ou interacteurs). En effet, la bibliothèque concrète devra contenir un ensemble d'interacteurs pouvant être utilisés pour réaliser les différentes tâches des opérateurs, la définition des fenêtres génériques et les liaisons entre ces dernières.
- ✓ *Final user interface*. L'interface concrète issue de l'étape précédente, sera utilisée pour la génération du code de l'interface finale. Ce code pourra être exploité par les logiciels de supervision industrielle.

4.1.3 Proposition d'un modèle de référence pour les applicatifs de contrôle-commande

Le modèle de référence proposé, nommé CAMADIA (Figure 49), présente une méthodologie unifiée et un processus unique pour la définition conjointe des applicatifs de contrôle-commande (Commande et Supervision) multiplateformes. Ce modèle a été construit en suivant plusieurs niveaux d'abstraction.

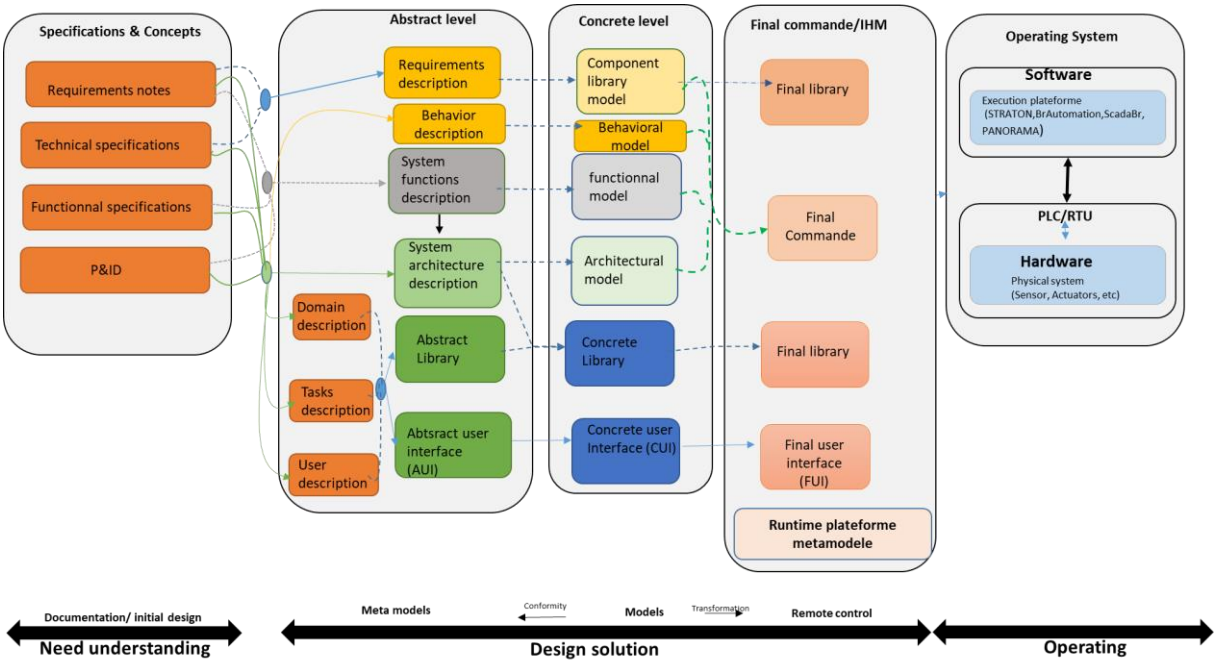


Figure 49 : Modèle de référence CAMADIA pour la définition conjointe de la supervision et commande multi-plateforme.

D'autre part, le passage d'un niveau d'abstraction à un autre (Figure 49) ne revient pas seulement à unifier les informations contenues dans les modèles de référence de la Figure 47 et Figure 48. En effet, une analyse approfondie a été réalisée sur chaque niveau d'abstraction des deux modèles de références précédents pour avoir une modélisation fine au niveau suivant du modèle de référence conjoint (Figure 49). Cela permet notamment de garantir la cohérence entre les parties du système (commande et supervision) ; et d'affiner les niveaux d'abstraction du modèle de référence conjoint afin de tenir compte de toutes les informations nécessaires à la conception du contrôle-commande. Par exemple, nous nous sommes aperçus après analyse des niveaux d'abstraction de chaque modèle que la seule utilisation du « *Domain description* » pour avoir le «

Abstract User Interface » puis le « *Concrete User Interface* » dans le modèle de référence de la supervision (Figure 48), ne suffit pas pour garantir la communication avec la partie commande. Par conséquent, dans le modèle de référence conjointe (Figure 49), nous avons pris en compte l'architecture (*system architecture description*) et la description fonctionnelle (*system functions description*) du système dans la description du « *Concrete User Interface* » ; ce qui permettra également d'avoir une interface de supervision plus globale.

5 Proposition d'un langage SCADA exécutable et multi-facettes

5.1 Etape 1 : Analyse du domaine de travail

L'analyse du domaine est une étape très importante pour garantir la couverture et la complétude de notre langage. L'analyse du domaine de travail permet d'extraire les différentes facettes du contrôle-commande, les différents concepts et règles du domaine. Deux types d'analyse ont été réalisées : l'analyse documentaire du domaine et l'analyse structurelle du domaine.

5.1.1 Analyse documentaire du domaine

L'analyse documentaire du domaine repose sur la lecture de la documentation technique des systèmes contrôle-commande, et des réunions avec les experts. Lors de cette analyse, toutes les facettes (architecture physique, comportement, exigences, interactions avec les utilisateurs) des systèmes SCADA ont été soigneusement étudiées. Notre analyse de l'architecture physique a pour objectif de comprendre le fonctionnement, la nature, la typologie, les caractéristiques physiques de tous les composants d'un système contrôle-commande. Cette analyse a révélé l'utilisation de deux normes (ISA5, ISO 1219) pour la conception des P&ID. Ces normes définissent le contenu, la forme et les règles de nommage des P&ID. Afin de permettre une plus large utilisation de notre langage pivot, nous avons combiné ces deux normes. La difficulté rencontrée lors de cette combinaison, porte essentiellement sur les différences entre les notations utilisées. Les échanges avec des experts du domaine et la consultation des ouvrages, la documentation et les recherches bibliographiques, nous ont permis de définir les différents composants avec leurs caractéristiques (ports, propriétés de fonctionnement, etc.). Cela nous a permis de combiner tous les composants de la même famille et de généraliser leurs points communs.

L'architecture logique ne manque pas de difficulté par rapport à l'architecture physique. En effet, la réalisation des objectifs du système est faite suivant un processus fonctionnel complexe. Cette complexité réside dans la manière d'ordonner les priorités d'exécution de ces fonctions, les contraintes métier à prendre en compte et les ressources partagées entre ces fonctions. La difficulté rencontrée ici réside dans l'harmonisation des différentes représentations des fonctions tout en les ordonnant, en les groupant, en les classifiant et en les caractérisant.

En ce qui concerne le comportement dynamique, nous l'avons réalisé dans le section 4 du chapitre 5. D'autre part, le système SCADA est soumis à de nombreuses exigences de domaine de différentes typologies, nous avons d'abord rassemblé ces exigences et produit des documents destinés à la description et à la classification de ces exigences. La rédaction et la collecte de ces exigences a permis de mettre en clair les différentes typologies de ces exigences. En effet, les exigences sont de trois familles (fonctionnelles, non fonctionnelles et contraintes) et sont appliquées à des entités différentes du système qu'elles soient physiques/procédurales ou fonction globale/fonction élémentaire.

Une fois que les facettes constituantes du système ont été analysées, nous avons abordé l'analyse des utilisateurs finaux qui interagissent avec ce système. Pour la compréhension des types de tâches utilisateurs et des informations qui les concernent, nous avons fait référence à des travaux

antérieurs (Bignon, 2012a; Goubali, 2017; Kesraoui, 2017; Prat et al., 2015) et les avons complétés par des travaux de la littérature.

5.1.2 Analyse structurelle du domaine

L'analyse documentaire du domaine de chaque facette a permis de récolter les données nécessaires. Cependant, ces données doivent être structurées d'une manière qui permet de les harmoniser afin de représenter le flux comportemental des différentes entités du système. De fait, nous avons procédé à la structuration et à l'harmonisation de ces données dans un format cohérent avec les différents niveaux du contrôle-commande. Cette structuration s'appuie à la fois sur les résultats de la gestion des modes suivant le modèle WDA et sur le modèle de référence proposé. L'analyse structurelle du domaine de l'architecture physique a été complétée par des propriétés physiques des composants du système SCADA. En effet, chaque composant, selon sa famille, possède des propriétés de fonctionnement qui permettent son exploitation. Nous avons récolté ces propriétés en tenant compte de tous les états des composants. La difficulté rencontrée ici concerne la schématisation de tous les composants dans les différentes manières d'exploitation, sans perte d'information. Afin de centraliser toutes ces informations, nous avons réalisé des documents techniques détaillant l'exploitation de ces propriétés.

Lors de l'analyse de l'architecture logique, nous avons classifié, harmonisé, lié les différentes fonctions selon différents niveaux d'abstraction, afin de représenter les informations de façon pertinente. Nous avons caractérisé 5 niveaux d'abstraction pour les fonctions. Chaque niveau d'abstraction permet de caractériser des fonctions particulières et d'illustrer les relations, les ressources partagées, les contraintes métier pour l'exploitation de ces fonctions. La schématisation/classification a pris plusieurs itérations et réunions de travail.

En ce qui concerne les exigences, nous avons réalisé une première classification basée sur les travaux de la littérature (fonctionnelle, non fonctionnelle et contraintes). Après la mise en œuvre de l'analyse structurelle du domaine nous avons constaté que les exigences du contrôle-commande étaient aussi liées aux niveaux de décomposition du système (système, fonction, composant) et aux niveaux d'abstraction des fonctions. Pour pouvoir proposer une classification qui convient aux systèmes SCADA, nous avons combiné les deux classifications et proposé une classification qui prend en compte le *type d'exigence* (fonctionnelle, non fonctionnelle et contrainte) et le *niveau d'abstraction/décomposition de l'entité*.

L'analyse du domaine portant sur les utilisateurs finaux des systèmes SCADA a quant à elle, été complétée par des informations issues du domaine. En effet, nous avons constaté lors de l'analyse structurelle que les utilisateurs étaient soumis à une politique de gestion des droits d'accès pour l'exploitation des fonctions dans des modes de fonctionnement bien précis. La récolte de ces informations était guidée par la schématisation qui illustre bien ces informations.

5.1.3 Analyse de l'architecture physique

Dans cette partie, nous avons analysé les éléments constitutifs de l'architecture physique d'un système SCADA, incluant les composants et leurs propriétés, les connecteurs et les règles métiers permettant de relier les composants les uns aux autres. Ces informations sont indispensables pour la création de notre langage pivot.

L'analyse des P&ID et des cahiers des charges des systèmes industriels nous a permis d'identifier chaque composant dans son contexte (pour chacun des deux systèmes utilisés), ses propriétés (contraintes, exigences, etc.) et les contraintes permettant de les relier. Toutes ces informations nous ont permis de créer des familles de composants sous forme d'une nomenclature standard

des différents composants des systèmes SCADA. Cette nomenclature est destinée à être réutilisable pour la conception de tout système SCADA industriel.

5.1.4 Analyse de l'architecture logique

Le but de cette partie est d'identifier les composantes de l'architecture logique d'un système SCADA. La documentation technique des spécifications fonctionnelles des systèmes étudiés a aussi mis en évidence que l'exploitation des fonctions des systèmes SCADA est souvent réalisée dans des modes de fonctionnement bien précis. Pour cela, nous exploitons la décomposition des fonctions présentée dans le paragraphe 5.1 du chapitre 5. Cette décomposition s'appuie sur les niveaux d'abstraction de la WDA avec une intégration des modes et des contraintes. Cette analyse nous a permis de proposer une taxonomie des fonctions classifiées.

5.1.5 Analyse du comportement

Dans cette partie, nous avons exploité le modèle de comportement issu des travaux sur la gestion des modes. Le modèle de comportement permet de caractériser l'état de l'aspect physique et procédural du système à n'importe quel instant. En effet, ce modèle renferme toutes les informations détaillées et contextuelles sur les familles de modes, les modes et les états. Cela permet de prendre en compte le comportement des systèmes dans leur globalité, sans perte d'information.

5.1.6 Analyse des exigences

Les systèmes industriels étudiés sont soumis à des exigences de leur domaine. Ces exigences sont différentes par nature et par niveau d'application. Pour récupérer ces exigences, nous avons étudié la documentation technique des deux systèmes. L'étude de la documentation technique nous a permis d'identifier les trois types d'exigences : fonctionnelles, non fonctionnelles et contraintes. Nous avons aussi constaté que chaque exigence est appliquée à un niveau d'abstraction différent (système, fonction, composant). Nous avons alors classé toutes les exigences des systèmes industriels.

5.1.7 Structuration des données

Dans cette partie nous avons utilisé le modèle conceptuel de modes et de contraintes pour structurer les données issues des différentes analyses décrites dans ces travaux. Cette structuration nous a permis d'identifier le type et le format de données à intégrer dans le langage pivot. En effet, les niveaux d'abstraction du modèle nous ont permis d'intégrer les fonctions, les contraintes et le comportement dans le langage pivot selon quatre niveaux d'abstraction. De plus, les niveaux de décomposition du modèle nous ont permis, quant à eux, d'établir les liens entre les différentes facettes par exemple : les fonctions physiques de la facette architecture logique ont été reliées aux composants élémentaires de la facette architecture physique.

5.2 Etape 2 : Création de la syntaxe abstraite statique (DDMM)

La deuxième étape de notre méthodologie consiste à développer la syntaxe abstraite du langage pivot, sous la forme d'un méta-modèle. Dans cette étape, nous avons utilisé les données récoltées et structurées issues de l'étape précédente et avons créé une syntaxe abstraite sous forme de classes et de liaisons entre les classes.

Tout d'abord, le méta-modèle de chaque facette a été créé à part. La création du méta-modèle a nécessité la modélisation de tous les concepts du domaine de cette facette. En effet, il fallait éviter la redondance et en même temps éviter les oublis. De plus, les règles du domaine ont été ajoutées pour chaque classe sous forme de contraintes OCL. Le méta-modèle obtenu pour chaque facette a

été testé sur des cas concrets. Les résultats de ces tests nous ont permis de corriger les manques/incohérences du méta-modèle.

Garantir les liens entre les différentes facettes nécessite une bonne réflexion. En effet, il fallait tout d'abord relier le méta-modèle de l'architecture physique à celui de l'architecture logique. Nous avons utilisé des liens (associations) entre la classe P&ID et la classe mère Fonction. Après l'exécution de différents tests, nous avons constaté que le méta-modèle ne permet de préciser que les fonctions de niveau physique qui sont réalisées par des composants physiques. De fait, nous avons réalisé des liaisons selon les niveaux d'abstraction de telle sorte que chaque fonction d'un niveau d'abstraction ait une liaison à l'entité physique de même niveau. Le méta-modèle résultat a été lié au méta-modèle des facettes « exigences », « comportement », « modèles de tâches » et « modèles utilisateurs ».

5.2.1 Proposition d'une syntaxe abstraite de l'architecture physique

Le but ici est de créer un méta-modèle capable de représenter les composants d'un système industriel ainsi que leurs connexions. La Figure 50 illustre la racine de la partie architecture physique. La classe principale, PID, contient toutes les autres classes. Elle possède quelques attributs repris des P&ID existants pour permettre une intégration à un système de gestion de documents. Les cardinalités des compositions partant de la classe principale ont volontairement une valeur minimum à zéro pour qu'un P&ID vide soit considéré comme valide. Le P&ID contient des composants complexes ou élémentaires. Chaque composant a des ports et est relié en utilisant ses ports à un autre composant par des liens (link). Les liens sont des tuyaux. Chaque composant de l'architecture physique possède des propriétés de domaine particulières, par exemple la pression, le débit, etc. La difficulté rencontrée est de pouvoir proposer pour chaque propriété une valeur et une unité. Nous avons également proposé 5 types de données et 13 énumérations (Figure 52).

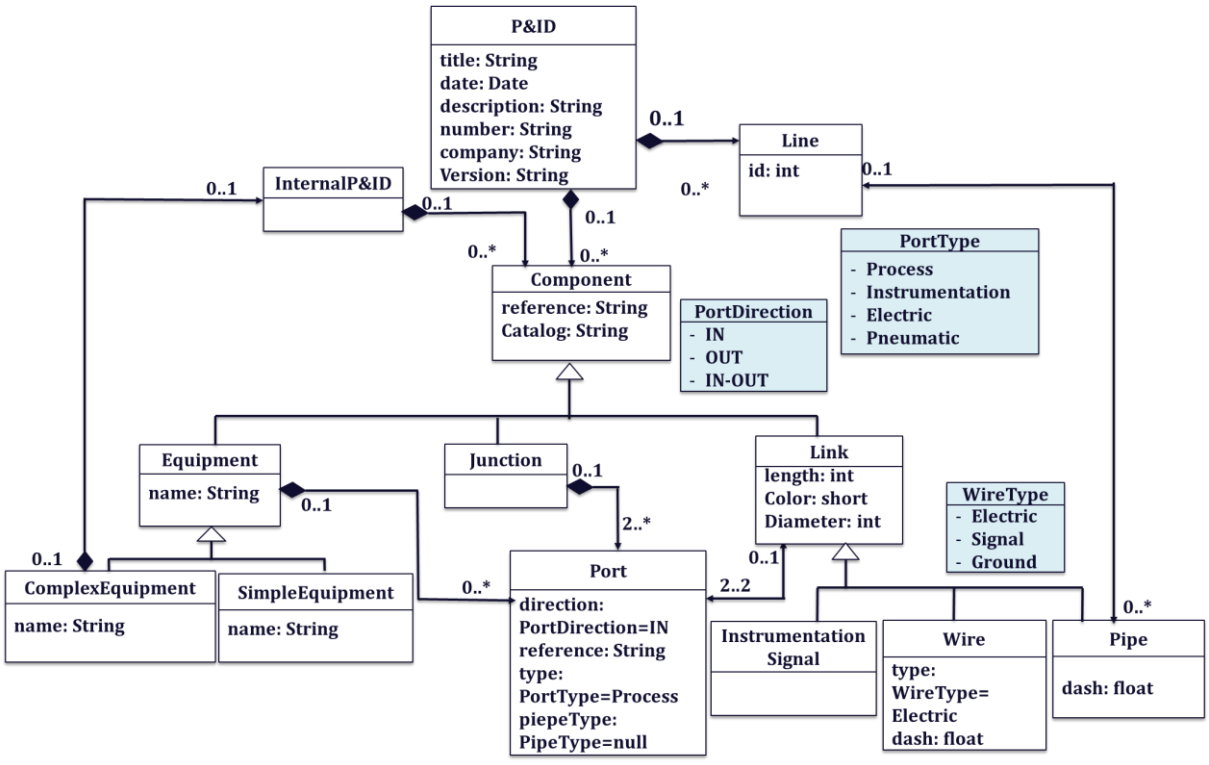


Figure 50 : Syntaxe abstraite de l'architecture physique proposée

La syntaxe abstraite de l'architecture physique proposée contient un nombre important de composants, de propriétés de composants, de contraintes OCL sur les ports, de liaisons, etc.

```
context ISAValve2Way

inv ports:
      self.port.type->count(metaModelArchitecture::PortType::Process) = 2
```

Figure 51 :ISA Valve 2 voies avec sa contrainte OCL de vanne.

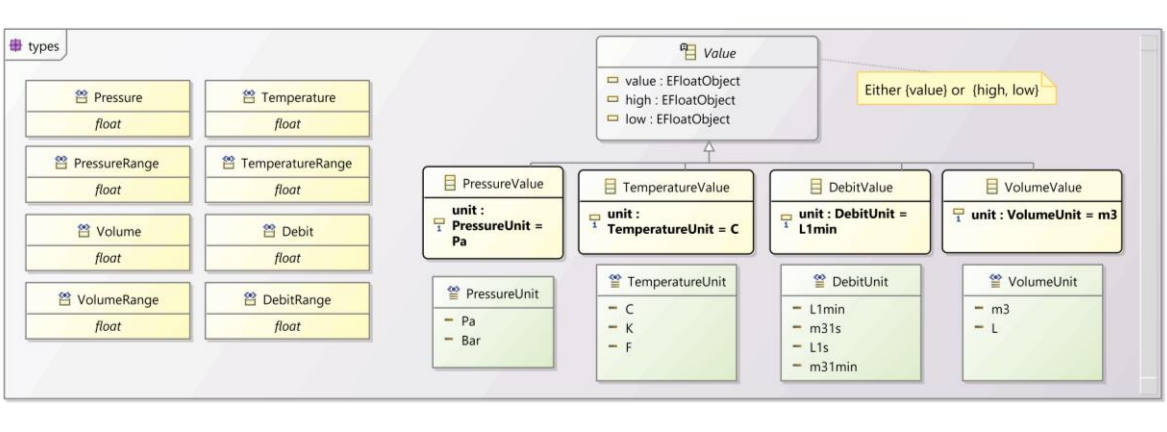


Figure 52 : Type de données proposé (attribut, valeur, unité) pour les équipements

5.2.2 Proposition d'une syntaxe abstraite de l'architecture logique

La deuxième partie est la définition de l'architecture logique, permettant la modélisation des fonctions du système (Figure 53). Ces dernières sont sur plusieurs niveaux d'abstraction, inspirées de la WDA.

