



HAL
open science

Contribution à la modélisation et à la vérification de processus workflow

Zohra Sbaï

► **To cite this version:**

Zohra Sbaï. Contribution à la modélisation et à la vérification de processus workflow. Modélisation et simulation. Conservatoire national des arts et métiers - CNAM; École nationale d'ingénieurs de Tunis (Tunisie), 2010. Français. NNT: 2010CNAM0731 . tel-00553383

HAL Id: tel-00553383

<https://theses.hal.science/tel-00553383>

Submitted on 7 Jan 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS



Ecole Doctorale Informatique, Télécommunications et Electronique de Paris
[Centre d'Etudes et De Recherche en Informatique du CNAM]

THÈSE présentée par :
[Zohra SBAÏ]

soutenue le : 13 novembre 2010

pour obtenir le grade de : Docteur du Conservatoire National des Arts et Métiers
Discipline/ Spécialité : Informatique

**Contribution à la Modélisation et à la Vérification
de Processus Workflow**

THÈSE DIRIGÉE PAR :

BARKAOUI Kamel
BEN AYED Rahma

Professeur, CNAM
Maître de Conférences, ENIT

RAPPORTEURS :

TATA Samir
ROBBANA Riadh

Professeur, TELECOM SudParis
Titre, Ecole Polytechnique de Tunis

JURY :

JAÏDANE Mériem
GUEMARA Sihem
ROBBANA Riadh
TATA Samir
BARKAOUI Kamel
BEN AYED Rahma

Professeur, ENIT, Président du jury
Professeur, SUP'COM de Tunis
Professeur, Ecole Polytechnique de Tunis
Professeur, TELECOM SudParis
Professeur, CNAM Paris
Maître de conférences, ENIT

Remerciements

Les travaux présentés dans cette thèse ont été effectués au Laboratoire Systèmes de Communication de l'Ecole Nationale d'Ingénieurs de Tunis, et au Centre d'Etude et de Recherche en Informatique du Conservatoire National des Arts et Métiers de Paris.

Mes remerciements vont à Madame Mériem JAIDANE, Professeur à l'Ecole Nationale d'Ingénieurs de Tunis, pour le grand honneur qu'elle m'a fait en acceptant de présider ce jury de thèse. Qu'elle trouve ici le témoignage de ma respectueuse gratitude et mon profond respect.

Je tiens à exprimer ma gratitude envers Monsieur Riadh ROBBANA, Professeur à l'Ecole Polytechnique de Tunis, qui m'a honoré en acceptant de juger ce travail. Qu'il trouve ici l'expression de ma profonde gratitude et mon respect.

Je suis également très reconnaissante envers Monsieur Samir TATA, Professeur à TELECOM SudParis, qui m'a fait un grand honneur en acceptant de juger ce travail. Je tiens à lui exprimer toute ma vive reconnaissance pour l'analyse rigoureuse qu'il a menée à ce mémoire et qui a contribué à son enrichissement.

Je remercie Madame Sihem GUEMARA, Professeur à l'Ecole Supérieure des Communications de Tunis (SUP'COM), d'avoir accepté de faire partie des membres du jury. Qu'elle trouve ici l'expression de mon profond respect.

Je remercie très vivement Madame Rahma BEN AYED, Maître des Conférences à l'Ecole Nationale d'Ingénieurs de Tunis, pour m'avoir dirigé pendant ces travaux et pour m'avoir accordé de son temps précieux tout au long de ces années.

Ma profonde gratitude s'adresse à Monsieur Kamel BARKAOUI, Professeur au Conservatoire National des Arts et Métiers de Paris, pour m'avoir guidé tout au long de ces années, pour la confiance et les conseils avisés qu'il m'a accordés pendant ce travail.

Je remercie vivement mes amis et collègues membres du Laboratoire Sys'Com de l'ENIT et membres du Laboratoire CEDRIC du CNAM pour le climat favorable qu'ils m'ont fourni.

REMERCIEMENTS

Résumé

La technologie de workflow, tendant à automatiser les processus d'entreprise et à fournir un support pour leur gestion, est aujourd'hui un secteur actif de recherche. C'est dans ce contexte que se situent ces travaux de thèse qui portent aussi bien sur la modélisation des processus workflow que sur leur vérification. Ces processus, pouvant être contraints par des ressources partagées ou encore par des durées de traitement, doivent être vérifiés avant d'être confiés aux systèmes de gestion de workflow. Nous nous sommes intéressés par la vérification de la propriété de cohérence (soundness) des réseaux de workflow (WF-net) : sous-classes des réseaux de Petri (RdPs) modélisant les processus workflow.

Dans ce cadre, en explorant la théorie structurelle des RdPs, nous avons identifié des sous-classes de WF-nets pour lesquelles la cohérence peut être vérifiée et caractérisée efficacement. Nous nous sommes focalisés en outre sur l'extension de ces sous-classes en tenant compte de la présence de ressources partagées et sur la propriété de cohérence en présence d'un nombre arbitraire d'instances à exécuter. Dans cette partie, nous avons dû automatiser le calcul des siphons minimaux dans un RdP. Pour cela, nous avons choisi un algorithme de la littérature et l'amélioré par la recherche et la contraction de circuits alternés.

Ensuite, nous avons abordé la modélisation et la vérification de processus workflow tenant compte des contraintes temporelles. Nous avons en premier lieu proposé un modèle de TWF-net (WF-net Temporisé). Pour ce modèle, nous avons défini la propriété de cohérence temporelle et proposé une condition nécessaire et suffisante pour la vérifier. En deuxième lieu, nous avons relaxé les contraintes temporelles adoptées par la proposition d'un modèle temporel visant des processus à contraintes temporelles variant dans des intervalles de temps. Nous avons défini formellement le modèle de ITWF-net (Interval Timed WF-net) et donné sa sémantique. Par ailleurs, nous avons développé et testé un prototype de modélisation et de simulation des ITWF-nets.

La dernière partie de cette thèse a concerné la vérification formelle des processus workflow par SPIN model checker. Nous avons dû en premier lieu traduire la spécification des workflows adoptée vers Promela. En second lieu, nous avons exprimé les propriétés de cohérence en Logique Linéaire Temporelle (LTL) et utilisé SPIN pour tester si chaque propriété est satisfaite. Enfin, nous avons exprimé les propriétés de k-cohérence pour les WF-nets modélisant plusieurs instances et de (k,R)-cohérence pour les processus workflow concurrents et qui possèdent des ressources partagées.

Mots clés : Processus workflow, Modélisation, Vérification, Réseaux de Petri, Model Checking.

Abstract

Workflow technology, whose role is to automate business processes and to provide a support for their management, is today an active sector of research. This thesis deals with the modelling of the workflow processes and their analysis. These processes, probably constrained by shared resources or by durations of treatment, must be checked before being executed by their workflow management systems. In this direction, we were interested by the checking of the soundness property of workflow nets (WF-nets) : subclasses of Petri nets modelling the workflow processes.

To begin with, by exploring the structure theory of Petri nets, we have identified subclasses of WF-nets for which soundness can be checked and characterized effectively. We also extended these subclasses by taking account of the presence of shared resources and we focused on the soundness property in the presence of an arbitrary number of instances ready to be carried out. In this part, we had to automate the computation of minimal siphons in a Petri net. For that, we chose an algorithm of the literature and improved it by the research and the contraction of alternate circuits.

Then, we were concerned by the modelling and the analysis of workflow processes holding temporal constraints. We initially proposed the model of TWF-net (Timed WF-net). For this model, we defined its soundness and proposed a method to check it. Then, we relaxed the adopted temporal constraints by the proposal of a model covering workflow processes for which temporal constraints vary in time intervals. We formally defined the model of ITWF-net (Interval Timed WF-net) and gave its semantics. In addition, we developed and tested a prototype of modelling and simulation of ITWF-nets.

The last part of this thesis concerns the formal analysis of workflow processes with SPIN model checker. We initially translated the workflow specification into Promela : the model description language used by SPIN. Then, we expressed the soundness properties in Linear Temporal Logic (LTL) and used SPIN to test if each property is satisfied by the Promela model of a given WF-net. Moreover, we expressed the properties of k -soundness for WF-nets modelling several instances and (k,R) -soundness for competitive workflow processes which share resources.

Keywords : Workflow process, Modelling, Analysis, Petri nets, Model Checking.

ABSTRACT

Table des matières

Introduction	15
1 Contexte	16
1.1 Emergence des workflows	16
1.2 Motivations	16
2 Problématique	17
3 Objectifs de la thèse	17
3.1 Etude des lacunes d'un certain nombre de WfMSs	18
3.2 Caractérisation Structurale de la propriété de cohérence	18
3.3 Vérification structurelle de la propriété de cohérence	19
3.4 Modélisation et analyse des processus workflow à contraintes temporelles	19
3.5 Vérification par model checking des processus workflow	20
4 Plan du mémoire	20
1 Technologie du workflow et contribution pratique	23
1.1 Introduction	23
1.2 Processus métier	23
1.2.1 Modélisation de processus métier	25
1.2.2 Implémentation de processus métier	25
1.3 Technologies de processus métier	26
1.3.1 Techniques Ad hoc	26
1.3.2 Workflows	26
1.3.3 Services Web	26
1.3.4 Autres Approches	27
1.4 La technologie de workflow	28
1.4.1 Le choix du workflow	28
1.4.2 Définitions de base du workflow	29
1.4.3 Concepts et terminologie de workflow	29
1.5 Classification des systèmes workflow	31

TABLE DES MATIÈRES

1.6	Modèle de référence des systèmes workflow	32
1.7	Modélisation des processus workflow	34
1.7.1	Aspects à modéliser	34
1.7.2	Techniques et outils de modélisation de workflow	35
1.7.3	Les aspects temporels de workflow	37
1.7.4	Structure organisationnelle	38
1.8	Outils logiciels pour décrire et gérer un workflow	38
1.9	Présentation de l'outil jPnet	40
1.9.1	Mapping entre jPdl et RdP	40
1.9.2	Mapping entre PNML et RdP	44
1.10	Conclusion	47
2	Vérification des processus workflow	49
2.1	Introduction	49
2.2	Les réseaux workflow : modèle de base	49
2.2.1	Choix des réseaux de Petri comme langage de modélisation de workflow	50
2.2.2	Notions de base des réseaux de Petri	51
2.2.3	Les réseaux workflow (WF-nets)	54
2.3	Vérification paramétrée de la cohérence des WF-nets	54
2.3.1	La k-cohérence	55
2.3.2	La Cohérence généralisée	55
2.3.3	Caractérisation structurelle des K-workflow-nets (KWF-nets)	55
2.4	Vérification sous contraintes de ressources	60
2.4.1	La (k, R)-cohérence	60
2.4.2	La R-cohérence	62
2.4.3	La cohérence généralisée sous contraintes de ressources	62
2.5	Procédure de vérification structurelle de la cohérence des processus workflow	63
2.5.1	Calcul de tous les siphons minimaux dans un RdP	63
2.5.2	Description du nouvel algorithme	67
2.5.3	Complexité et tests de l'algorithme optimisé	77
2.5.4	Procédure de vérification de la CS-property	80
2.6	Conclusion	80
3	Workflows à contraintes temporelles	83
3.1	Introduction	83
3.2	Introduction du temps dans les RdPs	84
3.2.1	Trois sémantiques Différentes	84
3.2.2	Sous-classes des RdPs augmentés avec le temps	86

3.3	Réseaux de workflow temporisés (TWF-nets)	88
3.3.1	Définition d'un TWF-net	90
3.3.2	Sémantique des états et leur évolution dans les TWF-nets	91
3.3.3	Espace d'états	92
3.4	Analyse des TWF-nets	93
3.4.1	Vérification de la cohérence d'un TWF-net	94
3.4.2	Exemple de modélisation et d'analyse par TWF-net	95
3.4.3	TWF-nets modélisant k instances	96
3.4.4	Etats des TWF-nets modélisant k instances	97
3.4.5	k -cohérence des TWF-nets	98
3.5	Critiques des TWF-nets	100
3.5.1	Applications des TWF-nets	100
3.5.2	Limites des TWF-nets	101
3.6	Vers un modèle temporel plus général de processus workflow	102
3.6.1	Définition des Interval Timed WorkFlow-nets (ITWF-nets)	102
3.6.2	Comportement dynamique d'un ITWF-net	105
3.6.3	Analyse des ITWF-nets	108
3.7	Simulation des ITWF-nets	109
3.7.1	Implémentation d'un module de simulation des ITWF-nets	109
3.7.2	Etudes de cas	114
3.8	Conclusion	117
4	Vérification par Model Checking	119
4.1	Introduction	119
4.2	Model checking	120
4.2.1	Intérêt des méthodes formelles	120
4.2.2	Principe du model checking	120
4.2.3	SPIN Model Checker	122
4.2.4	Logique Temporelle	123
4.3	Un modèle Promela des processus workflow	123
4.4	Formulation des propriétés de cohérence d'un WF-net en LTL	126
4.4.1	Formules LTL des sous propriétés de la cohérence d'un processus workflow	128
4.4.2	Formule LTL de la cohérence d'un processus workflow	131
4.4.3	Cohérence faible	132
4.5	Formules LTL de la cohérence des processus workflow complexes	137
4.5.1	k -cohérence des WF-nets modélisant des processus concurrents	137
4.5.2	(k,R) -cohérence des WF-nets avec ressources partagées	140

TABLE DES MATIÈRES

4.6	Conclusion	144
Conclusion		147
1	Résumé des contributions	147
1.1	Modélisation et vérification structurelle des processus workflow	148
1.2	Modélisation des processus workflow à contraintes temporelles	149
1.3	Plateforme logicielle de modélisation des processus workflow	150
1.4	Vérification de processus workflow à l'aide du model checker SPIN	150
2	Perspectives de recherche	150
2.1	Intégration des ressources lors de la gestion des workflows temporels	150
2.2	Préservation de la cohérence lors de la composition des processus	151
2.3	Workflow Mining	151

Table des figures

1.1	Dimensions des systèmes workflow	31
1.2	Une classification des systèmes workflow	32
1.3	Modèle de référence : Composants et Interfaces des systèmes workflow	33
1.4	Un exemple de modèle organisationnel (Structure d'arbre)	38
1.5	Fonctionnalités basiques de l'outil jPnet	40
1.6	DTD relatif à la définition de processus jPdl	41
1.7	Traduction en WF-net de a- start-state ; b- state ; c- end-state	42
1.8	Traduction d'une séquence simple d'états et de transitions	43
1.9	Traduction de : a-fork ; b-join	43
1.10	Traduction de : a- decision ; b- milestone	43
1.11	Mapping de jPdl vers workflow-net	44
1.12	DTD relative à la définition de processus PNML	45
1.13	Mapping entre PNML et workflow-net	46
2.1	Un WF-net à choix libre non cohérent	56
2.2	Un RootWF-net cohérent	58
2.3	Un CCWF-net cohérent	59
2.4	Un OWF-net cohérent	60
2.5	Un CCWFR-net cohérent	64
2.6	Description de la procédure Forward-Deletion	65
2.7	Comparaison entre les algorithmes de recherche de circuits dans un graphe	69
2.8	Réseau de Petri N_1	74
2.9	Réseau de Petri N_3	75
2.10	Réseau de Petri dont les circuits alternés ont été contractés	75
2.11	Réductions pour le calcul des siphons minimaux d'un RdP	76
3.1	Livraison d'un ordre d'achat	90
3.2	Les WF-nets T-Temporisé et P-Temporisé sont équivalents	92
3.3	Exemple de TWF-net et son espace d'états accessibles	93

TABLE DES FIGURES

3.4	Processus workflow temporisé pour une demande de crédit	96
3.5	Espace des états accessibles du TWF-net de la demande de crédit	97
3.6	TWF-net d'un processus de déroulement d'un cours	101
3.7	Un ITWF-net du processus de relecture d'un papier dans une conférence . .	104
3.8	Modélisation d'un processus workflow temporel sous jPnet	110
3.9	Simulation d'un processus workflow temporel sous jPnet	111
3.10	Exemple de modélisation de processus workflow sous jPnet	114
3.11	Modélisation du processus de conférence sous jPnet	115
3.12	Le graphe d'accessibilité du processus de la conférence généré - Partie1 . . .	116
3.13	Le graphe d'accessibilité du processus de la conférence généré - Partie2 . . .	116
3.14	Modélisation du processus de PFAII sous jPnet	117
4.1	Approche automate du model checking.	121
4.2	Un exemple de WF-net	126
4.3	Modèle Promela du WF-net de la figure 4.2	127
4.4	Un automate de büchi correspondant à F	127
4.5	Un automate de büchi correspondant à la négation de F	128
4.6	Le never claim correspondant à F	128
4.7	Un WF-net résumant la gestion de la chaîne logistique	133
4.8	Spécification Promela d'un processus de gestion de la chaîne logistique . . .	134
4.9	Un WF-net résumant une demande de prêt d'un crédit	134
4.10	Vérification d'un processus de demande de crédit (1)	135
4.11	Vérification d'un processus de demande de crédit (2)	135
4.12	Un exemple d'un WF-net non cohérent qui satisfait la cohérence faible . . .	136
4.13	Vérification de la cohérence faible d'un exemple de WF-net	137
4.14	Un imprimé écran vérifiant la non cohérence d'un WF-net	138
4.15	Un contre-exemple montrant la non cohérence d'un WF-net	138
4.16	Un exemple d'un WF-net non 2-cohérent	139
4.17	La description Promela d'un WF-net non 2-cohérent	140
4.18	Un exemple d'un WFR-net	142
4.19	Une capture d'écran de la vérification par SPIN d'un WFR-net(1)	143
4.20	Une capture d'écran de la vérification par SPIN d'un WFR-net(2)	144

Introduction

Aujourd'hui, l'automatisation des processus d'entreprise est un défi important. Avant tout développement et mise en œuvre d'un système d'information, il est fondamental de bien maîtriser les enchaînements des opérations de ces processus.

Les activités dans un domaine fonctionnel d'entreprise peuvent être aussi bien manuelles, qu'automatisées et sont souvent déployées dans des systèmes matériels, logiciels et distribués. Généralement, chaque sous-système est développé selon une approche modulaire, de sorte que tous ces systèmes ou modules doivent communiquer et se coordonner pour réaliser leurs objectifs fonctionnels.

L'automatisation de l'exécution de la coordination de ces activités est appelée workflow. La WfMC (Coalition de Gestion de Workflow) définit un workflow comme étant "l'automatisation totale ou partielle d'un processus d'entreprise, au cours duquel on échange d'un participant à un autre, des documents, des informations ou des tâches pour action, et ce selon un ensemble de règles procédurales".

Ainsi, des techniques et outils de modélisation de processus ont permis le développement de nombreux systèmes de gestion de workflow prenant en charge partiellement ou entièrement la responsabilité de cette exécution coordonnée. Toutefois, la vérification de la cohérence de cette exécution, la prise en compte de l'allocation des ressources (partagées) nécessaires à l'exécution de ces activités, des contraintes temporelles et de la sécurité ne sont pas encore assurées au sein de ces systèmes.

En raison de la diversité de ces problèmes ouverts, la technologie de workflow est aujourd'hui un secteur actif de recherche. C'est dans ce cadre que se situe le sujet de cette thèse de doctorat.

Dans cette introduction nous présentons le contexte général de cette thèse et nous détaillons nos objectifs. Ces derniers portent en premier lieu sur la vérification des processus workflow en utilisant la théorie structurelle des réseaux de Petri ; en second lieu sur la prise en compte des contraintes complexes (ajout de ressources partagées, incorporation du facteur temps) dans la modélisation et l'analyse des processus workflow ; et en dernier lieu sur la vérification des processus workflow par model checking. Nous décrivons nos contributions dans ce sujet ainsi que l'organisation générale de ce mémoire.

1 Contexte

Un processus d'entreprise est défini par un ensemble d'activités reliées, qui réalisent en commun un objectif d'entreprise. La technologie de workflow tend à automatiser les processus d'entreprise et fournir un support pour leur gestion.

De manière informelle, un workflow est un travail coopératif impliquant un nombre limité de personnes devant accomplir, en un temps limité, des tâches articulées autour d'une procédure définie et ayant un objectif global.

1.1 Emergence des workflows

A l'origine, les systèmes d'information ont été conçus pour supporter l'exécution de tâches individuelles. Les systèmes d'information d'aujourd'hui ont besoin de supporter et de manipuler les processus d'entreprise. Il ne suffit plus de se concentrer uniquement sur les tâches. Le système d'information doit également contrôler, surveiller et soutenir les aspects logistiques d'un processus métier. En d'autres termes, le système d'information doit également gérer le flux de travail grâce à l'organisation. De nombreuses organisations possédant des processus d'entreprise complexes ont identifié le besoin de concepts, de techniques et d'outils d'aide à la gestion des workflows.

De nos jours, les applications de workflows sont multiples. En effet, le workflow joue un rôle important au niveau des entreprises financières telles que les systèmes bancaires et les compagnies d'assurance. Le workflow peut aussi être utile pour les sociétés fournissant des services informatiques, par exemple le processus de développement d'un logiciel.

D'autres organisations que les entreprises peuvent bénéficier des rôles de workflow. Dans ce contexte, nous citons comme exemples de workflow, le suivi d'un dossier médical d'un patient, la planification des opérations chirurgicales, etc.

1.2 Motivations

Plusieurs raisons expliquent l'intérêt accru accordé au processus métier. Premièrement, l'émergence de philosophies de gestion [7] telles que la reconstruction de processus métier (Business Process Reengineering (BPR) [55]) et l'amélioration continue des processus (Continuous Process Improvement (IPC)) sont à l'origine de la prise de conscience de la part des organisations de l'intérêt des processus métier. Deuxièmement, les organisations, de nos jours, devraient pouvoir fournir un large éventail de services et de produits. Par conséquent, le nombre de processus au sein des organisations a augmenté, et donc le nombre de produits et de services a également augmenté. En contrepartie, la durée de vie des produits et services a diminué au cours des trois dernières décennies [7].

En conséquence, les processus métier actuels sont également soumis à de fréquents changements. En outre, la complexité de ces processus a considérablement augmenté. Tous ces changements dans l'environnement du système d'information d'une organisation de taille moyenne, ont fait des processus métier un défi important dans le développement des systèmes d'information. Le besoin pour un nouveau composant, nommé Système de Gestion de Workflows (WfMS) paraît donc évident. Le rôle principal de ce WfMS est de

supporter la définition, l'exécution, l'enregistrement et le contrôle des processus.

2 Problématique

La définition et la gestion des processus workflow se fait à travers un système de gestion de workflow (WfMS). Or, en dépit de l'existence d'une variété de systèmes de gestion de workflows [1], nous ne disposons pas d'outils efficaces d'analyse et de vérification correcte et complète des processus workflows.

Afin de faire face à ce genre de problèmes, des travaux antérieurs ont montré l'intérêt de l'utilisation des réseaux de Petri (RdPs) [46, 88] pour la modélisation des processus workflows. Parmi les raisons d'utilisation des RdPs [7], nous citons d'abord leur sémantique formelle combinée avec un formalisme graphique, ensuite nous mentionnons leur traitement adéquat du flot de contrôle en terme d'états et d'événements et enfin nous évoquons leur richesse par les techniques d'analyse. La modélisation de processus workflow par RdP a conduit à la naissance d'une sous-classe particulière des RdPs dite workflow-net (WF-net) [7, 8] permettant de modéliser et d'analyser formellement le comportement de processus workflows.

Les WF-nets sont connus par la gestion efficace du flot de contrôle entre les différentes tâches modélisées, cependant ils ne permettent pas de représenter des processus complexes, comme l'exigent les WfMSs actuels. Ces processus sont essentiellement ceux mettant en œuvre des contraintes complexes comme la gestion du temps et des ressources, et le support des situations imprévisibles.

Pour la classe des WF-nets et en vue de la vérification des processus modélisés par cette classe, une propriété de cohérence a été définie. Cette propriété assure la terminaison de chaque instance de processus workflow. En plus de la terminaison, il ne doit pas y avoir d'inter-blocages ni de tâches inutiles. Ces conditions caractérisent la propriété de cohérence.

La définition de la propriété de cohérence est comportementale, sa vérification est donc cruciale surtout quand on dispose d'un processus workflow complexe. Nous concluons donc l'importance de nous investir dans l'étude, l'extension et la vérification de la cohérence des processus workflow qu'ils soient simples ou complexes.

3 Objectifs de la thèse

Pour contribuer à pallier aux lacunes mentionnées dans la section précédente, nous résumons les objectifs de cette thèse par les points suivants :

- Etude d'un certain nombre de systèmes de gestion de workflow actuels existant en open source. Cette étude a pour but de faire ressortir et ressentir les lacunes de tels systèmes.
- Caractérisation de la propriété de cohérence des WF-nets en utilisant la théorie structurelle des RdPs.
- Vérification structurelle de la propriété de cohérence.
- Modélisation et analyse des processus workflow à contraintes temporelles.
- Vérification par model checking des processus workflow.

Nous détaillons ces différentes contributions dans les sous sections qui suivent.