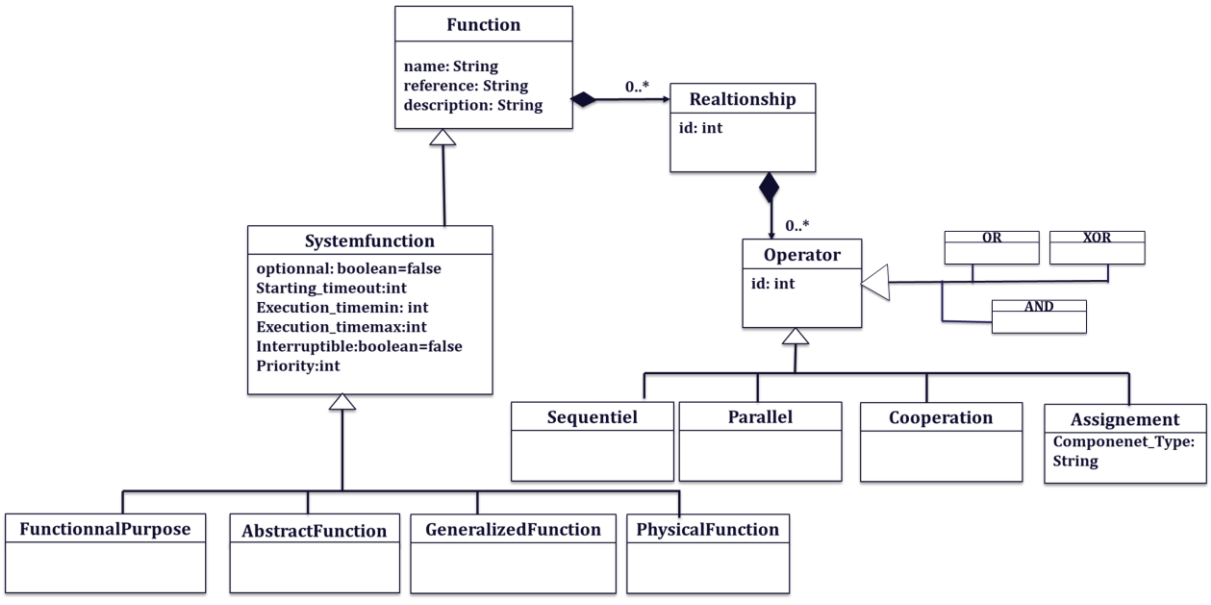


Figure 53 : Syntaxe abstraite pour la partie logique.

Le premier niveau représente l'objectif fonctionnel des systèmes SCADA (par exemple, « distribuer de l'eau douce »). Le deuxième niveau représente les fonctions nécessaires à la réalisation de l'objectif fonctionnel en utilisant les fonctionnalités de l'architecture physique choisie. Par exemple, la fonction « *Transfert* » est nécessaire pour distribuer l'eau douce car il y a

plusieurs réservoirs d'eau douce dans la solution choisie. Le troisième niveau représente les étapes intermédiaires pour la réalisation des fonctions abstraites. Ce sont les fonctions qui sont directement reliées aux composants, par exemple « Ouvrir les vannes ». Le quatrième niveau représente les fonctions des composants physiques par exemple, une vanne fait circuler l'huile d'un circuit hydraulique.

D'autre part, les fonctions possèdent des *paramètres*. Les fonctions peuvent posséder une *précondition* qui permet d'indiquer des prérequis pour leur exécution. Elles peuvent posséder une *post-condition* qui permet de vérifier que la fonction a bien eu l'impact escompté sur le système. Enfin, elles peuvent avoir une *condition d'arrêt*, qui permet de provoquer l'arrêt de la fonction.

Nous avons aussi modélisé les relations entre fonctions par des opérateurs tels que le AND, OR, SAND. .

```
context SystemFunction
      inv ExecutionTimeMinMax:
            self.execution_time_min < self.execution_time_max
```

Figure 54 : Contrainte OCL intervalle d'exécution d'une fonction

5.2.3 Proposition d'une syntaxe abstraite du comportement

La modélisation du comportement (Figure 55) permet de répondre au besoin informationnel sur l'état d'une entité physique ou procédurale à n'importe quel instant. Etant donné que les familles de modes, les modes et les états sont définis à l'avance dans de précédents travaux, nous avons utilisé les résultats de ces travaux pour modéliser la facette du comportement dynamique.

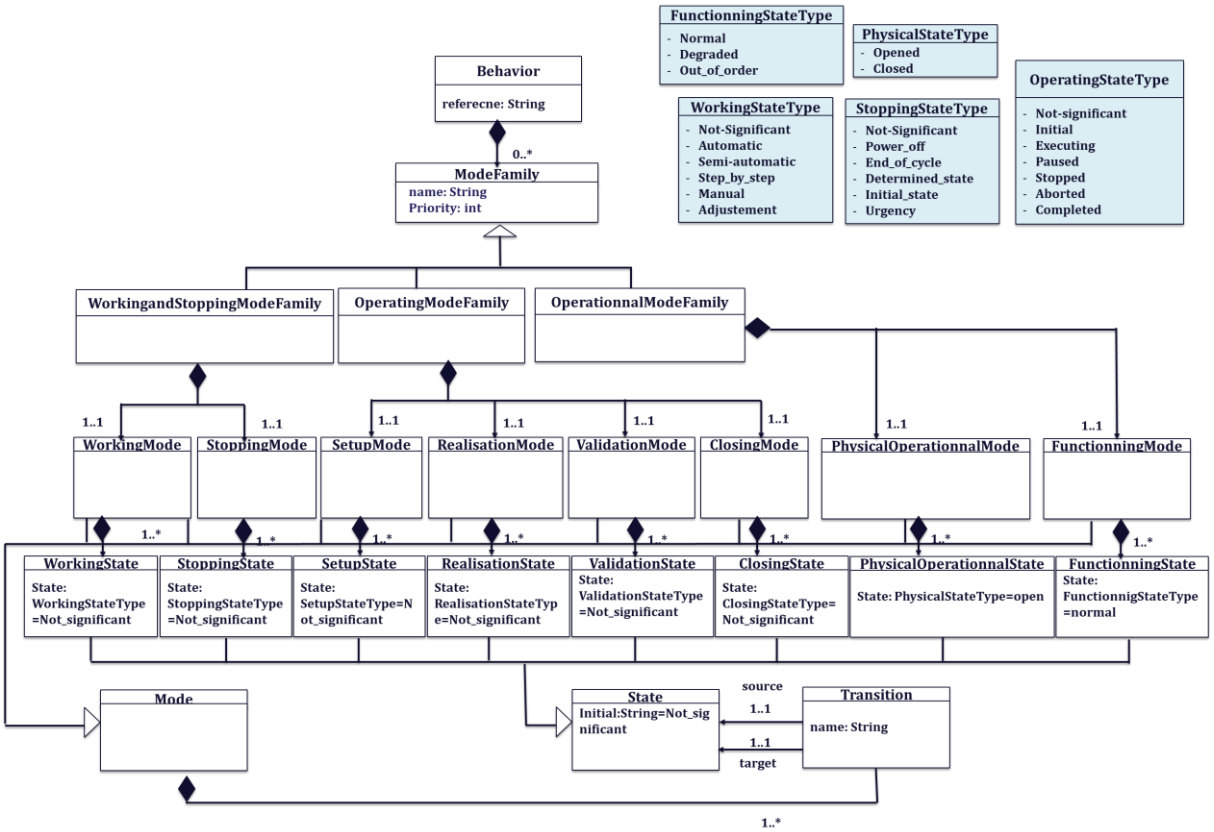


Figure 55 : Syntaxe abstraite pour la partie Comportement

La facette du comportement dynamique se compose de classes où chacune représente une famille de modes (par exemple la famille de modes Opérationnels). Chaque famille de modes se compose d'une ou plusieurs classes représentant ses modes. Chaque classe de mode est caractérisée par des états qui sont de plusieurs type.

```
context PhysicalOperationalMode
      inv atMostOneOfEachState:
            self.physicaloperationalstate->forAll(s1, s2 | s1 <> s2 implies
s1.state <> s2.state)
```

Figure 56 : Contrainte OCL pour les états du mode Opérationnel physique.

5.2.4 Proposition d'une syntaxe abstraite des contraintes

La modélisation des exigences proposée suit plusieurs niveaux d'abstractions (Figure 57). Une exigence est caractérisée par sa référence, son type (fonctionnel, non fonctionnel, contrainte), et au niveau d'abstraction/décomposition auquel elle est liée. La difficulté réside lors de la spécification de ces exigences par un expert du domaine d'une manière textuelle et en langage naturel. En effet, il a fallu chercher et trouver un moyen pour pouvoir assurer la cohérence des exigences spécifiées. Nous avons alors ajouté à la syntaxe abstraite des exigences, des classes qui permettent de décrire les exigences en utilisant un dictionnaire de données.

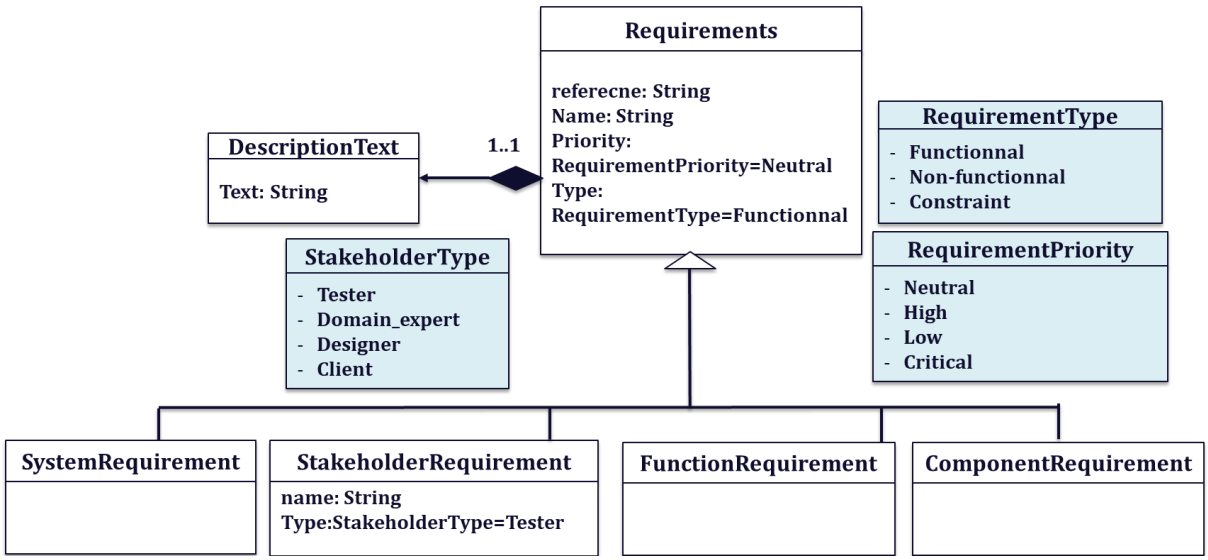


Figure 57 : Syntaxe abstraite partie Exigences.

5.2.5 Proposition de la syntaxe abstraite globale du xDSML

Toutes les facettes du système ont été liées et harmonisées dans une seule syntaxe abstraite qui représente le langage pivot proposé. En effet, la Figure 58 illustre les classes racines et les liaisons entre les différentes syntaxes abstraites des différentes facettes dans une vue *de* haut-niveau. En effet, le système représenté par la classe « System », possède une architecture. L'architecture système peut être une architecture physique « PhysicalArchitecture », qui est la racine de la syntaxe abstraite de l'architecture physique (présentée dans 5.2.1). Le système possède également une architecture logique « LogicalArchitetcure », qui est la racine de la syntaxe abstraite de la facette architecture logique (5.2.2). La classe « Behavior » représente le comportement dynamique du système et celui de ses différentes entités (physiques, logiques), modélisées dans 5.2.3. Les contraintes du système sont modélisées dans la facette « Requirements » (5.2.4).

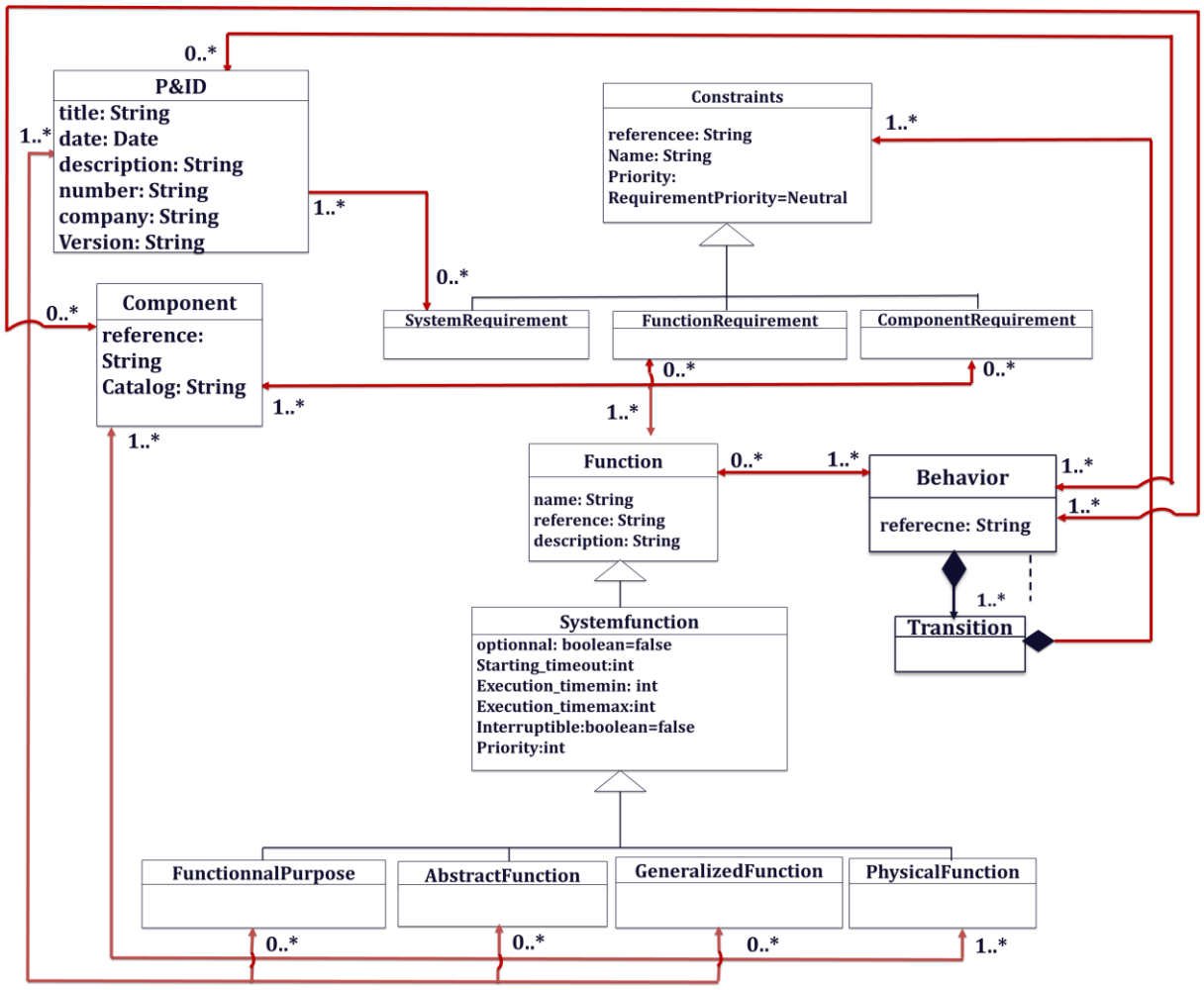


Figure 58 : Extrait de la syntaxe abstraite globale : liaisons de toutes les facettes du système SCADA dans une vue de haut-niveau

L'analyse du domaine nous a permis de collecter tous les concepts et toutes les terminologies pour la création de la syntaxe abstraite (méta-modèle). Le méta-modèle du langage pivot proposé englobe 179 classes, 47 énumérations et 226 contraintes OCL.

- L'architecture physique est composée de 69 classes et de 196 contraintes OCL.
- L'architecture logique est composée de 24 classes et de 10 contraintes OCL.
- La partie contrainte est composée de 14 classes et de 1 contrainte OCL.
- La partie comportement est composée de 30 classes et de 15 contraintes OCL.
- Le modèle de taches est composé de 30 classes et de 4 contraintes OCL.
- Le modèle de l'utilisateur est composé de 10 classes.
- La configuration système est définie par 2 classes.
- Le méta-modèle contient également 47 énumérations

5.3 Etape 3 : Création de la syntaxe abstraite dynamique (trace DTMM)

La trace (Figure 59) se compose de la séquence d'ordonnancement sous forme d'ordres de boucle (Loop). Chaque ordre de Loop a deux descriptions. *La première est abstraite*, définit l'ensemble des composants d'une boucle. En effet, chaque boucle démarre d'un composant *Source* qui se charge de la génération du fluide hydraulique, des composants *WayPoint* qui transfèrent l'huile d'un point A à un point B, d'un composant *Target* qui est le composant visé par cette boucle et un composant *Well* où la boucle s'arrête. *La deuxième description est concrète*, elle décrit les états

opérationnels nécessaires de chaque composant et fonction. Une loop se caractérise par sa durée (*Duration)*, son nom (*name)* et le temps d'exécution écoulé (*TimeFlushingDone)*.

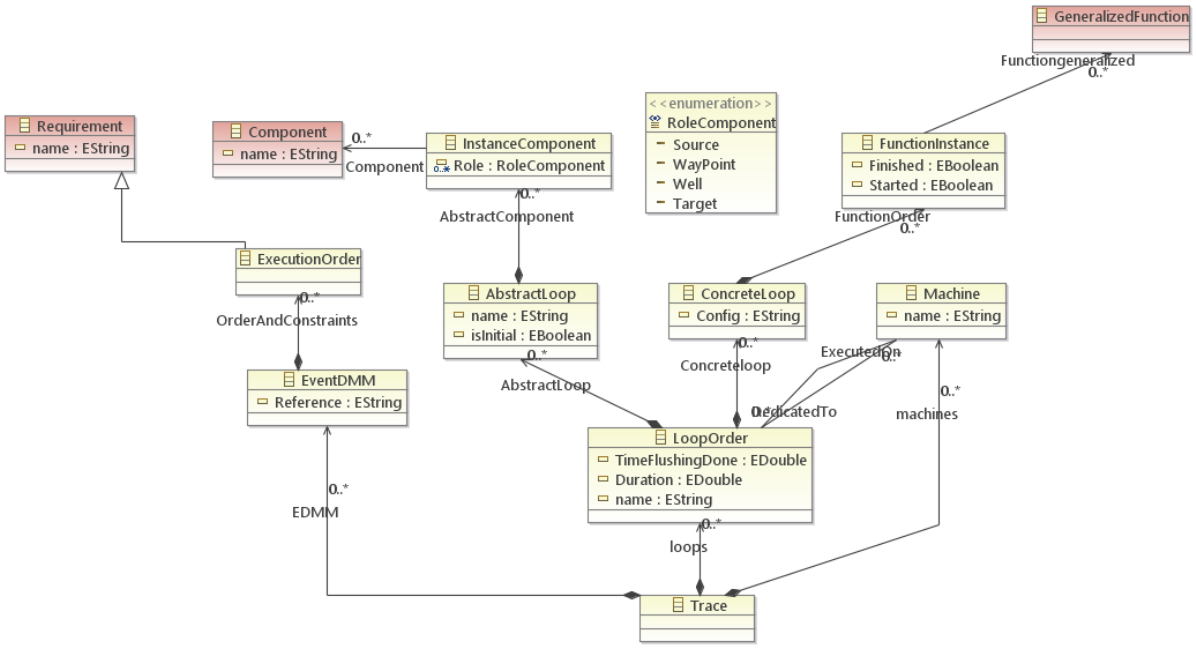


Figure 59 : Le méta-modèle de la trace (DTMM).

La classe *InstanceComponent* a un attribut *Role* (*source, WayPoint, Well, Target*). Elle permet d'étendre la classe *Component* avec le rôle du composant dans une loop donnée. La classe *FunctionInstance* vient aussi pour étendre la classe *GeneralizedFunction* par les informations d'exécution (*Finished, Started*).

Les évènements qui permettent d'exécuter une fonction ou manipuler un composant sont décrits par la classe *EventDMM* et *ExecutionOrder* qui hérite de *Requirement*.

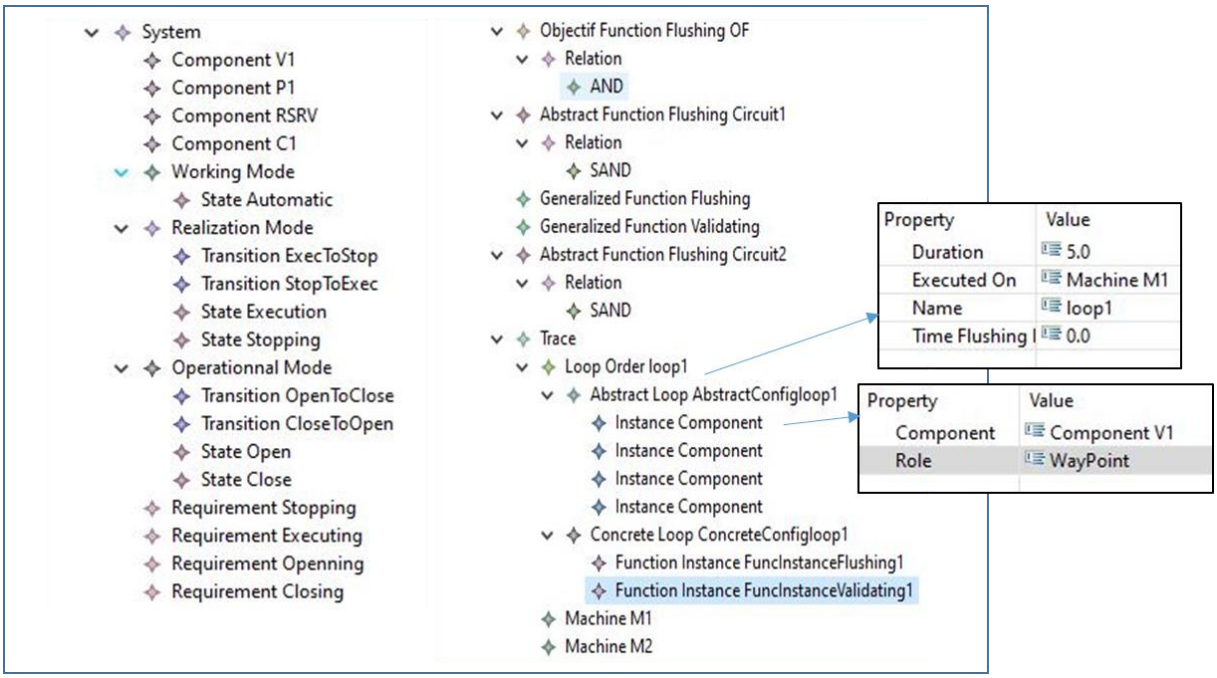


Figure 60 : Exemple d'instance

5.4 Etape 4 : La sémantique dynamique

La sémantique dynamique contient les règles qui assurent la cohérence et la validation du comportement. Ces règles ont été illustrées dans la partie (la section 4.3 du chapitre 6).

5.5 Etape 5 : Création de la syntaxe concrète du xDSML

5.5.1 Syntaxe concrète standard

A partir des différentes syntaxes abstraites, nous avons créé des instances avec l'outil EMF. Les instances EMF sont des syntaxes concrètes qui permettent de tester les méta-modèles. Cependant, ces instances ne sont pas compréhensibles par des experts du domaine car leur manipulation nécessite des compétences en méta-modélisation.

5.5.2 Création d'une syntaxe concrète conforme aux habitudes des experts

Le méta-modèle proposé permet la création des instances dynamiques (modèle). Cependant, une syntaxe concrète standard est difficile à utiliser par les experts du domaine.

Nous avons choisi de développer un logiciel pour se rapprocher du domaine et fournir une syntaxe concrète facile à manipuler par un expert et concentrer ainsi ses efforts sur la spécification. L'objectif est d'intégrer des techniques qui permettent à un expert non-programmeur d'écrire les spécifications sans qu'il soit obligé d'apprendre un nouveau langage. Nous avons choisi d'utiliser les techniques de l'EUD car ces dernières ont prouvé leur utilité dans la spécification fonctionnelle (Goubali & Goubali, 2017) des systèmes SCADA.

Pour la conception de ce logiciel, nous avons utilisé le diagramme de classe. Nous avons également choisi le modèle MVC[9] car le logiciel, par ses différents programmes, interagit avec des fichiers de données issus de la syntaxe abstraite du langage pivot et des interfaces à afficher à l'utilisateur. La Figure 61 illustre la structure du logiciel EUD. La spécification du système SCADA en utilisant ce logiciel est sur trois étapes. La première (zone (1) sur la Figure 61) permet la création du P&ID. Dans cette étape, l'expert sélectionne des composants de la bibliothèque (zone (4) sur la Figure 61) et les fait glisser dans la zone de dessin (zone (5) sur la Figure 61). Les propriétés des composants sont renseignées au moment de leur sélection. La liaison entre les composants instanciés est faite par des tuyaux (qu'il a aussi la possibilité de sélectionner de la librairie). Relier un composant à un autre doit être conforme aux contraintes OCL (imposées par le domaine) intégrées au méta-modèle. Avec notre outil, l'utilisation des contraintes OCL est complètement transparente pour les experts métier.

La deuxième étape (zone (2)) permet à l'expert de spécifier les fonctions, les exigences et le comportement. La spécification des fonctions passe par deux étapes. La première permet de choisir la configuration du composant. Un composant de l'architecture physique déjà créé (P&ID) joue un rôle (source, puit, goal, waypoints). L'opérateur choisit pour chaque composant le (les) rôle(s) adéquat(s). *La* deuxième *étape* est la spécification de toutes les fonctions du système. La spécification des fonctions est faite selon les niveaux d'abstraction de ces dernières. Ici l'opérateur peut spécifier l'architecture logique indépendamment de son métier. D'autre part, les modes proposés sont utilisés pour mieux exploiter les fonctions du système. En effet, pour chaque fonction l'expert doit sélectionner le mode approprié dans lequel elle s'exécute. La spécification du comportement dynamique en utilisant les modes, permet de créer un modèle du comportement (à partir des spécifications modes, des fonctions, des contraintes, de l'architecture

[9] Modèle Vue Contrôleur

physique). La spécification des exigences quant à elle, est faite en langage naturel. Cependant, afin de limiter les erreurs et assurer l'enregistrement des exigences bien formulées par l'expert, nous avons choisi d'utiliser un dictionnaire de spécification des contraintes pour les systèmes industriels SBVR (*Semantics of Business Vocabulary and Business Rules*). Nous avons enregistré dans le logiciel les mots clés de SBVR. Cela nous a permis de les proposer à l'expert sous forme de listes de mots à sélectionner pour formuler les exigences. L'utilisation des mots clés facilite d'une part , la spécification des exigences pour l'expert (écriture des phrases) et d'autre part, l'enregistrement des exigences correctes.

Dans la troisième étape (zone (3) sur la Figure 61) du logiciel proposé, permet de simuler l'exécution des séquences de commandes (boucles) .L 'expert visualisera, corrigera et validera le résultat de ses spécifications.

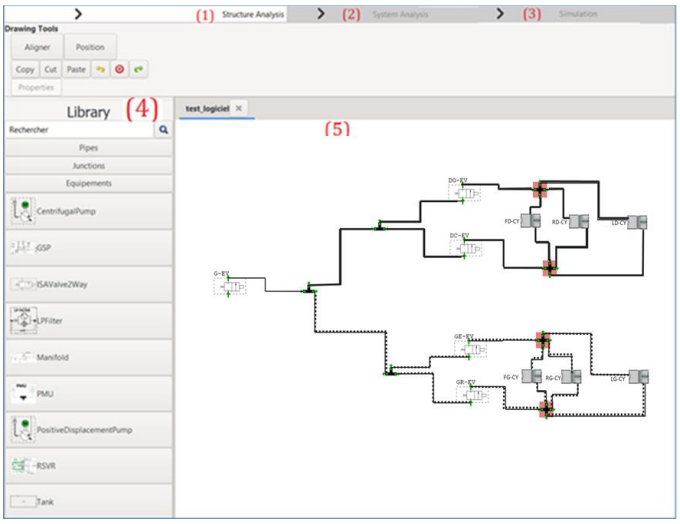


Figure 61 : Structure du logiciel de spécification multi-facettes

6 Conclusion

Un xDSML est un langage qui joue le rôle d'intermédiaire entre différents langages utilisés dans la conception des systèmes de type SCADA. Son objectif est de faciliter la spécification des systèmes SCADA. L'utilisation de notre langage dédié permet de garantir une plus grande homogénéité des informations, puisqu'elles sont toutes exprimées dans le même langage.

Le langage proposé se compose d'une interface de spécification construite autour de la syntaxe abstraite du langage pivot. L'interface de spécification offre la possibilité aux experts métier de construire une maquette du système (architecture ou P&ID) à partir d'une bibliothèque de composants prédéfinis, puis de spécifier les différentes facettes du système (contraintes, fonctions, comportement), d'intégrer l'ordonnancement des boucles et de simuler les séquences d'actions. En résumé, notre xDSML se compose de cinq composants :

1. la syntaxe abstraite statique (DDMM), qui contient les concepts du domaine : architecture physique, architecture logique, contraintes et comportement ;
2. la sémantique statique est ajoutée par des contraintes OCL ;
3. la syntaxe abstraite dynamique qui contient les concepts qui permettent d'étendre les concepts du DDMM et qui sont susceptible d'avoir un changement d'états (composants et fonctions) lors de l'exécution du DSL ;
4. la sémantique opérationnelle (dynamique) contient les règles d'exécution et changement d'états ;

5. la syntaxe concrète permet de créer un P&ID à partir d'une bibliothèque de composants, créer l'architecture logique, spécifier les contraintes et comportement, spécifier les boucles et de les simuler.

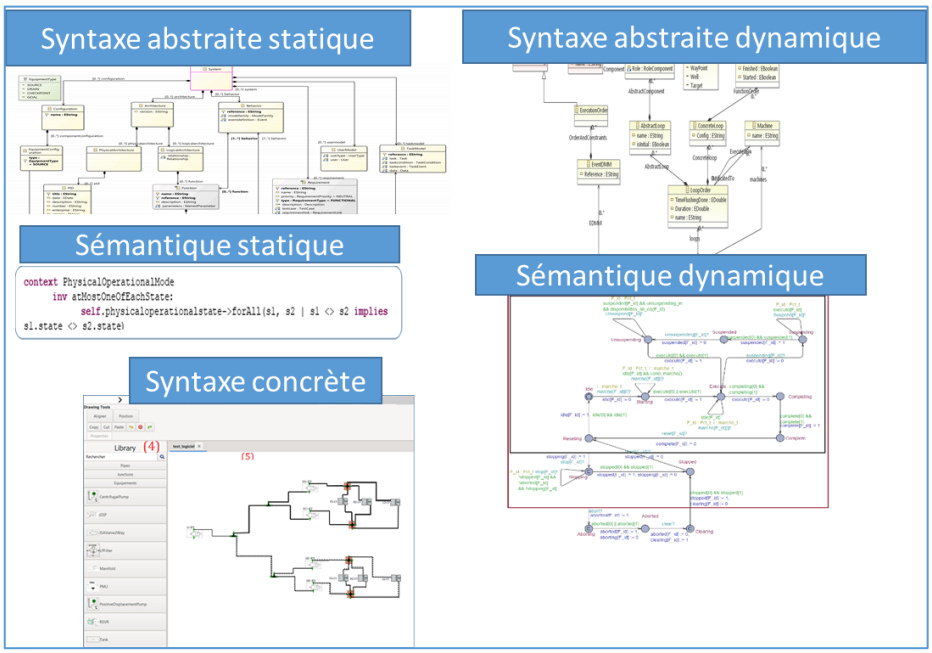


Figure 62 : Architecture du xDSML proposé.

Chapitre 8 : Conclusion et perspectives

1 Introduction

Les travaux présentés dans cette thèse ont permis de répondre à une problématique industrielle. En effet, pour réaliser des tests sur des systèmes hydrauliques, les testeurs utilisent des bancs de test SCADA. Ces derniers permettent de superviser et de contrôler le procédé à travers des ordres de commande. Trois problématiques de ce domaine :

- L'ordonnancement manuel des tests abstraits est très fastidieux et nécessite beaucoup d'efforts pour respecter les contraintes de précédences et des ressources partagées.
- L'ordre d'exécution de ces tests est défini manuellement par les experts du domaine, ce qui peut être une source d'erreurs. Des contraintes techniques et opérationnelles doivent être respectées pour assurer l'efficacité des tests et la sécurité des personnes et des installations.
- La spécification des informations de test est réalisée via des documents textuels. Chaque donnée est ensuite réécrite en langage spécifique puis les données sont utilisées pour générer les commandes. Cette combinaison nécessite d'homogénéiser les données.

Pour répondre à ces problématiques nous avons proposé trois contributions : l'ordonnancement de deux machines parallèles dédiées avec des contraintes de précédences et une ressource partagée, la génération automatique des séquences de commande SCADA opérationnelles et un xDSML pour faciliter la spécification et la simulation des séquences générées.

2 Rappel des contributions

Nous avons présenté et proposé dans cette thèse une approche qui permet de calculer un ordonnancement de tests, de générer les séquences de commandes pour l'exécution de ces tests selon l'ordonnancement et de spécifier les informations comme entrées pour l'ordonnancement et la génération de séquences afin de pouvoir simuler le résultat via des interfaces de supervision proche du réel.

2.1 Ordonnancement

Notre première contribution porte sur l'ordonnancement de deux machines parallèles dédiées avec une ressource partagée et des précédences locales. Un algorithme pour le problème $\boldsymbol{PD2|res1\cdot\cdot,localprec|Cmax}$ a été proposé. Cet algorithme est basé sur une formulation **disjonctive** pour représenter les précédences locales. La formulation disjonctive permet d'éviter la complexité des variables de décisions. Nous avons évalué l'algorithme selon plusieurs tailles d'instances et avons proposé le meilleur solveur pour ce genre d'algorithme. Vu la complexité du problème, nous avons également proposé des heuristiques pour le résoudre en temps polynomiale et pouvoir l'appliquer sur un nombre d'instances important.

2.2 Génération de séquences de commande SCADA

L'ordonnancement calculé peut s'avérer non-faisable opérationnellement si les contraintes opérationnelles ne sont pas respectées. Nous avons proposé une approche pour la génération de séquences de commande permettant l'exécution opérationnelle de l'ordonnancement calculé. Cette approche est basée sur une gestion des modes afin de spécifier les états et modes de chaque entité (composant, fonction, système), de modéliser le comportement de chaque entité à base de machine à états et de gérer les compatibilités entre les différents états d'une même entité et des entités entre elles. En effet, pour chaque entité des contraintes de compatibilités ont été ajoutées afin de garantir le respect des contraintes opérationnelles. Nous avons proposé un canevas qui permet de passer d'un ensemble de modes élémentaires à un modèle global opérationnel,

compatible et cohérent. Ce modèle a été notamment vérifié et validé formellement. Le modèle opérationnel est ensuite utilisé conjointement avec un modèle formel de l'ordonnancement calculé pour calculer automatiquement la séquences de commande opérationnelle permettant la réalisation de l'ordonnancement sur le modèle du système.

2.3 xDSML

Pour outiller les deux premières contributions, nous avons également proposé un xDSML. Ce dernier permet d'assister les opérateurs des systèmes SCADA dans le processus d'optimisation de leur système. Doté d'une syntaxe abstraite statique pour représenter le domaine, d'une syntaxe abstraite dynamique pour représenter les règles d'exécution et d'une syntaxe concrète fidèle aux domaine, notre xDSM permet d'une part de spécifier toutes les informations statiques du système SCADA telles que le P&ID, les fonctions et également la décomposition fonctionnelle. D'autre part, il permet la simulation les séquences de commandes et de valider les spécifications dans une étape antérieure à la conception. Le développement de ce xDSML est basé sur un modèle de référence SCADA. En effet, vu la complexité de tels systèmes, nous avons proposé un modèle de référence qui regroupe toutes les facettes d'un système SCADA ainsi que les liaisons entre ces facettes. Ce modèle de référence est une vraie aide pour combiner les données, structurer et homogénéiser la syntaxe abstraite statique du xDSML.

3 Perspectives

A l'issue de cette thèse, plusieurs perspectives ont émergé et qui tourne autour de l'optimisation des bancs de tests SCADA.

3.1 Calcule automatique des ordres de tests abstraits

Les travaux de cette thèse sont basés sur des tests abstraits déjà calculés. Il serait intéressant de proposer des approches qui permettent de calculer automatiquement ces tests abstraits. Par exemple, de convertir le P&ID en graphe (Bignon, 2012b) afin de pouvoir calculer les chemins des tests abstraits. Le calcul devra prendre en compte les contraintes de perte de charges, des longueurs de la boucle et le débit.

3.2 Ordonnancement stochastiques

Les durées des tests abstraits ont été introduites dans nos algorithmes d'ordonnancement comme des données connues, ce que veut dire que les tests ont été déjà réalisés. Toutefois, si un nouveau test est lancé, les durées seront inconnues. L'utilisation des algorithmes stochastiques dans ce cas peut être envisagée.

3.3 Intégration du mode dégradé

Dans la partie de génération de séquences et spécialement la gestion des modes, nous nous sommes concentrés sur le mode de fonctionnement normal, en supposant que tout marche bien. Il est nécessaire d'ajouter le mode dégradé afin de pouvoir garantir un changement de configuration en ligne, ce qui enclenchera également un recalcule d'ordonnancement en ligne.

3.4 Génération automatique des applicatifs de contrôle-commande

Notre xDSML comporte toutes les informations d'un banc de tests SCADA, il peut être utilisé pour générer les applicatifs de contrôle-commande.

.

Références Bibliographiques

14:00-17:00. (s. d.). *ISO 16431:2012*. ISO. Consulté 12 septembre 2023, à l'adresse https://www.iso.org/standard/50158.html

Abrahão, S., Insfran, E., Sluÿters, A., & Vanderdonckt, J. (2021). Model-based intelligent user interface adaptation : Challenges and future directions. *Software and Systems Modeling*, *20*(5), 1335-1349.

Adawiyah, R., Malida, T. R., & Muda, I. (2021). *The role of the Microsoft office Visio in producing accounting information*. https://www.academia.edu/download/65822748/60059CFD208C91610980605.pdf

ADEPA. (1981). *Le GEMMA: Le Guide d'Étude des Modes de Marche et Arrêt*.

Aggoune, R. (2002). *Ordonnancement d'ateliers sous contraintes de disponibilité des machines* [PhD Thesis]. Université Paul Verlaine-Metz.

Albeaik, S., Bayen, A., Chiri, M. T., Gong, X., Hayat, A., Kardous, N., Keimer, A., McQuade, S. T., Piccoli, B., & You, Y. (2022). Limitations and Improvements of the Intelligent Driver Model (IDM). *SIAM Journal on Applied Dynamical Systems*, *21*(3), 1862-1892. https://doi.org/10.1137/21M1406477

Almonte, L., Guerra, E., Cantador, I., & De Lara, J. (2022). Recommender systems in model-driven engineering : A systematic mapping review. *Software and Systems Modeling*, *21*(1), 249-280. https://doi.org/10.1007/s10270-021-00905-x

Anokhin, A., Ivkin, A., & Dorokhovich, S. (2018). Application of ecological interface design in nuclear power plant (NPP) operator support system. *Nuclear Engineering and Technology*, *50*(4), 619-626.

Arrêt d'urgence (automatique). (2020). In *Wikipédia*. https://fr.wikipedia.org/w/index.php?title=Arr%C3%AAt_d%27urgence_(automatique)&oldid=175833364

Audibert, L. (2007). Base de Données et langage SQL. *Developpez. com.[En ligne]. Disponible sur: http://laurent-audibert. developpez. com/Cours-BD/.[Consulté le: 06-août-2018]*. https://www.academia.edu/download/50923556/Cours-BD.pdf

Baker, K. R., & Trietsch, D. (2013). *Principles of sequencing and scheduling*. John Wiley & Sons.

Barbieri, G., Battilani, N., & Fantuzzi, C. (2015). A PackML-based design pattern for modular PLC code. *IFAC-PapersOnLine*, *48*(10), 178-183. https://www.sciencedirect.com/science/article/pii/S2405896315009957

Batteux, M., Prosvirnova, T., & Rauzy, A. (2018a). From models of structures to structures of models. *2018 IEEE International Systems Engineering Symposium (ISSE)*, 1-7. https://ieeexplore.ieee.org/abstract/document/8544424/

Batteux, M., Prosvirnova, T., & Rauzy, A. (2018b). Schemas de modelisation de politiques de maintenance de composants en Altarica 3.0. *Congrès Lambda Mu 21,«Maîtrise des risques et transformation numérique: opportunités et menaces»*. https://hal.science/hal-02063641/

Batteux, M., Prosvirnova, T., & Rauzy, A. (2020). Modélisation de combinaisons de maintenances en AltaRica 3.0. *Congrès Lambda Mu 22 «Les risques au cœur des transitions»(e-congrès)-22e Congrès de Maîtrise des Risques et de Sûreté de Fonctionnement, Institut pour la Maîtrise des Risques*. https://hal.science/hal-03462797/

Batteux, M., Prosvirnova, T., & Rauzy, A. B. (2019). AltaRica 3.0 in ten modelling patterns. *International Journal of Critical Computer-Based Systems*, *9*(1/2), 133. https://doi.org/10.1504/IJCCBS.2019.098809

BATTEUX, M., SystemX, I. R. T., PROSVIRNOVA, T., & RAUZY, A. (s. d.). *Modélisation de combinaisons de maintenances en AltaRica 3.0 Modeling combination of maintenance policies with AltaRica 3.0*. Consulté 10 octobre 2023, à l'adresse https://hal.science/hal-03462797v1/preview/LM22_COM_FULL_482836___20200709_.pdf

Bellman, R., & Kalaba, R. (1957). Dynamic programming and statistical communication theory. *Proceedings of the National Academy of Sciences of the United States of America*, *43*(8), 749.

Belloir, N., Bruel, J.-M., & Faudou, R. (2014). Modélisation des exigences en UML/SysML. *Revue Génie Logiciel*, *111*, 6-12. https://hal.science/hal-01085292/

Berruet, P. (1998). *Contribution au recouvrement des systèmes flexibles de production manufacturière : Analyse de la tolérance et reconfiguration* [PhD Thesis]. Lille 1.

Berruet, P., Toguyeni, A. K. A., Elkhattabi, S., and Craye, E. (1999). *Berruet, P., Toguyeni, A. K. A., Elkhattabi, S., and Craye, E. (1999). Tolerance evaluation of flexible manufacturing architectures. Journal of Intelligent Manufacturing, 10(6) :471–484.*

Bignon, A. (2012a). *Génération conjointe de commandes et d ' interfaces de supervision pour systèmes sociotechniques reconfigurables*.

Bignon, A. (2012b). *Génération conjointe de commandes et d'interfaces de supervision pour systèmes sociotechniques reconfigurables* [PhD Thesis, Université de Bretagne Sud]. https://theses.hal.science/tel-00735869/

Blazewicz, J., Brauner, N., & Finke, G. (2004). *Scheduling with Discrete Resource Constraints.*

Blazewicz, J., Dell'Olmo, P., Drozdowski, M., & Speranza, M. G. (1992). Scheduling multiprocessor tasks on three dedicated processors. *Inf. Process. Lett.*, *41*(5), 275-280.

Blazewicz, J., Dror, M., & Weglarz, J. (1991). Mathematical programming formulations for machine scheduling : A survey. *European Journal of Operational Research*, *51*(3), 283-300.

B\lażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., & Weglarz, J. (2001). *Scheduling computer and manufacturing processes*. springer science & Business media.

B\lażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., & Weglarz, J. (2007). *Handbook on scheduling : From theory to applications*. Springer Science & Business Media.

B\lazewicz, J., Kubiak, W., & Martello, S. (1993). Algorithms for minimizing maximum lateness with unit length tasks and resource constraints. *Discrete applied mathematics*, *42*(2-3), 123-138.

Blazewicz, J., Lenstra, J. K., & Kan, A. R. (1983). Scheduling subject to resource constraints : Classification and complexity. *Discrete applied mathematics*, *5*(1), 11-24.