3.1 Etude des lacunes d'un certain nombre de WfMSs

La première phase de cette thèse a consisté en l'exploration de l'état de l'art sur les concepts utilisés ainsi que les produits existant sur le marché. Cette phase nous a permis de comprendre les principes des systèmes de gestion de workflow existants et essentiellement jBPM [17].

Au cours de cette étape de travail, nous avons pu rapidement remarquer l'absence de la composante qui s'intéresse à l'analyse des processus workflow qui seront exécutées par le WfMS. Par exemple, pour le cas de jBPM, il suffit de donner la description du processus dans la syntaxe du langage jPdl [66]. D'une part, le développeur de processus doit impérativement connaître le langage jPdl. D'autre part, le processus décrit ne sera pas analysé et par conséquent, s'il contient des erreurs, il sera géré et exécuté à l'aide de jBPM tout en comportant ces erreurs.

Pour pallier ce problème, nous avons étudié et analysé le moteur de workflow jBPM afin de dégager sa conception. L'étude de cette conception nous a amené à apporter des améliorations. La solution proposée consiste en la conception et l'implémentation d'un outil dénommé jPnet qui permet de faciliter la tâche du développeur de processus à travers l'introduction du processus d'entreprise sous forme de graphe de réseau de Petri (plus particulièrement workflow-net). Ce graphe sera traduit en un fichier XML conforme à la spécification du moteur de workflow étudié [40, 54, 120]. En plus, jPnet comble une des déficiences du moteur jBPM à savoir la vérification de la cohérence du processus introduit par le développeur de processus soit sous forme d'un réseau de Petri soit sous forme d'un fichier XML. Enfin, nous avons proposé une traduction du langage jPdl vers les réseaux de Petri. Notre approche facilite la vérification de la cohérence des processus d'entreprise donnés en jPdl dans le cas général. Pour passer d'un fichier écrit en langage jPdl à un graphe de réseau de Petri, nous avons effectué une analyse sémantique du fichier donné en jPdl tout en supposant qu'il est syntaxiquement correct.

3.2 Caractérisation Structurale de la propriété de cohérence

Cette contribution vise le problème de la vérification de la cohérence des processus workflows en faisant d'abord abstraction des ressources et en se focalisant ensuite sur l'influence des ressources sur la préservation de la cohérence.

En se basant sur des résultats avancés de la théorie structurale des réseaux de Petri [21], nous avons identifié de nouvelles classes de WF-nets : OWF-nets, RootWF-nets, CCWF-nets pour lesquelles la cohérence est structurellement caractérisable et décidable efficacement.

L'intérêt principal de ces classes est qu'elles permettent la prise en compte cohérente des ressources ainsi que la modélisation de patrons de synchronisation complexe rencontrés en pratique, en particulier dans le contexte des systèmes de gestion de workflow collaboratifs. Ce travail est une extension et généralisation des résultats décrits dans [4, 22, 58, 59].

3.3 Vérification structurelle de la propriété de cohérence

Les systèmes de gestion de workflows modernes doivent supporter de complexes contraintes (grande taille, partages des ressources, etc); d'où l'importance d'outils plus riches et plus efficaces pour la vérification de la cohérence des processus workflows gérés.

La caractérisation structurelle de la cohérence est basée sur une propriété importante qui est la CS-Property. Cette dernière signifie le contrôle de tous les siphons minimaux d'un RdP. L'investissement dans le calcul de tous les siphons minimaux d'un WF-net paraît donc important pour réduire le coût de la vérification de la propriété de cohérence.

Dans cette direction, nous avons été en mesure d'effectuer la recherche et l'optimisation d'un algorithme qui calcule tous les siphons minimaux. L'algorithme retenu a été optimisé par les notions de réduction et de compactage assurées suite à l'exploration du concept de circuits alternés.

L'implémentation de l'algorithme choisi avec des optimisations évidentes a été faite sur jPnet. Ceci a été accompagné par l'implémentation de toute l'approche de vérification structurelle de la cohérence d'un WF-net donné.

L'algorithme de recherche de circuits alternés et de compactage du réseau après chaque circuit trouvé a été également implémenté et testé.

3.4 Modélisation et analyse des processus workflow à contraintes temporelles

La gestion du temps est importante pour les processus d'entreprise et en même temps très complexe. Aujourd'hui, les WfMSs disponibles offrent une gestion de temps très limitée. Elle est faite principalement sous forme de calendrier ou de liste "to-do" pour une gestion de temps personnalisée incluant le contrôle des deadlines proches, la génération d'alertes et les déclencheurs d'une exception du processus spécifique quand un deadline est raté. Par contre, la dépendance temporelle entre deadlines n'est pas connue par les WfMSs. [82]

Après avoir effectué une étude et une classification des différentes extensions des RdPs par le facteur temps, nous avons proposé un modèle temporisé de workflow. Ce modèle consiste en l'extension des WF-nets par association d'une durée à chaque tâche. Pour ce modèle, que nous avons appelé Timed Workflow Net (TWF-net), nous avons défini sa sémantique en termes d'états et en termes de transitions entre ces états et par conséquent de l'espace des états accessibles.

Outre la définition formelle du modèle, nous avons défini les propriétés de cohérence et de k-cohérence d'un TWF-net et donné une condition nécessaire et suffisante (en fonction d'autres propriétés également définies) pour la vérification de chacune de ces propriétés.

Le modèle de TWF-net pouvant être limité dans certains cas à cause de son adoption de durées fixes d'exécution des transitions, nous nous sommes orientés vers la relaxation des contraintes temporelles adoptées en vue de la proposition d'un modèle temporel, plus général, de modélisation de processus workflow. Les processus nouvellement visés sont contraints par des durées de séjour (durée de rester dans un état bien défini). Nous avons défini en premier lieu formellement le modèle de ITWF-net (Interval Timed Workflow net) et donné sa sémantique de franchissement et d'évolution d'états. En second lieu, nous nous

sommes intéressés à l'analyse des ITWF-nets et leur simulation.

3.5 Vérification par model checking des processus workflow

Les tests et la simulation avaient depuis longtemps été utilisés dans la vérification de systèmes. Cependant, ces techniques permettent d'explorer uniquement une partie des comportements possibles. Elles diffèrent en cela des techniques de vérification formelle. En effet, ces dernières garantissent qu'une propriété est vérifiée par toutes les exécutions possibles du système.

Dans ce cadre, nous nous sommes orientés vers l'utilisation d'une des méthodes formelles puissantes en vue de la vérification des processus workflow qui est le model checking. Cette méthode regroupe plusieurs techniques entièrement automatiques dans lesquelles la propriété à vérifier est testée exhaustivement sur l'ensemble des exécutions possibles du système. Dans ce volet, nous avons utilisé SPIN model checker pour vérifier les propriétés de cohérence des processus workflow.

Nous avons en premier lieu traduit la spécification des processus workflow adoptés, à savoir les workflow nets, vers Promela : le langage de description de modèles à vérifier par SPIN. En second lieu, nous avons exprimé les propriétés de cohérence en Logique Linéaire Temporelle (LTL) et utilisé SPIN pour tester si chaque propriété est satisfaite par le modèle Promela du WF-net en question.

En outre, nous avons exprimé les propriétés de k -cohérence pour les WF-nets modélisant plusieurs instances et de (k, R) -cohérence pour les processus workflow concurrents et qui possèdent des ressources partagées. Nous avons par ailleurs testé tous les résultats trouvés sur des exemples concrets.

4 Plan du mémoire

Le présent mémoire est constitué de six chapitres. Le premier est représenté par cette introduction générale.

Le deuxième chapitre est essentiellement dédié à une introduction aux concepts de base et terminologie de workflow. Nous exposons d'abord les notions de processus métier, des technologies utilisées pour leur gestion et plus particulièrement de la technologie de workflow. Ensuite, nous étalons les techniques de modélisation de processus workflow. Enfin, après une brève présentation de quelques systèmes de gestion de workflow existants, nous présentons notre outil jPnet de modélisation et de vérification de processus workflow.

Dans le troisième chapitre, nous abordons notre approche de vérification des processus workflow. Nous donnons en premier lieu une caractérisation structurelle de la propriété de cohérence et ce en se basant sur la théorie structurelle des réseaux de Petri. Cette caractérisation est basée sur l'identification de nouvelles sous-classes de workflow-nets pour lesquelles la propriété de cohérence est fortement liée à la CS-Property (propriété de siphons contrôlés). Cette dernière nous a poussé à nous focaliser sur l'algorithmique de calcul des siphons d'un RdP : objectif de la dernière partie de ce chapitre.

Le quatrième chapitre est consacré à l'extension des processus workflow par l'ajout des

contraintes temporelles. Nous exposons en premier lieu les différentes extensions temporelles existantes des réseaux de Petri et nous détaillons en deuxième lieu les modèles que nous proposons.

Dans le cinquième chapitre, nous présentons notre approche de vérification des processus workflow par le model checker SPIN. Nous commençons par définir le modèle Promela de processus workflow à vérifier par SPIN. Ensuite, nous montrons comment formuler les propriétés de cohérence souhaitées par la Logique Linéaire Temporelle et nous les illustrons par des exemples. Enfin, nous exposons l'extension des résultats trouvés pour vérifier la cohérence des processus workflow possédant plusieurs instances prêtes à l'exécution et partageant des ressources.

Dans le dernier chapitre, nous récapitulons le travail réalisé et donnons quelques perspectives futures.

4. PLAN DU MÉMOIRE

Chapitre 1

Technologie du workflow et contribution pratique

1.1 Introduction

La modélisation par les processus dont les domaines d'application étaient relativement restreints à l'origine, n'a cessé de progresser. Elle s'impose, aujourd'hui, en tant qu'outil incontournable, placé au cœur de la gestion des organisations. La déclinaison des processus organisationnels au niveau des architectures logicielles a donné, entre autres, naissance au workflow afin de répondre à des besoins divers dont celui d'optimisation des processus opérationnels, support et de pilotage.

Ce chapitre est dédié à une étude détaillée de la notion de workflow d'une part et à la présentation de notre outil jPnet qui comble une des déficiences du WfMS jBPM d'autre part. Il est composé de trois parties principales. Dans la première partie, nous rappelons les définitions et la terminologie de base du workflow tout en se référant au modèle de la Coalition de Gestion de Workflow. Dans la deuxième partie, nous présentons les modèles existants en vue de la modélisation des processus workflow. Dans la dernière partie, nous proposons d'étudier des outils logiciels de description et de gestion de workflow, de cerner leur problématique majeure et de contribuer à pallier cette problématique. Notre contribution consiste en l'implémentation d'un outil de modélisation et de vérification des processus workflow spécifiés en WF-nets. Nous présentons dans la section 1.9 les fonctionnalités de notre outil qui ont été initialement conçues pour répondre au besoin des WfMSs étudiés à savoir la vérification de la cohérence des processus workflow avant de commencer leur exécution.

1.2 Processus métier

La gestion de processus métier est basée sur l'observation que chaque produit fourni par une société au marché est le fruit d'un nombre d'activités établies. Les processus métier constituent l'instrument clé pour organiser ces activités et prouver la compréhension de leurs relations.

La technologie de l'information en général et les systèmes d'information en particulier joue un rôle important dans la gestion de processus métier. En effet, plusieurs activités qu'une compagnie exécute sont supportées par des systèmes d'information. Les activités de processus métier peuvent être exécutées par les employés d'une compagnie manuellement ou bien à l'aide de systèmes d'information. On trouve aussi des activités de processus métier qui peuvent être exécutées automatiquement par les systèmes d'information, sans aucune intervention humaine.

Une entreprise peut atteindre ses objectifs métier d'une manière efficace et effective seulement si les humains et d'autres ressources d'entreprise, tels que les systèmes d'information, travaillent bien ensemble. Les processus métier forment un concept important pour faciliter cette collaboration effective.

Dans plusieurs compagnies, il y a une séparation entre les aspects métier organisationnels et la technologie d'information mise en place. Rétrécir cette séparation entre organisation et technologie est important, car dans les marchés dynamiques actuels, les entreprises sont constamment forcées à fournir des produits meilleurs et plus spécifiques pour leurs clients.

Au niveau organisationnel, les processus métier sont essentiels pour comprendre comment les compagnies opèrent. De plus, ils jouent un rôle important dans la conception et la réalisation de systèmes d'information flexibles. Ces systèmes d'information fournissent la base technique pour la création rapide de nouvelles fonctionnalités afin de réaliser de nouveaux produits en vue de la satisfaction de nouvelles exigences du marché.

La gestion des processus métier est influencée par les concepts et les technologies de différents domaines d'administration de l'entreprise et de l'informatique. Basée sur des travaux récents dans l'organisation et la gestion, la gestion des processus métier trouve ses racines dans la tendance de l'orientation processus des années quatre vingt dix, où un nouveau chemin d'organisation des compagnies se basant sur les processus métier a été proposé.

Un processus métier a été défini, par Michael Hammer et James Champy dans [56], comme une collection d'activités qui prennent un ou plusieurs types d'entrée et créent une sortie qui vaut une valeur pour le client. Cette définition met l'accent sur le comportement de l'entrée/sortie d'un processus métier en mentionnant ses pré-conditions (entrées) et ses post-conditions (sorties). Le processus lui-même est décrit de façon abstraite par une collection d'activités. Notons que le terme "collection" n'implique ni un classement des activités ni toutes autres contraintes d'exécution, la définition par Hammer et Champy est tout à fait libérale quant à l'aspect du processus.

Les contraintes d'exécution entre activités sont identifiées dans [43], qui définit un processus métier comme "un ensemble de tâches, logiquement reliées, exécutées afin d'accomplir un objectif métier pour un client particulier ou un marché".

En se basant sur ces caractérisations des processus métier, on adopte la définition suivante :

Définition 1.1 *Un processus métier est un ensemble d'activités qui sont exécutées en coordination dans un environnement organisationnel et technique. Ces activités réalisent en commun un objectif d'entreprise (commercial). Chaque processus métier est exécuté par*

1.2. PROCESSUS MÉTIER

une seule organisation, mais il peut interagir avec d'autres processus métier réalisées par d'autres organisations.

Après une première considération des processus métier, leurs composants, et leurs interactions, la vue est élargie. La gestion de processus métier ne couvre pas seulement les processus métier mais aussi d'autres activités.

Définition 1.2 *La gestion de processus métier inclut les concepts, les méthodes et les techniques pour supporter la conception, l'administration, la configuration, l'exécution et l'analyse de processus métier.*

La base de la gestion de processus métier est la représentation explicite de processus avec leurs activités et les contraintes de l'exécution entre elles. Une fois les processus métier sont définis, ils peuvent faire l'objet d'analyse, d'amélioration et d'exécution.

1.2.1 Modélisation de processus métier

Un modèle de processus métier consiste en un ensemble de modèles d'activités et de contraintes d'exécution entre eux. Un exemple de processus métier représente un cas concret dans le processus opérationnel d'une compagnie, consistant en des instances d'activités. Chaque modèle de processus métier opère comme un plan pour un ensemble d'instances du processus métier, et chaque modèle d'activité opère comme un plan pour un ensemble d'instances de l'activité.

Si aucune confusion n'est possible, le terme processus métier est utilisé pour se reporter à aussi bien au modèle de processus métier qu'à une instance de processus métier. D'une manière analogue, l'activité est utilisée pour faire référence au modèle de l'activité ou à une instance de l'activité. Les modèles du processus métier sont les objets principaux pour mettre en œuvre ces processus métier. Cette mise en œuvre peut être faite par des règles d'organisation et des politiques, elle peut aussi être faite par un système logiciel en utilisant un système de gestion de processus métier. Dans ce cas, ce système logiciel est conduit par des représentations explicites du processus.

Puisque par définition les processus métier sont exécutés dans une seule organisation, le cheminement d'activités peut être contrôlé par un système de gestion de processus métier comme un composant logiciel centralisé exécuté par le revendeur de la compagnie. Ce contrôle centralisé est très semblable à un conducteur qui contrôle, centralement, les musiciens dans un orchestre ; par conséquent, les processus métier sont aussi appelés des orchestrations de processus.

1.2.2 Implémentation de processus métier

Implémenter un processus métier, c'est utiliser les ressources logicielles et matérielles adéquates pour exécuter ses activités. Les activités sont implémentées comme des processus matériels et/ou électroniques. Les processus matériels incluent la production, la transformation, la mesure et l'assemblage d'objets physiques. Un exemple de processus matériel est l'assemblage de voitures. Les processus électroniques, d'un autre côté, impliquent des

systèmes logiciels pour créer, gérer et fournir de l'information. Un exemple de processus électronique est l'accès à des catalogues de produits d'une vente en ligne. Nous nous intéressons dans cette thèse aux activités qui peuvent être implémentées en tant que processus électroniques.

1.3 Technologies de processus métier

Construire des processus métier inter-organisationnels requière une intégration à des niveaux différents, et plus spécialement aux niveaux communication et processus métier. La couche communication est concernée par l'échange de messages entre partenaires. L'objectif d'intégration de cette couche est de permettre des interactions transparentes entre partenaires. Celle-ci, étant une tâche difficile, peut être assurée en utilisant les API (Application Programming Interface) et les workflows.

Dans cette section, nous présentons quelques technologies qui implémentent et gèrent les processus métier.

1.3.1 Techniques Ad hoc

Les processus métier peuvent être implémentés en utilisant les langages de programmation tels que C et Java. Dans cette approche, le processus métier est modélisé (par exemple comme un document Word). Ensuite, les activités principales sont implémentées par des programmes. L'interaction entre ces programmes est supportée par des interconnexions entre des protocoles de communication appropriés (par exemple, les sockets TCP/IP).

Le problème de cette solution est que tous les protocoles de communication utilisés doivent être comprises. En même temps, toute modification dans le protocole de communication ou la logique métier requière la recompilation des programmes.

1.3.2 Workflows

Un workflow [124] est l'automatisation d'un processus métier, en tout ou en partie, durant lequel des documents, informations, ou tâches sont passés d'un participant à un autre pour action, d'après un ensemble de règles procédurales [118]. Ayant la définition de workflow, il y a un grand nombre d'activités métier dans une organisation qui se classifient dans la catégorie de workflow. Elles incluent un ordre d'achat, une demande de crédit, une location de voitures, etc.

1.3.3 Services Web

Le concept de service Web est devenu récemment très populaire. Un service Web (ou simplement service) est une application autonome ou un composant, i.e., une fonctionnalité sémantiquement bien définie, uniquement identifiée par un identificateur uniforme de ressource (ou Uniform Resource Identifier URI). Comme exemples typiques de services Web, nous citons les réservations de billets en lignes, la gestion des relations entre clients, la facturation, la comptabilité et la chaîne de la provision.

Une définition plus précise des services Web est fournie par le dictionnaire Webopedia [125]. Il définit un service Web comme "une manière standardisée d'intégration des applications basées sur le Web en utilisant les standards ouverts XML, SOAP, WSDL, UDDI et les protocoles de transport de l'Internet. XML est utilisé pour représenter les données, SOAP pour transporter les données, WSDL pour décrire les services disponibles, et UDDI pour lister les fournisseurs de services et les services disponibles". Les services Web présentent l'application la plus concrète de SOA (Service Oriented Architecture).

D'après cette définition, nous retenons qu'un service Web est décrit dans un document WSDL (Web Service Description Language), précisant les méthodes (fonctionnalités) invoquées, leur signature et leurs points d'accès du service (URL, port, etc.). Ces méthodes peuvent être accessibles à travers des requêtes via une couche dédiée au transport de messages : SOAP (Simple Object Access Protocol). La requête et la réponse forment des messages XML transportés par HTTP (Hyper Text Transfer Protocol). Un service Web est référencé dans une sorte d'annuaire UDDI (Universal Description, Discovery and Integration) qui facilite aux utilisateurs l'accès à ce service. Un service Web peut être utilisé pour exporter des fonctionnalités à partir d'une application d'entreprise et pour les rendre accessibles via des protocoles standards. Le service Web forme donc l'interface d'accès et de dialogue avec l'application.

1.3.4 Autres Approches

Autres technologie de processus métier incluent Electronic Business XML et Web Ontology Language for Services. Elles sont brièvement décrites comme suit :

- **Electronic Business XML (ebXML)** : ebXML définit un ensemble de spécifications pour permettre des interactions B2B entre des organisations. La partie basique de l'infrastructure ebXML est le dépôt (ou repository). Il sauvegarde des informations d'entreprise importantes avec les produits et services qu'elles offrent. A la couche communication, l'échange de messages se fait à travers le service de messagerie. Le dernier permet l'utilisation de protocoles communs tels que SMTP, HTTP et FTP. A la couche de processus métier, ebXML définit un schéma de la spécification du processus métier disponible en UML ou XML. Elle fournit un ensemble de spécifications de processus métier qui sont partagées entre plusieurs industries en vue de construire des processus métier personnalisés. Les interactions entre les processus métier sont représentées à travers les chorégraphies. Une chorégraphie spécifie l'ordre et les transitions entre les transactions métier. Pour modéliser la collaboration, ebXML définit les Collaborations Protocol Agreements (CPAs). Un CPA est un accord entre deux partenaires qui spécifie d'avance les conditions sous lesquelles ils vont collaborer (par exemple termes de paiement).
- **Web Ontology Language for Services (OWL-S)** (connu anciennement par DAML-S) vise à réaliser une vision de Web sémantique pour rendre les ressources Web disponibles aussi bien par contenu que par mots clés. OWL-S organise les services Web en ontologies. Une ontologie est définie par une spécification formelle et explicite d'une conception partagée [50]. OWL-S divise les descriptions de service en profil de service, modèle et fondement. Le profil de service fournit une description haut niveau d'un service Web. Il exprime l'entrée requise du service et les sortie que le

service va fournir au demandeur. Le modèle de service définit les opérations et leurs flots d'exécution dans le service Web. Le fondement de service fournit un mapping entre OWL-S et le standard WSDL et décrit comment le service est invoqué.

1.4 La technologie de workflow

1.4.1 Le choix du workflow

Le workflow est un outil de groupe qui fait appel à un groupe d'acteurs de l'entreprise. Il s'intéresse surtout aux processus administratifs qui traitent un événement extérieur dès son apparition jusqu'à son traitement complet. Une application de workflow connaît les tâches et la procédure à appliquer pour les cas qu'elle sait traiter. Ainsi, elle a pour rôle de décomposer les cas en tâches, et d'affecter chaque tâche à un acteur selon les règles de gestion prédéfinies. L'introduction du workflow constitue pour cette raison un changement capital dans la façon d'assister les entreprises. En effet, le workflow s'intéresse directement aux processus dans leur intégralité, à leur suivi, à l'affectation des tâches aux acteurs, au suivi des échéances et au traitement des exceptions. Ceci est accompagné par l'enregistrement des données permettant d'analyser les coûts, les charges et la qualité essentiels à une approche industrielle. Une application de workflow, associée le plus souvent à une application dédiée à la gestion électronique de documents, amène des gains de productivité allant de 20 à 50% sur la part des tâches automatisées, et permet de réduire des délais de 30 à 90% [8].

Les apports du workflow sont multiples. En effet, un workflow permet de :

1. Représenter la structure d'un groupe en termes d'acteurs, de groupes d'acteurs et de rôles tenus par chaque acteur. C'est une représentation de l'organisation du groupe.
2. Assurer le traitement de chaque cas en conformité avec les procédures de l'entreprise. Il "applique" les procédures.
3. Préparer les tâches à exécuter au fur et à mesure de l'avancement des cas. Il assure ainsi la planification du travail.
4. Attribuer les tâches aux acteurs en fonction des critères prédéfinis de rôle, d'appartenance aux divers groupes, et de charge. Il peut assurer la répartition automatique du travail.
5. Enchaîner automatiquement les appels aux outils bureautiques ou informatiques que requiert chaque tâche. Il assiste ainsi chaque acteur dans l'exécution de son travail.
6. Maintenir et rendre immédiatement accessibles tous les documents utiles au traitement de chaque cas. Il assiste ainsi les acteurs en regroupant autour de chacune des tâches tous les éléments qui lui sont pertinents.
7. Tenir une comptabilité détaillée de l'exécution de chaque tâche. Il permet de fournir des statistiques riches sur les procédures et l'activité des acteurs.

1.4.2 Définitions de base du workflow

Face à un grand nombre de produits workflow, chacun utilisant sa propre terminologie, un groupe de vendeurs et de consultants ont créé en 1993 la coalition de gestion de workflow (WfMC acronyme de Workflow Management Coalition). La WfMC regroupe plus de 250 vendeurs, utilisateurs, consultants et institutions de recherche qui ont intérêt au domaine de la gestion de workflow. Cette coalition a pour but de promouvoir l'utilisation de workflow par l'établissement de standards pour les systèmes de gestion de workflow. Ceci a été concrétisé par la publication de glossaires de référence comportant les terminologies utilisées dans le domaine du workflow. Pour le reste de cette section, nous nous sommes essentiellement référés au glossaire "WFMC-TC-1011" [118] pour définir les termes qui nous intéressent.

Avant de définir le terme workflow, il est à noter qu'un problème de confusion persiste entre les termes : workflow (processus workflow), technologie workflow et système workflow. En ce qui suit, nous allons définir chacun de ces termes [118].

Définition 1.3 *Un **workflow** est la forme exécutable d'un processus d'une organisation, gérable par un système workflow. Il permet d'automatiser l'exécution du processus ou encore sa simulation.*

Définition 1.4 *Un **système workflow** (ou WfMS pour Système de Gestion de Workflow) est un système informatique permettant la gestion des processus métiers. Les services proposés par un WfMS sont au minimum l'exécution d'un processus et sa gestion (contrôle et suivi) en plus de la mise à disposition des outils et des documents nécessaires à la réalisation des différentes étapes du processus.*

Définition 1.5 *La **technologie workflow** est la technologie informatique du TCAO (Travail Coopératif Assisté par Ordinateur), qui s'intéresse à la gestion des processus de l'organisation. C'est l'ensemble des moyens utilisés pour automatiser et gérer un processus. Cette gestion est garantie vu qu'il est possible de présenter un modèle de processus sous une forme exécutable.*

La relation entre ces trois termes est la suivante : Une entreprise peut introduire une **technologie workflow** dans son système d'information en installant un **système de gestion de workflow** qui gère ses processus, automatisés en **workflow**.

1.4.3 Concepts et terminologie de workflow

Nous proposons dans cette section de rappeler les termes de base liés aux processus workflows. De nombreuses publications proposent une terminologie relative aux concepts ainsi que les relations entre eux. Les concepts définis par la WfMC [118], puis affinés par van der Aalst et Van Hee dans [1], sont les plus largement appliqués dans la communauté de gestion des processus métier. La liste suivante présente les concepts de base de workflow et les structures de base pour la conception de workflow et le contrôle de processus comme le suggère la WfMC :

- Une **activité** (tâche) est une description d'une partie du travail qui constitue une étape logique dans un workflow. Elle peut être manuelle ou automatique [1]. Une activité manuelle est entièrement réalisée par une ou plusieurs personnes, sans aucune utilisation d'une application. En revanche, une activité automatique est effectuée par une application, sans aucune intervention des personnes, en se basant sur des données déjà enregistrées. Les activités sont classées en fonction des mutuelles dépendances imposées par des aspects structurels et de données (flot de contrôle et flot de données entre les activités). Différentes configurations permettent de couvrir les aspects structurels : séquence, sélection, itération, et concurrence. Pour la représentation des données, deux approches sont les plus utilisées : soit par le biais des flux de données entre les activités, soit par l'intermédiaire des services de fourniture des données des (ou vers les) activités .
- Une **instance** (instance de workflow (un cas) ou instance d'activité) est la représentation d'une exécution unique d'un workflow ou d'une activité dans un workflow.
- Un **participant** (acteur, agent, utilisateur, entité de traitement, ressource) est une entité qui exécute une instance d'activité. Cette entité peut être un être humain ou un système logiciel.
- Un **élément de travail (work-item)** est la représentation du travail à traiter (par un participant) dans le cadre d'une activité d'une instance de workflow. Une liste des éléments de travail associée avec un participant de workflow donné (ou groupe de participants) est appelé une **liste de travail (work-list)**.
- Un **état** de workflow (resp. d'activité) est lié à des conditions internes définissant l'état d'une instance du workflow (resp. de l'activité) à un moment donné. Dans le cas d'un workflow, l'état pourrait être "initié", "en exécution", "actif", "suspendu", "achevé", "terminé" et "archivé". Dans le cas d'une activité, il pourrait être "inactive", "active", "en exécution", "suspendue", "sautée" et "terminée".

En résumé, nous distinguons dans un workflow des cas, des éléments de travail (work-items) et des ressources. Les work-items lient les cas et les tâches, les activités lient les cas, les tâches et les ressources.

La figure 1.1 [7] montre qu'un workflow comporte trois dimensions : (1) la dimension de cas, (2) la dimension du processus et (3) la dimension des ressources. La dimension de cas signifie le fait que tous les cas sont traités individuellement. Du point de vue workflow, les cas ne s'influencent pas des autres, mais ils s'influencent les uns des autres indirectement via le partage des ressources et des données. Dans la dimension de processus, il est spécifié le processus de workflow, c'est à dire les tâches et l'acheminement de ces tâches. Dans la dimension des ressources, les ressources sont regroupées dans des classes particulières nommées les rôles et les unités organisationnelles. Une classe de ressource est un ensemble de ressources présentant des caractéristiques similaires. Si une classe de ressource est basée sur les capacités (exigences fonctionnelles) de ses membres, elle est appelée un rôle. Si le classement est basé sur la structure de l'organisation, une classe de ressource est appelée une unité organisationnelle (par exemple une équipe ou un département).

Nous pouvons visualiser un workflow comme un certain nombre de points dans la vue en trois dimensions de la figure 1.1. Chaque point représente soit un élément de travail (cas + tâche) ou une activité (cas + tâche+ressource).

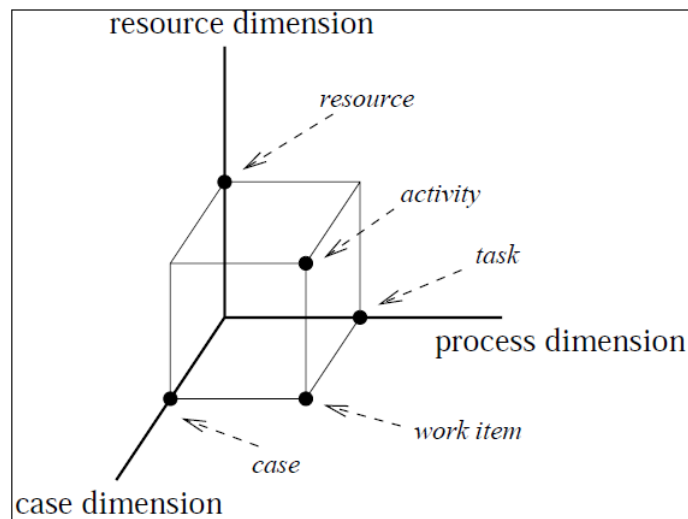


FIGURE 1.1 – Dimensions des systèmes workflow

Notons qu'ici, van der Aalst [7] différencie entre tâche et activité par le fait qu'une activité est un élément de travail en cours d'exécution par une ressource spécifique.

1.5 Classification des systèmes workflow

Deux schémas majeurs de classification des systèmes workflow ont été proposés dans la littérature. [18, 52, 84]

1. Dans [84], McReady a classé les workflows en :
 - workflows Ad hoc dédiés à la gestion des processus de base dans une entreprise.
 - workflows collaboratifs gérant des processus qui évoluent assez fréquemment.
 - workflows administratifs correspondant aux processus orientés états qui suivent une procédure bien définie.
 - workflows de production qui concernent les processus fortement structurés avec presque pas de variations.

Ces quatre types de workflow sont classés (Figure 1.2) en fonction de leur valeur commerciale et de leur répétitivité. Les workflows ad-hoc et les workflows collaboratifs impliquent les participants collaborant pour atteindre un certain but. Souvent, pour ces deux types, aucun modèle de workflow n'est (complètement) défini à l'avance en raison de la faible répétitivité. Les workflows collaboratifs (par exemple la préparation d'une documentation de produits) ont une valeur commerciale supérieure à celle des workflows ad hoc (par exemple la planification de réunions). Les workflows administratifs et les workflows de production ont une haute répétitivité. Des modèles de workflow peuvent être prédéfinis pour ces deux types. Les workflows de production (renfermant les activités essentielles de l'organisation, par exemple, de traitement des réclamations dans une compagnie d'assurance) ont une valeur commerciale supérieure à celle des processus administratifs (par exemple, le calcul des salaires).

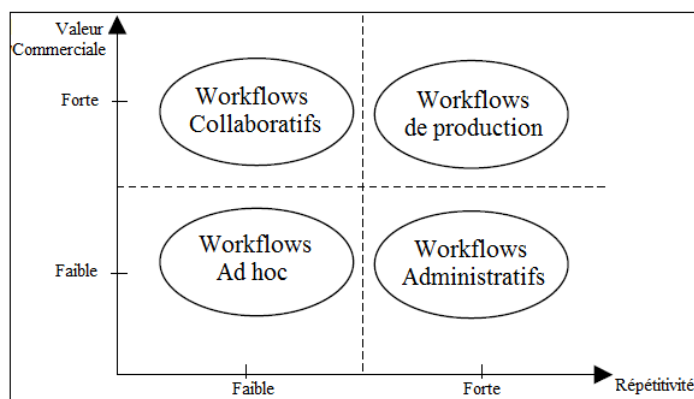


FIGURE 1.2 – Une classification des systèmes workflow

Les workflows collaboratifs ne sont pas conformes à la définition de workflow proposée par la WfMC. Un workflow collaboratif met l'accent sur la communication et le partage de l'information plutôt que la définition des processus.

Dans cette thèse, nous nous limitons aux systèmes de gestion de workflow réels, c'est à dire les systèmes qui supportent les workflows de production, les workflows administratifs et/ou les workflows ad hoc. La WfMC met également l'accent sur ce type d'outils logiciels.

- La deuxième classification des workflow reportée dans la littérature est la suivante : workflows orientés vers l'Homme, workflows orientés vers le système et workflows transactionnels [52]. Les activités de workflows orientés vers l'Homme sont menées par les humains. Ces workflows sont comparables aux workflows ad hoc et collaboratifs. Les workflows orientés système impliquent des systèmes informatiques qui exécutent les opérations de calculs intensifs et les tâches spécialisées des logiciels. Les workflows transactionnels [14] sont un type particulier des workflows orientés système. La communauté de base de données met l'accent sur ce type de workflow. La motivation principale pour l'introduction de workflows transactionnels a été d'aborder l'incapacité des WfMSs d'assurer l'exactitude et la fiabilité des exécutions de workflows en présence d'exécutions concurrentes et d'échecs.

Dans cette thèse, nous considérons une vue conceptuelle des workflows sans mettre l'accent sur les workflows transactionnels. Un workflow est tout simplement considéré comme étant composé d'un ensemble d'activités avec des dépendances de flux de contrôle entre elles, où les activités sont exécutées par les participants qui peuvent être des humains ou bien des agents logiciels.

1.6 Modèle de référence des systèmes workflow

La WfMC a développé un modèle de référence pour la technologie de workflow [118]. La figure 1.3 [123] illustre ce modèle. L'objectif principal du modèle de référence est de fournir un standard pour l'interopérabilité entre les sous-systèmes de workflow. Il se compose d'une description générale de la structure d'un WfMS, dans laquelle cinq principaux éléments

sont présentés (des outils de définition de processus, des applications clientes de workflow, des applications invoquées, des outils d'administration et de contrôle et d'autres services d'exécution de workflow).

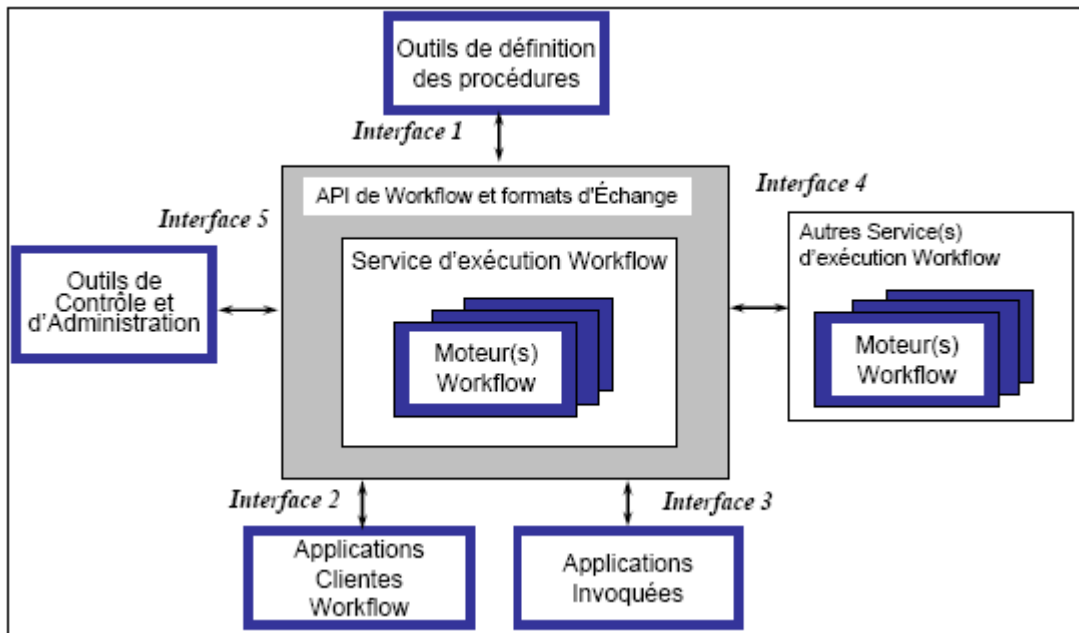


FIGURE 1.3 – Modèle de référence : Composants et Interfaces des systèmes workflow

Ces éléments sont liés au service d'exécution de workflow via l'incorporation d'interfaces, qui sont assurées par une série d'appels à des API (Workflow Application Programming Interface WAPI). Nous décrivons dans les points qui suivent ces cinq interfaces.

– **Interface 1 : Outils de définition de procédures**

- Les outils de définition de processus sont utilisés pour spécifier et analyser les définitions des processus workflow et/ou les classifications des ressources. La plupart des systèmes de gestion de workflow fournissent trois outils de définition des processus :
- un outil avec une interface graphique pour définir les processus workflow,
 - un outil pour spécifier les classes de ressources (modèle d'organisation), et
 - un outil de simulation pour analyser un workflow spécifié.

– **Interface 2 : Applications clientes workflow**

L'utilisateur final communique avec le système workflow via les applications clientes. Un exemple d'application cliente workflow est la liste d'éléments de travail ou encore la corbeille. Via une telle corbeille les éléments de travail sont offerts à l'utilisateur final. En sélectionnant un élément de travail, l'utilisateur peut exécuter une tâche pour un cas spécifique.

– **Interface 3 : Applications invoquées**

L'interface 3 permet à un système workflow l'invocation d'applications si nécessaire. Par exemple, dans le cas des applications manipulant des données fortement typées, un composant externe, supplémentaire, est ajouté. Ce composant, dit agent d'appli-

cation, est chargé de traduire ces données dans un format compréhensible par WAPI.

– **Interface 4 : Autres services d'exécution de workflows**

Via l'interface 4, un système workflow peut échanger des informations avec d'autres moteurs de workflow. Cette interface assure par exemple les fonctions de manipulation des éléments de travail ou des listes d'éléments de travail et de leur transmission entre deux systèmes workflow.

– **Interface 5 : Outils d'administration et de contrôle**

Les outils d'administration et de contrôle sont utilisés pour surveiller et contrôler le workflow. Ces outils sont utilisés pour enregistrer l'avancement des cas et pour détecter les exceptions. En outre, ces outils sont utilisés pour définir les paramètres, allouer les ressources et gérer les anomalies.

1.7 Modélisation des processus workflow

La modélisation est une activité qui précède toute décision ou formulation, elle permet de représenter la description du système réel. Tout comme un système informatique, le système workflow comporte un certain nombre d'aspects à modéliser. Nous présentons en premier lieu ces aspects, nous décrivons en second lieu les principales techniques de modélisation utilisées dans le domaine de workflow et nous terminons cette section par évoquer certains aspects temporels et organisationnels des workflows.

1.7.1 Aspects à modéliser

– **L'aspect fonctionnel**

L'aspect fonctionnel concerne l'identification des activités des processus que l'on souhaite modéliser. Il est important de comprendre qu'il ne s'agit pas uniquement d'identifier les fonctions des différents départements d'une organisation mais aussi de distinguer les activités composant un processus. La modélisation fonctionnelle doit également permettre d'établir la hiérarchie des activités, i.e. d'exprimer de possibles décompositions en termes de sous-processus. Enfin, le modèle fonctionnel doit aussi représenter le flux de données associées aux activités et les interdépendances de données entre les activités (data flow).

– **L'aspect comportemental**

L'aspect comportemental est un aspect primordial du workflow puisqu'il correspond à la dynamique du processus. Le comportement s'exprime par la modélisation d'un contrôle de flux entre les activités. Ce dernier permet d'indiquer la chronologie de l'exécution des activités, leur flux (séquentiel ou parallèle), les points de synchronisation entre activités ou au contraire, les points de disjonction. De plus, le modèle comportemental doit représenter les événements qui permettent de déclencher les activités. Nous soulignons l'importance de ce modèle, qui permet l'exécution du workflow. L'aspect comportemental est également appelé aspect de coordination.

– **L'aspect informationnel (données)**

Cet aspect concerne l'ensemble des informations et des données qui sont associées

aux activités. Le modèle informationnel, souvent négligé lors de l'implémentation d'un workflow [68], décrit en détail les relations qui existent entre les données, leur type et leur structure.

– **L'aspect organisationnel**

Comme son nom l'indique, la partie organisationnelle concerne la description de l'organisation des acteurs de l'entreprise. Le modèle organisationnel peut soit refléter fidèlement l'organigramme de l'entreprise, c'est à dire la décomposition hiérarchique de celle-ci en départements et services soit décrire des unités organisationnelles dans lesquelles on identifie des acteurs. Selon la méthode choisie, la description est plus ou moins détaillée et permet d'établir des liens hiérarchiques entre les acteurs ainsi que des relations entre unités organisationnelles ou départements. Toutefois, quelle que soit la méthode retenue, la description des rôles associés aux différentes activités reste invariante. Les rôles créent l'interface entre le modèle organisationnel et les modèles représentant les activités.

1.7.2 Techniques et outils de modélisation de workflow

Associés aux aspects à modéliser définis précédemment, un certain nombre d'outils de modélisation peuvent être employés pour décrire le comportement des flux de travail.

1.7.2.1 Réseaux de Petri et workflow

Les Réseaux de Petri (RdPs) [89] constituent un formalisme majeur pour modéliser les processus workflows. Une des forces des RdPs est la base mathématique forte qu'ils offrent avec une représentation graphique. Dans cette section, nous résumons la projection topographique entre concepts de workflow et de RdP [1, 11, 65].

Un processus définit les tâches aussi bien que les conditions pour leur exécution. En utilisant les RdPs, un processus est représenté en conversant sa seule entrée (i.e., un nœud du début) dans une place sans arcs entrants, et sa seule sortie (i.e., nœud de la fin) dans une place sans arcs sortants. Les conditions sont représentées par des places, et les tâches par des transitions. Habituellement, un processus spécifié par RdP devrait accomplir deux exigences : (1) il doit être possible d'accéder à tout moment un état pour lequel il y a un jeton dans la place finale, et (2) quand il y a un jeton dans la place finale, tous les autres jetons auraient dû disparaître.

Dans un processus modélisé par un RdP, une transition active correspond à un work-item, et le tir d'une transition à une instance de l'activité. Certains work-items peuvent seulement être transformés dans une instance d'activité une fois ils sont déclenchés. Un déclencheur pourrait correspondre à une initiative du participant, à un événement externe ou à un signal du temps initié par l'environnement. A chaque transition correspondante à une tâche qui exige un déclencheur, une autre place d'entrée est ajoutée. Une occurrence du déclencheur apporte un jeton dans cette place supplémentaire. Le jeton est consommé une fois la transition appropriée est franchie. Un échec pendant l'exécution d'une tâche exige un 'rollback' (i.e., revenir à l'état antérieur au début de l'activité). Quand une activité sera complétée avec succès, des changements deviennent définitifs.

1.7.2.2 UML et workflow

Les diagrammes d'états/transitions sont un autre formalisme majeur pour la modélisation des processus workflows. Ils ont été inventés par Harel [57], et ont été incorporés dans UML (Unified Modelling Language) [105] dans une forme légèrement différente. Weissenfels et al. [117] ont investi sur l'usage des diagrammes d'états/transitions pour modéliser les processus workflows (le projet Mentor WfMS). Nous pouvons aussi citer Blake [32] qui a présenté une approche systématique pour la modélisation des workflows en utilisant UML (le projet WARP).

Dans le projet Mentor WfMS [117], les activités reflètent la décomposition fonctionnelle d'un système et dénotent les composants actifs d'une spécification ; elles correspondent directement aux activités d'un workflow. Un diagramme d'activités spécifie le flot de données entre les activités, dans la forme d'un graphe orienté avec les éléments de données comme annotations des arcs. Les diagrammes d'états capturent le comportement d'un système en spécifiant le flot de contrôle entre les activités.

Pour le projet WARP (Workflow Automation through Agent-based Reflective Processes) [32], l'approche utilisée distingue entre les vues structurelles, fonctionnelles, non-fonctionnelles, et opérationnelles. Les vues structurelles informe sur les activités, la définition des rôles, et la composition du workflow. Elles sont représentées dans UML par les diagrammes des classes. Les vues fonctionnelles montrent le flot de données et de contrôle du workflow en utilisant les diagrammes d'activités UML. Les vues non-fonctionnelles (traitement des erreurs, concurrence, atomicité, etc.) utilisent le flot de données et de contrôle et peuvent être modélisées aussi par les diagrammes d'activités. Finalement, les vues opérationnelles sont reliées à l'initiation des instances du workflow, et la coordination pour l'achèvement du workflow. Ces vues peuvent être modélisées par les diagrammes de séquence UML.

1.7.2.3 YAWL

YAWL (Yet Another Workflow Language) [2] est basé sur les RdPs avec des caractéristiques additionnelles pour faciliter la modélisation des workflows complexes. YAWL hérite de la classe des RdPs et s'étend par la modélisation de l'instance multiple, les tâches composites, le retrait des jetons et les transitions connectées directement.

La spécification du workflow dans YAWL est une sorte de réseaux de workflow étendu (EWF-nets) qui forme une hiérarchie. Une tâche est soit atomique soit composite. Chaque tâche composite fait référence à un unique EWF-net dans le niveau le plus bas de la hiérarchie. Les tâches atomiques forment les feuilles de l'arbre. Il y a un seul EWF-net non référencé par une tâche composite. Cet EWF-net est nommée le niveau le plus haut du workflow et forme la racine.

Chaque EWF-net consiste en des tâches (composites ou atomiques) et des conditions qui peuvent être interprétées comme des places. Chaque EWF-net a une unique condition d'entrée et une unique condition de sortie. Contrairement aux RdPs, il est possible de connecter les transitions. Sémantiquement, cette construction peut être interprétée comme une condition ignorée.

Chaque tâche peut avoir des instances multiples. Nous pouvons créer un seuil maximal et un seuil minimal pour le nombre d'instances après l'initiation d'une tâche. En plus, il est possible d'indiquer que la tâche se termine au moment où certaines instances seuils sont achevées.

1.7.3 Les aspects temporels de workflow

Le modèle de workflow doit être capable de capturer différents aspects du processus métier incluant les propriétés de structure, des données et des ressources, mais aussi des propriétés temporelles. La modélisation du temps dans les workflows a été très peu enquêtée dans le contexte de projets de recherche. Marjanovic et Orłowska ont annoncé que trois contraintes temporelles peuvent être spécifiées [81] : (1) une contrainte de durée qui modélise la durée attendue d'une activité dans un workflow (un temps relatif simple ou bien un intervalle de deux valeurs relatives de temps) ; (2) une contrainte de deadline qui peut être spécifiée en terme de limite de temps absolu quand une activité doit débiter ou finir durant l'exécution de workflow ; (3) une contrainte temporelle interdépendante qui détermine quand une activité doit commencer/finir relativement au commencement/fin d'une autre activité (une valeur relative de temps).

La consistance temporelle joue un rôle crucial dans la modélisation des contraintes de temps. Elle doit être vérifiée plusieurs fois durant le cycle de vie du workflow : durant la modélisation de workflow et puis durant l'exécution du workflow à plusieurs points de contrôle (habituellement après chaque nœud de décision) pour s'assurer que les activités sont entrain de s'exécuter comme prévu.

L'assignation du temps aux activités dans un workflow est une tâche semblable à la planification des systèmes temps réel. Une différenciation est faite entre gestion du temps au temps de construction et celle au temps d'exécution. Au temps de construction, et utilisant le modèle de workflow et les durées assignées aux activités dans le modèle, les temps de début et de fin relatifs pour toutes les activités sont calculés (en ce qui concerne le commencement du workflow). De tels calculs sont menés en utilisant une traversée avancée et une traversée moins évoluée du modèle de workflow. Au temps d'instanciation, un calendrier est utilisé pour convertir toute l'information du temps relatif spécifié pendant le temps de construction aux points du temps absolus.

Si une date limite est dépassée, un échec du temps est généré et des actions spéciales peuvent être déclenchées, connues sous le nom d'actions de l'escalade [48] : extension de la date limite, sélection alternative, suppression facultative et erreur de temps.