Brucker, P. (1999). Scheduling algorithms. *Journal-Operational Research Society*, *50*, 774-774.

Bruno, J., Coffman Jr, E. G., & Sethi, R. (1974). Scheduling independent tasks to reduce mean finishing time. *Communications of the ACM*, *17*(7), 382-387.

Bussenot, R. (2018). *Rendre agile les tests d'intégration des systèmes avioniques par des langages dédiés* [PhD Thesis]. Université Paul Sabatier-Toulouse III.

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., & Vanderdonckt, J. (2003). A unifying reference framework for multi-target user interfaces. *Interacting with computers*, *15*(3), 289-308.

Campos Ciro, G. (2015). *Développement de méthodes d'ordonnancement efficaces et appliquées dans un système de production mécanique* [PhD Thesis]. Troyes.

Capawa Fotsoh, E., Mebarki, N., Castagna, P., Berruet, P., & Gamboa, F. (2020). Towards a reference model for configuration of reconfigurable manufacturing system (RMS). *Advances in Production Management Systems. The Path to Digital Transformation and Innovation of Production Management Systems: IFIP WG 5.7 International Conference, APMS 2020, Novi Sad, Serbia, August 30–September 3, 2020, Proceedings, Part I*, 391-398.

Chapurlat, V., Daclin, N., Chuche, P., Marzin, S., & Vien, P. (2013). Proposition d'un Guide d'Etude des Modes Opérationnels des Systèmes (GEMOS) pour l'Ingénierie Système. *Conférence Francophone de Génie Industriel, La Rochelle*.

Cheng, T. C. E., & Sin, C. C. S. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, *47*(3), 271-292.

Christofides, N., Alvarez-Valdés, R., & Tamarit, J. M. (1987). Project scheduling with resource constraints : A branch and bound approach. *European Journal of Operational Research*, *29*(3), 262-273.

Cochard, T. (2017). *Contribution à la génération de séquences pour la conduite de systèmes complexes critiques* [PhD Thesis, Université de Lorraine]. https://theses.hal.science/tel-01754706/

Cochard, T., Gouyon, D., & Pétin, J.-F. (2015). Génération de séquences d'actions sûres par recherche d'atteignabilité. *Génie logiciel*, *112*, 43-50. https://hal.science/hal-01138524/

Coello, C. A. C. (2005). An introduction to evolutionary algorithms and their applications. *International Symposium and School on Advancex Distributed Systems*, 425-442.

Coffman, E. G., & Graham, R. L. (1972). Optimal scheduling for two-processor systems. *Acta informatica*, *1*(3), 200-213.

Collignon, B., Vanderdonckt, J., & Calvary, G. (2008). Model-driven engineering of multi-target plastic user interfaces. *Fourth International Conference on Autonomic and Autonomous Systems (ICAS'08)*, 7-14. https://ieeexplore.ieee.org/abstract/document/4488315/

Combemale, B., Crégut, X., & Pantel, M. (2010). A design pattern for executable DSML. *Event (London)*, 1-23. https://www.researchgate.net/profile/Marc-Pantel/publication/265031853_A_Design_Pattern_for_Executable_DSML/links/5676cbfe08ae0ad265c5a5bb/A-Design-Pattern-for-Executable-DSML.pdf

Combemale, B., Crégut, X., & Pantel, M. (2012). A design pattern to build executable DSMLs and associated V&V tools. *2012 19th Asia-Pacific Software Engineering Conference*, *1*, 282-287. https://ieeexplore.ieee.org/abstract/document/6462664/

Consel, C., Latry, F., Réveillère, L., & Cointe, P. (2005). A Generative Programming Approach to Developing DSL Compilers. In R. Glück & M. Lowry (Éds.), *Generative Programming and Component Engineering* (Vol. 3676, p. 29-46). Springer Berlin Heidelberg. https://doi.org/10.1007/11561347_4

Dangoumau, N. (2000a). *Contribution à la gestion des modes des systèmes automatisés de production* [PhD Thesis]. Lille 1.

Dangoumau, N. (2000b). *Contribution à la gestion des modes des systèmes automatisés de production* [PhD Thesis, Lille 1]. https://www.theses.fr/2000LIL10184

Dangoumau, N. (2000c). *Contribution à la gestion des modes des systèmes automatisés de production*. Lille 1.

Dangoumau, N., La, C. À., Des, G., Systèmes, D. E. S., & Production, A. D. E. (2000). *CONTRIBUTION À LA GESTION DES MODES DES SYSTÈMES AUTOMATISÉS DE PRODUCTION M.E.CRAYE*.

Daniels, R. L., Hoopes, B. J., & Mazzola, J. B. (1996). Scheduling parallel manufacturing cells with resource flexibility. *Management science*, *42*(9), 1260-1276.

De Lamotte, F. F. (2006). Proposition d'une approche haut niveau pour la conception, l'analyse et l'implantation des systèmes reconfigurables. *LESTER, Université de Bretagne Sud, Lorient*. http://www-labsticc.univ-ubs.fr/~lamotte/documents/memoire_lamotte_final.pdf

Des, E., Marche, M. D. E., Gemma, L., Gemma, L., Gemma, L., Gemma, L., Gemma, L., Gemma, L., Gemma, L., & Gemma, L. (s. d.). *GUIDE D ' ETUDE DES MODES DE MARCHE ET D ' ARRET*. 1-17.

Deuerlein, J., Simpson, A. R., & Korth, A. (2014). Flushing planner : A tool for planning and optimization of unidirectional flushing. *Procedia Engineering*, *70*, 497-506. https://www.sciencedirect.com/science/article/pii/S1877705814000575

EDİS, E. B. (2009). *Resource constrained parallel machine scheduling problems with machine eligibility restrictions : Mathematical and constraint programming based approaches* [PhD Thesis]. DEÜ Fen Bilimleri Enstitüsü.

Edis, E. B., Oguz, C., & Ozkarahan, I. (2013). Parallel machine scheduling with additional resources : Notation, classification, models and solution methods. *European Journal of Operational Research*, *230*(3), 449-463.

Faraut, G. (2010). *Commutations sûres de mode pour les systèmes à événements discrets* [PhD Thesis]. INSA de Lyon.

Faraut, G., Piétrac, L., & Niel, E. (2008). Identification of incompatible states in mode switching. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, *September 2014*, 121-128. https://doi.org/10.1109/ETFA.2008.4638382

Gabow, H. N. (1982). An almost-linear algorithm for two-processor scheduling. *Journal of the ACM (JACM)*, *29*(3), 766-780.

Gacias, B., Artigues, C., & Lopez, P. (2010). Parallel machine scheduling with precedence constraints and setup times. *Computers & Operations Research*, *37*(12), 2141-2151.

Garey, M. R., & Johnson, D. S. (1975a). Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, *4*(4), 397-411.

Garey, M. R., & Johnson, D. S. (1975b). Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, *4*(4), 397-411.

Garey, M. R., & Johnson, D. S. (1978). "strong"np-completeness results : Motivation, examples, and implications. *Journal of the ACM (JACM)*, *25*(3), 499-508.

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability* (Vol. 174). freeman San Francisco.

Geurtsen, M., Didden, J. B., Adan, J., Atan, Z., & Adan, I. (2023). Production, maintenance and resource scheduling : A review. *European Journal of Operational Research*, *305*(2), 501-529. https://www.sciencedirect.com/science/article/pii/S0377221722002673

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, *13*(5), 533-549.

Götz, S., & Tichy, M. (2020). Investigating the Origins of Complexity and Expressiveness in ATL Transformations. *J. Object Technol.*, *19*(2), 12-1. https://www.jot.fm/issues/issue_2020_02/article12.pdf

Goubali, O. (2017). *Apport des techniques de programmation par démonstration dans une démarche de génération automatique d'applicatifs de contrôle-commande* [PhD Thesis]. ISAE-ENSMA Ecole Nationale Supérieure de Mécanique et d'Aérotechique-Poitiers.

Goubali, O., Berruet, P., Bignon, A., Girard, P., & Guittet, L. (2014). Anaxagore, un exemple d'ingénierie dirigée par les modèles pour la supervision industrielle. *Proc. Ergonomie et Informatique Avancée (ERGO'IA 2014)*. https://hal.science/hal-04205794/document

Goubali, O., & Goubali, O. (2017). *Apport des techniques de programmation par démonstration dans une démarche de génération automatique d ' applicatifs de contrôle-commande To cite this version : HAL Id : Tel-01505594 Présentée par : AUTOMATIQUE D ' APPLICATIFS DE CONTRÔLE-COMMANDE*.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling : A survey. In *Annals of discrete mathematics* (Vol. 5, p. 287-326). Elsevier.

Greiner, S., Wiesmayr, B., Feichtinger, K., Meixner, K., Konersmann, M., Pfeiffer, J., Oberlehner, M., Schmalzing, D., Wortmann, A., & Rumpe, B. (2023). *Maturity Evaluation of Domain-Specific Language Ecosystems for Cyber-Physical Production Systems*. https://seg.inf.unibe.ch/papers/etfa23.pdf

Habibi, F., Barzinpour, F., & Sadjadi, S. (2018). Resource-constrained project scheduling problem : Review of past and recent developments. *Journal of project management*, *3*(2), 55-88.

Hervo, C., & Le Frapper, O. (2007). *Visio 2007*. Editions ENI. https://books.google.com/books?hl=fr&lr=lang_fr&id=IDGSv51RbIUC&oi=fnd&pg=PA7&dq=ms+visio&ots=cBBOjiDqlz&sig=OKqTaouEbCb1XMwWQ4lahxycNBs

Hoogeveen, J. A., van de Velde, S. L., & Veltman, B. (1994). Complexity of scheduling multiprocessor tasks with prespecified processor allocations. *Discrete Applied Mathematics*, *55*(3), 259-272.

International Society of Automation, & American National Standards Institute (Éds.). (2010). *Batch control : American national standard: ANSI/ISA-88. Part 1: Models and terminology: ANSI/ISA-88.00-01-2010*. ISA.

Issad, M., Koul, L., & Rauzy, A. (2015). A contribution to safety analysis of railway CBTC systems using Scola. *Safety and Reliability of Complex Engineered Systems: ESREL 2015*. https://hal.science/hal-01246161/

J.-C.Gentina. (2000, juillet). De l'apport spécifique des réseaux de Petri temporisé à l'optimisation de performances des SFPM en fonctionnement cyclique et à l'ordonnancement cyclique de production. *Conférence Internationale Francophone d'Automatique (CIFA'2000), Lille*.

Jézéquel, J.-M., Barais, O., & Fleurey, F. (2011). Model Driven Language Engineering with Kermeta. In J. M. Fernandes, R. Lämmel, J. Visser, & J. Saraiva (Éds.), *Generative and Transformational*

Techniques in Software Engineering III (Vol. 6491, p. 201-221). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-18023-1_5

Jézéquel, J.-M., Combemale, B., & Vojtisek, D. (2012). *Ingénierie Dirigée par les Modèles : Des concepts à la pratique...* Ellipses. https://inria.hal.science/hal-00648489/document

Jouault, F., Allilaire, F., Bézivin, J., & Kurtev, I. (2008). ATL : A model transformation tool. *Science of computer programming*, *72*(1-2), 31-39. https://www.sciencedirect.com/science/article/pii/S0167642308000439

Kamach, O., & Piétrac, L. (2006). Multi-model approach to discrete events systems : Application to operating mode management. *Mathematics and Computers in Simulation*, *70*(5-6), 394-407.

Kan, A. R. (2012). *Machine scheduling problems : Classification, complexity and computations*. Springer Science & Business Media.

Kanso, M. (2010). *Contribution à la construction des configurations des systèmes manufacturiers : Une approche basée sur la décision multi-critère* [PhD Thesis, Lorient]. https://www.theses.fr/2010LORIS212

Kellerer, H., & Strusevich, V. A. (2003a). Scheduling parallel dedicated machines under a single non-shared resource. *European Journal of Operational Research*, *147*(2), 345-364.

Kellerer, H., & Strusevich, V. A. (2003b). Scheduling problems for parallel dedicated machines under multiple resource constraints. *Discrete Applied Mathematics*, *133*(1-3), 45-68.

Kellerer, H., & Strusevich, V. A. (2003c). Scheduling problems for parallel de dicated machines under multiple resource constraints. *Discrete Applied Mathematics*, *133*(1-3), 45-68.

Kellerer, H., & Strusevich, V. A. (2008). Scheduling parallel dedicated machines with the speeding-up resource. *Naval Research Logistics (NRL)*, *55*(5), 377-389.

Kermad, L. (1996). *Contribution à la supervision et à la gestion des modes et des configurations des systèmes flexibles de production manufacturière* [PhD Thesis]. Lille 1.

Kermani, M., Adelmanesh, B., Shirdare, E., Sima, C. A., Carnì, D. L., & Martirano, L. (2021). Intelligent energy management based on SCADA system in a real Microgrid for smart building applications. *Renewable Energy*, *171*, 1115-1127. https://www.sciencedirect.com/science/article/pii/S0960148121003566

Kesraoui, S. (2017). *Intégration des techniques de vérification formelle dans une approche de conception des systèmes de contrôle-commande : Application aux architectures SCADA*. https://lilloa.univ-lille.fr/bitstream/handle/20.500.12210/23378/https:/tel.archives-ouvertes.fr/tel-01738049/document?sequence=1

King, B. J., Read, G. J., & Salmon, P. M. (2022). Clear and present danger? Applying ecological interface design to develop an aviation risk management interface. *Applied Ergonomics*, *99*, 103643.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671-680.

Kondili, E., & Sargent, R. W. H. (1988). *A general algorithm for scheduling batch operations*. Department of Chemical Engineering, Imperial College.

Koné, O., Artigues, C., Lopez, P., & Mongeau, M. (2013). Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources. *Flexible Services and Manufacturing Journal*, *25*(1), 25-47.

Kopka, H., & Daly, P. W. (2003). *Guide to LATEX*. Pearson Education. https://books.google.com/books?hl=fr&lr=&id=BTPgB6dzhG4C&oi=fnd&pg=PA1980&dq=latex+&ots=H850b7dlMP&sig=bHbrXglxc4ICN5qZAiCfLrXyEDo

Kosar, T., Bohra, S., & Mernik, M. (2016). Domain-specific languages : A systematic mapping study. *Information and Software Technology*, *71*, 77-91. https://www.sciencedirect.com/science/article/pii/S0950584915001858

Ku, W.-Y., & Beck, J. C. (2016). Mixed integer programming models for job shop scheduling : A computational analysis. *Computers & Operations Research*, *73*, 165-173.

Land, A. H., & Doig, A. G. (2010). An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008* (p. 105-132). Springer.

Lawler, E. L., Lenstra, J. K., Kan, A. H. R., & Shmoys, D. B. (1993). Sequencing and scheduling : Algorithms and complexity. *Handbooks in operations research and management science*, *4*, 445-522.

Le Frapper, O. (2006). *AutoCAD 2007 : Tous les outils, de la conception jusqu'au dessin et à la présentation détaillée*. Éditions Eni. https://books.google.com/books?hl=fr&lr=lang_fr&id=4v1ORfD2v84C&oi=fnd&pg=PA12&dq=autocad&ots=SBJrWRZ1PX&sig=kHsDzxbf3DL31luwFswFBtVD49A

Lebeaupin, B. (2017). *Vers un langage de haut niveau pour une ingénierie des exigences agile dans le domaine des systèmes embarqués avioniques* [PhD Thesis, Université Paris Saclay (COmUE)]. https://theses.hal.science/tel-01761690/

Lenstra, J. K., Kan, A. R., & Brucker, P. (1977). Complexity of machine scheduling problems. In *Annals of discrete mathematics* (Vol. 1, p. 343-362). Elsevier.

Leroy, D., Bousse, E., Wimmer, M., Mayerhofer, T., Combemale, B., & Schwinger, W. (2020). Behavioral interfaces for executable DSLs. *Software and Systems Modeling*, *19*(4), 1015-1043. https://doi.org/10.1007/s10270-020-00798-2

Li, K., Shi, Y., Yang, S., & Cheng, B. (2011). Parallel machine scheduling problem to minimize the makespan with resource dependent processing times. *Applied Soft Computing*, *11*(8), 5551-5557.

Maqrot, S. (2019). *Méthodes d'optimisation combinatoire en programmation mathématique : Application à la conception des systèmes de verger-maraîcher* [PhD Thesis]. Université Paul Sabatier-Toulouse III.

Marangé, P., Pétin, J.-F., Manceaux, A., & Gouyon, D. (2011). Contribution à la reconfiguration des systèmes de production : Ordonnancement par recherche d'atteignabilité. *Journal Européen des Systèmes Automatisés (JESA)*, *45*(1/3), 45-60.

Melouk, M., Rhazali, Y., & Youssef, H. (2020). An Approach for Transforming CIM to PIM up To PSM in MDA. *Procedia Computer Science*, *170*, 869-874. https://www.sciencedirect.com/science/article/pii/S1877050920305603

Mesli-Kesraoui, O., Ouhammou, Y., Goubali, O., Berruet, P., Girard, P., & Grolleau, E. (2021). Towards a Model-Based Approach to Support Physical Test Process of Aircraft Hydraulic Systems. *International Conference on Model and Data Engineering*, 33-40.

Mitchell, J. E. (2002). Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, *1*, 65-77.

Mohanan, M. (2020). Automated transformation of NL to OCL constraints via SBVR. *International Journal of Advanced Intelligence Paradigms*, *16*(3/4), 229. https://doi.org/10.1504/IJAIP.2020.107524

Nastov, B., Chapurlat, V., Dony, C., & Pfister, F. (2015). A Verification Approach from MDE Applied to Model Based Systems Engineering : xeFFBD Dynamic Semantics. In F. Boulanger, D. Krob, G. Morel, & J.-C. Roussel (Éds.), *Complex Systems Design & Management* (p. 225-238). Springer International Publishing. https://doi.org/10.1007/978-3-319-11617-4_16

Ott, A. (2007). *System testing in the avionics domain* [PhD Thesis]. Universität Bremen.

Oulamara, A. (2009). *Contribution à l'étude des problèmes d'ordonnancement flowshop avec contraintes supplémentaires : Complexité et méthodes de résolution* [PhD Thesis]. Institut National Polytechnique de Lorraine-INPL.

Özdamar, L., & Ulusoy, G. (1995). A survey on the resource-constrained project scheduling problem. *IIE transactions*, *27*(5), 574-586.

Pedroza, G., Apvrille, L., & Knorreck, D. (2011). AVATAR : A SysML environment for the formal verification of safety and security properties. *2011 11th Annual International Conference on New Technologies of Distributed Systems*, 1-10. https://ieeexplore.ieee.org/abstract/document/5957992/

Pinedo, M., & Hadavi, K. (1992). Scheduling : Theory, algorithms and systems development. In *Operations Research Proceedings 1991* (p. 35-42). Springer.

Prat, S. (2018). *Intégration de techniques de vérification par simulation dans un processus de conception automatisée de contrôle commande To cite this version : HAL Id : Tel-01691344 Sophie PRAT Intégration de Techniques de Vérification par Simulation dans un Processus d*.

Prat, S., Rauffet, P., Bignon, A., & Berruet, P. (2015). Vers l'intégration d'une approche de génération automatique de mod ele de simulation dans un flot de conception de contrôle -commande. *JDJN MACS*. https://inria.hal.science/hal-01251052/

Prosvirnova, T. (2014). *AltaRica 3.0 : A model-based approach for safety analyses* [PhD Thesis, Ecole Polytechnique]. https://www.researchgate.net/profile/Tatiana-Prosvirnova/publication/278827421_AltaRica_30_a_Model-Based_approach_for_Safety_Analyses/links/56670e6608ae8905db8b8d79/AltaRica-30-a-Model-Based-approach-for-Safety-Analyses.pdf

Prosvirnova, T., & Rauzy, A. (2013). AltaRica 3.0 project : Compile guarded transition systems into fault trees. *European Safety and Reliability Conference, ESREL*, *2013*. http://www.lix.polytechnique.fr/~prosvirnova/PR13_ESREL2013_FTCompiler.pdf

Ramalho, F., Robin, J., & de Barros, R. S. M. (2003). XOCL-an XML Language for Specifying Logical Constraints in Object Oriented Models. *J. Univers. Comput. Sci.*, *9*(8), 956-969. https://www.academia.edu/download/33673797/jucs2003-oficial.pdf

Rasmussen. (1984). Coping with Complexity. *Annals of the New York Academy of Sciences*, *426*(1), 11-18. https://doi.org/10.1111/j.1749-6632.1984.tb16506.x

Roques, P. (2017). *Systems architecture modeling with the Arcadia method : A practical guide to Capella*. Elsevier. https://books.google.com/books?hl=fr&lr=&id=Qj3jCwAAQBAJ&oi=fnd&pg=PP1&dq=arcadia+capella&ots=-2wKpWGoHs&sig=6jAnfL4WatLyuXXkMKUcw3Kay8k

Roques, P. (2016). MBSE with the ARCADIA Method and the Capella Tool. *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*.

Roselli, S. F., Bengtsson, K., & Åkesson, K. (2018). SMT solvers for job-shop scheduling problems : Models comparison and performance evaluation. *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, 547-552.

Rouhi, A., & Lano, K. (2020). Formalizing the main characteristics of QVT-based model transformation languages. *Journal of Computing and Security*, *7*(1), 35-62. https://jcomsec.ui.ac.ir/article_24645_4491.html

Roy Chaudhuri, S., Natarajan, S., Banerjee, A., & Choppella, V. (2019). Methodology to develop domain specific modeling languages. *Proceedings of the 17th ACM SIGPLAN International Workshop on Domain-Specific Modeling*, 1-10.

Sallaou, M. (2009). *Taxonomie des connaissances et exploitation en conception pr ´ eliminaire : Application `a un syst `eme Docteur l ' École Nationale Supérieure d ' Arts et Métiers Spécialité " Mécanique "*.

Sanctorum, A., & Signer, B. (2019). A unifying reference framework and model for adaptive distributed hybrid user interfaces. *2019 13th International Conference on Research Challenges in Information Science (RCIS)*, 1-6.