En dépit de l'importance du temps pour la coordination et l'exécution de processus métier, le support de la gestion du temps dans les workflows actuellement disponibles est plutôt rudimentaire. Comme mentionné dans [80], les exigences pour la modélisation et la visualisation du temps dépassent les capacités fournies par les WfMSs (commerciaux) actuels.

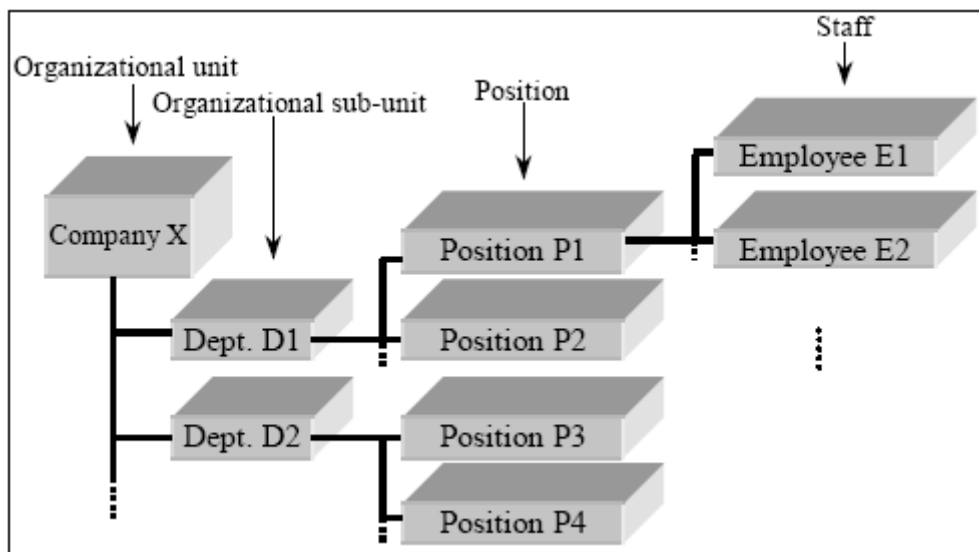


FIGURE 1.4 – Un exemple de modèle organisationnel (Structure d'arbre)

1.7.4 Structure organisationnelle

La spécification des participants dans le workflow exige habituellement une spécification à l'avance d'une structure d'organisation (rôles, capacités, positions, hiérarchies, etc.). Un aspect important d'une structure d'organisation est la division d'autorités et de responsabilités. Un exemple d'autorité est d'assigner un travail à d'autres membres du personnel. La forme de structure d'organisation la plus largement utilisée est l'organisation hiérarchique caractérisée par une structure d'arbre où chaque nœud montre soit (1) la personne qui est responsable de tous les gens en dessous d'elle dans l'arbre, soit (2) le département (i.e., l'unité organisationnelle) qui assemble des sous-départements définis en dessous de lui dans l'arbre jusqu'à atteindre la portée individu dans le staff (figure 1.4).

1.8 Outils logiciels pour décrire et gérer un workflow

Pour représenter et gérer les workflows, un certain nombre de logiciels a été développé. L'ensemble de ces logiciels dénombre actuellement plus de deux cents logiciels sur le marché et de plus en plus d'entreprises envisagent l'utilisation de tels logiciels [1]. Ce choix important de logiciels provient de la jeunesse du domaine et du manque de standards conceptuels formellement décrits.

Dans le but de donner une impression de la génération actuelle des systèmes de gestion de workflow, nous présentons brièvement cinq d'entre eux : jBPM [17], OpenWFE [87], FlowMind [12], COSA workflow [108] et Webflow [47].

- **jBPM** (java Business Process Management) est un système de gestion de workflow qui décrit le processus métier avec le langage jPdl [66]. Ce système a été introduit par JBoss, il offre des structures basiques pour modéliser le processus d'entreprise. Il est

basé sur le formalisme de RdP, open source et extensible ; de plus sa documentation est disponible. L'inconvénient de ce moteur est que la description du processus métier en question doit être faite en jPdl, donc la connaissance de ce langage, spécifique à ce moteur, est requise. De plus, la cohérence du processus, décrit en jPdl, ne sera pas faite par le moteur de workflow.

- **OpenWFE** (Open source WorkFlow Environment) est un moteur de workflow existant en open source. Il permet la gestion des processus qui vont être exécutés par des participants qui pourraient être soit des humains soit d'autres services. Cet environnement est écrit en java mais il n'est pas limité à ce langage, il comporte des connecteurs permettant son utilisation à partir de Python, Perl, C# (.NET), PHP et Ruby.
- **FlowMind** est un WfMS écrit en Java qui utilise le Web pour adapter les processus collaboratifs d'une entreprise aux exigences du marché. Parmi les fonctionnalités de ce système nous citons : la modélisation, le déploiement, l'automatisation et l'administration des processus métier. En outre, FlowMind propose des outils adaptables à tous les profils afin de garantir une efficacité maximale et une pertinence des processus d'entreprise.
- **COSA Workflow** (COSA Solutions) rejoint un modèle organisationnel puissant utilisant les RdPs. COSA offre un traiteur de liste de travail HTML et une intégration de Lotus Notes. Les activités peuvent être programmées à travers une API. Le workflow COSA est sélectionné par Baan, un éditeur de progiciels de gestion intégrés (ERP), en 1999 comme le produit workflow standard le plus intégré aux ERPs. En résumé, COSA présente ces avantages :
 - Il permet de gérer une organisation complexe avec des unités, des groupes, des projets et plusieurs rôles.
 - Il permet de modéliser des procédures complexes.
 - Il offre un mécanisme de contrôle de temps puissant en utilisant le calendrier.
 - Il exige un débit raisonnable.
 - Il est basé sur la notation par RdP.
- **Webflow** (SAP) est conçu avec le concept orienté objet. Webflow offre un haut niveau de réutilisabilité. Il s'intéresse aux procédures de gestion et offre un outil de modélisation organisationnel puissant en laissant l'implémentation des activités aux programmeurs. Il est compatible avec Lotus Notes. Ce moteur de workflow peut coopérer avec d'autres moteurs à travers XML. Les caractéristiques de Webflow peuvent se résumer en ces points :
 - Il est un workflow de production avec l'intégration des applications, qui est robuste et complexe.
 - Il modélise les procédures relatives aux changements d'organisations.
 - Il gère un nombre important de différentes procédures et activités réutilisables.
 - Les activités gérées sont complexes et doivent être programmées pour convenir aux exigences.

Les moteurs de workflow actuels ne décrivent pas tous un processus métier avec un langage de modélisation formel comme les RdPs et essentiellement ils ne permettent pas de vérifier la cohérence de tel processus. D'où apparaît l'importance de développer un outil de modélisation et de vérification de processus workflow qui ait un format universel

d'échange de processus. Ceci sera utile pour mettre en œuvre l'interface de communication entre différents moteurs de workflow qui a été décrite dans la section 1.6 (interface 4).

1.9 Présentation de l'outil jPnet

Nous avons mené une étude détaillée sur le moteur jBPM de JBoss afin de dégager sa conception. L'étude de cette conception a mené à tirer les lacunes de ce moteur de workflow, dégagées dans la section précédente, et par ailleurs de proposer des améliorations pour remédier à ces lacunes. La solution proposée consiste en la conception et l'implémentation d'un outil dénommé jPnet qui permet de faciliter la tâche du développeur de processus à travers l'introduction du processus d'entreprise sous forme de graphe de réseau de Petri (plus particulièrement workflow-net (WF-net) : RdP simplifié pour représenter une instance du workflow). Ce graphe sera traduit en un fichier XML conforme à la spécification du moteur de workflow étudié.

Les différentes fonctionnalités initialement implémentées sous notre outil jPnet sont exposées dans le diagramme des cas d'utilisation UML de la figure 1.5. Ce diagramme reflète uniquement les fonctionnalités basiques, pour lesquelles jPnet a été proposé. D'autres modules ont été implémentés sous jPnet au cours de cette thèse et vont être détaillés dans les chapitres qui suivent.

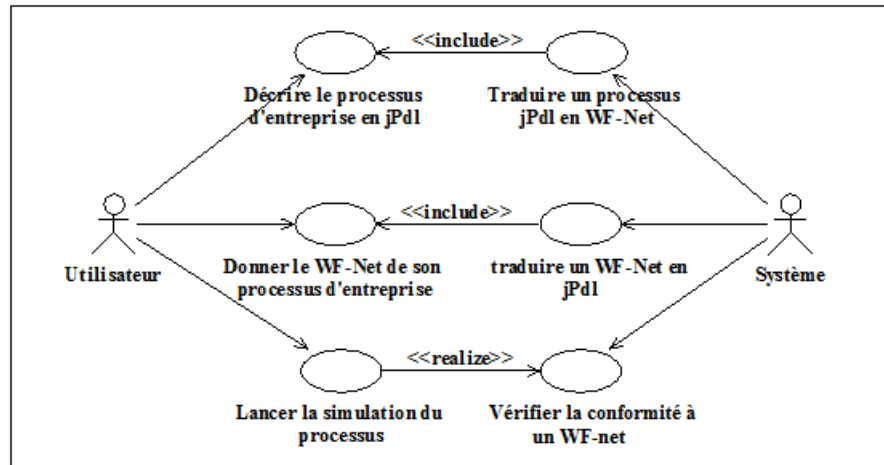


FIGURE 1.5 – Fonctionnalités basiques de l'outil jPnet

1.9.1 Mapping entre jPdl et RdP

jPnet comble une des déficiences du moteur jBPM à savoir la vérification de la cohérence du processus introduit par le développeur de processus soit sous forme d'un réseau de Petri soit sous forme d'un fichier XML. Ensuite, nous avons proposé une traduction du langage jPdl vers les réseaux de Petri. Notre approche facilite la vérification de la cohérence des processus d'entreprise donnés en jPdl en cas général.

JPdl est un langage de description de processus d'entreprise spécifique au moteur de workflow jBPM de JBoss. La description d'un tel processus en JPdl fait l'objet d'un fichier XML conforme à la spécification JPdl. A cet effet, une DTD (Document Type Definition) a été mise en place. Elle est détaillée dans [66]. Le fragment DTD relatif à la définition de processus 'Process Definition' est donné dans la figure 1.6. Dans cette DTD, $x?$ signifie zéro ou une occurrence de x , x^* signifie zéro ou plusieurs occurrences de x et $x|y$ signifie x ou y .

```
<!ELEMENT process-definition (
  description? ,
  swimlane* ,
  type* ,
  start-state ,
  ( state |
    milestone |
    process-state |
    decision |
    fork |
    join
  )* ,
  end-state ,
  action* ) >
<!ATTLIST process-definition name CDATA #REQUIRED >
```

FIGURE 1.6 – DTD relatif à la définition de processus JPdl

Cette DTD indique qu'en définissant son processus d'entreprise dans le langage JPdl, le développeur peut :

- décrire le processus (description?),
- assigner des états à des acteurs (swimlane*),
- définir les types des données qui seront enregistrées dans le moteur jBPM (types*),
- ajouter des fragments de code java dont l'exécution sera déclenchée lors d'événements spécifiques réalisés lors de l'exécution du processus (action*).

!ATTLIST définit la liste des attributs alors que #REQUIRED après un attribut indique que celui-ci est obligatoire.

Dans les points qui suivent, nous résumons les constructions : state, start-state, milestone, process-state, decision, fork, join et end-state.

- state : c'est le concept central de jBPM. C'est un état dans lequel peut se trouver un processus d'entreprise. Le fragment DTD relatif à cette construction est le suivant :

```
<!ELEMENT state (
  description?, assignment?,
  action*, transition+ ) >
<!ATTLIST state name CDATA #REQUIRED >
```

Cette DTD indique qu'en définissant un état, le développeur peut lui associer une description, un assignement (une dépendance d'un ou plusieurs acteurs externes), des actions et au moins une transition (i.e. une connexion dirigée entre l'état en cours et

- un autre).
- start-state : c'est l'unique état, dans le processus, à partir duquel toutes les instances de ce dernier sont générées. Plusieurs transitions peuvent sortir du start-state,
 - milestone : est un type spécial d'état utilisé pour synchroniser entre deux chemins d'exécution concurrents,
 - process-state : le processus père lance un sous processus quand l'exécution arrive au process-state,
 - decision : il s'agit d'une structure alternative permettant le choix d'un chemin d'exécution parmi plusieurs. Une décision modélise une situation où le moteur de workflow décide de la route à prendre, en se basant sur le contexte, et éventuellement de certaines ressources externes,
 - fork : il génère plusieurs chemins d'exécution concurrents,
 - join : il réunit plusieurs chemins d'exécution concurrents. Un join ne peut avoir qu'une transition sortante,
 - end-state : un processus possède exactement un état de fin (end-state). Il marque la fin d'une instance du processus.

Pour passer d'un fichier écrit en langage jPdl à un graphe de réseau de Petri, nous avons effectué une analyse sémantique du fichier donné en jPdl tout en supposant qu'il est syntaxiquement correct.

Le mapping de jPdl vers réseau de Petri est effectué en traduisant les constructions de jPdl en termes de réseaux de Petri.

1.9.1.1 Traduction des états

En jBPM, le terme state possède la même définition que dans une machine à état fini (Finite State Machine). Il est donc possible de traduire les états en des places dans les réseaux de Petri. En particulier, l'état de début (start-state) et l'état de fin (end-state) sont respectivement traduits en place source et place destination dans les WF-nets (figure 1.7).

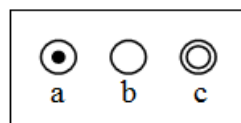


FIGURE 1.7 – Traduction en WF-net de a- start-state ; b- state ; c- end-state

1.9.1.2 Traduction des liens

jPdl définit les flux de contrôle entre les états avec des transitions, décisions, forks, joins et milestones. En examinant la séquence simple d'états et leurs relations, nous déduisons facilement la réalisation de cette séquence par le pattern 1 de la liste des patterns définis dans [3]. Ce pattern nommé "Sequence" informe qu'une activité dans le workflow ne se déclenche qu'après la terminaison d'une autre activité dans le même processus. La figure 1.8 présente une séquence simple d'états et de transitions.

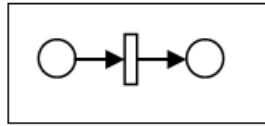


FIGURE 1.8 – Traduction d'une séquence simple d'états et de transitions

1.9.1.3 Traduction des structures de contrôle

Pour modéliser les structures de contrôle, jPdl dispose des constructions fork, join, decision et milestone.

- fork produit, à partir d'un état plusieurs chemins d'exécution concurrents. Cette structure est traduite par "AND-SPLIT" (figure 1.9(a)). Dans ce contexte, la fin de l'exécution d'un état provoque le déclenchement de deux ou plusieurs états.
- join relie plusieurs chemins d'exécution concurrents dans un état. Ainsi join se traduit naturellement par une "Synchronization" (figure 1.9(b)) : le passage du processus à l'état suivant dépend de la réalisation de deux ou plusieurs états précédents.

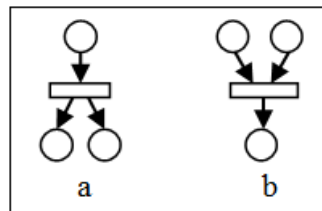


FIGURE 1.9 – Traduction de : a-fork ; b-join

- decision indique la possibilité de décider après la terminaison d'un état, l'exécution d'un état parmi deux ou plusieurs. Nous l'avons donc traduite par la structure OU EXCLUSIF (XOR-SPLIT) représentée dans la figure 1.10(a).
- milestone : cette construction est traduite par la structure (OR-JOIN) puisqu'elle ne permet le passage à l'état suivant que si un des états précédents est terminé (figure 1.10.b).

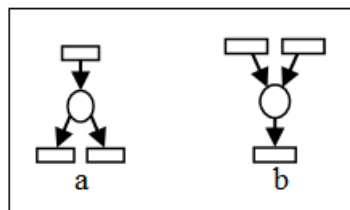


FIGURE 1.10 – Traduction de : a- decision ; b- milestone

Le détail du mapping entre jPdl et WF-net est présenté dans [102] et est résumé dans la figure 1.11.



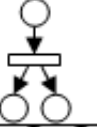
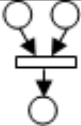

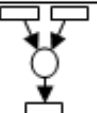


jPdl	construction WF-Net
Start-state	Place Source i 
State	Place 
Fork	Diffusion 
Join	Synchronisation 
Decision	Choix/conflit 
Milestone	Convergence 
Transition	Transition avec deux arcs 
End-state	Place Finale f 

FIGURE 1.11 – Mapping de jPdl vers workflow-net

Le mapping résumé dans la figure 1.11 a été implémenté sous jPnet et testé sur des exemples concrets.

1.9.2 Mapping entre PNML et RdP

Outre jBPM, jPnet peut être utile pour d'autres moteurs de workflow puisqu'il permet de modéliser un processus selon le formalisme graphique WF-net et de vérifier sa cohérence. Pour que le processus modélisé soit compréhensible par ces moteurs, nous avons implémenté la génération d'un fichier de description de processus conforme à la spécification PNML (Petri Net Markup Language) [31, 74, 77] à partir d'un workflow-net. Cette génération ne sera réalisée qu'après vérification de la cohérence du processus en question. Dans le sens inverse, un fichier donné en PNML est traduit en workflow-net en vue de la vérification de sa cohérence.

Petri Net Markup Language (PNML) est un format d'échange largement accepté basé sur XML pour les réseaux de Petri, qui est actuellement standardisé ISO/IEC 15909-2 [75]. Le principal problème avec l'élaboration d'un format d'échange standard pour les réseaux de Petri est la multitude de versions et des variantes existantes des réseaux de Petri. Afin de remédier à ce problème, PNML fournit des interfaces pour définir de nouvelles fonctionnalités et de nouveaux types de réseaux de Petri : interface de définition des fonctionnalités et interface de définition des types.

Un document qui satisfait aux exigences du modèle PNML est appelé un document de réseau de Petri (PetriNetDoc). Il peut contenir plusieurs réseaux de Petri (PetriNet). Chaque réseau de Petri comporte un identifiant unique et un type. Le type est une URL de référence pour le nom du package à sa définition.

Un réseau de Petri se compose de quelques pages à un niveau supérieur qui sont, à leur tour composées de plusieurs objets. Ces objets représentent la structure graphique du réseau de Petri. Chaque objet dans un PetriNetDoc possède un identifiant unique, qui peut être utilisé pour faire référence à cet objet. En outre, chaque objet peut être équipé d'informations graphiques définissant sa position, taille, couleur, forme et d'autres informations sur son aspect graphique.

Un objet d'un réseau est une place, une transition, un arc, ou autre type d'objet (une page ou un nœud de référence). Par convention, les places et les transitions sont généralisées à des nœuds qui peuvent être reliés par des arcs.

Un fichier PNML est un fichier écrit en langage PNML qui décrit les propriétés graphiques des éléments du RdP modélisant le processus workflow.

La description d'un processus en PNML fait l'objet d'un fichier XML conforme à la spécification PNML. Nous nous sommes inspirés de [74] pour comprendre le mapping du modèle PNML vers la syntaxe XML. Par ailleurs, nous avons défini une DTD (Document Type Definition) spécifique à jPnet à travers laquelle la traduction vers WF-net est aisée. Cette DTD est détaillée dans la figure 1.12.

```
<!ELEMENT pnml (net)>

<!ELEMENT net (place+,transition+,arc+)>
<!ATTLIST net id CDATA #REQUIRED>
<!ATTLIST net type CDATA #REQUIRED>

<!ATTLIST place id CDATA #REQUIRED>
<!ELEMENT place (graphics?,name?) >
<!ELEMENT graphics (offset)>

<!ELEMENT offset EMPTY>
<!ATTLIST offset x CDATA #IMPLIED>
<!ATTLIST offset y CDATA #IMPLIED>
<!ELEMENT name (text)>
<!ELEMENT text (#PCDATA)>
<!ATTLIST transition id CDATA #REQUIRED>
<!ELEMENT transition (graphics?,name?)>

<!ATTLIST arc id CDATA #REQUIRED>
<!ATTLIST arc source CDATA #REQUIRED>
<!ATTLIST arc target CDATA #REQUIRED>
<!ELEMENT arc (graphics?) >
```

FIGURE 1.12 – DTD relative à la définition de processus PNML

La DTD de la figure 1.12 définit la structure minimale d'un fichier PNML. !ELEMENT définit les éléments enfants de l'élément racine. Ces enfants sont cités entre parenthèses. Le caractère virgule donne l'ordre de citation des éléments enfants. La présence ou non d'un des caractères ?, * et + à la suite d'un élément enfant détermine le nombre de ses occurrences. #PCDATA, indique qu'un élément contient des données textuelles et EMPTY indique qu'un élément est vide. !ATTLIST définit la liste des attributs alors que #REQUIRED après un attribut indique que celui-ci est obligatoire et #IMPLIED pour dire optionnel.

Un élément *pnml* réfère à un document de réseau de Petri (PetriNetDoc) et doit contenir un élément *net*. Un élément *net* doit contenir au moins un élément *place* suivi par au moins un élément *transition* lui même suivi par au moins un élément *arc*. Les attributs *id* et *type* de l'élément *net* sont obligatoires.

Les éléments *place* et *transition* possèdent un attribut obligatoire *id*. De même, ils possèdent l'élément *graphics* suivi de l'élément *name* ; ces deux éléments peuvent avoir occurrence zéro ou plusieurs fois. Un élément *name* contient l'élément *text*, ce dernier contient des données textuelles. Un élément *graphics* contient un élément *offset* qui réfère à la position définie par les attributs optionnels *x* et *y*. Un élément *arc* possède trois attributs obligatoires *id*, *source* et *target*. Il peut contenir l'élément *graphics* zéro ou plusieurs fois.

À partir de cette DTD, nous proposons le mapping entre PNML et WF-net résumé dans la figure 1.13.

PNML	WF-net
Place	Place d'entrée
Place	Place
Place	Place de ressource
Transition	Transition
Arc	Arc
Place	Place de fin

FIGURE 1.13 – Mapping entre PNML et workflow-net

Le mapping entre PNML et WF-net résumé dans la figure 1.13 a été implémenté sous *jPnet* et testé sur des exemples concrets.

1.10 Conclusion

Dans ce chapitre, nous avons en premier lieu rappelé les concepts fondamentaux relatifs à la gestion des processus workflow et introduit la terminologie qui va être utilisée dans le reste de cette thèse. Nous avons tout d'abord rappelé les principales notions relatives aux processus métier et à leur implémentation. Nous nous sommes ensuite centrés sur les technologies de processus métier et aux détails de la technologie de workflow. Puis, après avoir résumé les différentes classes de workflows existantes, nous avons détaillé le modèle de référence de la WfMC décrivant les interfaces liant un système workflow et son environnement. Enfin, nous avons présenté les principaux formalismes et langages de modélisation de processus workflow.

En deuxième lieu, nous avons étudié quelques systèmes de gestion de workflow et dégagé leur problématique majeure à savoir le manque d'outils de modélisation et de vérification des processus confiés à ces systèmes. A l'issue de cette étude, nous avons exposé notre première contribution à la modélisation et à la vérification de processus workflow. Cette contribution a consisté en l'implémentation d'un outil de modélisation de processus workflow par des workflow-nets, de vérification de leur cohérence et de leur traduction vers des langages de définition de processus connu par des systèmes de gestion de workflow existants. Notre outil, dénommé jPnet [99], permet d'autres fonctionnalités qui seront exposées dans les chapitres qui suivent.

Ce premier travail nous a permis de nous assurer du besoin de vérification des processus workflow et de souligner les insuffisances des langages de description des processus utilisés par les WFMSs actuels en général et le langage jPdl en particulier. Ainsi, nous notons le manque d'outils et de langages qui proposent des structures supportant les constructeurs avancés tels que les états à instances multiples, la possibilité de passer des données d'une instance d'état précédent vers un état suivant capable de supporter des instances d'exécution multiples et enfin la gestion des ressources partagées.

1.10. CONCLUSION

Chapitre 2

Vérification des processus workflow

2.1 Introduction

Les techniques et outils de modélisation de processus ont permis le développement de nombreux systèmes de gestion de workflow prenant en charge partiellement ou entièrement la responsabilité de l'exécution coordonnée des activités. Toutefois, la vérification de la cohérence de cette exécution, la prise en compte de l'allocation des ressources (partagées) nécessaires à l'exécution de ces activités ne sont pas encore assurées au sein de ces systèmes.

La propriété de cohérence (ou soundness) a fait l'objet de plusieurs travaux de recherche. Pour vérifier cette propriété, une condition nécessaire et suffisante a été établie. En effet, démontrer qu'un WF-net est cohérent revient à montrer que son réseau fermeture est vivant et borné. Cette vérification s'effectue de façon comportementale puisqu'elle fait recours au calcul de l'espace d'états. Donc, cette vérification est de complexité exponentielle qui augmente en fonction de la taille du réseau. Nous proposons donc de nous investir dans la recherche d'autres caractérisations de la cohérence qui diminuent le coût de sa vérification. Pour ce faire nous nous sommes basés sur des résultats récents de la théorie structurelle des réseaux de Petri.