Šeda, M. (2007a). Mathematical models of flow shop and job shop scheduli ng problems. *International Journal of Applied Mathematics and Computer Sciences*, *4*(4), 241-246.

Šeda, M. (2007b). Mathematical models of flow shop and job shop scheduling problems. *International Journal of Applied Mathematics and Computer Sciences*, *4*(4), 241-246.

Sethi, R. (1976). Scheduling graphs on two processors. *SIAM Journal on Computing*, *5*(1), 73-82.

S\lowiński, R. (1980). Two approaches to problems of resource allocation among project activities—A comparative study. *Journal of the Operational Research Society*, *31*(8), 711-723.

Soriano, P., & Gendreau, M. (1997). Fondements et applications des méthodes de recherche avec tabous. *RAIRO-Operations Research*, *31*(2), 133-159.

Stéphane, B. (1991). *Intégration de la gestion des modes de marche dans le pilotage d'un système automatisé de production*.

Symbols, I. (2009). Instrumentation Symbols and Identification ANSI/ISA-5.1. *International Society of Automation*.

Temate Ngaffo, S. H. G. (2012). *Des langages de modélisation dédiés aux environnements de méta-modélisation dédiés* [PhD Thesis]. https://oatao.univ-toulouse.fr/8841/

Unlu, Y., & Mason, S. J. (2010). Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, *58*(4), 785-800.

Vincent, B., Duhamel, C., Ren, L., & Tchernev, N. (2016). An efficient heuristic for scheduling on identical parallel machines to minimize total tardiness. *IFAC-PapersOnLine*, *49*(12), 1737-1742.

Wagner, H. M. (1959). An integer linear-programming model for machine scheduling. *Naval research logistics quarterly*, *6*(2), 131-140.

Wang, H., Lau, N., & Gerdes, R. M. (2018). Examining cybersecurity of cyberphysical systems for critical infrastructures through work domain analysis. *Human factors*, *60*(5), 699-718.

Wasson, C. S. (2011). System Phases, Modes, and States Solutions to Controversial Issues. *Proceedings of the 21th Annual International Symposium of the International Council of Systems Engineering*.

Wolny, S., Mazak, A., Carpella, C., Geist, V., & Wimmer, M. (2020). Thirteen years of SysML : A systematic mapping study. *Software and Systems Modeling*, *19*, 111-169.

Xie, X., Nachabe, M., & Zeng, B. (2015). Optimal Scheduling of Automatic Flushing Devices in Water Distribution System. *Journal of Water Resources Planning and Management*, *141*(6), 04014081. https://doi.org/10.1061/(ASCE)WR.1943-5452.0000477

Zdun, U., Strembeck, M., & Group, D. S. (2009). Reusable Architectural Decisions for DSL Design. *Proceedings of 14th European Conference on Pattern Languages of Programs (EuroPLoP)*, 1-37.

Annexes

4 Annexe 1 : Guideline pour l'obtention du modèle opérationnel

4.1 Partie1 : Modélisation

Notre démarche suit un flot de six (06) phases permettant d'aider un concepteur quelconque à spécifier le comportement global de son système et d'obtenir un modèle formel utilisable pour la conception.

4.1.1 Phase 1 Extraction et classification des fonctions et composants

L'objectif de cette phase est d'extraire les fonctions de haut niveau et élémentaires ainsi que les composants contenus dans le cahier de charge et de les assigner à différents niveaux de la WDA en s'appuyant sur guideline que nous avons défini préalablement.

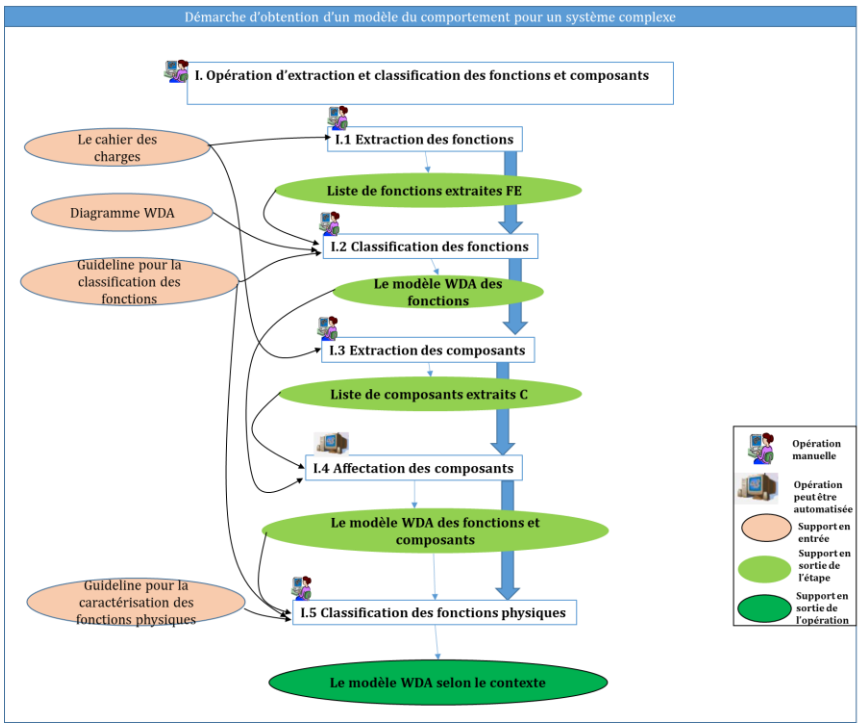


Figure 63 : Entrées/Sorties des étapes de l'opération 1 de notre démarche.

4.1.2 Etape 1 : Extraction des fonctions

Lors de cette étape, nous procédons à l'extraction des fonctions à partir du cahier des charges.

Nous définissons l'ensemble de fonctions du cahier des charges $FCDC = \{f_1, f_2, \dots, f_n\}$.

4.1.2.1 Entrées

Cahier de charges.

4.1.2.2 Opérations à réaliser par le concepteur

Une fonction est définie comme « un service délivré par le système » (Prat, 2018).

Le concepteur doit :

A. Pour chaque fonction $f_i, (i = 1, \dots, n)$ du cahier des charges $FCDC$, faire :
   a. Extraire la fonction f_i du cahier des charges $FCDC$.

b. Sauvegarder la fonction f_i dans la liste des fonctions extraites $FE = \{f_1, f_2, \dots, f_n\}$.

Règle générale de cette étape :

$\forall f_i \in FCDC, (i = 1, \dots, n) => f_i \in FE, (i = 1, \dots, n)$.

4.1.2.3 Sorties

Liste de fonctions extraites FE.

4.1.3 Etape 2 : Classification des fonctions

Lors de cette étape, on classifie les fonctions extraites du cahier des charges $FE = \{f_1, f_2, \dots, f_n\}$ selon les trois haut niveaux d'abstraction de la WDA, $niveau_k, k \in \{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques\}$.

4.1.3.1 Entrées

Liste de fonctions extraites FE.

4.1.3.2 Diagramme WDA.

	Système	Sous-système	Composant
Objectif fonctionnel			
Fonction abstraite			
Fonction générique			
Fonction physique			
Formes physiques			

Tableau 11 : Diagramme WDA (Rasmussen, 1984).

4.1.3.3 Guideline pour la classification des fonctions :

Niveau d'abstraction	**Objectifs fonctionnels**		**Fonctions abstraites**				**Fonctions génériques**			**Fonctions physiques**		**Formes physiques**
Définition	Fonctions haut niveau justifiant la conception du système.		Fonctions imposées par le contexte.				Fonctions décrivant le processus fonctionnel.			Fonctions propres aux composants		Composants physiques
Niveau de décomposition	Le système en entier		Le système en entier				Un sous-système, sous ensemble de composants			Un seul composant		Le composant
Question de spécification	- Pour quelle raison le système va être conçu?	-Quels sont les fonctions qui permettent de répondre aux besoins client.	- Quels sont les fonctions haut niveau nécessaires pour répondre à chacun des objectifs client.	- Quels sont les fonctions imposées par le contexte réalisé par tout le système pour répondre aux objectifs?	- En posant la question pourquoi?, on obtient l'objectif fonctionnel associé.		- Quels sont les fonctions réalisées par les sous-système.	- Obtenues en posant la question: comment? À chaque fonction abstraite.	- En posant la question pourquoi?, on obtient la fonction abstraite associée.	- Quels sont les fonctions réalisées par chacun des composants.	- Obtenues en se basant sur le tableau 3.	- Du cahier des charges.
Liaison entre les fonctions	-Décomposées en sous fonctions par l'opérateur (OU) et (ET +).	-Reliées aux fonctions abstraites par l'opérateur de raffinement.	-Décomposées en sous fonctions par l'opérateur (OU -) et (ET +),		-Reliées aux fonctions génériques par l'opérateur de raffinement.	Et aux fonctions objectives par l'opérateur d'abstraction.	-Décomposées en sous fonctions par l'opérateur (OU -) et (ET +).		-Reliées aux fonctions physiques par l'opérateur de raffinement, reliées aux fonctions abstraites par l'opérateur d'abstraction et aux sous-ensembles de composants par l'opérateur coopération.	-Décomposées en sous fonctions par l'opérateur (OU -) et (ET +),	-Reliées aux fonctions génériques par l'opérateur d'abstraction affectées aux composant par l'opérateur assignement.	
Ordre d'exécution	- Le choix de l'opérateur ET (+) mène à spécifier l'ordre d'exécution: parallèle, séquentiel.		- Le choix de l'opérateur ET (+) mène à spécifier l'ordre d'exécution: parallèle, séquentiel.				- Le choix de l'opérateur ET (+) mène à spécifier l'ordre d'exécution: parallèle, séquentiel.			- Le choix de l'opérateur ET (+) mène à spécifier l'ordre d'exécution: parallèle, séquentiel.		
Type de fonctions	Selon le contexte		Selon le contexte				- Fonction de transfert, fonction de routage. (définies dans le Tableau 3)			Stockage, libre passage, contrôle, générateur, transformation(définies dans le Tableau 3)		Selon le contexte

Tableau 12 : guideline pour la classification des fonctions.

Fonctions génériques. (Prat, 2018)		
Fonction de routage	Le but d'une fonction de Routage est d'aiguiller le fluide à travers la tuyauterie d'un point X vers un point Y, selon le contexte. Une fonction de Routage est mise en œuvre par une ou plusieurs opérations de Libre Passage par des vannes, et potentiellement une ou plusieurs opérations de Stockage Actif par des vannes.	
fonction de Transfert	Le but d'une fonction de Transfert est de déplacer le fluide d'un point X vers un point Y. Une telle fonction requiert la mise en œuvre d'une fonction de Routage X ! Y et d'une fonction de Générateur.	
Fonctions physiques. (Prat, 2018)		
Une fonction stockage	Une fonction de Stockage garde le fluide dans une zone donnée. Différents types d'opérations peuvent implémenter une telle fonction. Une opération de Stockage passif est utilisée lorsque la ressource réalisant l'opération n'est pas commandée (cas d'un réservoir ou d'une soute). Au contraire, une opération de Stockage Actif nécessite de commander la ressource (fermeture d'une vanne par exemple). (Prat, 2018)	Stockage passif : réservoir, soute. Stockage actif : vanne fermée
Une fonction libre passage	Une fonction de Libre Passage permet le passage du fluide en un lieu donné. Elle est implémentée par une opération de Libre Passage qui est réalisée par une ressource qui permet au fluide de passer (comme une vanne ouverte). Cette opération est similaire à une opération de Stockage Actif mais avec une durée de stockage nulle. (Prat, 2018)	Libre passage : vanne ouverte.
Une fonction de contrôle	Une fonction de Contrôle mesure certains paramètres du fluide. Cette fonction est implémentée par une opération de Contrôle réalisé par un capteur de mesure. (Prat, 2018)	Capteur de débit, capteur de pression, capteur de température.
Une fonction de générateur	Une fonction de Générateur génère un mouvement du fluide (en imposant une pression ou un débit). Une telle fonction est implémentée par une opération de Générateur par une ressource (par exemple une pompe) qui génère une pression ou un débit. (Prat, 2018)	Pompe, générateur, EMP, EDP. ACCU
Une fonction de Transformation	Modifie les caractéristiques du fluide (par exemple : traitement de l'eau, désalinisation de l'eau de mer). Elle est mise en œuvre par une opération de Transformation réalisée par une ressource qui modifie une ou plusieurs propriétés du fluide. (Prat, 2018)	Module de chloration, stérilisateur

Tableau 13 : Taxonomie de fonctions génériques et physiques.

4.1.3.4 Opérations à réaliser par le concepteur

Le concepteur procède comme suit :

a) Classification des fonctions selon les niveaux d'abstraction de la WDA

B. Pour chaque niveau d'abstraction $niveau_k, k = \{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques\}$ faire :
   a. Se positionner dans la guideline au même niveau d'abstraction $niveau_k$.
   b. Répondre aux questions de la guideline de ce niveau $niveau_k$.
   c. Si une réponse correspond à une fonction $f_i, (i = 1, \dots, n)$ de la liste des fonctions extraites FE, faire :
      1. Récupérer la fonction f_i.
      2. Placer cette fonction dans le diagramme WDA dans le niveau d'abstraction $niveau_k$.

      Cette affectation est définie comme suit :

Règle générale de cette étape :

$$\forall f_i \in FE, (i = 1, \dots, n)$$

$$affectation_{f_i, niveau_k} = \begin{cases} 1\ si\ f_i\ correspond\ à\ une\ réponse\ des\ questions\ de\ guideline\ du\ niveau_k, f_i\ est\ de\ niveau_k \\ 0\ sinon \end{cases}$$

b) Liaisons entre fonctions de même niveau d'abstraction

C. Pour chaque fonction f_i du niveau d'abstraction $niveau_k$, $affectation_{f_i, niveau_k} = 1$, faire:
   - Pour toute fonction f_j , $affectation_{f_j, niveau_k} = 1\ et\ j \neq i$ faire :
     - Si f_j est une sous fonction de f_j faire :
       a. Relier f_i et f_j par l'opérateur : (.) ET logique.

$$liaison_{f_i, f_j} = \begin{cases} ET\ si\ f_j\ est\ une\ sous\ fonction\ de\ f_i \\ 0\ sinon \end{cases}$$

       b. Si, il existe une fonction f_l tel que la réalisation de f_i nécessite f_j **ou** f_l alors les fonctions f_j et $f_l\ sont\ liées\ par\ un\ l'opérateur\ OU\ logique\ (+)$:
       $\exists f_l, liaison_{f_i, f_l} = 1 j \neq l,\quad realisation_{f_i} = (f_j \vee f_l) \Rightarrow liaison_{f_i, f_j, f_l} = OU\ logique\ (+)$.

       c. Sinon $si\ la\ réalisation\ def_i\ nécessite\ f_j$ ***et*** f_l:
       $\exists f_l, liaison_{f_i, f_l} = 1\ j \neq l,\quad realisation_{f_i} = (f_j \wedge f_l) \Rightarrow liaison_{f_i, f_j, f_l} = ET\ logique\ (.)$
          1. Spécifier l'ordre d'exécution des fonctions f_j, f_l : parallèle $(||)$, séquentiel $(\rightarrow)$.
          $liaison_{f_i, f_j, f_l} = ET\ logique \Rightarrow ((f_j || f_l) \vee (f_j \rightarrow f_l) \vee (f_l \rightarrow f_j))$
     - Sinon : Spécifier l'ordre d'exécution des fonctions f_j, f_i : parallèle $(||)$, séquentiel $(\rightarrow)$.
     $liaison_{f_i, f_j} = ((f_i || f_j) \vee (f_i \rightarrow f_j) \vee (f_j \rightarrow f_i))$

Règle générale de cette étape :

$$\forall f_i, f_j, f_l \in FE, (i = 1, \dots, n), i \neq j \neq l :$$

$$affectation_{f_i, niveau\ k} = 1, affectation_{f_j, niveau\ k} = 1, affectation_{f_l, niveau\ k}, \forall k \in \{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques\}$$

$$liaison_{f_i, f_j} = 1\ et$$

$$liaison_{f_i, f_l} = 1$$

$$si\ liaison_{f_i, f_j, f_l} = ET\ logique \Rightarrow ((f_j \| f_l) \vee (f_j \rightarrow f_l) \vee (f_l \rightarrow f_j))$$

Nous proposons les annotations suivantes :

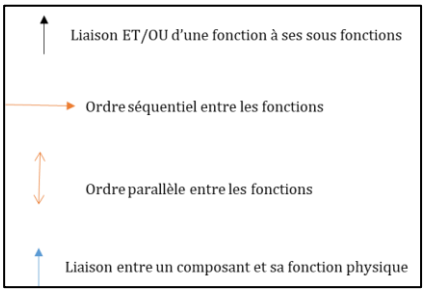


Figure 64 : Annotations pour liaisons entre fonctions.

4.1.3.5 Sorties

Diagramme WDA des fonctions : remplissage des niveaux $\{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques\}$.

4.1.4 Etape 3 : Extraction des composants

Lors de cette étape, nous procédons à l'extraction des composants élémentaires à partir du cahier des charges.

4.1.4.1 Entrées

Cahier des charges

4.1.4.2 Opérations à réaliser par le concepteur

Nous définissons l'ensemble des composants $CCDC = \{c_1, c_2, \dots, c_n\}$.

A. Pour chaque composant c du cahier des charges $CCDC$, faire :
   a. Extraire le composant c du cahier des charges $CCDC$.
   b. Sauvegarder le composant $c$dans la liste des composants extraits $C = \{c_1, \dots, c_n\}$.

4.1.4.3 Sorties

Liste de composants C.

4.1.5 Etape 4 : Affectation des composants

Tous les composants extraits seront affectés au niveau le plus bas « formes physiques » du diagramme WDA.

4.1.5.1 Entrées

Liste de composants C.

4.1.5.2 *Modèle WDA des fonctions.*

4.1.5.3 *Opérations à réaliser par le concepteur*

a. Pour chaque composant $c \in C$ faire :
b. Affecter le composant c au niveau d'abstraction 'formes physiques'.

Règle générale de cette étape :

$$\forall c \in C, affectation_{c,niveau_{formesphysiques}} = 1$$

4.1.5.4 *Sorties*

Diagramme WDA des fonctions et composants : remplissage des niveaux $\{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques, formes\ physiques\}$

4.1.6 Etape 5 : Classification des fonctions physiques

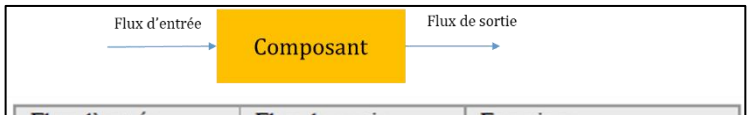
Dans cette étape, le concepteur affecte pour chaque composant c, les fonctions physiques qui les réalise.

4.1.6.1 *Entrées*

Modèle WDA des fonctions et composants.

Taxonomie des fonctions.

Guideline pour la caractérisation des fonctions physiques.



Flux d'entrée	Flux de sortie	Fonctions
Etat 1	Etat 2	Convertir/Transformer
Etat 1	Etat 1	Transmettre
Pas d'entrée	Etat 2	Fournir/Approvisionner
Etat 1	Pas de sortie	Stocker/Accumuler

Tableau 14 : Guideline pour la caractérisation des fonctions physiques. (Sallaou, 2009)

4.1.6.2 *Opérations à réaliser par le concepteur*

Pour chaque composant du modèle WDA des fonctions et composants, tel que : $affectation_{c,niveau_{formesphysiques}} = 1$, Faire :

a. Spécifier l'ensemble de ces états opérationnels $S_c = \{s_1, \dots, s_n\}$.
b. Pour chaque état opérationnel $s_h, h \in \{1, \dots, n\}$ de ce composant faire :
    1. Sélectionner la fonction physique f_h de l'ensemble des fonctions élémentaires $F_{physiques}$ existante dans la taxonomie (Tableau 13), telle que le composant c réalise f_h dans l'état s_h .
    2. Si le concepteur n'arrive pas à caractériser la fonction f_h du composant c, faire :
        - Se référer au guideline pour la caractérisation des fonctions physiques.
    3. Affecter la fonction f_h au niveau 'fonctions physiques' du diagramme WDA.

Règle générale de cette étape :

$$\forall c \in C, affectation_{c,niveau_{formesphysiques}} = 1,$$

$$\forall s_h \in S_c, \exists f_h \in F_{physiques}(c): f_h(c, s_h)$$

$$affectation_{f_h, niveau_{fonctionsphysiques}} = 1$$

4.1.6.3 *Sorties*

Diagramme WDA selon le contexte : remplissage des niveaux $\{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques, fonctions\ physiques, formes\ phys$

4.2 Phase 2 : Identification et affectation des propriétés comportementales

L'objectif de cette opération est de sélectionner les modes de la bibliothèque des modes et de les affecter aux fonctions du modèle WDA issus de la phase précédente, en se basant sur un Guideline et règles que nous avons proposées.

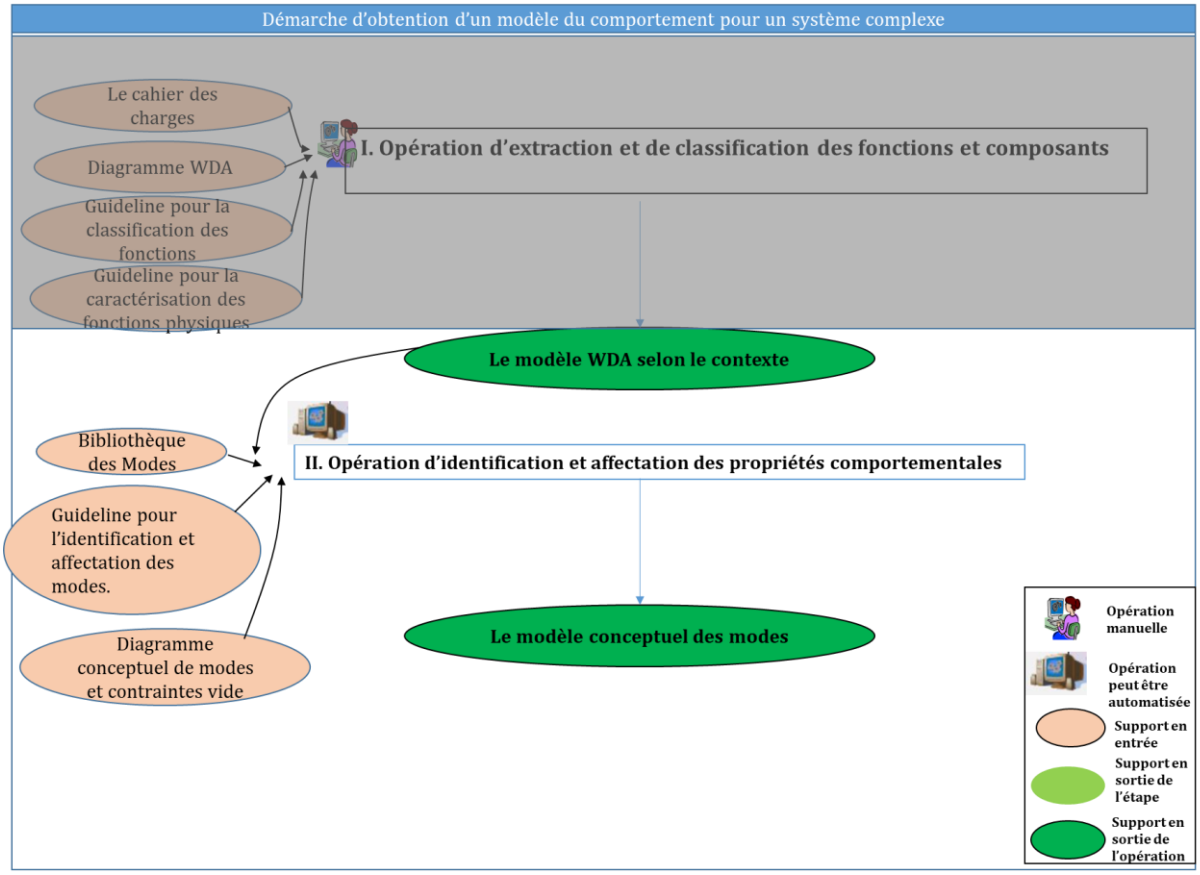


Figure 65 : Entrées/Sorties des étapes de l'opération 1 de notre démarche.

4.2.1 Etape 1 : Sélection et affectation des modes

Dans cette étape, le concepteur sélectionne les modes de la liste des modes (de la bibliothèque des modes) en se basant sur la guideline. Soit : $FM_i \in \{FM_1, \dots, FM_n\}$ l'ensemble des familles de modes de la bibliothèque, et $M_{i,j} \in \{M_{i,1}, \dots, M_{i,k}\}$ l'ensemble des modes de la famille de modes FM_i.

4.2.1.1 *Entrées*

4.2.1.1.1 La bibliothèque des modes.

La bibliothèque des modes est une libraire qui contient des familles de modes, modes et états définis génériquement. Pour chaque mode nous avons proposé un pattern (modèle) basé sur les liaisons intra-modes définies à travers des matrices de transitions. Une matrice définit les autorisations/interdictions des liaisons inter-modes.

4.2.1.1.2 Liste de modes

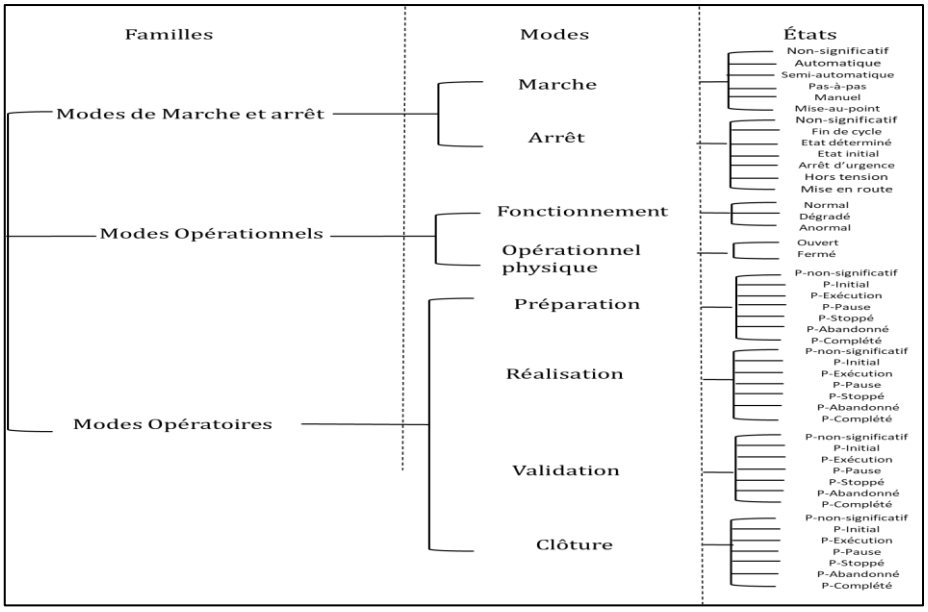


Figure 66 : Liste de modes de la bibliothèque de modes.

4.2.1.2 Patterns de modes

4.2.1.2.1 La modélisation du mode d'arrêt

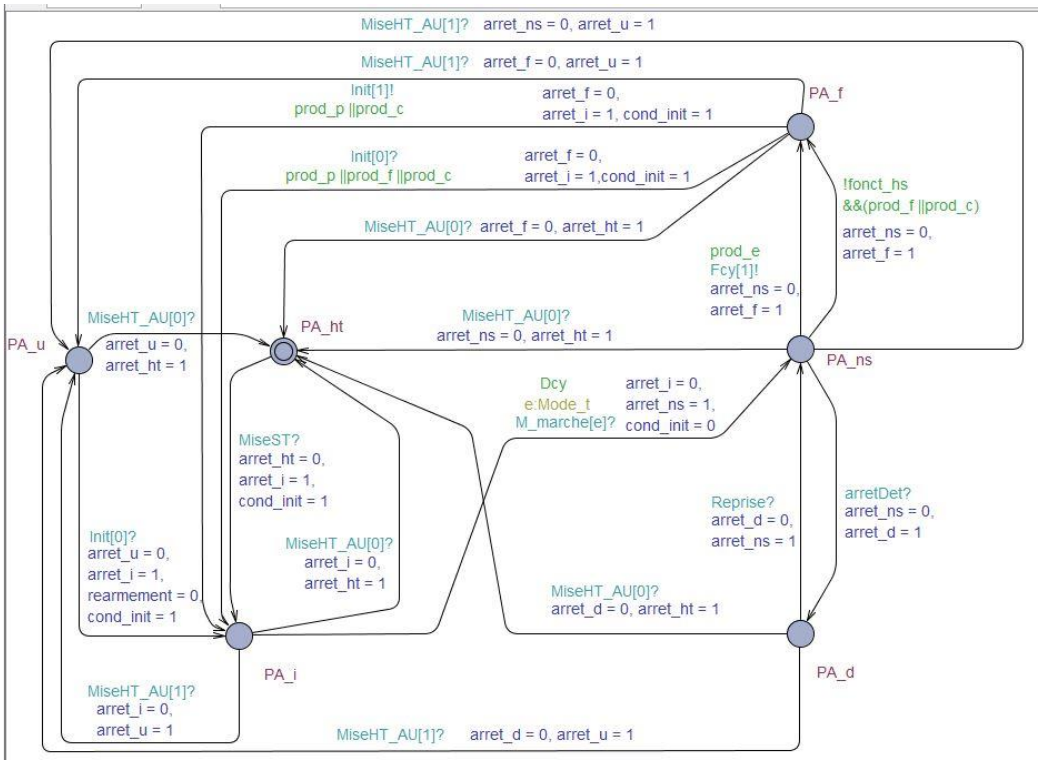


Figure 67 : Modèle du mode d'arrêt.

4.2.1.2.2 La modélisation du mode marche

La figure (Figure 68) présente la modélisation du mode marche selon les transitions présentées ci-dessus.

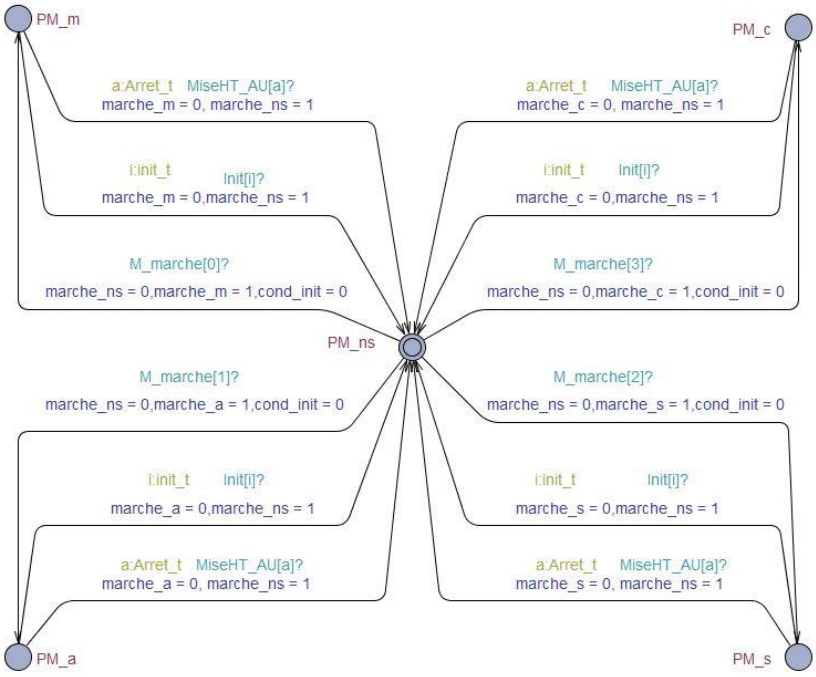


Figure 68 : Modélisation du mode marche.

4.2.1.2.3 Modélisation du mode de fonctionnement

Selon les transitions présentées dans la figure ci-dessus, nous modélisons le mode de fonctionnement.

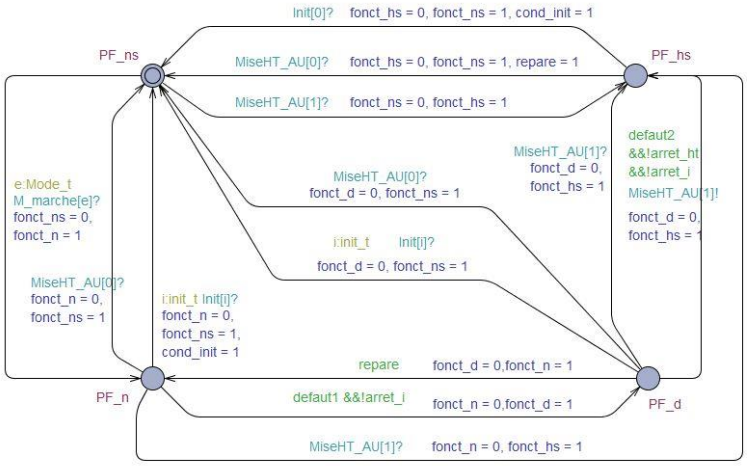


Figure 69: modélisation du mode de fonctionnement.

4.2.1.2.4 Modélisation du mode opérationnel physique

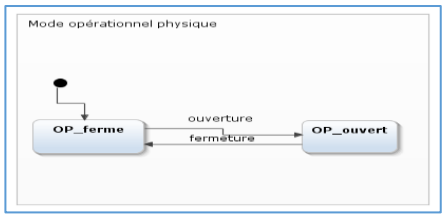


Figure 70 : modélisation du mode opérationnel physique.

4.2.1.2.5 Modélisation du mode réalisation

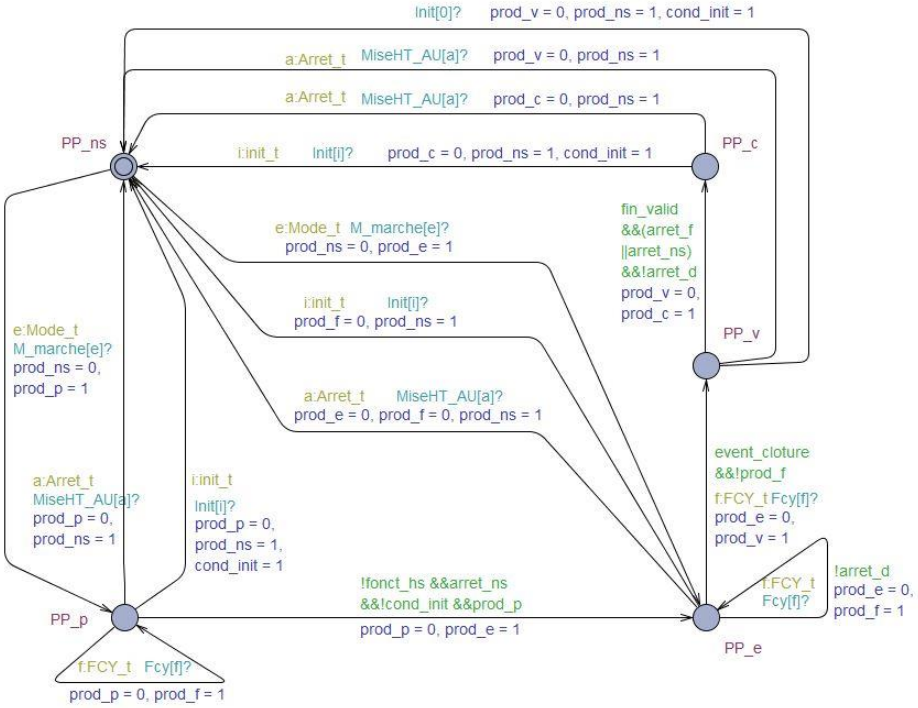


Figure 71 : Modélisation du mode réalisation.

4.2.1.2.6 Modélisation du pupitre de commande

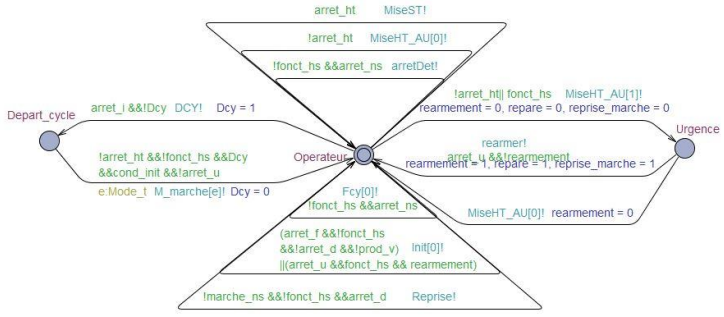


Figure 72 : modélisation du pupitre de commande.

4.2.1.2.7 Liaisons inter-modes

Pour assurer la sécurité des personnes et du système, nous avons pré-remplie la matrice la matrice de cohérence des modes (Dangoumau et al., 2000). Cela permet de faciliter aux concepteurs la validation compatibilité des modes entre eux. En effet, la matrice de cohérence des modes (MCM) permet d'interdire/autoriser que deux états de deux modes différents soient actifs en même temps. Le (**Erreur ! Source du renvoi introuvable.**) présente la matrice de cohérence d es modes pré-rempli proposé. Les 0 et 1 écrits en noirs sont fixes et ne changeront pas d'une conception à une autre. A contrario les cases écrites en vers 0/1 sont spécifiques au système à concevoir.

		A							**M**					
		A-ns	A-d	A-i	A-ht	A-fc	A-u	A-mr	M-ns	M-a	M-sa	M-pp	M-m	M-ma
	A-ns	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
	A-d		**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1/0**	**1/0**	**1/0**	**1/0**	**1/0**
	A-i			**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
	A-ht				**1**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
	A-fc					**1**	**0**	**0**	**0**	**1/0**	**1/0**	**1/0**	**1/0**	**1/0**
	A-u						**1**	**0**	**0**	**1/0**	**1/0**	**1/0**	**1/0**	**1/0**
	A-mr							**1**	**0**	**1/0**	**1/0**	**1/0**	**1/0**	**1/0**

Tableau 15 : Matrice de cohérence entre modes.

4.2.1.2.8 Diagramme conceptuel de modes et contraintes vide

	Système	Sous-système	Composant
Objectif fonctionnel	Fonctions : Modes : Contraintes :		
Fonction abstraite	Fonctions : Modes : Contraintes :		
Fonction générique		Fonctions : Modes : Contraintes :	
Fonction physique			Fonctions : Modes : Contraintes :
Formes physiques			Composants : Modes : Contraintes :

Tableau 16 : modèle conceptuel de modes et contraintes.

4.2.1.2.9 Modèle WDA selon le contexte.

Guideline pour l'identification et affectation des modes.

A. Se positionner sur le $niveau_{objectifs\ fonctionnels}$.
   a. Sélectionner de la bibliothèque des modes, la famille de modes de marche et d'arrêt.
      1. Sélectionner le mode de marche.
      2. Identifier comment le système peut fonctionner.
      3. Sélectionner les états de marche correspondants. (L'état M-ns est sélectionné automatiquement).
      4. Sélectionner le mode d'arrêt.
      5. Identifier comment le système peut être arrêté.
      6. Sélectionner les états d'arrêt correspondants. (L'état A-ht est sélectionné automatiquement).

7. Affecter les modes et états sélectionnés au $niveau_{objectifs\ fonctionnels}$ dans la case *modes* (Tableau 16).

b. Sélectionner de la bibliothèque des modes, la famille de modes opératoires.
   1. S'il existe une étape de préparation :
   2. Sélectionner le mode préparation.
   3. Sélectionner les états du mode préparation nécessaires.
   4. Sélectionner le mode réalisation.
   5. Sélectionner les états du mode réalisation nécessaires.
   6. S'il existe une étape de validation :
   7. Sélectionner le mode validation.
   8. Sélectionner les états du mode validation nécessaires.
   9. S'il existe une étape de clôture :
   10. Sélectionner le mode clôture.
   11. Sélectionner les états du mode clôture nécessaires.
   12. Affecter les modes et états sélectionnés au $niveau_{objectifs\ fonctionnels}$ dans la case *modes* (Tableau 16).

c. Sélectionner de la bibliothèque des modes, la famille de modes opérationnels.
   1. Sélectionner le mode de fonctionnement.
   2. Si le système peut fonctionner en 'dégradé':
   3. Sélectionner l'état 'dégradé' en plus des états 'normal' et 'anormal' .
   4. Affecter les modes et états sélectionnés au $niveau_{objectifs\ fonctionnels}$ dans la case *modes* (Tableau 16).

B. Se positionner sur le $niveau_{fonctions\ abstraites}$.
   a. Sélectionner de la bibliothèque des modes, la famille de modes de marche et d'arrêt.
      1. Sélectionner le mode de marche.
      2. Identifier comment les fonctions du niveau $niveau_{fonctions\ abstraites}$. Peuvent fonctionner.
      3. Sélectionner les états du mode de marche correspondants.
      4. Sélectionner le mode d'arrêt.
      5. Identifier comment les fonctions du niveau $niveau_{fonctions\ abstraites}$. Peuvent être arrêtées.
      6. Sélectionner les états correspondants.
      7. Affecter les modes et états sélectionnés au $niveau_{fonctions\ abstraites}$ dans la case *modes* (Tableau 16).
   b. Sélectionner de la bibliothèque des modes, la famille de modes opératoires.
      1. S'il existe une étape de préparation :
         - Sélectionner le mode préparation.
         - Sélectionner les états du mode préparation nécessaires.
      2. Sélectionner le mode réalisation.
      3. Sélectionner les états du mode réalisation nécessaires.
      4. S'il existe une étape de validation :
         - Sélectionner le mode validation.
         - Sélectionner les états du mode validation nécessaires.
      5. S'il existe une étape de clôture :
         - Sélectionner le mode clôture.
         - Sélectionner les états du mode clôture nécessaires.
      6. Affecter les modes et états sélectionnés au $niveau_{fonctions\ abstraites}$ dans la case *modes* (Tableau 16).

C. Se positionner sur le $niveau_{fonctions\ génériques}$.

a. Sélectionner de la bibliothèque des modes, la famille de modes de marche et d'arrêt.
    1. Sélectionner le mode de marche.
    2. Sélectionner comment les fonctions du niveau $niveau_{fonctions\ génériques}$. Peuvent fonctionner.
    3. Sélectionner les états du mode de marche correspondants.
    4. Sélectionner le mode d'arrêt.
    5. Identifier comment les fonctions du niveau $niveau_{fonctions\ génériques}$. Peuvent être arrêtées. Et sélectionner les états correspondants.
    6. Affecter les modes et états sélectionnés au $niveau_{fonctions\ génériques}$ dans la case *modes* (Tableau 16).

b. Sélectionner de la bibliothèque des modes, la famille de modes opératoires.
    1. S'il existe une étape de préparation :
        - Sélectionner le mode préparation.
        - Sélectionner les états du mode préparation nécessaires.
    2. Sélectionner le mode réalisation.
    3. Sélectionner les états du mode réalisation nécessaires.
    4. S'il existe une étape de validation :
        - Sélectionner le mode validation.
        - Sélectionner les états du mode validation nécessaires.
    5. S'il existe une étape de clôture :
        - Sélectionner le mode clôture.
        - Sélectionner les états du mode clôture nécessaires.
    6. Affecter les modes et états sélectionnés au $niveau_{fonctions\ génériques}$ dans la case *modes* (Tableau 16).

D. Se positionner sur le $niveau_{fonctions\ physiques}$.
a. Sélectionner de la bibliothèque des modes, la famille de modes de marche et d'arrêt.
    1. Sélectionner le mode de marche.
    2. sélectionner les états du mode de marche les fonctions du niveau $niveau_{fonctions\ physiques}$. Peuvent fonctionner.
    3. Sélectionner le mode d'arrêt.
    4. Identifier comment les fonctions du niveau $niveau_{fonctions\ physiques}$. Peuvent être arrêtées. Et sélectionner les états correspondants.
    5. Affecter les modes et états sélectionnés au $niveau_{fonctions\ physiques}$ dans la case *modes* (Tableau 16).