Dans ce chapitre nous commençons par introduire dans la section 2.2 le modèle de base de notre travail à savoir le WF-net ainsi que les caractéristiques structurelles des réseaux de Petri. Ensuite, nous exposons notre approche concernant la vérification paramétrée de la cohérence (section 2.3). Puis, nous étendons cette approche pour qu'elle puisse être exploitée dans le cas de WF-nets avec ressources partagées (section 2.4). Enfin, nous présentons notre algorithme de calcul des siphons minimaux dans un WF-net (section 2.5) utile pour l'implémentation des résultats des sections précédentes.

2.2 Les réseaux workflow : modèle de base

Le modèle choisi comme modèle de base est celui des workflow-nets (WF-nets) : sous-classe des réseaux de Petri (RdPs) adaptée aux processus workflow. Nous justifions tout d'abord ce choix, puis nous présentons les notions de réseaux de Petri utiles pour notre travail et enfin nous définissons les réseaux workflow (WF-net).

2.2.1 Choix des réseaux de Petri comme langage de modélisation de workflow

Un processus workflow définit les activités ainsi que leur ordre d'exécution. En utilisant les RdPs ; un processus pourrait être représenté en remplaçant son unique entrée i.e. son nœud initial par une place ne possédant aucun arc d'entrée, et son unique nœud de sortie par une place ne possédant aucun arc de sortie. Les conditions sont représentées par les places et les tâches par les transitions.

Dans [7] trois raisons pour l'utilisation du formalisme des RdPs dans les WfMS sont données. Nous présentons dans ce qui suit ces raisons :

- **Sémantique formelle en plus du formalisme graphique** : La première raison de l'utilisation des RdPs pour la modélisation des processus workflow est le fait que la logique d'un certain fonctionnement pourrait, à travers ce formalisme, être représentée par un langage, à la fois, formel et graphique. De plus, l'association de plusieurs autres options (temps, couleur et hiérarchie) a été définie formellement. En effet, une procédure de workflow spécifiée en termes de RdP est non ambiguë, ceci veut dire que chaque construction est assez claire et qu'il n'y aura pas de conflits d'interprétation du sens de la spécification d'une certaine procédure de workflow. En outre, la description d'un workflow à l'aide d'un RdP pourrait être utilisée comme un contrat entre différents tiers.
La sémantique formelle peut être utilisée dans le but de résoudre les conflits à propos de l'interprétation des procédures de workflow communes. De plus, l'interprétation des procédures de workflow basée sur les RdPs est indépendante des outils : elle ne change jamais quand une nouvelle version d'un certain WfMS est délivrée. Cette sémantique formelle permet le raisonnement à propos des propriétés d'une procédure de workflow donnée ; elle permet de prouver l'existence ou l'absence de propriétés dynamiques comme les deadlocks ou les livelocks.
- **Modélisation orientée états avec modélisation orientée événements** : Contrairement aux autres techniques de modélisation des processus workflows, l'état d'un cas peut être modélisé explicitement avec les RdPs. En premier lieu, une description basée sur les états permet une claire distinction entre l'activation d'une tâche et son exécution puisque l'activation d'une tâche n'implique pas son exécution. Une autre raison pour l'explicitation de la modélisation des états est la possibilité de l'existence de tâches compétitives (des tâches actives simultanément et dont une seule pourrait être exécutée).
- **Variété des techniques d'analyses** : Les RdPs sont distingués par la disponibilité de plusieurs techniques d'analyse. En effet, en construisant le graphe des marquages, nous sommes capables de prouver qu'une certaine propriété est vérifiée pour une procédure de workflow donnée. Par exemple, nous pouvons utiliser le graphe des marquages pour détecter les deadlocks et les états non désirés. Il est aussi possible d'utiliser des techniques exploitant la structure du RdP en question. Nous pouvons par exemple, calculer les p-invariants pour décider à propos de la finitude ou bien calculer les siphons et les trappes pour décider à propos de la vivacité.

La modélisation des workflows par les RdPs est fait à travers une sous classe particulière des RdPs nommée workflow-net (WF-net). Cette sous-classe sera présenté dans la section

2.2.3.

2.2.2 Notions de base des réseaux de Petri

Cette section expose les définitions de base et les notations des réseaux de Petri (RdPs) ainsi que les résultats principaux de leur théorie structurelle utilisés dans notre travail.

2.2.2.1 Réseaux de Petri

Définitions

Un *réseau de Petri* est un quadruplet $N = (P, T, F, W)$ où

- P et T sont des ensembles finis non vides (respectivement de places et transitions),
- $F \subseteq P \times T \cup T \times P$ est la relation de flot,
- $W : F \rightarrow \mathbb{N}$ est la fonction de valuation $W(x, y) = 0 \Leftrightarrow (x, y) \notin F$.

Si $W(u) = 1 \quad \forall u \in F$, N est dit ordinaire et est dénoté par $N = (P, T, F)$.

Un *marquage* de N est une fonction $M : P \rightarrow \mathbb{N}$. M_0 désigne le marquage initial.

On dénote par M_p le marquage pour lequel $M(p) = 1$ et $M(q) = 0 \quad \forall q \neq p$.

Représentation graphique

Les RdPs sont représentés comme suit : les places sont représentées par des cercles, les transitions par des rectangles, la relation de flot est représentée en dessinant un arc entre x et y chaque fois que (x, y) est dans la relation, et la fonction de valuation marque les arcs par leurs poids si ces derniers sont plus grands que 1.

Un marquage M d'un RdP est représenté en dessinant, pour chaque place p , $M(p)$ jetons noirs dans le cercle représentant la place p .

Notations

Pour chaque nœud x de $P \cup T$, nous notons par $\bullet x$ l'ensemble de ses nœuds entrants et par x^\bullet l'ensemble de ses nœuds sortants.

Formellement : $\forall x \in P \cup T$

$$\begin{aligned}\bullet x &= \{y \in P \cup T / (y, x) \in F\} \\ x^\bullet &= \{y \in P \cup T / (x, y) \in F\}.\end{aligned}$$

Franchissement

Une transition $t \in T$ est *franchissable* dans un marquage M (noté par $M[t)$) si et seulement si $\forall p \in \bullet t : M(p) \geq W(p, t)$.

Son franchissement conduit au marquage M' défini par : $\forall p \in P : M'(p) = M(p) + C(p, t)$. C étant la matrice $P \times T$ définie par $C(p, t) = W(t, p) - W(p, t)$ et appelée matrice d'incidence de N .

Le franchissement est noté par $M[t) M'$. L'ensemble des marquages accessibles de M est noté par $[M)$.

2.2.2.2 Propriétés comportementales de base

Un marquage M_h est un *état d'accueil* si et seulement si $\forall M \in [M_0)$, $M_h \in [M)$;

(N, M_0) est *borné* si et seulement si $\forall p \in P : [\exists k \in \mathbb{N} : \forall M \in [M_0], M(p) \leq k]$ i.e. l'ensemble des marquages accessibles de M_0 est fini.

(N, M_0) est *quasi-vivant* $\Leftrightarrow \forall t \in T : \exists M \in [M_0], M[t]$;

(N, M_0) est *pseudo-vivant* $\Leftrightarrow \forall M \in [M_0] : \exists t \in T, M[t]$;

(N, M_0) est *vivant* $\Leftrightarrow \forall t \in T : [\forall M \in [M_0] : \exists M' \in [M], M'[t]]$;

N est *structurellement vivant* si et seulement si $\exists M_0/(N, M_0)$ est vivant.

2.2.2.3 Propriétés structurelles de base

Un vecteur $f \in \mathbb{Z}^P$, $f \neq 0$, est un *P-invariant* si et seulement si ${}^t f.C = 0$.

On dénote par $\|f\|^+ = \{p \in P/f(p) > 0\}$ et par $\|f\|^- = \{p \in P/f(p) < 0\}$.

N est *conservatif* si et seulement si il existe un P-invariant f tel que $\|f\| = \|f\|^+ = P$.

$\|f\| = \{p \in P/f(p) \neq 0\}$ est appelé support de f . Si N est conservatif alors N est structurellement borné (i.e. borné pour tout M_0).

Un vecteur $g \in \mathbb{N}^T$, $g \neq 0$, est un *T-invariant* si et seulement si $C.g = 0$.

La propriété de vivacité, qui correspond à l'absence de situations globales ou locales de blocage, est étroitement liée à l'évolution des marquages des siphons. Un *siphon* est un sous-ensemble de places tel que l'ensemble de ses transitions en entrée soit inclus dans l'ensemble de ses transitions en sortie.

Un *siphon* est un ensemble non vide $S \subseteq P$ tel que $\bullet S \subseteq S^\bullet$. S est dit minimal si et seulement si il ne contient aucun autre siphon que lui-même.

On dénote par : $\max_{p^\bullet} = \max_{t \in p^\bullet} W(p, t)$ et $\min_{t \in \bullet p} W(t, p)$.

Une place p est dite *non-bloquante* si $\min_{t \in \bullet p} W(t, p) \geq \max_{p^\bullet}$.

Dans ce qui suit, on considère des réseaux à valuation homogène, i.e. $\forall p \in P, \forall t, t' \in p^\bullet, W(p, t) = W(p, t') = W(p)$

Un siphon S est dit *contrôlé* si et seulement si S est marqué dans tout marquage accessible i.e.

$\forall M \in [M_0], \exists p \in S/M(p) \geq W(p)$.

Conditions suffisantes de contrôle de siphons Si un siphon S de (N, M_0) satisfait une des deux conditions structurelles suivantes, alors S est contrôlé :

1. $\exists R \subseteq S$ tel que $R^\bullet \subseteq \bullet R$, R est initialement marqué et toutes ses places sont non-bloquantes.
2. \exists un P-invariant $f \in \mathbb{Z}^P$ tel que $S \subseteq \|f\|$ et $\forall p \in (\|f\|^- \cap S), W(p) = 1, \|f\|^+ \subseteq S$ et $\sum_{p \in P} [f(p)M_0(p)] > \sum_{p \in S} [f(p).(W(p) - 1)]$.

Un siphon vérifiant la condition 1 (resp. 2) est dit *contrôlé par trappe* (resp. *par invariant*).

Remarque Contrairement au contrôle par trappe, le contrôle par invariant est non monotone (i.e non préservation du contrôle lors de l'accroissement du marquage). Ceci explique largement la non monotonie de la vivacité des RdPs en général.

Un RdP (N, M_0) est dit *contrôlé* (ou satisfaire la CS-property) si et seulement si tous ses siphons minimaux sont contrôlés [23].

Un RdP (N, M_0) est dit *assez fortement contrôlé* si et seulement si tous ses siphons minimaux sont contrôlés soit par trappe soit par invariant.

Si tous les siphons sont contrôlés par trappe, (N, M_0) est dit *fortement contrôlé* (ou satisfaire la propriété de Commoner).

Nous rappelons ci-dessous deux relations bien connues entre vivacité et contrôlabilité d'un réseau :

Si (N, M_0) est contrôlé alors il est pseudo-vivant.

Si (N, M_0) est vivant alors il est contrôlé.

Ainsi, pour les RdPs où le contrôle est non seulement une condition nécessaire mais suffisante de vivacité, on a équivalence entre vivacité et contrôlabilité. C'est dans ce sens que les K-systèmes ont été récemment définis dans [21]. Ils forment la classe des RdPs où il y a équivalence entre contrôlabilité (CS-property), pseudo-vivacité et vivacité.

2.2.2.4 K-systèmes

Nous rappelons ici les principaux résultats concernant les K-systèmes et utiles à notre travail.

- **Marquage stable** : Soit (N, M_0) un RdP. Un marquage accessible $M^* \in [M_0]$ est *stable* si et seulement si T est partitionné en deux sous-ensembles T_D et T_L , où toutes les transitions de T_L (resp T_D) sont vivantes (resp mortes) pour M^* .
- **K-système** : (N, M_0) est un *K-système* si et seulement si pour tout marquage stable M^* , $T_D = T$ ou $T_L = T$. Ceci signifie que dès qu'une transition est vivante, tout le réseau est vivant.

Cette définition étant purement comportementale, dans [21] les auteurs ont identifié des classes de K-systèmes reconnaissables structurellement. Elles sont basées sur les concepts de place racine et de transition ordonnée.

- **Place racine** : r est une *place racine* pour $t \in r^\bullet$ si et seulement si $r^\bullet \subseteq p^\bullet, \forall p \in \bullet t$.
- **Transition ordonnée** : t est une *transition ordonnée* si et seulement si $\forall p, q \in \bullet t, p^\bullet \subseteq q^\bullet$ ou $q^\bullet \subseteq p^\bullet$.
- **Réseau ordonné** : N est un *RdP ordonné* si et seulement si toutes ses transitions sont ordonnées.
- **Réseau racine** : N est un *réseau racine* si et seulement si pour toute transition $t \in T$, il existe une place $r \in P$ qui est racine pour t .
- **Composante racine** : Soit N un réseau racine et $Root_N$ l'ensemble de ses places racines. La *composante racine de N* est le sous réseau défini comme suit :
 $Root(N) = (Root_N, T^*, F^*, W^*)$
 1. $T^* = Root_N^\bullet$.
 2. $F^* \subseteq (F \cap ((Root_N \times T^*) \cup (T^* \times Root_N)))$, tel que $(p, t) \in F^*$ si et seulement si $(p, t) \in F$ et p est une place racine pour t , et $(t, p) \in F^*$ si et seulement si $(t, p) \in F$.
 3. W^* est la restriction de la fonction de valuation sur F^* .
- **Réseau à fermeture consistante** : N est un RdP à *fermeture consistante* si et

seulement si N est conservatif et l'ensemble de ses transitions non ordonnées est inclus dans le support de chaque T-invariant.

Pour les réseaux ordonnés, les réseaux racines dont les composantes racines sont conservatives et fortement connexes, et les réseaux à fermeture consistante, on a l'équivalence entre vivacité et contrôlabilité.

Dans ce qui suit, nous exploitons et étendons cette caractérisation structurelle de la vivacité sur ces trois classes de K-systèmes dans le contexte des réseaux workflow.

2.2.3 Les réseaux workflow (WF-nets)

Nous avons évoqué dans la section 1.7.2.1 la modélisation d'un processus workflow par un RdP. Cette modélisation a abouti à la proposition d'une sous-classe des RdPs, dite réseau de workflow (ou encore workflow-net : WF-net) [8], dédiée à la définition des processus workflow. Dans un WF-net, les tâches sont modélisées par les transitions et les dépendances causales sont modélisées par les places et les arcs. Formellement, un WF-net est défini comme suit :

Définition 2.1 *Un réseau de Petri (N, M_0) est un WF-net si et seulement si :*

1. N admet une seule place source i (i.e. $\bullet i = \emptyset$) dite place initiale.
2. N admet une seule place puit f (i.e. $f \bullet = \emptyset$) dite place finale.
3. Pour chaque nœud $n \in P \cup T$, il existe un chemin de i à n et un chemin de n à f .

Un WF-net devrait satisfaire la condition suivante : Pour n'importe quelle instance, il va éventuellement terminer et au moment où il termine, il y a un jeton dans la place f et toutes les autres places sont vides. De plus, il ne devrait y avoir aucune tâche morte, i.e. il devrait être possible d'exécuter une tâche arbitraire en suivant un itinéraire approprié sur le WF-net. Ces deux conditions correspondent à la propriété de cohérence [7, 8].

Définition 2.2 *Un WF-net $N = (P, T, F)$ est cohérent si et seulement si :*

1. Pour chaque marquage M accessible du marquage initial M_i , il existe une séquence de franchissement ramenant du marquage M au marquage final M_f .
Formellement : $\forall M \in [M_i], M_f \in [M]$.
2. L'état final est l'unique état accessible de l'état initial avec au moins un jeton dans la place f . Formellement : $\forall M \in [M_i] : M(f) \geq 1 \Rightarrow M = M_f$.
3. Il n'y a pas de transitions mortes dans N . Formellement : $\forall t \in T, \exists M \in [M_i] / M[t]$.

2.3 Vérification paramétrée de la cohérence des WF-nets

Les WF-nets, initialement introduits, traitent un seul cas (instance) de la procédure modélisée (i.e. un seul jeton existe au début dans la place i). Dans notre cas, nous considérons les processus workflow modélisant k instances.

Un WF-net (N, M_0) est caractérisé par :

1. N est connexe et admet une seule place source i (i.e. $\bullet i = \emptyset$) et une seule place puit f (i.e. $f\bullet = \emptyset$) et pour marquage initial paramétré $M_0 = ki$.
2. k désigne le nombre d'instances du processus workflow modélisé.
3. La place source i modélise l'état initial des processus à instancier.
4. La place f modélise leur terminaison.
5. N^* le réseau obtenu à partir de N par ajout d'une transition t^* et deux arcs (i, t^*) et (t^*, f) , est fortement connexe. N^* est appelé fermeture de N .

La propriété de cohérence des k instances du processus workflow correspond à la propriété de k -cohérence.

2.3.1 La k -cohérence

Un WF-net est *k-cohérent* si et seulement si :

1. A partir de tout état accessible de $M_0 = ki$, il existe une séquence de franchissements conduisant au marquage final (terminaison des k instances).
Formellement, $\forall M \in [ki], kf \in [M]$;
2. L'état final est le seul état accessible de l'état initial avec au moins k jetons dans la place f . Formellement, $\forall M \in [ki] : M(f) \geq k \Rightarrow M = kf$;
3. Toute transition (activité) peut être franchie (exécutée) : (N, ki) est quasi-vivant.

2.3.2 La Cohérence généralisée

Un WF-net est *cohérent* si et seulement si il est k -cohérent $\forall k \in \mathbb{N}^*$ [59]. Cette propriété est prouvée décidable dans [58].

Dans ce qui suit, et en exploitant des résultats récents de la théorie structurale des RdPs, nous visons d'une part à caractériser structurellement la k -cohérence et d'autre part à établir une condition structurale nécessaire et/ou suffisante à la cohérence généralisée. Pour ce faire, nous rappelons d'abord la propriété non structurale reliant la k -cohérence à la propriété comportementale de vivacité.

Théorème 2.1 *Un WF-net N est k -cohérent si et seulement si (N^*, ki) est bien formé (i.e. N^* est vivant et borné).*

Preuve : donnée dans [22].

Le contrôle étant une condition nécessaire de vivacité, on peut énoncer ce qui suit : si N est k -cohérent alors (N^*, ki) est borné et contrôlé.

Dans ce qui suit, nous exploitons la caractérisation structurale des K-Systèmes et l'étendons dans le contexte des workflows.

2.3.3 Caractérisation structurale des K-workflow-nets (KWF-nets)

Ce travail vise la vérification de la cohérence des workflows en s'appuyant sur la théorie structurale des RdPs. Les travaux précédents et s'inscrivant dans cette même direction

[24] ont identifié principalement deux classes de WF-nets pour lesquelles la k-cohérence est caractérisée structurellement et décidable en temps polynomial. Il s'agit des réseaux workflow à choix libre (FCWF-nets) et des réseaux workflow ENSeC.

Un WF-net est un WF-net à choix libre si et seulement si pour toute paire de transitions t_1 et t_2 , $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implique $\bullet t_1 = \bullet t_2$.

Un WF-net est un WF-net ENSeC (Elementary Non-Self Controlling) si et seulement si pour chaque paire de transitions t_1 et t_2 tel que $\bullet t_1 \cap \bullet t_2 \neq \emptyset$, il n'existe pas de chemin sans-conflit menant de t_1 à t_2 .

Les chemins connectent les nœuds par une séquence d'arcs. Un chemin C d'un nœud n_1 à un nœud n_k est une séquence $\langle n_1, n_2, \dots, n_k \rangle$ tel que $(n_i, n_{i+1}) \in F$ pour $1 \leq i \leq k-1$. C est sans-conflit (conflict-free) si et seulement si pour chaque place n_j sur C et chaque transition n_i sur C , $j \neq i-1 \Rightarrow n_j \notin \bullet n_i$.

Pour les FCWF-nets et les WF-nets ENSeC, la vérification de la k-cohérence se ramène à décider si la fermeture est conservative et si chacun de ses siphons est également une trappe. Ce problème peut être résolu en temps polynomial à l'aide des algorithmes développés dans [16].

La figure 2.1 illustre un FCWF-net. Sa fermeture N^* possède un siphon minimal $(i, p_1, p_2, p_3, p_5, f)$ contenant strictement une trappe (i, p_2, p_3, p_5, f) . Donc en vertu du théorème 2.1, N ne peut être k-cohérent.

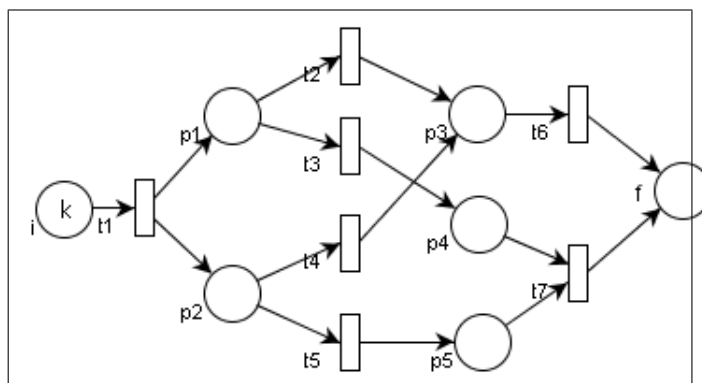


FIGURE 2.1 – Un WF-net à choix libre non cohérent

Partant des K-Systèmes, nous avons identifié trois nouvelles classes de KWF-nets qui étendent le pouvoir de description des classes précédentes tout en étant reconnaissables de manière structurelle et efficace.

Définition 2.3 Soit N un WF-net. N est appelé un K-workflow-net (KWF-net) si et seulement si N^* est un K-système.

Théorème 2.2 Soit N un KWF-net. N est k-cohérent si et seulement si N^* est borné et contrôlé.

Preuve : Conséquence immédiate de l'équivalence entre la vivacité et la CS-property pour les K-systèmes [21] et théorème 2.1.

Définition 2.4 *Soit N un WF-net. N est un WF-net ordonné (OWF-net) si et seulement si N^* est un réseau ordonné (i.e. N est ordonné).*

Définition 2.5 *Soit N un WF-net. N est un RootWF-net si et seulement si N^* est un réseau racine dont la composante racine est conservative et fortement connexe.*

Définition 2.6 *Soit N un WF-net. N est un WF-net à fermeture consistante (CCWF-net) si et seulement si N^* est un RdP à fermeture consistante.*

L'intérêt majeur de ces trois nouveaux réseaux workflow est double. D'une part, ils autorisent la modélisation de patrons de synchronisation complexes rencontrés dans le contexte pratique des workflows ; et d'autre part, leur cohérence généralisée est vérifiable efficacement.

Remarque La classe des OWF-nets contient strictement la sous classe des réseaux workflow à choix libre (FCWF-nets). Aussi, Il n'y a aucune relation d'inclusion entre les RootWF-nets et les CCWF-nets. Par exemple, le RootWF-net de la figure 2.2 n'est pas un CCWF-net et le CCWF-net de la figure 2.3 n'est pas un RootWF-net.

La classe des RootWF-nets contient strictement les FCWF-nets (leur composante racine coïncide avec la fermeture du réseau). La sous-classe de CCWF-nets contient strictement la sous classe des CFWF-nets (WF-nets sans circuit).

Naturellement, ces trois réseaux workflow (RootWF-net, CCWF-net et OWF-net) étant des sous classes de KWF-nets, leur k-cohérence est équivalente aux propriétés de finitude (le caractère borné) et de contrôlabilité (CS-property) (théorème 2.2).

Bien que la vivacité ne soit pas une propriété monotone pour les RdPs en général y compris les K-Systèmes, nous avons établi des conditions sous lesquelles la cohérence des KWF-nets est monotone (i.e préservée lors de l'accroissement du marquage initial).

Théorème 2.3 *Soit N un KWF-net. Si N^* est conservatif et assez fortement contrôlé alors N est cohérent pour tout $k > 0$.*

Pour démontrer ce résultat, nous exploitons des caractéristiques structurelles propres aux WF-nets.

Lemme 2.1 *Soit N un WF-net cohérent alors :*

1. *Tout siphon de N^* contient la place source i (et la place finale f puisque $\bullet i = \{t^*\}$ et $\bullet t^* = \{f\}$).*
2. *Toute trappe de N^* contient la place finale f (et la place source i puisque $f^\bullet = \{t^*\}$ et $t^{*\bullet} = \{i\}$).*

Preuve :

1. Par contradiction, supposons que N^* contient un siphon S ne contenant pas les places i et f alors S est initialement non contrôlé et donc N ne peut être cohérent.
2. Par contradiction, supposons que N^* admet initialement une trappe T non marquée. N étant cohérent (à fortiori k-cohérent), on assure par la condition (3) de la k-cohérence, l'existence d'un marquage accessible M dans lequel T serait marquée, ce qui contredirait la condition (1).

Lemme 2.2 *Soit N un WF-net et S un siphon de N^* . Si S satisfait l'une des deux conditions structurelles suivantes alors S est contrôlé pour tout $k > 0$.*

1. $\exists R \in S$ tel que : $R^\bullet \subseteq \bullet R$ (S contient une trappe R).
2. \exists un P -invariant I tel que $S \subseteq \|I\|$, $\|I\|^+ \subseteq S$.

Preuve :

1. Soit S un siphon de N^* contenant une trappe. La place i étant incluse dans S , S est contrôlé dès que $k > 0$.
2. Soit S un siphon dans N^* vérifiant la condition (2). La place i étant incluse dans S , la seconde condition de contrôle est satisfaite $\forall k > 0$. S est donc contrôlé par invariant $\forall k > 0$.

Une conséquence immédiate du lemme précédent est que la 1-cohérence implique la cohérence généralisée pour tout WF-net dont la fermeture est conservative et assez fortement contrôlée.

Nous illustrons ci-dessous chacune des trois sous classes identifiées par un exemple.

Exemple 2.1

La figure 2.2 illustre un exemple de RootWF-net.

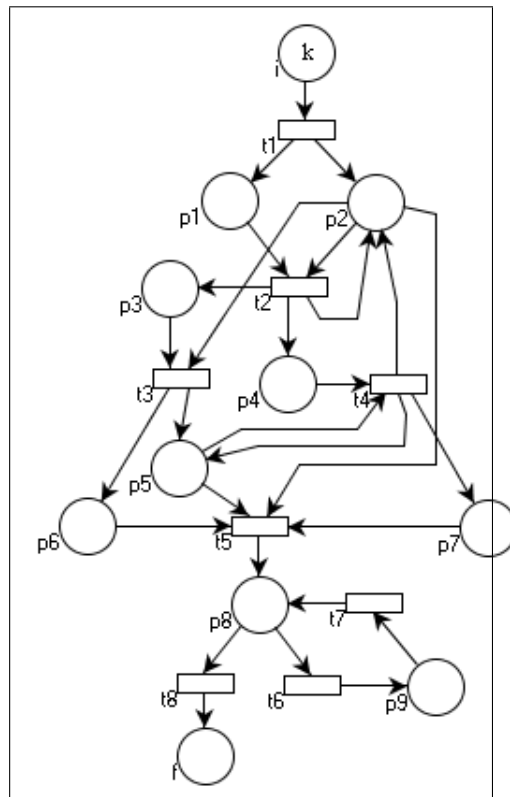


FIGURE 2.2 – Un RootWF-net cohérent

2.3. VÉRIFICATION PARAMÉTRÉE DE LA COHÉRENCE DES WF-NETS

Ce WF-net n'est pas un pas un CCWF-net. Sa fermeture N^* est un réseau racine et sa composante racine $N^* - (p_2, p_5)$ est fortement connexe. De plus, N^* est conservatif bien que non décomposable en machine à états. N^* admet cinq siphons minimaux :

$$S_1 = i, p_1, p_3, p_5, p_8, p_9, f ;$$

$$S_2 = i, p_1, p_3, p_6, p_8, p_9, f ;$$

$$S_3 = i, p_1, p_4, p_7, p_8, p_9, f ;$$

$$S_4 = i, p_2, p_5, p_8, p_9, f ;$$

$$S_5 = i, p_2, p_4, p_8, p_9, f .$$

On peut vérifier que les quatre siphons S_1, S_2, S_3 et S_4 sont contrôlés par trappe et S_5 contrôlé par l'invariant $i + p_2 + p_4 + f - p_3$. N^* est donc assez fortement contrôlé. En vertu du théorème 2.3, N est cohérent pour tout $k > 0$.

Remarque. On peut noter que pour le marquage initial $i + p_3 > i$ (exclu dans le contexte de workflow où seule la place source est initialement marquée), le siphon S_5 serait non contrôlé. La vivacité de N^* n'est plus assurée (en effet l'état d'inter-blocage $p_1 + p_5 + p_6$ est accessible).

Exemple 2.2

La figure 2.3 illustre un exemple de CCWF-net. Ce WF-net n'est pas un RootWF-net. Sa fermeture N^* est conservative et assez fortement contrôlée dès que $k > 0$. En effet, elle contient deux siphons minimaux contrôlés aussi bien par trappe que par invariant. En vertu du théorème 3, N est cohérent pour tout $k > 0$.

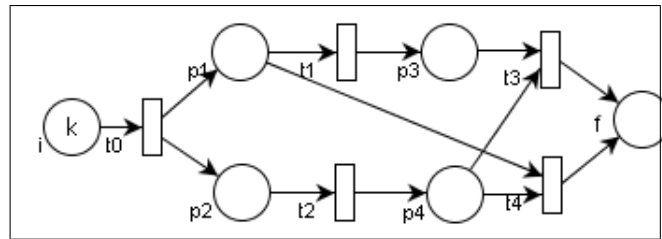


FIGURE 2.3 – Un CCWF-net cohérent

Exemple 2.3

La figure 2.4 illustre un exemple de OWF-net. Sa fermeture N^* n'est ni un WF-net FC ni un réseau ENSeC [26]. N^* est conservatif et fortement contrôlé. En effet, tout siphon minimal de N^* est une trappe. En vertu du théorème 2.3, N est cohérent pour tout $k > 0$.

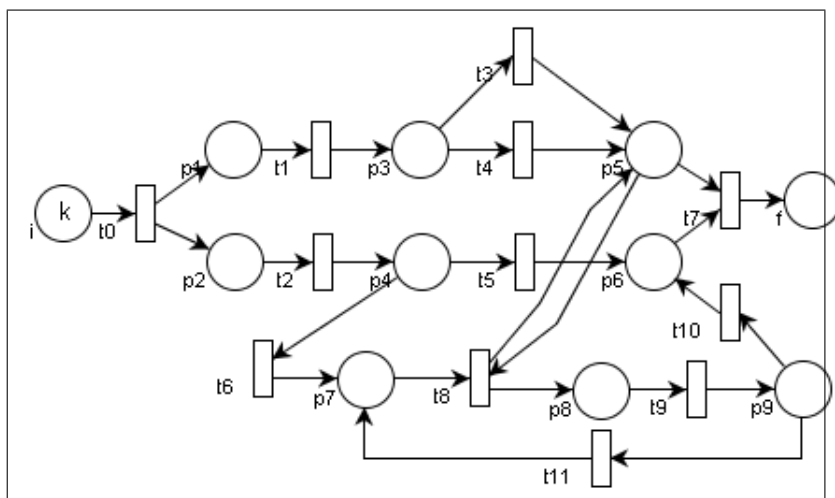


FIGURE 2.4 – Un OWF-net cohérent

2.4 Vérification de la cohérence sous contraintes d'allocation de ressources

Dans la section précédente, nous avons considéré le problème de la cohérence des workflows en faisant abstraction des ressources (partagées) nécessaires pour l'exécution des tâches. Les ressources sont durables tout au long de l'exécution du workflow. Une ressource acquise sera par la suite libérée et une ressource libérée a été précédemment requise. Un processus workflow sous contraintes de ressources (WFR-net) peut être représenté à l'aide du RdP suivant, appelé WFR-net [22, 58].

Un WFR-net est un tuple $N_R = (P \cup P_R, T, F \cup F_R, W \cup W_R)$ où :

1. $N = (P, T, F, W)$ est un WF-net borné
2. $P_R \neq \emptyset \wedge P \cap P_R = \emptyset$ (P_R est l'ensemble des ressources)
3. $F_R \subseteq (P_R \times T) \cup (T \times P_R)$ (relation flot pour les ressources)
4. $\forall r \in P_R, \exists I_r \geq 0 : {}^t I_r.C = 0 \wedge \|I_r\| \cap P_R = \{r\}$ (préservation des ressources).

On désigne par $ki + R$ le marquage initial du WFR-net où R est le marquage initial des places ressources. Il faut noter que dans ce modèle, une activité (transition) peut se référer à plusieurs ressources et que les sous réseaux induits par les invariants ne sont pas nécessairement des machines à états.

2.4.1 La (k, R) -cohérence

Dans un premier temps, nous étendons la propriété de cohérence en intégrant la disponibilité initiale des ressources [58].

Un WFR-net N_R est dit (k, R) -cohérent si et seulement si :

2.4. VÉRIFICATION SOUS CONTRAINTES DE RESSOURCES

1. A partir de tout état accessible de $M_0 = ki + R$, il existe une séquence de franchissements conduisant au marquage final $kf + R$ (terminaison des k instances et libération de l'ensemble des ressources). Formellement : $\forall M \in [ki + R], kf + R \in [M]$.
2. L'état final est le seul état accessible de l'état initial avec au moins k jetons dans la place f . Formellement : $\forall M \in [ki + R] : M(f) \geq k \Rightarrow M = kf + R$.
3. Toute transition (activité) peut être franchie (exécutée) : $(N, ki + R)$ est *quasi-vivant*.

Remarques En raison des invariants associés aux places ressources (condition 4 de la définition des WFR-nets), il est aisé de vérifier que sur chaque état M accessible du marquage initial, nous avons $M_R \leq R$ (où M_R est le sous-marquage obtenu par projection de M sur les places ressources). Par ailleurs, les places ressources étant additionnelles au processus workflow sous-jacent N , elles ne peuvent que restreindre son comportement. La k -cohérence de N est alors une condition nécessaire à la (k, R) -cohérence de N_R .

Proposition 2.1 *Soit N_R un WFR-net. Si N_R est (k, R) -cohérent alors N est k -cohérent.*

Preuve : N_R étant (k, R) -cohérent, nous pouvons assurer que N^* est quasi-vivant et que ki est un état d'accueil (par franchissement de t^* nous revenons à ki). N^* est donc borné et vivant. En vertu du théorème 2.1, N est k -cohérent.

Dans ce qui suit, on dénote par N_R^* le réseau obtenu à partir de N_R en substituant à N sa fermeture N^* . N_R^* est appelé fermeture de N_R .

Théorème 2.4 *Soit N_R un WFR-net. N_R est (k, R) -cohérent si et seulement si $(N_R^*, ki + R)$ est vivant et borné.*

Preuve :

(\Rightarrow)

$(N_R, ki + R)$ est (k, R) -cohérent. Montrons d'abord que $(N_R^*, ki + R)$ est borné. Supposons que N_R n'est pas borné, donc $\exists M_1 \in [ki + R]$ et $\exists M_2 \in [M_1]$ tel que $M_2 > M_1$.

Par la condition 1 de la (k, R) -cohérence, il existe une séquence de transitions $\sigma \in T^*/M_1[\sigma]kf + R$ et donc puisque $M_2 > M_1$, $\exists M/M_2[\sigma]M/M > kf + R$. Ceci contredit la condition 2 de la (k, R) -cohérence. $(N_R, ki + R)$ est donc borné (N et les places ressources sont bornés) et par conséquent $(N_R^*, ki + R)$ est aussi borné.

Montrons maintenant que $(N_R^*, ki + R)$ est vivant : comme $(N_R, ki + R)$ est (k, R) -cohérent, nous avons $\forall M \in [ki + R] : kf + R \in [M]$ (condition 1). Alors, par franchissement de t^* , nous obtenons : $\forall M \in [ki + R] : ki + R \in [M]$, i.e. $ki + R$, est un état d'accueil de $(N_R^*, ki + R)$. Aussi, toutes les transitions sont quasi-vivantes (condition 3), $(N_R^*, ki + R)$ est donc vivant.

(\Leftarrow)

Si $(N_R^*, ki + R)$ est vivant et borné alors il est trivial que N^* soit bien formé (vivant et borné) et donc N est k -cohérent et N_R est (k, R) -cohérent.

Corollaire 2.1 *Soit N_R un WFR-net. Si N_R est (k, R) -cohérent alors $(N_R^*, ki + R)$ est borné et contrôlé.*

2.4.2 La R-cohérence

Soit N_R un WFR-net. N_R est *R-cohérent* si et seulement si il existe un marquage R_0 tel que N_R est (k, R) -cohérent pour tout $R \geq R_0$.

Théorème 2.5 *Soit $(N_R, ki + R)$ un WFR-net conservatif. Si N est k -cohérent alors il existe R_0 tel que pour tout $R \geq R_0$, N_R est contrôlé.*

Preuve :

N est k -cohérent, il suffit alors de montrer que $\exists R_0$ tel que tout siphon S dans N_R contenant (au moins) une place ressource soit contrôlé dès que $R \geq R_0$.

Considérons un tel siphon S et supposons qu'il soit non contrôlé. On dénote par $I(r)$ le P-invariant minimal associé à la ressource r , et par $f(p)$ le P-invariant minimal associé à p ($f(p)$ existe puisque N_R est conservatif).

Soit $I_S = \sum_{r \in S} I(r)$, $Out(S) = \|I_S\| \setminus S$, $H_S = \sum_{p \in Out(S)} f(p)$, $\lambda_S = \max_{p \in Out(S) \cap \|H_S\|} (I_S(p))$, $Z_S = I_S - \lambda_S \cdot H_S$

Nous pouvons toujours trouver R_0 satisfaisant la condition suivante :

${}^t Z_S \cdot M_0 > \sum_{p \in S} (Z_S(p) \cdot (\max_{p \bullet} - 1))$ et tel que $\forall R > R_0$, tout siphon S de N_R est contrôlé.

2.4.3 La cohérence généralisée sous contraintes de ressources

Soit N_R un WFR-net. N_R est *cohérent* ssi il existe R_0 tel que N_R est (k, R) -cohérent pour tout k et pour tout $R \geq R_0$.

Corollaire 2.2 *Soit N_R un WFR-net conservatif. Si N_R est cohérent alors N est cohérent.*

Preuve : Conséquence de la proposition 2.1 et de la définition de la cohérence.

En se basant sur les résultats précisés ci-dessus, nous définissons des classes de WFR-nets pour lesquelles des conditions structurelles nécessaires et suffisantes à la cohérence sont établies et peuvent être vérifiées efficacement.

Définition 2.7 *Soit N_R un WFR-net. N_R est un OWFR-net si et seulement si N_R^* est un réseau ordonné.*

Définition 2.8 *Soit N_R un WFR-net. N_R est un RootWFR-net si et seulement si N_R^* est un réseau racine et sa composante racine est bornée et fortement connexe.*

Proposition 2.2 *Soit N_R un WFR-net. Si N est un CCWF-net alors N_R est aussi un CCWF-net. Il sera dénommé CCWFR-net.*

Preuve : Nous devons montrer que l'ensemble des T-invariants de N coïncide avec celui de N_R i.e $C_R \cdot g = 0 \Leftrightarrow C \cdot g = 0$.

$C_R \cdot g = 0 \Rightarrow C \cdot g = 0$ est trivial.

Il reste à montrer que $C \cdot g = 0 \Rightarrow C_R \cdot g = 0$. Cette implication est assurée grâce aux invariants associés aux places ressources (condition 4 de la définition des WFR-nets).

2.5. PROCÉDURE DE VÉRIFICATION STRUCTURELLE DE LA COHÉRENCE DES PROCESSUS WORKFLOW

Théorème 2.6 *Soit N_R un OWFR-net, un RootWFR-net ou un CCWFR-net. N_R est R-cohérent si et seulement si N_R^* est borné et contrôlé.*

Preuve : Conséquence du théorème 2.5 et de l'équivalence entre la contrôlabilité (CS-property) et la vivacité pour les K-systèmes.

Corollaire 2.3 *Soit N_R un OWFR-net, un RootWFR-net ou un CCWFR-net conservatif et assez fortement contrôlé alors N_R est cohérent.*

Preuve : Similaire à la preuve du théorème 2.3 (la R-cohérence implique la cohérence).

La figure 2.5 illustre un exemple de CCWFR-net conservatif. Ses siphons sont :

$$S_1 = i, p_1, p_2, p_3, p_4, p_5, p_6, f ;$$

$$S_2 = r_2, p_2, p_3, p_5 ;$$

$$S_3 = r_1, p_2, p_3, p_4, p_5, p_6.$$

S_1 et S_2 sont supports de flots positifs. S_1 est contrôlé dès que $M_0(i) > 0$, S_2 est contrôlé dès que $M_0(r_2) > 0$. S_3 est un siphon minimal strict (il n'est pas une trappe) :

$$I_{S_3} = I_{r_1} = r_1 + 2p_1 + 4p_2 + 3p_3 + 5p_4 + 3p_5 + 3p_6$$

$$Out(S_3) = \{p_1\} ;$$

$$\lambda_{S_3} = 2 ;$$

$$H_{S_3} = 1 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + f ;$$

$$Z_{S_3} = r_1 - 2i + 2p_2 + p_3 + 3p_4 + p_5 + p_6 - 2f.$$

Par conséquent, S_3 est contrôlé pour tout k satisfaisant $r_1 > 2k$.

Alors N_R est assez fortement contrôlé. Il est donc cohérent, i.e. (k,R)-cohérent $\forall k$ et $\forall R \geq R_0$ avec $R_0 = (2k + 1)r_1 + r_2$.

2.5 Procédure de vérification structurelle de la cohérence des processus workflow

Dans cette section, nous nous proposons d'implémenter l'approche structurelle de la vérification de la cohérence des processus workflow. Pour ce faire, nous établissons tout d'abord une procédure de calcul de tous les siphons minimaux dans un WF-net. Ensuite, nous abordons les procédures de vérification de la CS-Property et de vérification structurelle de la cohérence pour certaines sous-classes des WF-nets.

2.5.1 Calcul de tous les siphons minimaux dans un RdP

Le calcul des siphons minimaux n'est pas simple. En effet, leur calcul explore $2^{|P|}$ sous ensembles de l'ensemble des places P du RdP en question et vérifie la structure de chacun de ces sous ensembles. Il est donc clair que ce problème est d'une complexité importante relativement à la taille du réseau. Pour réduire cette complexité plusieurs algorithmes ont été proposés. Nous décrivons brièvement ceux les plus répandus.

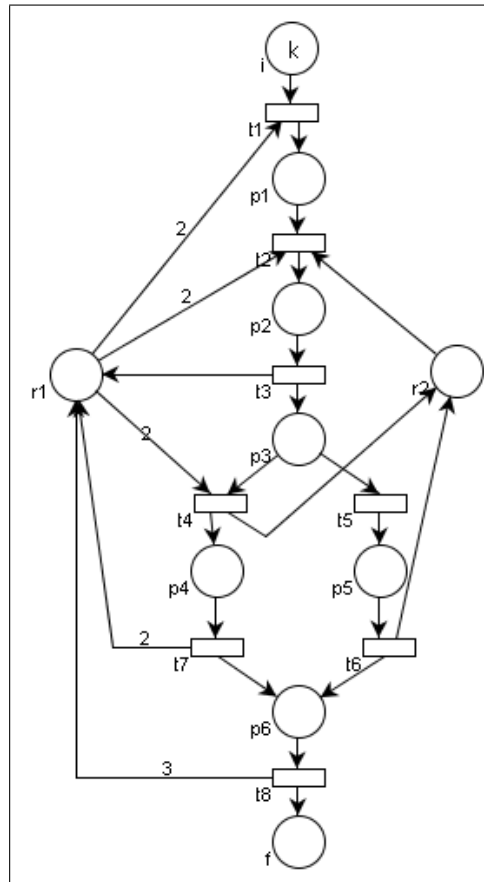


FIGURE 2.5 – Un CCWFR-net cohérent

2.5.1.1 Algorithmes existants

– La procédure Forward-Deletion

La procédure Forward-Deletion a été adoptée comme outil pour l'extraction d'un seul siphon minimal pour un RdP N ordinaire [109]. L'idée de la procédure se résume comme suit :

Soient un RdP $N = (P, T, E)$ et un ensemble de places $P_I \subset P$, avec P_I peut être vide. Si $N \setminus P_I$ possède au moins une transition $t \in T$ telle que $L_N(t) \cap (P - P_I) = \emptyset$, avec $L_N(t)$ est l'input de t et $R_N(t)$ est l'output de t ; alors $\forall P_S \subset (P - P_I)$, nous avons $R_N(t) \cap P_S = \emptyset$ et donc $t \notin L_N(P_S)$.

La description formelle de la procédure est donnée dans la figure 2.6.

La complexité de Forward-Deletion (N, P_I) est de $O(|P| + |T| + |E|)$ [109].

Il y a plusieurs autres procédures qui lui sont relatives et qui l'utilisent pour extraire plus de siphons mais qui ont des complexités supérieures à celle de Forward-Deletion.

– La procédure EST

La procédure EST (Extracting Siphon Trap) [111] trouve une classe maximale de siphons et de trappes mutuellement disjoints. Elle utilise la procédure $Find_NWC$

2.5. PROCÉDURE DE VÉRIFICATION STRUCTURELLE DE LA COHÉRENCE DES PROCESSUS WORKFLOW

```

Procédure Forward-Deletion (N, PI);
/* Entrée N = (P, T, E) et PI */
/* Sortie NF = (PF, TF, EF) ⊂ N */
begin
1. NF = (PF, TF, EF) ← N \ PI;
2. Tsrc ← {t ∈ TF / LNF(t) = ∅}
3. While (Tsrc ≠ ∅) do
    begin
4. Choisir t ∈ Tsrc; Tsrc ← Tsrc \ {t}
5. P' ← RNF(t); Ttemp ← RNF(P');
6. NF ← NF \ ((t) ∪ P');
7. Tsrc ← Tsrc ∪ {t ∈ Ttemp ∩ TF | LNF(t) = ∅}
    end
end;

```

FIGURE 2.6 – Description de la procédure Forward-Deletion

(N', P_x) avec $P_x \subset P'$. Cette procédure trouve, s'il existe, au moins un sous réseau faiblement connecté N'' de N' ; $N'' = (P'', T'', E'')$ avec $P_x \subset P''$ si $P_x \neq \emptyset$ ou bien elle fait sortir un sous réseau faiblement connecté N' si $P_x = \emptyset$. $P'' = \emptyset$ sera l'output si aucun tel ensemble n'existe.

La complexité de la procédure EST est de l'ordre de $O(|P|^2(|P| + |T| + |E|))$ [111].

– La procédure FDST

Le principe de la procédure FDST (Find Disjoint Minimal Siphons Traps) est de trouver les siphons et les trappes minimaux et disjoints. Cette procédure extrait une classe maximale de siphons et de trappes minimaux mutuellement disjoints [110].

La sous procédure *Find – N_{WC}* (N', P_x) avec $P_x \subset P'$ cherche s'il existe un sous réseau représentant une composante faiblement connectée du réseau original N' ; $N'' = (P'', T'', E'')$ avec $P_x \subset P''$ si $P \neq \emptyset$; sinon elle fait retourner un sous réseau faiblement connecté N' si $P_x = \emptyset$. $P'' = \emptyset$ sera l'output si aucun tel ensemble n'est retrouvé.

Ayant un sous réseau faiblement connecté $N' = (P', T', E')$ de N , la procédure *Find – S*(N') cherche un siphon P_S de N' . Pour chaque siphon faiblement connecté N_S de N , *Reduction*(N_S, P_x) extrait un ensemble de siphons minimaux $P_{MS} \subset P_S$ de N avec l'utilisation de FB-Deletion (Forward-Back Deletion) et *Find – N_{wc}* (Find a weakly connected component).

La complexité de FDST est de $O(|P|^2(|P| + |T| + |E|))$ [13].

2.5. PROCÉDURE DE VÉRIFICATION STRUCTURELLE DE LA COHÉRENCE DES PROCESSUS WORKFLOW

– La procédure GMST

GMST est l'abréviation de Generating-disjoint Minimal Siphons Traps. Cette procédure génère une classe maximale de siphons et de trappes minimaux mutuellement disjoints [109].

La complexité de GMST est de $O(|P|^2(|P| + |T| + |E|))$, elle est égale à celle de FDST mais cette procédure a prouvé une performance plus élevée que celle de FDST. Ceci a été montré par des résultats expérimentaux dans [109].

– L'algorithme de R.Cordone et al.

R.Cordone, L.Ferrarini et L.Piroddi proposent dans [37, 38] des résultats intéressants pour le calcul de l'ensemble des siphons minimaux au sein d'un RdP ordinaire. Cet algorithme est basé sur des propriétés théoriques des RdPs qui permettent la réduction du problème du calcul des siphons minimaux et ce par une réduction du RdP à chaque itération pour simplifier la tâche.

L'objectif de l'algorithme est de trouver tous les siphons minimaux dans un RdP contenant les places d'un ensemble donné \tilde{P} ; l'idée de l'algorithme peut se résumer en trois étapes :

1. **Etape 1** : Trouver un siphon générique (qui n'est pas nécessairement minimal).
2. **Etape 2** : Trouver un siphon minimal contenu dans ce siphon générique.
3. **Etape 3** : Décomposer le problème en sous problèmes correspondant chacun à la recherche dans un sous réseau, pour chaque sous problème appliquer la même procédure à partir de l'étape 1.