E. Se positionner sur le $niveau_{formes\ physiques}$.
a. Sélectionner de la bibliothèque des modes, la famille de modes opérationnels.
    1. Pour chaque composant c faire :
    2. Sélectionner le mode opérationnels physiques.
    3. Identifier ces états.
    4. Affecter les modes et états sélectionnés au $niveau_{formes\ physiques}$ dans la case *modes* (Tableau 16).

Opérations à réaliser par le concepteur

- Pour chaque niveau d'abstraction k du tableau WDA selon le contexte, faire :
- Sélectionner la famille de modes $FM_i \in \{FM_1, \dots, FM_n\}$. en se basant sur la guideline.
    - Pour chaque mode $M_{i,j} \in \{M_{i,1}, \dots, M_{i,k}\}$ de la famille de modes FM_i faire :

- Identifier chaque état $s_j \in S_{M_{i,j}}$ de ce mode en se basant sur la guideline.
- Sélectionner s_j.

Règle générale de cette étape :

$$\forall\, niveau_k, k =$$
$$\{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques, fonctions\ physiques, formes\ physiques\}$$
$$\exists\, FM_i \in \{FM_1, \dots, FM_n\},$$
$$\forall\, M_{i,j} \in \{M_{i,1}, \dots, M_{i,k}\},$$
$$\exists\, s_j \in S_{M_{i,j}} :$$
$$selectionner_{s_j}$$
$$affectation_{M_{i,j}, s_j, niveau_k} = 1$$

4.2.1.3 Sorties

Modèle conceptuel de modes selon le contexte.

4.2.2 Phase 3 : identification et affectation des contraintes du domaine

L'objectif de cette phase est d'extraire les contraintes du domaine et de les affecter aux différents niveaux d'abstraction du modèle conceptuel des modes et contraintes issu de la phase précédente.

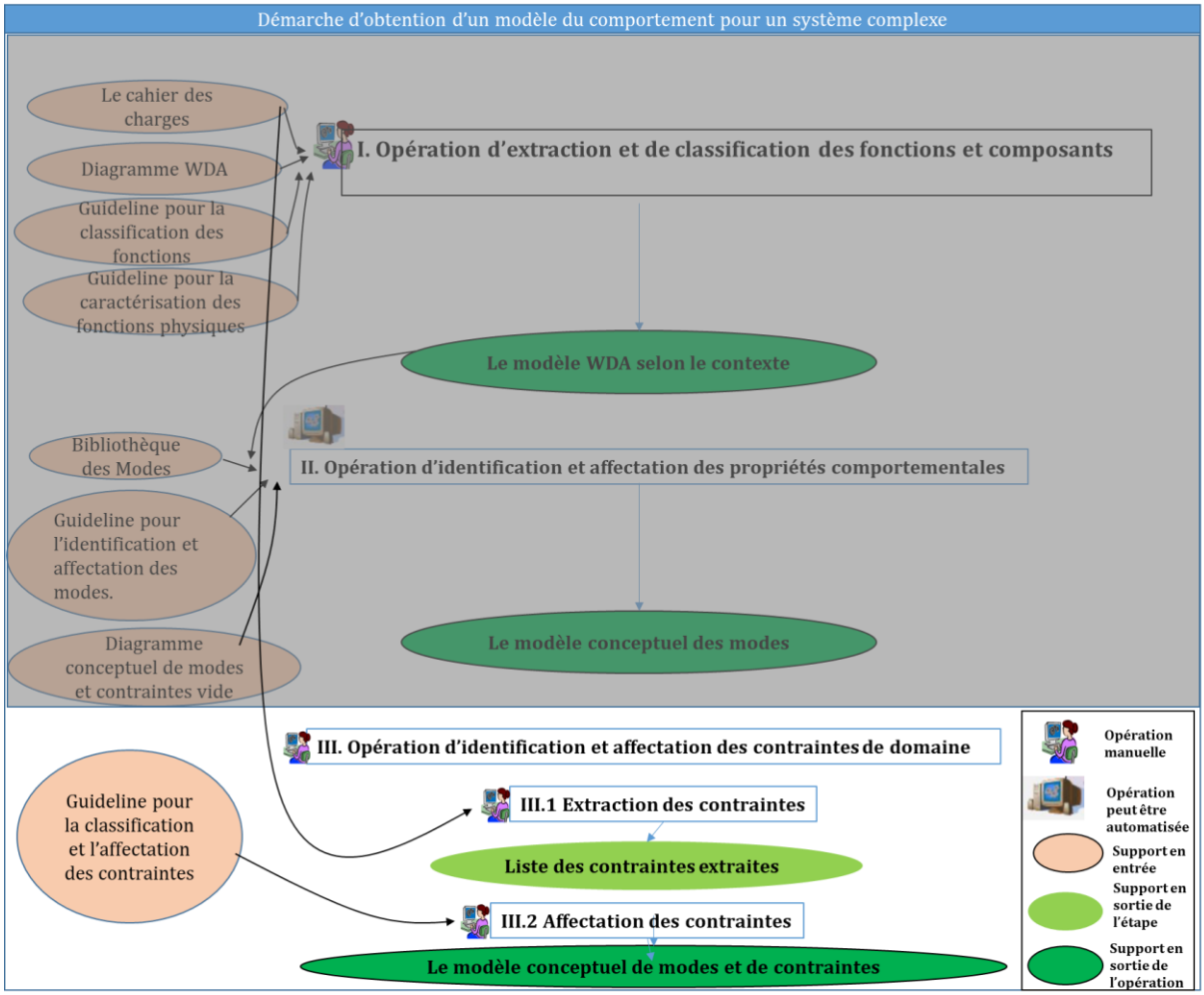


Figure 73 : Entrées/Sorties de l'opération d'affectation des contraintes.

4.2.2.1 Etape 1 : Extraction des contraintes

Lors de cette étape, nous procédons à l'extraction des contraintes du domaine à partir du cahier des charges. Nous définissons l'ensemble de contraintes du cahier des charges $ContraintesCDC = \{Contrainte_1, \dots, Contrainte_n\}$.

I.1.1.1.1 Entrées

Cahier des charges

I.1.1.1.2 Opérations à réaliser par le concepteur

Une contrainte du domaine est une contrainte qui limite la valeur d'une donnée.

A. Pour chaque contrainte $Contrainte_i$ du cahier des charges $ContraintesCDC$, faire :
   a. Extraire la $Contrainte_i$.
   b. Sauvegarder la $Contrainte_i$ dans la liste des contraintes extraites $ContraintesE = \{Contrainte_1, \dots, Contrainte_n\}$. .

I.1.1.1.3 Sorties

Liste de contraintes extraites $ContraintesE$.

I.1.1.2 Etape 2 : Affectation des contraintes

Toutes les contraintes extraites seront affectées aux différents niveaux d'abstraction du modèle conceptuel de modes et de contraintes.

I.1.1.2.1 Entrées

Liste de contraintes extraites $ContraintesE$.

Modèle conceptuel de modes et de contraintes.

Guideline pour la classification et l'affectation des contraintes.

	Système	Sous-système	Composant
Objectif fonctionnel	Quelles sont les contraintes du domaine nécessaires pour l'accomplissement des objectifs ? Quelles sont les contraintes pour avoir l'environnement adéquat à la réalisation des objectifs ?		
Fonction abstraite	Quelles sont les contraintes imposées par le contexte ?		
Fonction générique		Quelles sont les contraintes nécessaires à la réalisation de ces fonctions ?	
Fonction physique			Quelles sont les contraintes appliquées sur les composants pour réaliser les fonctions physiques ?
Formes physiques			Quelles sont les contraintes qui permettent de conditionner l'utilisation des composants ?

Tableau 17 : guideline pour l'affectation des contraintes du domaine.

I.1.1.2.2 Opérations à réaliser par le concepteur

A. Pour chaque niveau d'abstraction $niveau_k, k = \{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques, fonctions\ physiques, form$ faire :

a. Se positionner dans la guideline au même niveau d'abstraction $niveau_k$.
b. Répondre aux questions de la guideline de ce niveau $niveau_k$.
c. Si une réponse correspond à une contrainte $contrainte_i, (i = 1, \dots, n)$ de la liste des contraintes extraites $ContraintesE$, faire :
    1. Récupérer la $contrainte_i$.
    2. Placer cette contrainte dans le modèle conceptuel des modes et contraintes à la case *contraintes* du niveau d'abstraction $niveau_k$.

   Cette affectation est définie comme suit :

Règle générale de cette étape :

$$\forall contrainte_i \in ContraintesE, (i = 1, \dots, n)$$

$$affectation_{contrainte_i, niveau_k} = \begin{cases} 1 \; si \; contrainte_i \; correspond \; à \; une \; réponse \; des \; questions \; de \; guideline \; du \; niveau_k \\ 0 \; sinon \end{cases}$$

I.1.1.2.3 Sorties

Modèle conceptuel de modes et de contraintes rempli.

4.2.3 Phase 4 : Adaptation de la matrice de cohérence des modes

L'objectif de cette phase est d'adapter la matrice de cohérence des modes (compatibilité entre états) selon le contexte.

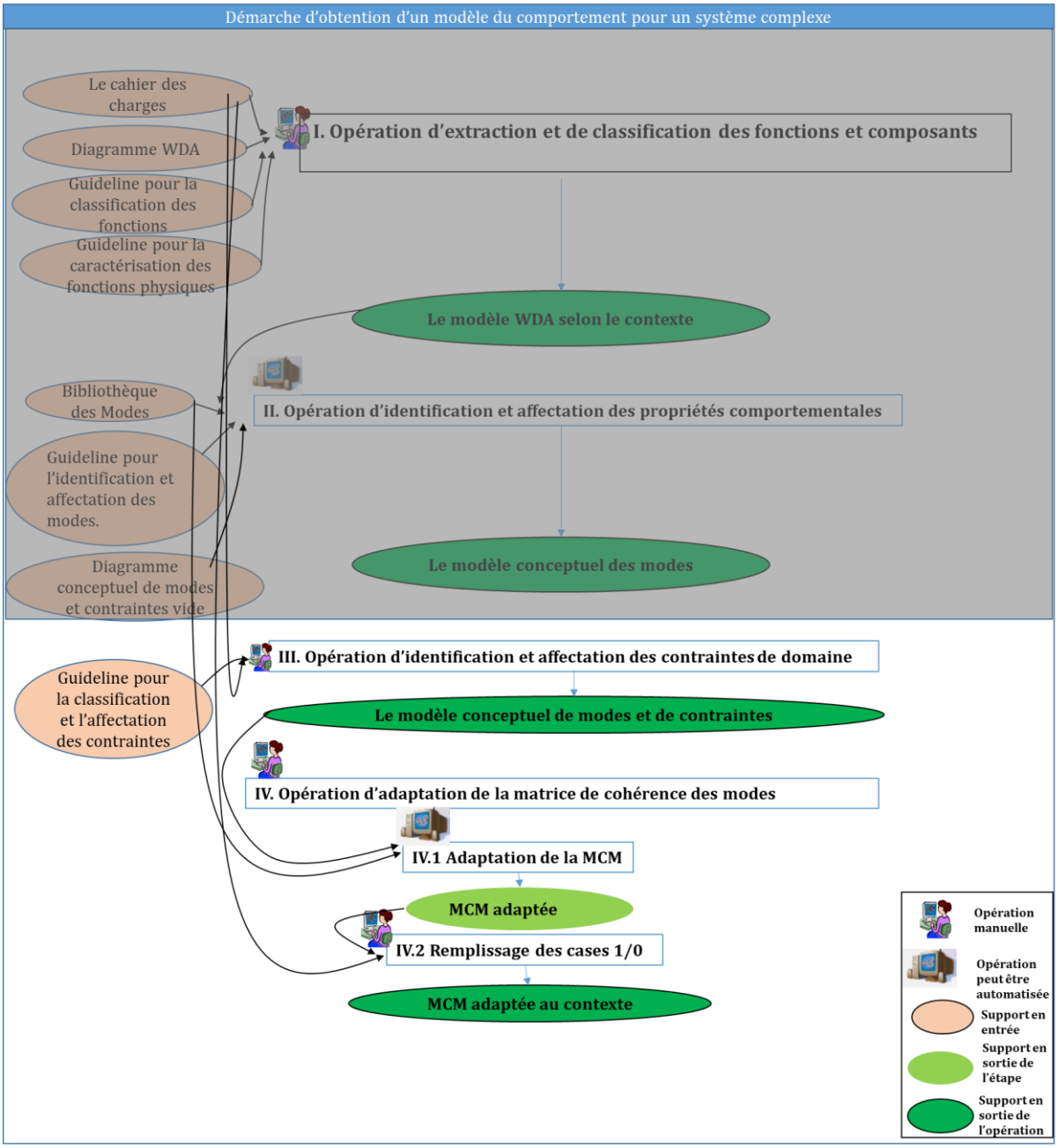


Figure 74 : Entrées/Sorties de l'opération adaptation de la matrice de cohérences des modes.

I.1.1.3 Etape 1 : Adaptation de la MCM

L'objectif de cette étape est de supprimer les lignes et les colonnes correspondants à chaque état qui n'a pas été sélectionné dans la phase 4.2. C'est-à-dire : supprimer de la MCM, tous les états qui n'apparaissent pas dans le modèle conceptuel de modes et de contraintes.

I.1.1.3.1 Entrées

Modèle conceptuel de modes et de contraintes.

La bibliothèque de modes.

I.1.1.3.2 Opérations à réaliser par le concepteur

$$\forall FM_i \in \{FM_1, \dots, FM_n\},$$

$$si\ affectation_{FM_i, niveau_k} = 0\ \forall\ niveau_k, k = \{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques, fonctions\ physiques, formes\ physiques\}$$

$$supprimer_{FM_i MCM}$$

Sinon :

$$\forall\ M_{i,j} \in \{M_{i,1}, \dots, M_{i,k}\},$$

$$, si\ affectation_{M_{i,j}, niveau_k} = 0\ \forall\ niveau_k, k = \{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques, fonctions\ physiques, formes\ physiques\}$$

$$supprimer_{M_{i,j} MCM}$$

Sinon

Si

$\exists S_j \in S_{M_{i,j}}$ et $affectation_{S_j, niveau_k} = 0$

$$\forall\ niveau_k, k = \{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques, fonctions\ physiques, formes\ physiques\}$$

Alors

$$supprimer_{S_j MCM}$$

I.1.1.3.3 Sorties

MCM adaptée au contexte.

		A							M					
		A-ns	A-d	A-i	A-ht	A-fc	A-u	A-mr	M-ns	M-a	M-sa	M-pp	M-m	M-ma
A	A-ns	1	0	0	0	0	0	0	0	1	1	1	1	1
	A-d		1	0	0	0	0	0	0	1/0	1/0	1/0	1/0	1/0
	A-i			1	0	0	0	0	1	0	0	0	0	0
	A-ht				1	0	0	0	1	0	0	0	0	0
	A-fc					1	0	0	0	1/0	1/0	1/0	1/0	1/0
	A-u						1	0	0	1/0	1/0	1/0	1/0	1/0
	A-mr							1	0	1/0	1/0	1/0	1/0	1/0

Tableau 18 : MCM adaptée au contexte.

I.1.1.4 Etape 2 : Remplissage des cases 1/0

I.1.1.4.1 Entrées

MCM adaptée au contexte.

Cahier des charges.

I.1.1.4.2 Opérations à réaliser par le concepteur

A. Pour chaque $Case_{i,j}\ \forall\ i,j\ \in \{1,\dots,n)$ de la MCM faire :
   a. Si $Case_{i,j} = 1/0$ faire :
      1. Remplir la $Case_{i,j}$ par 1 si la $ligne_i\ et\ colonne_j$ correspondent à deux états de deux modes différentes et sont compatibles.
      2. 0 Sinon.

$$\forall\ Case_{i,j}\ \forall\ i,j\ \in \{1,\dots,n)$$

$Case_{i,j} = 1/0$ Alors :

$$Case_{i,j} = \begin{cases} 1\ si\ ligne_i\ et\ colonne_j\ \ sont\ compatibles \\ 0\ sinon \end{cases}$$

I.1.1.4.3 Sorties

MCM adaptée au contexte.

4.2.4 Phase 5 : Adaptation des modèles de modes

L'objectif de cette opération est d'instancier les patterns de modes puis les adapter selon la MCM issue d la phase précédente.

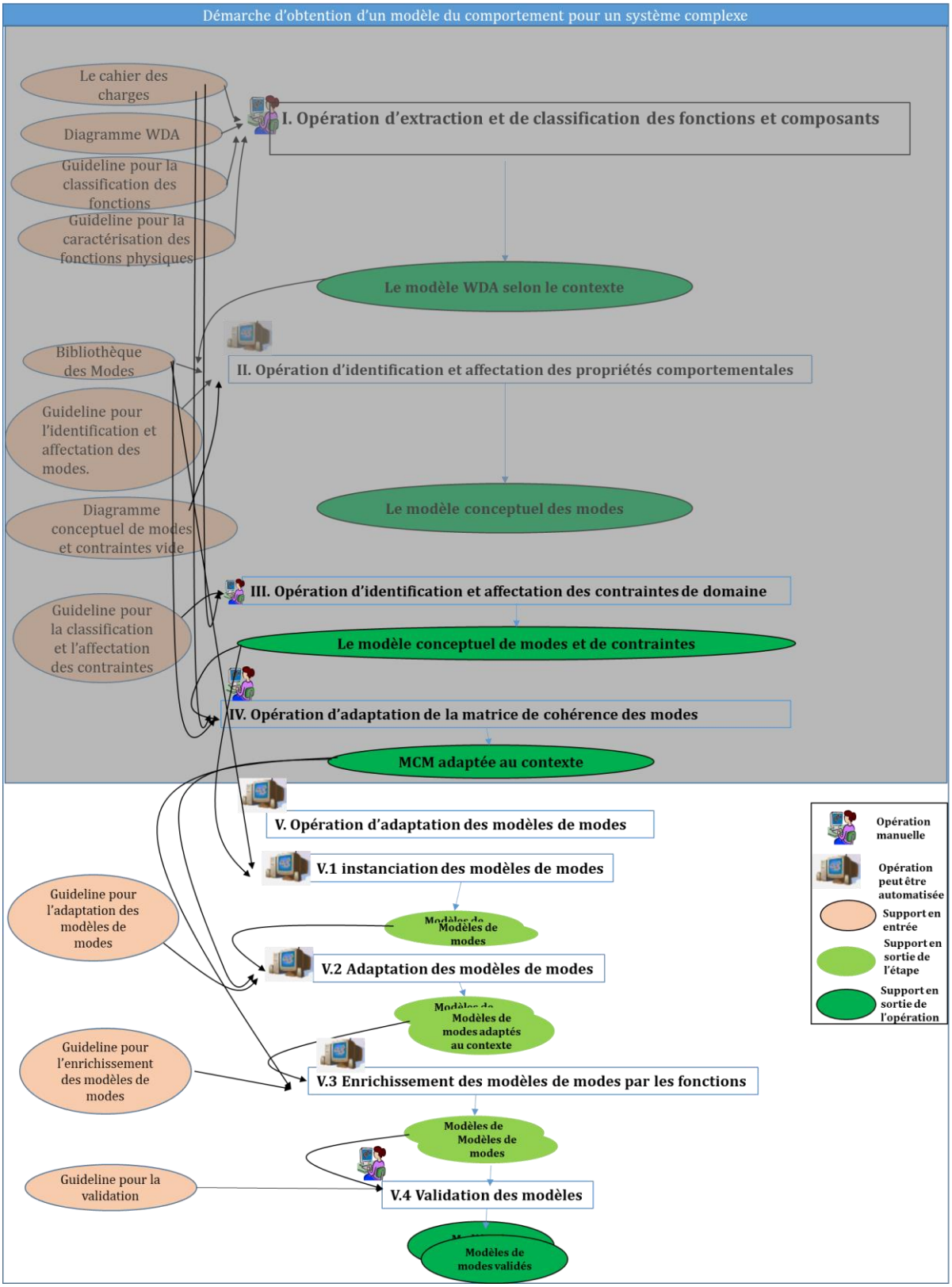


Figure 75 : Entrées/Sorties de l'opération d'adaptation des modèles de modes.

I.1.1.5 Etape 1 : Instanciation des patterns de modes

I.1.1.5.1 Entrées

Bibliothèque de modes.

Modèle conceptuel de modes et de contraintes.

I.1.1.5.2 Opérations à réaliser par le concepteur

A. Pour chaque mode $M_{i,j}\ \forall\, j\ \in \{1,\dots,n), \forall\, i\ \in \{1,\dots,m)$ existant dans le modèle conceptuel de modes et de contraintes faire :
   a. Instancier de la bibliothèque le pattern $P^{M_{i,j}}$.
      Formellement, cet automate est défini par (Faraut et al., 2008) comme suit :
      $P^{M_{i,j}} = (Q, \Sigma, \delta, q_0)$ Tel que :
      - Q : est l'ensemble des états du mode $M_{i,j}$.
      - Σ: Est l'ensemble des événements de commutation intra-mode.
      - $\delta : Q \times \Sigma \rightarrow Q$ Est la fonction de transition intra-mode.
      - q_0 Est l'état initial du mode.

$\forall\, M_{i,j}, i, j\ \in \{1,\dots,n)\ existant\ dans\ le\ modèle\ conceptuel\ de\ modes\ et\ de\ contraintes\ faire$: $Instancier\ P^{M_{i,j}}$ Tel que : $P^{M_{i,j}} = (Q, \Sigma, \delta, q_0)$

I.1.1.5.3 Sorties

Ensemble de patterns de modes instanciés.

I.1.1.6 Etape 2 : Adaptation des patterns de modes instanciés

Dans cette étape, on enrichit les transitions intra-modes par les conditions de compatibilité inter-modes.

I.1.1.6.1 Entrées

MCM adaptée au contexte.

Ensemble de patterns de modes instanciés.