En notant par n le nombre des places et m celui des arcs, la recherche d'un siphon générique peut être effectué en un temps linéaire relativement au nombre des places n i.e. sa complexité est de l'ordre de $O(n)$; la recherche du premier siphon minimal au sein du siphon donné est donc de l'ordre de $O(nm)$. Le nombre des appels récursifs à la principale procédure est au maximum 2^n puisque chaque appel fait générer un siphon distinct et il existe au maximum 2^n siphons dans le réseau. Ainsi la complexité totale de cet algorithme est de l'ordre de $O(nm2^n)$.

2.5.1.2 Choix d'un algorithme

Nous récapitulons dans le tableau 2.1 l'étude bibliographique faite sur les algorithmes de calcul des siphons ainsi que leurs complexités.

Procédure	Complexité
Forward-Deletion	$O(P + T + F)$
EST	$O(P ^2(P + T + F))$
FDST	$O(P ^2(P + T + F))$
GMST	$O(P ^2(P + T + F))$
Algorithme de Cordone et al.	$O(P F 2^{ P })$

TABLE 2.1 – Différentes procédures de calcul des siphons dans un RdP

2.5. PROCÉDURE DE VÉRIFICATION STRUCTURELLE DE LA COHÉRENCE DES PROCESSUS WORKFLOW

Nous remarquons que la procédure Forward-Deletion possède un temps optimum par rapport à FDST et GMST mais il faut souligner que cette procédure extrait un unique siphon minimal donc pour extraire tous les siphons minimaux à l'aide de cette procédure il faut l'exécuter itérativement jusqu'à vider l'ensemble de places ce qui rendra sa complexité la pire entre les complexités des autres algorithmes trouvés.

Ayant comme objectif primordial de cette étape, le calcul des siphons minimaux en un meilleur temps, nous trouvons que même si l'algorithme EST possède un temps d'exécution équivalent à celui de FDST et GMST, ceci ne nous présente pas un grand intérêt puisque cette procédure engendre une classe maximale de siphons qui ne sont pas nécessairement tous minimaux.

Les procédures FDST et GMST possèdent des temps d'exécution égaux mais l'expérience a approuvé l'efficacité de GMST par rapport à FDST. Une description formelle complète de GMST n'a pas été donnée.

Le dernier algorithme possède un temps optimum par rapport aux autres, en plus il est le seul qui calcule tous les siphons minimaux.

Nous proposons donc d'adopter l'algorithme de Cordone et al. pour les raisons suivantes :

- Il possède une complexité optimale par rapport aux autres algorithmes. Cette optimalité est due au fait que cet algorithme est basé sur une réduction du réseau en appliquant quelques propriétés simples et en dissociant le problème en sous problèmes possédant de plus petites complexités.
- Cet algorithme est bien adapté à un calcul facile des siphons contenant un ensemble particulier de places (qui est utile pour la prévention de deadlocks au sein des systèmes de workflows).
- Cet algorithme assure le calcul direct de tous les siphons minimaux sans avoir à calculer tous les siphons du réseau puis vérifier s'ils sont minimaux.
- Notre dernière raison est que, contrairement aux autres algorithmes, cet algorithme est applicable à n'importe quelle classe de RdP.

A travers cette étude nous avons énuméré, étudié et critiqué différents algorithmes d'extraction de siphons et de siphons minimaux au sein des RdPs. Parmi ces derniers, nous avons choisi un algorithme, auquel nous proposons d'y intégrer une méthode d'optimisation afin d'améliorer sa complexité et d'implémenter cet algorithme optimisé et de prouver sa fiabilité sur le plan pratique.

2.5.2 Description du nouvel algorithme

La notion de circuit alterné est d'une ultime importance quand au calcul des siphons minimaux d'un RdP. En effet, il a été démontré dans [25] et [26] que si une place p d'un circuit c appartient à un siphon alors toutes les places de c appartiennent aussi à ce siphon. Nous rappelons ci-dessous la définition d'un circuit alterné ainsi que celle d'une transition propre.

Définition 2.9 *Soit t une transition, t est dite transition de sortie propre d'une place p si $|\bullet t| = 1$ (i.e. $\bullet t = p$) et $t^\bullet \neq \{p\}$.*

Définition 2.10 Soit c un circuit composé de places et de transitions, si $\forall t \in c, |\bullet t| = 1$ alors c est dit circuit alterné.

Vue l'importance des circuits alternés dans le calcul des siphons minimaux, nous proposons d'effectuer la recherche des circuits alternés et la contraction des places de chaque circuit avant la recherche de siphons elle-même.

Le nouvel algorithme de calcul de tous les siphons minimaux, que nous proposons, consiste donc en deux phases :

- **Phase 1.** Réduction du problème par contraction du réseau :
 - a. Trouver un circuit alterné
 - b. Contracter ce circuit en une seule place
 - c. Répéter la recherche et la contraction tant que possible
- **Phase 2.** Décomposition du problème selon le principe de "diviser pour régner" de façon récursive en sous-problèmes, résolution de chacun de ces sous-problèmes puis fusionnement des résultats partiels pour obtenir une solution générale (même principe que celui du tri fusion).

2.5.2.1 Phase 1 : Recherche et contraction des circuits alternés

Soit $N = (P, T, F)$ un RdP.

La recherche de circuits alternés s'effectue en suivant les étapes suivantes :

- **Étape 1 :** élimination des transitions non propres.
Il s'agit de trouver les transitions t tel que $|\bullet t| = 1$.
La recherche se fait maintenant dans le réseau réduit : $N_1 = (P_1, T_1, F_1)$ avec :

$$T_1 = \{t \in T \mid |\bullet t| = 1\}$$

$$P_1 = \bullet T_1 \cup T_1^\bullet$$

$$F_1(p, t) = F(p, t) \text{ et } F_1(t, p) = F(t, p) \quad \forall t \in T_1 \text{ et } p \in P_1$$
- **Étape 2 :** élimination des places qui ne peuvent pas appartenir à un circuit : $\bullet p = \emptyset$ ou $p^\bullet = \emptyset$
Nous réduisons ensuite le réseau en éliminant ces places pour obtenir le réseau $N_2 = (P_2, T_2, F_2)$ réduit comme suit :

$$P_2 = P \setminus (\{p \in P \mid \bullet p = \emptyset\} \cup \{p \in P \mid p^\bullet = \emptyset\})$$

$$T_2 = P_2^\bullet \cup \bullet P_2$$

$$F_2(p, t) = F(p, t) \text{ et } F_2(t, p) = F(t, p) \quad \forall t \in T_2 \text{ et } p \in P_2$$
- **Étape 3 :** élimination des transitions qui ne peuvent pas appartenir à un circuit : $\bullet t = \emptyset$ ou $t^\bullet = \emptyset$. Après élimination de ces transitions, nous obtenons le réseau réduit $N_3 = (P_3, T_3, F_3)$
- **Étape 4 :** Recherche d'un circuit dans N_3 et contraction de ses places en une seule dans le réseau initial N .

Nous présentons dans ce qui suit la procédure de recherche de circuits alternés suivie de la procédure de contraction d'un circuit.

A. Procédure de recherche de circuits alternés

Pour le problème de la recherche de circuits alternés dans un RdP, nous avons dû en premier lieu explorer les algorithmes de parcours et de recherche de cycles dans les graphes [13, 33, 64, 86, 122]. En second lieu, nous nous sommes inspirés de tels algorithmes et nous les avons adapté dans le contexte de graphe biparti (RdP).

En se basant sur les résultats présentés dans le tableau de la figure 2.7, nous sommes parti de l'algorithme de vérification de l'existence d'un circuit élémentaire dans un graphe orienté.

	Graphes non orientés			Graphes orientés		
	cycle standard	cycle élémentaire	cycle sans corde	circuit standard	circuit élémentaire	circuit sans corde
Existence	$n + m$	$n + m$	$n + m$ trouver le cycle : $n + m^2$	$n + m$	$n + m$	
Longueur maximum	NP-complet	NP-complet		NP-complet	NP-complet	
Longueur minimum	$n(n + m)$	$n(n + m)$		$n^\omega \log(n)$ ou nm	$n^\omega \log(n)$ ou nm	
Longueur pour k fixé	$n^\omega \log(n)$ ou nm		$n^{(k-3)}n^\omega$			

FIGURE 2.7 – Comparaison entre les algorithmes de recherche de circuits dans un graphe

L'algorithme de recherche des circuits alternés et contraction que nous proposons est donc le suivant :

Recherche et contraction des circuits alternés

```

1  Tant que  $|\bullet p| > 1$  Faire
2      Etape 1
3      Etape 2
4      Etape 3
5   $S = P_3 \cup T_3$ 
6  Si  $S = \emptyset$  Alors Fin
7  Choisir au hasard un nœud  $s$  de  $S$ 
8   $a\_traiter = \{s\}$ 
9   $OK = faux$ 
10  $circuit = \emptyset$ 
11 Tant que  $a\_traiter \neq \emptyset$  Faire
12  $v = premier(a\_traiter)$ 
13  $a\_traiter = a\_traiter \setminus \{v\}$ 
14 Pour tout  $x \in v \bullet$  Faire
15 Si  $x = s$  Alors  $OK = vrai$  Sinon  $a\_traiter = \{x\} \cup a\_traiter$ 
16 Fin Pour
17  $circuit = circuit \cup \{v\}$ 
18 Fin Tant que
19 Si  $OK = vrai$  Alors  $N = Contraction(N, circuit)$ 
20 Fin Tant que

```

$Contraction(N, circuit)$ représente un appel à la procédure de contraction du circuit $circuit$ dans le réseau N .

B. Procédure de contraction d'un circuit c

La procédure de Contraction du circuit c dans un réseau N s'effectue en suivant les étapes suivantes :

- Toutes les transitions de c dont les places d'entrée et les places de sortie sont dans c sont éliminées ainsi que leurs arcs adjacents.
- Toutes les places de c sont contractées en une seule place $cont$
- Tous les arcs joignant les places de c vers une même transition des transitions restantes sont remplacés par un seul arc $(cont, t)$; et tous les arcs joignant la même transition t restante vers les places de c sont remplacés par un seul arc $(t, cont)$.

2.5.2.2 Phase 2 : Recherche de siphons minimaux par décomposition du problème

A. Présentation de l'algorithme

L'algorithme de calcul de tous les siphons minimaux d'un RdP (libres de circuits alternés) que nous développons ici est inspiré d'un algorithme qui a été initialement proposé par R.Cordone et al. dans [37] et [38]. Ils proposent des résultats intéressants pour le calcul de l'ensemble des siphons minimaux au sein d'un réseau de Petri ordinaire. Cet algorithme commence par la recherche d'un siphon générique Sg , pour ensuite trouver un

siphon minimal contenu dans Sg . Puis, le problème original est découpé en un ensemble de sous-problèmes qui sont faciles à résoudre. Chaque sous-problème requiert la recherche d'un siphon minimal dans un réseau plus petit.

Les étapes de l'algorithme sont donc les suivantes :

1. Trouver un siphon générique (par réduction du réseau initial en éliminant les places inutiles)
2. Chercher un siphon minimal dans ce siphon générique
3. Décomposer le problème en sous-problèmes : recherche dans un réseau réduit en suivant quelques contraintes spécifiques aux places
4. Appliquer la même procédure à chaque réseau réduit.

B. Détails de l'algorithme

Soit $N = (P, T, F)$ un réseau de Petri et $\tilde{P} \subset P$.

La réduction de N à un ensemble de places \tilde{P} est donnée par $\tilde{N} = red(N, \tilde{P})$ qui est le réseau $(\tilde{P}, \tilde{T}, \tilde{F})$ défini par :

- 1) $\tilde{T} = \{t \in T / (\bullet t \cup t \bullet) \cap \tilde{P} \neq \emptyset\}$
- 2) $\tilde{F}(p, t) = F(p, t)$ et $\tilde{F}(t, p) = F(t, p)$, $\forall p \in \tilde{P}$ et $\forall t \in \tilde{T}$.

Nous présentons en ce qui suit les sous-procédures permettant l'énumération des siphons minimaux d'un RdP N . Nous exposons aussi les différentes optimisations utilisées par ces sous-procédures en vue de calculer les siphons minimaux et qui sont numérotées de 1) à 5).

Soit Σ_N l'ensemble de ces siphons.

- 1) Recherche de siphons singletons :

Avant tout calcul, nous pouvons ressortir les siphons singletons (s'ils existent) qui sont évidemment minimaux. Ces siphons sont faciles à caractériser car il suffit de chercher les places p vérifiant $\bullet p = \emptyset$. Soit \hat{P} cet ensemble : $\hat{P} = \{p \in P / \bullet p = \emptyset\}$. Le problème est donc réduit à la recherche dans $\tilde{N} = red(N, \tilde{P})$ avec $\tilde{P} = P - \hat{P}$.

$\Sigma_N = \Sigma_{\tilde{N}} \cup \{p_1\} \cup \{p_2\} \cup \dots \cup \{p_n\}$ où $p_i \in \hat{P}$, $i = 1, \dots, n$

Un exemple de telle place siphon est la place source i d'un WF-net.

Maintenant, pour le calcul des siphons de $\tilde{N} = red(N, P - \hat{P})$, nous passons par les étapes proposés dans [37].

B.1. Calcul d'un siphon générique

Le calcul d'un siphon générique dans un RdP $N = (P, T, F)$ se fait par actions différentes révélant de résultats évidents ou bien de réduction du RdP en vue de la simplification du problème de recherche au maximum possible, et ce en éliminant les places qui ne peuvent pas être contenues dans un siphon.

- 2) Vérification si P est siphon :

Si toute transition possède au moins une place en entrée, alors l'ensemble de toutes les places est un siphon qui peut être retenu comme un siphon générique :

Si $\bullet t \neq \emptyset, \forall t \in T$ alors P est un siphon.

2.5. PROCÉDURE DE VÉRIFICATION STRUCTURELLE DE LA COHÉRENCE DES PROCESSUS WORKFLOW

3) 1^{ère} réduction du nombre de places :

La première réduction consiste en l'élimination des places qui sont en sortie des transitions qui n'ont pas d'entrées, i.e. éliminer $\bar{P} = \bar{T}^\bullet$ avec $\bar{T} = \{t \in T / \bullet t = \emptyset\}$.

La recherche se reporte donc à la recherche dans le réseau réduit $\tilde{N} = red(N, \tilde{P})$ avec $\tilde{P} = P - \bar{P}$.

D'après les étapes 1) et 2), l'ensemble des places qui restent après élimination de toutes les places de sortie des transitions dont l'entrée est vide est un siphon.

Si nous ne trouvons pas de siphon générique, alors $\Sigma_{\tilde{N}} = \emptyset$.

Nous décrivons ci-dessous la fonction relative au calcul d'un siphon générique dans un RdP.

Fonction TrouverUnSiphon(N)

Entrée : un RdP $N = (P, T, F)$
Sortie : un Siphon S
1 Si $\exists p \in P / \exists t \in \bullet pett \notin P^\bullet$, Alors $S = \emptyset$; Fin
2 Si $\exists p \in P / \exists t \in \bullet pet^\bullet t = \emptyset$ Alors goto 3 Sinon $S = P$; Fin
3 $N = red(N, P \setminus \{p\})$
4 goto 1

Une fois un siphon générique est trouvé, nous passons à chercher un siphon minimal dans ce siphon. Pour cela, nous présentons les réductions possibles relatives à la recherche d'un siphon minimal au sein d'un autre siphon.

4) 2^{ème} réduction du nombre de places :

Soit $S \subseteq P$ un siphon de N . S'il existe une place $\bar{p} \in S /$ soit $(\bullet t \cap S) \supset \{\bar{p}\}$ soit $(t^\bullet \cap S) = \emptyset \forall t \in \bar{p}^\bullet$, alors $S \setminus \{\bar{p}\}$ est aussi un siphon de N .

B.2. Recherche d'un siphon minimal

Le siphon obtenu après l'étape 4 n'est pas nécessairement minimal. Pour tester si un siphon S est minimal, nous utilisons la définition d'un siphon minimal.

Soit $S \subseteq P$ un siphon de N . S est un siphon minimal pour N si et seulement si pour toute place $p \in S$ le réseau réduit de N , $\tilde{N}_p = red(N, S \setminus \{p\})$ ne contient aucun siphon.

5) Décomposition du réseau et recherche dans les réseaux réduits :

Pour chaque réseau réduit N_i nous cherchons un siphon S avec la méthode de recherche d'un siphon générique ; si $S \neq \emptyset$ nous décomposons N_i en $|S|$ réseaux réduits : $\tilde{N}_p = red(N_i, S \setminus \{p\})$ où chaque \tilde{N}_p correspond à un sous-réseau de N_i ne contenant que les places du siphon S privées de la place p .

Pour chacun de ces nouveaux réseaux, nous testons s'il contient un siphon (par la même méthode de recherche d'un siphon générique) ; si oui S devient ce siphon et nous relançons la décomposition selon le nouveau S et la recherche dans les réseaux réduits ; si non nous marquons S comme un siphon minimal trouvé.

Nous résumons ci-dessous les étapes de la fonction de calcul d'un siphon minimal au sein d'un siphon quelconque.

Fonction TrouverUnSiphMin(N,S)

Entrée : un RdP $N = (P, T, F)$ et S un Siphon de N
Sortie : un siphon minimal S_m

- 1 Si $|S| = 1$ Alors $S_m = S$; Fin
- 2 Si $\exists p \in S / \forall t \in \bullet p, p \in (\bullet t \cap S)$, ou $(t \bullet \cap S) = \emptyset$ Alors goto 3 Sinon goto 4
- 3 $S = S - \{p\}$
- 4 goto 1
- 5 Si $S \subset P$ Alors $N = red(N, S)$
- 6 $P_{new} = P$
- 7 Si $P_{new} = \emptyset$ Alors $S_m = S$; Fin
- 8 $N_p = red(N, P \setminus \{p\}), p \in P_{new}$
- 9 $P_{new} = P_{new} \setminus \{p\}$
- 10 $S_p = TrouverUnSiphon(N_p)$
- 11 Si $S_p \neq \emptyset$, Alors $S_m = S_p$; goto 5 Sinon goto 7.

B.3. Recherche de tous les siphons minimaux

Quand un Siphon minimal S est trouvé, la recherche d'autres siphons minimaux peut exclure tous les siphons contenant $S = p_1, p_2, \dots, p_n$. Par ailleurs, la décomposition (proposée par Cordone et al.) consiste en la division du problème de recherche en plusieurs sous-problèmes correspondant chacun à la recherche dans le réseau réduit $red(N, P - p_i)$; $i = 1, \dots, n$.

Ci-dessous sont présentées les étapes de l'algorithme de calcul de tous les siphons minimaux présents dans un RdP.

Algorithme TrouverTousSiphMin

Entrée : un RdP $N = (P, T, F)$
Sortie : Σ_N ensemble des siphons minimaux de N

- 1 $\Sigma_N = \emptyset$
//Recherche de siphons singletons
- 2 Pour tout $p \in P$
Si $\bullet p = \emptyset$ Alors $\Sigma_N = \Sigma_N \cup \{p\}$; $N = red(N, P \setminus \{p\})$
- 3 Pour tout $p \in P$
Si $|\bullet p| = 1$ et $\bullet p \subseteq p \bullet$ Alors $\Sigma_N = \Sigma_N \cup \{p\}$; $N = red(N, P \setminus \{p\})$
//Recherche d'un siphon générique
- 4 $S = TrouverUnSiphon(N)$
//Recherche d'un siphon minimal
- 5 $S_m = TrouverUnSiphMin(N, S)$, $\Sigma_N = \Sigma_N \cup S_m$
//Recherche des autres siphons minimaux par décomposition/réduction
- 6 $P_{new} = S_m$
- 7 Tant que $P_{new} \neq \emptyset$ Faire
- 8 $N_p = red(N, P \setminus \{p\}), p \in P_{new}$
- 9 $\Sigma_{N_p} = TrouverTousSiphMin(N_p)$
- 10 $\Sigma_N = \Sigma_N \cup \Sigma_{N_p}$
- 11 $P_{new} = P_{new} \setminus \{p\}$
- 12 goto 7

2.5.2.3 Exemple d'application

Exemple 2.4

Soit le RdP de la figure 2.8 pour lequel nous envisageons de calculer l'ensemble de ses siphons minimaux.

– **Phase 1 : Recherche et Contraction des circuits alternés**

Après toutes les réductions possibles, nécessaires avant la recherche de circuits alternés, nous obtenons le réseau N_3 de la figure 2.9.

La recherche de circuits se fait uniquement sur N_3 et la contraction se fait dans le réseau initial N_1 .

Il est clair que N_3 contient deux circuits alternés (les plus grands) qui sont :

$$C_1 = \{p_7, t_7, p_9, t_9, p_8, t_8\} \text{ et}$$

$$C_2 = \{p_{11}, t_{13}, p_{12}, t_{14}, p_{10}, t_{12}\}.$$

La contraction de C_1 et C_2 dans N_1 ramène à un réseau qui contient lui-même un circuit alterné formé de la place contraction de C_1 , la place contraction de C_2 et des transitions t_{10} et t_{11} . La contraction de ce nouveau circuit ramène à un réseau ne contenant aucun circuit alterné.

Nous obtenons à la fin le réseau de la figure 2.10 dans lequel les places $p_7, p_8, p_9, p_{10}, p_{11}$ et p_{12} ont été contractées en une seule (p_4).

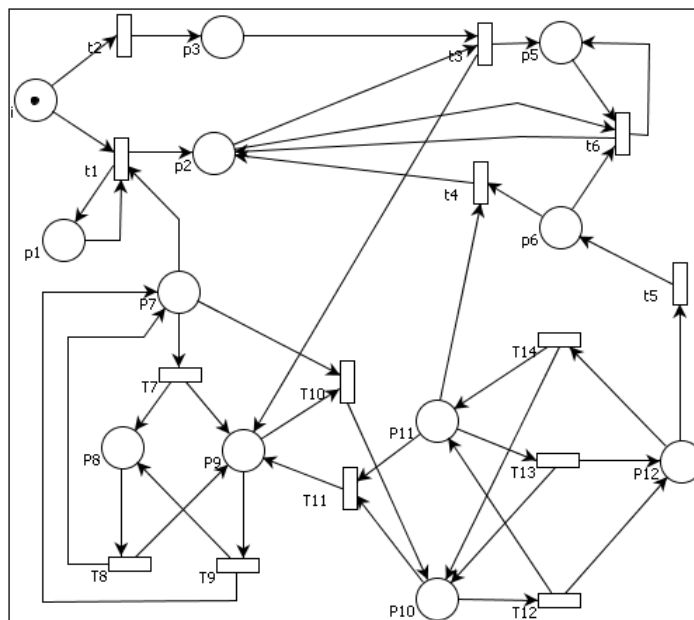


FIGURE 2.8 – Réseau de Petri N_1

2.5. PROCÉDURE DE VÉRIFICATION STRUCTURELLE DE LA COHÉRENCE
DES PROCESSUS WORKFLOW

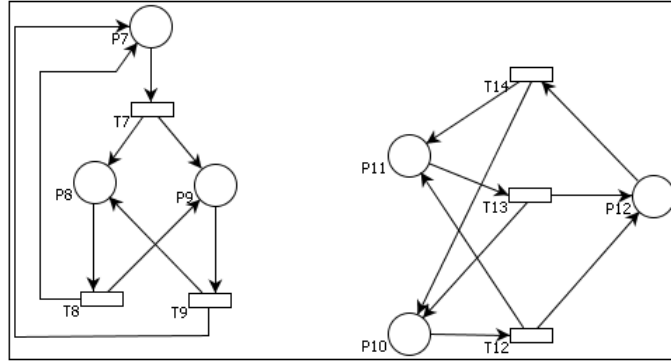


FIGURE 2.9 – Réseau de Petri N_3

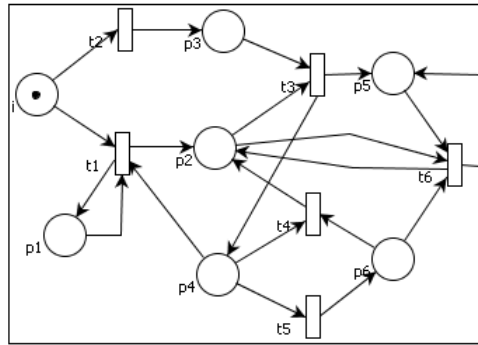


FIGURE 2.10 – Réseau de Petri dont les circuits alternés ont été contractés

– **Phase 2 : Calcul des siphons minimaux**

Soit N le RdP de la figure 2.10 pour lequel nous souhaitons calculer tous les siphons minimaux.

L'ensemble des siphons minimaux trouvés en appliquant l'algorithme présenté plus haut est

$$\Sigma_N = \{\{i\}, \{p_1\}, \{p_2, p_4\}\}$$

Ce résultat est obtenu en suivant les étapes suivantes :

Initialement, l'ensemble des siphons minimaux est $\Sigma_N = \emptyset$.

Cherchons les siphons singletons dans $N = (P, T, F)$ avec $P = \{i, p_1, p_2, p_3, p_4, p_5, p_6\}$.