I.1.1.6.2 Opérations à réaliser par le concepteur

B. Pour chaque $Case_{i,j}\ \forall\, i, j\ \in \{1,\dots,n)$ de la MCM faire :
C. Si $Case_{i,j} = 0$ faire :
   b. Si $l'état_i\ \in M_{i,k}$ est plus prioritaire que $l'état_j\ \in\ M_{j,u}$ faire :
   c. Modifier $P^{M_{i,k}} = (Q, \Sigma, \delta, q_0)$ comme suit :
      1. Ajouter la condition $état_i$ *sur toutes les transitions* $t\ \in \Sigma$ tel que : $\delta : l'état_j \times t \rightarrow q_0$ (toutes les transitions qui contiennent la condition $état_i$ mènent à l'état initial du mode $M_{i,k}$ qui est l'état non significatif)
      2. Ajouter la condition *not* $état_i$ *sur toutes les transitions* $t\ \in \Sigma$ tel que : $\delta : q_0 \times t \rightarrow l'état_j$ (toutes les transitions qui contiennent la condition *not* $état_i$ mènent à $l'état_j$ du mode $M_{i,k}$ à condition que $état_i$ n'est pas actif)

$\exists\, Case_{i,j}\ \forall\, i, j\ \in \{1,\dots,n)$: $Case_{i,j} = 0\ alors$:

Ajouter la condition $état_i$ *sur toutes les transitions* $t \in \Sigma$ tel que : $\delta : l'état_j \times t \rightarrow q_0$ Ajouter la condition *not* $état_i$ *sur toutes les transitions* $t \in \Sigma$ tel que : $\delta : q_0 \times t \rightarrow l'état_j$

I.1.1.6.3 Sorties

Ensemble de modèles de modes adaptés.

I.1.1.7 Etape 3 : Validation des modèles

Une fois que les modèles ont été instanciés, modifiés et enrichis, le concepteur aura besoin de confirmer la validité de ces modèles avant leur utilisation pour la conception. De fait, nous proposons une guideline pour faciliter cette opération.

I.1.1.7.1 Entrées

Ensemble de modèles de modes enrichis.

Guideline pour la validation.

Nous proposons deux types de validation : une validation numérique et une validation formelle.

I.1.1.7.2 Propriétés structurelles : après instanciation

La validation numérique est basée sur des calculs permettant de garantir la bonne construction des modèles.

I.1.1.7.2.1 Liaisons intra-modes

Un mode contient plusieurs états reliés entre eux par des transitions. Afin de garantir la bonne structure du modèle de mode, nous devons vérifier certaines propriétés:

I.1.1.7.2.1.1 La non existence d'états puits :

Le modèle de mode i ne contient pas d'état puits si et seulement si : dans la MCP de ce mode i, pour chaque ligne i, il existe au moins une colonne j, tel que : la condition $c_{i,j} \neq \emptyset$.

I.1.1.7.2.1.2 La non existence d'états non accessibles

Le modèle de mode i ne contient pas d'état inaccessible si et seulement si : dans la MCP de ce mode i, pour chaque colonne i, il existe au moins une ligne j, tel que : la condition $c_{i,j} \neq \emptyset$.

I.1.1.7.2.2 Liaisons inter-modes

Liaisons entre états de deux modes différents.

I.1.1.7.2.2.1 Parallélisme entre deux modes de même famille

Pour chaque famille de mode k, pour chaque deux modes différents i, j de cette famille de modes, il existe un état s du mode i, il existe un état s1 du mode j, tel que s et s1 sont compatibles.

$$i \neq j \; et \; i,j \in M_{k,l} \; \forall \, l = \{0, \dots, n\} : \exists \; s \in S_i, \exists \; s1 \in S_j \; tel \; que : compatibilité_{s,s1} = 1$$

I.1.1.7.2.2.2 Conditions de compatibilité entre deux modes

Pour chaque deux modes différents i et j : la somme de toutes les conditions de compatibilité de la ligne s *représentant l'état* s: *tel que* $s \in S_i$ avec toutes les colonnes représentant les états du mode j est différente de zéro. C'est-à-dire que l'état s doit être compatible avec au moins un état d'un autre mode j. (Dangoumau et al., 2000)

$$\sum_{k=0}^{|m_j|} MCM_{s_{i,k},s_{j,l}}^{(m_i,m_j)} \neq 0 \; et \; \sum_{l=0}^{|m_i|} MCM_{s_{i,k},s_{j,l}}^{(m_i,m_j)} \neq 0$$

I.1.1.8 Etape 4 : enrichissement des modèles de modes par les fonctions et contraintes

Dans cette étape, on enrichit les états des modèles de modes par les actions et contraintes du domaine.

I.1.1.8.1 Entrées

Ensemble de modèles de modes adaptés.

Modèle conceptuel de modes et de contraintes.

I.1.1.8.2 Opérations à réaliser par le concepteur

D. Pour chaque niveau d'abstraction $niveau_k, k = \{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques\}$ faire :
- Se positionner dans le modèle conceptuel de modes et de contraintes au même niveau d'abstraction $niveau_k$.
- Se positionner dans la familles de modes opératoires tel que : $affectation_{FM_{opéraoires}, niveau_k} = 1$

d. Pour chaque mode de la famille de modes opératoires $M_{i,famille\ de\ modes\ opéraoires}$ faire :
e. Prendre son modèle $P^{M_{i,famille\ de\ modes\ opéraoires}} = (Q, \Sigma, \delta, q_0)$.
f. Pour chaque état $S_h \in Q : S_h \neq q_0$ faire :
g. Ajouter les fonctions f_r du $niveau_k$ du mode $M_{i,famille\ de\ modes\ opéraoires}$ à l'état S_h comme actions comme suit :
h. $P^{M_{i,famille\ de\ modes\ opéraoires}} = (Q, \Sigma, \delta, F, \gamma, q_0)$ Tel que :
i. $F : l'ensemble\ des\ actions\ f_r$
j. $f_r \in \gamma(S_h)$.
k. Si deux fonctions sont séquentielles : $si\ \exists\ f_j, f_i \in F\ tel\ que : liaison_{f_i, f_j} = (f_i \rightarrow f_j \vee f_j \rightarrow f_i) faire$:
l. Ajouter la contrainte $fin\ f_i$ ou $fin\ f_j$ sur la transition qui active l'état exécutant la fonction f_i ou f_j.
m. Pour chaque $contrainte_h$ du $niveau_k$ faire :
n. Ajouter $contrainth$ à l'ensemble des conditions de transitions Σ tel que :
o. $\delta : contrainte_h \times S_r \rightarrow S_h$: c'est-à-dire la contrainte h est une condition pour la transition de tout état S_r à l'état S_h qui réalise les actions.

E. Se positionner au niveau d'abstraction $niveau_k, k = \{formes\ physiques\}$, faire :
p. Pour chaque mode opérationnel physique, faire :
q. Prendre son modèle $P^{M_{opérationnel\ physiques, famille\ de\ modes\ opérationnels}} = (Q, \Sigma, \delta, q_0)$.
r. Pour chaque état $S_h \in Q : S_h \neq q_0$ faire :
s. Ajouter les fonctions f_r du $niveau_k$ du mode $M_{opérationnel\ physique, famille\ de\ modes\ opérationnels}$ à l'état S_h comme actions comme suit :
t. $P^{M_{opérationnel\ physique, famille\ de\ modes\ opérationnels}} = (Q, \Sigma, \delta, F, \gamma, q_0)$ Tel que :
u. $F : l'ensemble\ des\ actions\ f_r$ Réalisés par le composant dans cet état.
v. $f_r \in \gamma(S_h)$
w. Pour chaque $contrainte_h$ du $niveau_k$ faire :
x. Ajouter $contrainth$ à l'ensemble des conditions de transitions Σ tel que :
y. $\delta : contrainte_h \times S_r \rightarrow S_h$: c'est-à-dire la contrainte h est une condition pour la transition de tout état S_r à l'état S_h qui réalise les actions.

I.1.1.8.3 Sorties

Modèles de modes enrichis.

4.2.5 Phase 6 : obtention du modèle formel

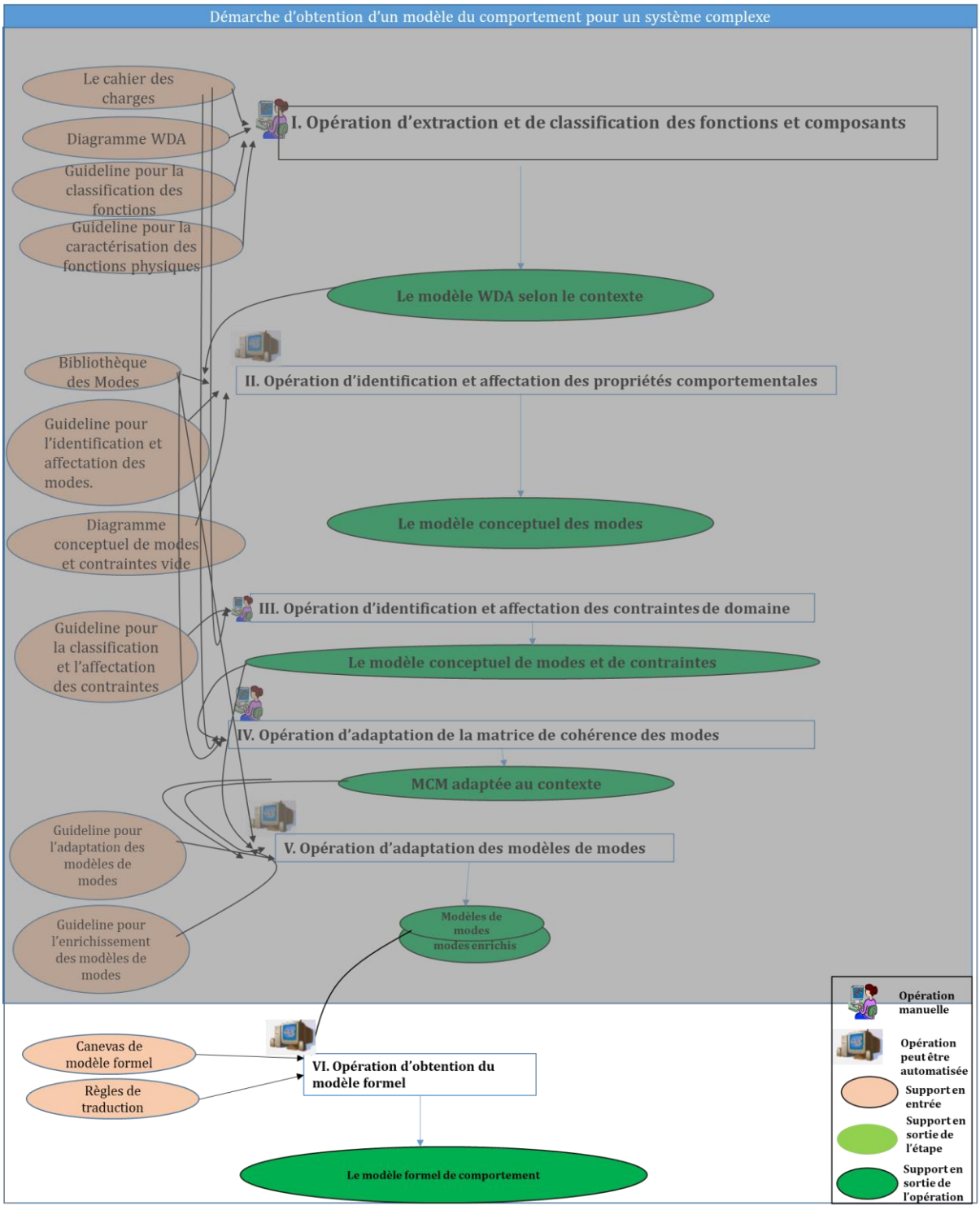


Figure 76 : Entrées/Sorties de l'opération d'obtention du modèle de comportement.

I.1.1.9 Besoin d'un canevas

L'utilisation des niveaux d'abstractions conjointement avec les modes et contraintes a permis de bien spécifier le comportement dynamique du système, en plusieurs modèles de modes. Cependant, ces modèles transitent entre eux pour permettre la réalisation des objectifs d'un

niveau jusqu'au niveau élémentaire. La combinaison de ces modèles doit alors suivre les mêmes niveaux d'abstraction pour permettre de garder les liens hiérarchiques entre les fonctions. La combinaison entre ces modèles doit aussi respecter la nature concurrentielle entre les modes.

I.1.1.10 Proposition d'un canevas

F. Créer un état orthogonal et nommer le Modèle du comportement.

G. Pour chaque niveau d'abstraction $niveau_k, k = \{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques, fonctions\ physiques, formes\ physiques\}$ faire :

H. Créer une région et nommer la par l'indice k.

I. Créer un état orthogonal et nommer le par l'indice k.

J. Si $niveau_k = niveau_{objectifs\ fonctionnels}$ faire :

- Créer une région et nommer la famille de mode de marche et d'arrêt.
- Créer un état orthogonal et nommer le, famille de mode de marche et d'arrêt.
  - Créer une région Mode d'arrêt.
  - Créer une région mode de marche.
- Créer une région famille de modes opératoires.
- Créer un état orthogonal et nommer le, famille de modes opératoires.
  - Créer une région Mode de préparation.
  - Créer une région mode de réalisation.
  - Créer une région Mode de validation.
  - Créer une région mode de clôture.
- Créer une région famille de modes opérationnels.
- Créer un état orthogonal et nommer le, famille de modes opérationnels.
  - Créer une région Mode de fonctionnement.

K. Si $niveau_k = niveau_{fonctions\ physiques}$ faire :

- Créer une région et nommer la famille de mode de marche et d'arrêt.
- Créer un état orthogonal et nommer le, famille de mode de marche et d'arrêt.
  - Créer une région Mode d'arrêt.
  - Créer une région mode de marche.
- Créer une région famille de modes opérationnels.
- Créer un état orthogonal et nommer le, famille de modes opérationnels.
  - Pour chaque composant faire :
  - Créer une région Mode opérationnel physique de ce composant.

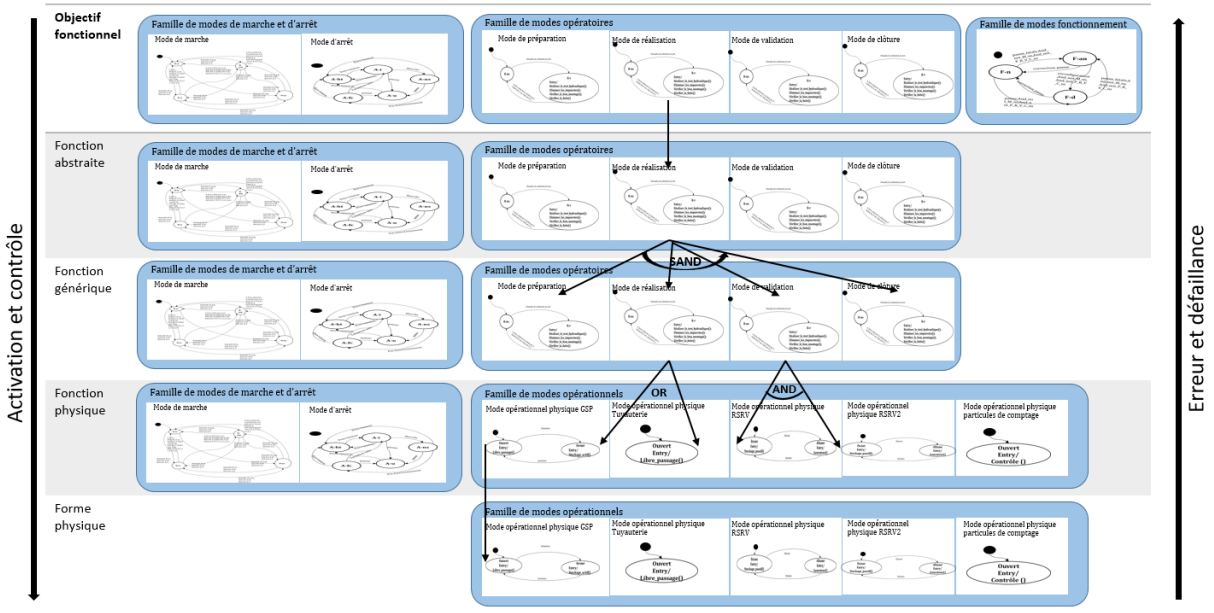


Figure 77 : Canevas proposé.

I.1.1.11 Etape d'obtention du modèle de comportement

I.1.1.11.1 Entrées

Modèles de modes.

Canevas.

I.1.1.11.2 Opérations à réaliser par le concepteur

L. Pour chaque niveau d'abstraction $niveau_k, k = \{objectifs\ fonctionnels, fonctions\ abstraites, fonctions\ génériques, fonctions\ physiques, formes\ physiques\}$ faire :
M. Pour chaque région de mode $M_{i,j}\ faire$:
N. Affecter $P^{M_{i,j}} = (Q, \Sigma, \delta, F, \gamma, q_0)$ à cette région.

Résumé

Les systèmes industriels sont généralement des systèmes complexes comportant plusieurs composants et offrant plusieurs fonctionnalités ou fonctions. Vu le nombre important des composants de ces systèmes, dans le monde industriel, des bancs de tests SCADA sont souvent développés pour les tester. Un banc de test SCADA (Supervisory Control and Data Acquisition) est un système SCADA qui a pour rôle de surveiller l'état du procédé et d'envoyer des commandes pour le manipuler. Pour répondre à un objectif de test sur ces bancs, les experts métiers se chargent du lancement des différentes fonctions de test. Pour cela, ils définissent un ordre précis de ces fonctions. Toutefois, certaines contraintes viennent compliquer la définition de cet ordre. En effet, les fonctions peuvent se réaliser en parallèle ou séquentiellement. D'autre part, certaines fonctions se partagent les mêmes composants et sont régies par les mêmes contraintes physiques comme la pression, le débit, etc. La définition manuelle de cet ordre est très fastidieuse et peut être une source d'erreur. Une fois l'ordre déterminé, les experts sont en charge d'identifier les séquences d'actions élémentaires permettant la réalisation de la fonction. Par exemple, afin de réaliser une fonction de transfert d'eau d'une soute à une autre, les experts sont obligés de réaliser la séquence d'actions élémentaire suivante : ouvrir les vannes, enclencher la pompe, choisir la soute de départ/d'arrivé, etc. Cependant, il faut s'assurer que les commandes envoyées tiennent compte des multiples contraintes de fonctionnement opérationnelles. De fait, la réalisation de ces actions peut devenir difficile et source d'erreur.

Pour répondre à ces problématiques, l'objectif des travaux de cette thèse consiste à proposer des solutions permettant l'optimisation des bancs de tests SCADA. Notre approche comporte trois contributions majeures.

4) Afin d'optimiser la réalisation des tests, notre première contribution porte sur la proportion une solution pour l'ordonnancement mathématique de deux machines parallèles dédiées avec une ressource partagée et des précédences locales. Des algorithmes et procédures ont été proposés afin d'ordonnancer l'exécution des fonctions d'un banc de tests SCADA.
5) Dans la deuxième contribution, nous proposons une approche pour la génération de séquences de commande permettant l'exécution opérationnelle de l'ordonnancement calculé. Basée sur les modes et états afin de spécifier, modéliser et déduire un modèle opérationnel pour l'exécution de l'ordonnancement sur un banc de test SCADA. Le modèle opérationnel est ensuite utilisé pour la génération automatique des séquences de commande permettant la réalisation de l'ordonnancement mathématique.
6) Dans la troisième contribution**,** nous proposons un xDSML qui permet d'outiller les deux premières contributions. Notre xDSML permet d'assister les experts non-informaticiens des systèmes SCADA dans le processus d'optimisation de leur système. Ce qui, permet d'une part de spécifier toutes les informations statiques du système SCADA telles que le P&ID, les fonctions et également la décomposition fonctionnelle. D'autre part, il permet la spécification des informations dynamiques comme les modes des composants et des fonctions.

Mots-clés : SCADA, Tests industriels, Ordonnancement, Méta-modélisation, Séquences de commande, Modélisation, Vérification formelle, Contrôle-Commande

Abstract

Industrial systems are usually complex, multi-component systems offering a wide range of functions. Given the large number of components in these systems, SCADA test benches are often developed in the industrial world to test them. A SCADA (Supervisory Control and Data Acquisition) test bench is a SCADA system whose role is to monitor process status and send commands to manipulate it. To meet a test objective on these benches, business experts take charge of launching the various test functions. To do this, they define a precise order for these functions. However, certain constraints complicate the definition of this order. Functions can be run in parallel or sequentially. On the other hand, some functions share the same

components and are governed by the same physical constraints, such as pressure, flow rate and so on. The manual definition of this order can be a source of error, and also very tedious. Once the order has been determined, the experts are responsible for identifying the elementary action sequences required to perform the function. For example, in order to carry out a function involving the transfer of water from one bunker to another, the experts have to carry out the following elementary sequence of actions: open the valves, switch on the pump, select the departure/arrival bunker, and so on. However, it must be ensured that the multiple commands sent take into account the multiple operating constraints. As a result, carrying out these actions can become difficult and error-prone.

To answer these problems, the objective of this thesis is to propose solutions allowing the optimization of SCADA test benches. Our approach has three major contributions.

1) In order to optimize the realization of the tests, our first contribution focuses on the proportion of a solution for the mathematical scheduling of two parallel dedicated machines with a shared resource and local precedents. Algorithms and procedures have been proposed to schedule the execution of the functions of a SCADA test bench.
2) In the second contribution, we propose an approach for the automatic generation of command sequences allowing the operational execution of the calculated scheduling. Based on modes and states in order to specify, model and deduce an operational model for executing scheduling on a SCADA test bench. The operational model is used for the automatic generation of control sequences allowing the realization of mathematical scheduling.
3) In the third contribution, we propose an xDSML that allows equiping the first two contributions. Our xDSML allows assisting the experts of SCADA systems in the optimization process of their system. This, on the one hand, allows to specify all the static information of the SCADA system such as the P&ID, the functions and also the functional decomposition. On the other hand, it allows the specification of dynamic information such as modes of components and functions.

Keywords : SCADA, Industrial test, Scheduling, Modelling, Operation sequences, Formal verification, Control-Command