1. Elimination de la place i car elle n'admet pas de transitions d'entrée. Elle forme donc un siphon singleton qui sera ajouté à Σ_N .

$$\Sigma_N = \{\{i\}\}$$

2. Elimination de p_1 qui forme un siphon singleton puisque $\bullet p_1 = p_1^\bullet$.

$$\Sigma_N = \{\{i\}, \{p_1\}\}$$

3. Une fois i éliminée nous aurons dans le nouveau réseau réduit $\bullet t_2 = \emptyset$ donc p_3 (sortie de t_2) sera aussi éliminée.

4. La recherche se fait maintenant sur $N = red(N, P \setminus \{i, p_1, p_3\})$ (figure 2.11(a)) donc

2.5. PROCÉDURE DE VÉRIFICATION STRUCTURELLE DE LA COHÉRENCE DES PROCESSUS WORKFLOW

le nouvel ensemble de places est $P = \{p_2, p_4, p_5, p_6\}$.

Cherchons un siphon dans P .

Nous remarquons que toutes les transitions sont en sortie d'au moins une place de P donc $S = \{p_2, p_4, p_5, p_6\}$ est un siphon.

5. Cherchons un siphon minimal S_m dans S

Cherchons les places inutiles :

Les conditions d'élimination des places p_2 , p_4 , p_5 et p_6 ne sont pas satisfaites donc ces places ne peuvent pas être éliminées.

Décomposition en $4(= |P|)$ sous-problèmes de recherche (figure 2.11) :

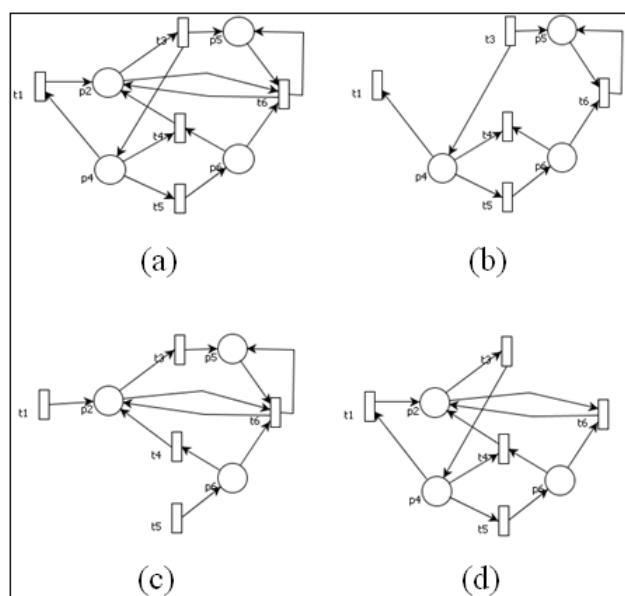


FIGURE 2.11 – Réductions pour le calcul des siphons minimaux d'un RdP

i/ Eliminons p_2 : le réseau réduit est donné par la figure 2.11(b) avec $P = \{p_4, p_5, p_6\}$.

Nous avons $t_3 \in \bullet P$ et $t_3 \notin P^\bullet$ donc p_4 et p_5 seront éliminées. De plus nous avons $\bullet p_6 \notin p_6^\bullet$, alors P ne contient pas un siphon générique.

ii/ Eliminons p_4 : le réseau réduit est donné par la figure 2.11(c) avec $P = \{p_2, p_5, p_6\}$.

Nous avons $\bullet t_3 = \emptyset$ et $\bullet t_5 = \emptyset$ donc p_2 et p_6 seront éliminées, dans le nouveau réseau réduit : $\bullet t_3 = \emptyset$ donc $\{p_5\}$ n'est pas un siphon. Alors P ne contient pas un siphon générique.

iii/ Eliminons p_5 : le réseau réduit est donné par la figure 2.11(d) avec $P = \{p_2, p_4, p_6\}$.

$S = \{p_2, p_4, p_6\}$ est un siphon générique, cherchons un siphon minimal dans S . Pour cela éliminons les places de S une à une et testons si le reste de places constitue un siphon.

iii.1. Eliminons p_2 , $\{p_4, p_6\}$ n'est pas un siphon car $\bullet t_3 = \emptyset$ et $\bullet t_5 = \emptyset$.

iii.2. Eliminons p_4 , $\{p_2, p_6\}$ n'est pas un siphon car $\bullet t_1 = \emptyset$ et $\bullet t_5 = \emptyset$.

iii.3. Eliminons p_6 , $\{p_2, p_4\}$ est un siphon

- Eliminons p_2 , $\{p_4\}$ n'est pas un siphon car $\bullet t_3 = \emptyset$

- Eliminons p_4 , $\{p_2\}$ n'est pas un siphon car $\bullet t_4 = \emptyset$

Donc $\{p_2, p_4\}$ est un siphon minimal

$$S_m = \{p_2, p_4\}$$

$$\Sigma_N = \{\{i\}, \{p_1\}, \{p_2, p_4\}\}$$

6. $N = (P, T, F)$ avec $P = \{p_2, p_4, p_5, p_6\}$ contient un siphon minimal $S_m = \{p_2, p_4\}$

Donc nous décomposons ce réseau en deux sous-réseaux N_2 et N_4 avec $P_2 = \{p_4, p_5, p_6\}$

et

$P_4 = \{p_2, p_5, p_6\}$ et nous relançons la recherche dans ces deux réseaux.

Les étapes mentionnées dans i/ et ii/ sont refaits ici et résultent en :

$$\Sigma_{N_2} = \emptyset \text{ et } \Sigma_{N_4} = \emptyset.$$

$$\text{Donc } \Sigma_N = \Sigma_N \cup \Sigma_{N_2} \cup \Sigma_{N_4} = \{\{i\}, \{p_1\}, \{p_2, p_4\}\}$$

Maintenant nous devons décontracter la place p_4 pour trouver les siphons minimaux du RdP de la figure 2.8 :

$$\text{Nous obtenons } \Sigma_N = \Sigma_N \cup \Sigma_{N_2} \cup \Sigma_{N_4} = \{\{i\}, \{p_1\}, \{p_2, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}\}\}$$

2.5.3 Complexité et tests de l'algorithme optimisé

2.5.3.1 Nouvelle Complexité

Commençons par calculer la complexité de l'algorithme de calcul des siphons minimaux dans un RdP. En notant par n le nombre des places et m celui des arcs, la recherche d'un siphon générique peut être effectué en $O(m)$.

La recherche du premier siphon minimal au sein du siphon générique nécessite au maximum n appels à la fonction de recherche d'un siphon générique et elle est donc de l'ordre de $O(nm)$.

Enfin pour trouver tous les siphons minimaux et en suivant le partitionnement proposé, nous pouvons avoir en tout 2^n sous problèmes de recherche. Ainsi la complexité totale de cet algorithme est de l'ordre de $O(nm2^n)$.

Notons par c cette complexité relative à la recherche de tous les siphons minimaux dans un RdP N et cherchons l'influence de la recherche et contraction des circuits alternés sur cette complexité.

En général, après avoir trouvé chaque circuit nous opérons une contraction du graphe du RdP : nous remplaçons toutes les places du circuit par une seule place connectée aux autres nœuds du réseau comme indiqué dans la procédure de réduction. Puis, nous continuons à rechercher l'existence de circuit alterné sur le réseau ainsi réduit tant que possible.

Cette opération de réductions/contractions successives est polynomiale. Nous pouvons la ramener à au maximum n détections d'un circuit dans un graphe, une détection se fait en $O(m)$ par des techniques classiques de parcours dans les graphes tel que le parcours en profondeur d'abord. Donc au total la complexité de cette opération est au maximum de $O(nm)$ où m est le nombre d'arcs et n le nombre de places.

Ainsi par exemple si nous trouvons un circuit alterné de 2 places, la complexité c sera

2.5. PROCÉDURE DE VÉRIFICATION STRUCTURELLE DE LA COHÉRENCE DES PROCESSUS WORKFLOW

divisée par 2. Au meilleur des cas, s'il existe un circuit alterné passant par toutes les places (non nécessairement simple et obtenu après contractions successives) alors la complexité c est divisée par 2 puissance n ainsi du premier coup on saura qu'il y a un seul siphon minimal dans le réseau courant et ce en $O(mn)$.

Donc en général, si le nombre de places du réseau a passé de n à $n1$, la complexité sera divisé par 2^{n-n1} .

2.5.3.2 Implémentation et tests

L'algorithme proposé s'avère très utile dans le cas où le réseau contient des circuits alternés. En effet, la complexité de l'algorithme de calcul des siphons minimaux diminue avec la procédure de la recherche et la contraction de circuits alternés. Ainsi, pour l'exemple 2.4, le calcul des siphons minimaux est ramené, après contraction des circuits alternés, à un calcul sur un réseau de 7 places et 6 transitions (figure 2.10) au lieu du calcul sur le réseau initial formé de 12 places et 14 transitions (figure 2.8).

Afin de mesurer l'efficacité de l'algorithme proposé, nous avons implémenté les fonctions présentées plus haut et donc tout l'algorithme de recherche et contraction des circuits alternés et de calcul des siphons minimaux dans le réseau réduit à l'aide du langage C. Les tests ont été effectués sur une machine Pentium 4 dotée d'un CPU de 3.06 GHz et d'une RAM de 504 Mo.

Les RdPs que nous avons utilisés pour les tests sont les mêmes utilisés par Cordone et al. [37] Ces tests, consistant en 2400 fichiers, représentent des RdPs choisis aléatoirement et pour lesquels le nombre de places et de transitions varient de 5 à 75. Ces tests sont groupés en 15 classes de fichiers possédant chacune une taille, i.e. la 1^{ère} classe contient les RdPs de taille 5, la 2^{ème} classe contient les RdPs de taille 10 et ainsi de suite jusqu'à la 15^{ème} classe qui contient des réseaux de taille 75.

Chaque classe contient elle-même 16 sous-classes possédant chacune une probabilité de connectivité (densité du RdP représentant le nombre d'arcs) des entrées (d_i) et des sorties (d_o) différente des autres. Chaque d_i et chaque d_o valent 0.2, 0.4, 0.6 ou bien 0.8 d'où le nombre 16 des sous-classes correspondant à toutes les combinaisons possibles entre d_i et d_o . Et enfin, dans chaque sous-classe il y a 10 tests différents.

Cordone et al. ont démontré que le taux de croissance du temps de calcul des siphons minimaux dépend fortement de la densité des arcs d'entrées; il y a une corrélation inverse entre la densité d'entrée et le nombre de siphons minimaux. Ils ont remarqué aussi que le nombre de siphons minimaux pour les tests de taille supérieure ou égale à 40 peut varier d'une façon aléatoire d'un test à un autre en fonction de d_i .

L'application de notre algorithme sur ces différents fichiers de tests a donné de bons résultats uniquement pour les réseaux où il y'avait des circuits alternés dont le nombre n'est pas malheureusement important. Ceci est dû essentiellement au fait que les RdPs de tests sont générés aléatoirement et donc pouvant référer à des réseaux quelconques pour lesquels il y'avait par exemple des transitions non connectées, des transitions à plusieurs entrées et sans sortie ou l'inverse, etc.

Le tableau 2.2 présente les valeurs trouvées par notre programme qui sont le nombre de siphons minimaux ainsi que le temps de calcul (en secondes) pour chaque densité d_i . Le

2.5. PROCÉDURE DE VÉRIFICATION STRUCTURELLE DE LA COHÉRENCE DES PROCESSUS WORKFLOW

tableau 2.3 présente les valeurs trouvées par Cordone et al.

Taille RdP	$d_i = 0.2$		$d_i = 0.4$		$d_i = 0.6$		$d_i = 0.8$	
	%	Temps CPU	%	Temps CPU	%	Temps CPU	%	Temps CPU
5	100	0.00	100	0.00	100	0.00	100	0.00
10	100	0.00	100	0.00	100	0.00	100	0.00
15	100	0.00	100	0.00	100	0.00	100	0.00
20	100	0.00	100	0.00	100	0.00	100	0.00
25	100	0.00	100	0.00	100	0.00	100	0.00
30	100	8.515	100	8.965	100	2.115	100	1.43
35	100	120.34	100	70.31	100	8.87	100	2.86
40	100	945.23	100	145.654	100	27.43	100	15.21
45	100	1985.41	100	299.56	100	40.03	100	26.15
50	-	-	13	2598.57	100	324.4	100	39.35
55	-	-	-	-	100	443.76	100	125.02
60	-	-	-	-	100	465.32	100	131.87
65					100	3456.73	100	155.94
70					100	4563.07	100	245.05
75	-	-	-	-	-	-	100	269.03

TABLE 2.2 – Pourcentages des siphons minimaux et temps de calcul trouvés par notre programme

Taille RdP	$d_i = 0.2$		$d_i = 0.4$		$d_i = 0.6$		$d_i = 0.8$	
	%	Temps CPU	%	Temps CPU	%	Temps CPU	%	Temps CPU
5	100	0.00	100	0.00	100	0.00	100	0.00
10	100	0.00	100	0.00	100	0.00	100	0.00
15	100	0.00	100	0.01	100	0.01	100	0.00
20	100	0.03	100	0.13	100	0.08	100	0.02
25	100	0.52	100	1.22	100	0.44	100	0.10
30	100	8.47	100	8.72	100	1.97	100	0.30
35	100	102.81	100	61.78	100	7.89	100	0.91
40	100	1134.07	100	292.33	100	28.57	100	2.35
45	7.69	3531.79	100	1468.27	100	79.02	100	5.20
50	-	-	15	3446.83	100	250.38	100	11.11
55	-	-	-	-	100	664.04	100	23.88
60	-	-	-	-	100	1536.79	100	43.81
65					47.5	2994.33	100	81.03
70					17.5	3519.87	100	145.98
75	-	-	-	-	-	-	100	289.08

TABLE 2.3 – Pourcentages des siphons minimaux et temps de calcul trouvés par Cordone et al.

2.5.4 Procédure de vérification de la CS-property

Pour examiner si un WF-net donné vérifie la CS-property, nous devons vérifier que chaque siphon minimal est contrôlé. Dans un souci de ne pas avoir à faire une vérification comportementale du contrôle des siphons, i.e. celle qui nécessite le calcul de tous les marquages accessibles, nous nous sommes limités au contrôle de siphons par trappe ou par invariant. Pour le contrôle par trappe, il nous a suffi de chercher une trappe dans le siphon en question. Et pour le contrôle par invariant, nous étions en mesure de calculer tous les invariants du réseau et de vérifier après la condition nécessaire.

La recherche de trappe est importante dans la vérification de la contrôlabilité d'un siphon. Par analogie avec les résultats trouvés lors du calcul des siphons, nous avons proposé une procédure de calcul d'une trappe au sein d'un ensemble de places particulières.

Pour cela, comme premier résultat, nous pouvons affirmer que s'il existe une place qui possède une transition de sortie qui ne fait entrer à aucune place du réseau, alors supprimer cette place du réseau n'affectera pas l'ensemble de ses trappes.

Lemme 2.3 *Soit $N = (P, T, F)$ un réseau de Petri. Si $\exists p \in P / \exists t \in p^\bullet$ et $t^\bullet = \emptyset$, alors l'ensemble des trappes de N est égal à celui du réseau N_p obtenu après avoir éliminé p . (p ne peut être contenu dans une trappe).*

Preuve

Soit T une trappe donc $T^\bullet \subset \bullet T$. Soit $p \in P$ tel que $\exists t \in p^\bullet$ vérifiant $t^\bullet = \emptyset$.

Supposons que $p \in T$, alors $p^\bullet \subset T^\bullet \subset \bullet T \Rightarrow t \in \bullet T$ par conséquent $\exists q \in T$ telle que $q \in t^\bullet$ ce qui est impossible puisque $t^\bullet = \emptyset$.

La fonction de recherche d'une trappe (qui n'est pas nécessairement minimal) dans un ensemble spécifique de places est décrite ci-dessous.

Fonction FindTrap(N, P')

Entrée : un RdP $N = (P, T, F)$ et un ensemble de places $P' \subseteq P$

Sortie : une trappe $Tr \subseteq P'$

- 1 Si $\exists p \in P \cap P' / \exists t \in p^\bullet$ et $t \notin \bullet P$ Alors $Tr = \emptyset$; Fin
- 2 Si $\exists p \in P \cap P' / \exists t \in p^\bullet$ et $t^\bullet = \emptyset$ Alors $N = red(N, P - \{p\})$; goto 1 Sinon $Tr = P$

2.6 Conclusion

Ayant comme objectif principal la contribution à la vérification de la propriété de cohérence des processus workflow. Nous nous sommes intéressés dans un premier volet à identifier de nouvelles sous-classes de WF-nets pour lesquelles la cohérence peut être caractérisée d'une manière structurelle et donc plus efficace. En effet, pour les sous-classes OWF-nets, RootWF-nets et CCWF-nets, nous avons montré que la cohérence est fortement liée à la propriété structurelle de siphon contrôlé (CS-Property). Ces résultats sont aussi généralisés pour couvrir les propriétés de k -cohérence (présence de k instances du workflow prêtes à s'exécuter) et de R-cohérence (présence des ressources partagées dans le workflow en question).

Dans un deuxième volet, nous avons abordé le calcul des siphons minimaux dans un

2.6. CONCLUSION

RdP nécessaires pour la vérification de la propriété de siphon contrôlé. Pour ce faire, nous avons en premier lieu étudié les algorithmes de calcul d'un ou de tous les siphons d'un réseau (qu'ils soient minimaux ou non). En second lieu, nous avons choisi l'algorithme de Cordone et al. et nous l'avons augmenté par une procédure de compactage du réseau initial en vue de simplifier la recherche. Ce compactage est fait en explorant les résultats relatifs à la relation entre circuits alternés et siphons minimaux.

2.6. CONCLUSION

Chapitre 3

Modélisation et Analyse sous contraintes temporelles

3.1 Introduction

La gestion du temps dans les processus workflow est cruciale dans la détermination et le contrôle du cycle de vie des activités. Elle rassemble l'assignation des deadlines, la vérification des inconsistances de temps ou bien le calcul de la durée de tout le processus. De plus, l'importance croissante et la dominance des systèmes informatiques en général a exigé l'introduction de nouveaux, plus riches, et plus expressifs modèles temporels, constituant une évolution des modèles temporels de base.

Dans un souci de tenir compte de manière efficiente et explicite aussi bien les caractéristiques comportementales du système que ses caractéristiques temporelles, de nombreuses extensions des RdPs ont été proposées dans la littérature. Ces extensions peuvent se regrouper en deux grandes classes distinctes : il s'agit d'une part des Réseaux Temporisés (ou Timed Petri Nets) [91], et d'autre part des Réseaux Temporels (ou Time Petri Nets) [85].

Dans ce chapitre, nous proposons un modèle de workflow basé sur les réseaux de Petri temporisés résultant en un WF-net Temporisé (ou simplement TWF-net) dans lequel nous associons à chaque transition une quantité de temps qui représente la durée de la tâche modélisée par cette transition.

Incorporer des contraintes de temps dans les processus workflow résulte en des modèles plus complexes et rend crucial le besoin de techniques et outils convenables pour leur vérification et validation. Dans ce chapitre, nous nous focalisons aussi sur la vérification de la propriété de la cohérence des TWF-nets proposés. Dans cette direction, nous proposons une caractérisation de la propriété de la cohérence du TWF-net en fonction des propriétés de vivacité et de finitude qui seront aussi définies pour ce modèle.

Vu que le modèle de TWF-net, que nous avons proposé, peut être limité dans certains cas à cause de son adoption de durées fixes d'exécution des transitions, nous avons opté pour la relaxation des contraintes temporelles fixées en vue de la proposition d'un modèle plus général de modélisation de processus workflow temporels. Dans la dernière partie de ce

chapitre, nous définissons formellement le modèle de ITWF-net (Interval Timed WF-net) et donnons sa sémantique de franchissement et d'évolution d'états. Nous discutons ensuite l'analyse des ITWF-nets et leur simulation. A la fin de cette partie, nous présentons le module de modélisation et de simulation des ITWF-nets que nous avons développé, testé et intégré à jPnet.

3.2 Introduction du temps dans les RdPs

Les RdPs représentent un formalisme puissant et reconnu pour la modélisation et l'analyse des systèmes workflow. Dans un souci de tenir compte de manière efficiente aussi bien des différentes fonctionnalités du système, que de ses caractéristiques temporelles de manière explicite, de nombreuses extensions des RdPs ont été proposées dans la littérature. Ces extensions peuvent se regrouper en deux grandes classes distinctes : les modèles temporisés [1], [2], [3] où les attributs temporels sont des valeurs ponctuelles, et les modèles temporels [5] où ces attributs prennent la forme d'intervalles de temps pouvant traduire des plages de faisabilité aussi bien que des incertitudes sur le processus. En outre, certains de ces modèles ont été appliqués aux processus workflow afin de leur permettre de supporter le facteur temps.

Les modèles de RdP intégrant la dimension de temps permettent de traiter les problèmes liés à l'analyse et l'évaluation des performances aux travers des méthodes analytiques.

Dans ce cadre, cette section est consacrée à l'étude des différentes sémantiques et des principaux types de RdPs utilisés pour la modélisation des systèmes à contraintes temporelles.

3.2.1 Trois sémantiques Différentes

Etant donné qu'une transition représente un événement, il est naturel que les délais temporels devraient être associés avec les transitions, et ce en assignant à chaque transition la durée que prend l'événement pour se produire. C'est l'approche adoptée par la plupart des modélisateurs. Cependant, quelques modélisateurs assignent le temps aux places ou aux arcs au lieu des transitions. Bien qu'initialement cela puisse paraître contradictoire, il peut être une façon utile pour représenter le temps. L'assignation de durées aux places et aux arcs change simplement la façon avec laquelle le RdP est interprété. Dans les deux cas, la sémantique du RdP est définie d'une façon semblable à celle quand le temps est assigné aux transitions.

Le temps dans les RdPs est représenté selon trois sémantiques différentes : durées de franchissement, durées d'exploitation et durées d'activation [34].

– Durées de franchissement

Dans la sémantique des RdPs non temporisés, les transitions peuvent être tirées n'importe quand après être activées, enlevant des jetons de leurs places d'entrée et créant des jetons dans leurs places de sortie. Quand des durées de franchissement sont incluses, la sémantique du réseau est changée. Chaque transition a un délai associé avec elle. Quand une transition est activée, elle enlève les jetons des entrées immédiatement mais elle ne crée les jetons dans les places de sortie que lorsque la

durée du tir s'est écoulée.

Non seulement les règles d'exécution qui ont été changées par rapport aux RdPs non temporisés, mais aussi de nouveaux états ont été créés. Cela crée le besoin de redéfinir l'espace des états produit par les RdPs. Chaque état du RdP est défini non seulement par le marquage, mais aussi par la condition de chacune des transitions. L'inconvénient principal de l'utilisation des durées de franchissement est d'agrandir l'espace des états. Les marquages du RdP non temporisé peuvent maintenant avoir de multiples états associés avec eux dans le RdP augmenté avec le temps correspondant.

– Durées d'exploitation

Le principe de l'adoption des durées d'exploitation consiste en le classement des jetons en deux types : disponible et indisponible. Les jetons disponibles peuvent être utilisés pour permettre le franchissement d'une transition, alors que les jetons indisponibles ne le peuvent pas. A chaque transition est assignée une durée, et quand une transition est tirée, l'action d'enlever et de créer des jetons est faite instantanément. Cependant, les jetons créés ne sont pas disponibles pour permettre de nouvelles transitions jusqu'à ce qu'ils aient séjourné dans leurs places de sortie pour le temps spécifié par la transition qui les a créés. L'usage de durées d'exploitation devient plus populaire en particulier dans les réseaux de haut niveau où le temps pour lequel un jeton devient disponible peut être enregistré comme partie du jeton.

Les durées d'exploitation et les durées de franchissement sont en fait la même façon de représentation du temps. La seule différence est que dans un cas les jetons sont 'tenus' dans les transitions, alors que dans l'autre cas ils sont 'tenus' dans les places. De ce point, il n'y aura aucune distinction entre les durées d'exploitation et les durées de franchissement.

Les durées d'exploitation seront utilisées quand on n'a pas de transitions qui restent activées sur des périodes de temps. Donc, la sémantique des durées d'exploitation est plus proche à celle des RdPs non temporisés. Aussi, leurs graphes d'accessibilité sont en général plus petits. Chaque marquage est plus compliqué que celui d'un RdP non temporisé, puisqu'il implique des jetons non disponibles pour une certaine période.

Les durées d'exploitation ont été assignées, au début, aux places [97]. Les durées d'exploitation assignées aux places ont la même sémantique que celles associées avec les transitions. Quand le temps est associé avec une place, il fait référence au temps pendant lequel les jetons créés dans cette place par toute transition ne sont pas disponibles pour permettre le tir des transitions. La différence entre ces deux approches est la source de laquelle les jetons obtiennent leur durée d'indisponibilité. Quand le temps est assigné aux transitions, chaque jeton créé par une transition donnée a le même temps d'exploitation. Quand le temps est associé avec les places, chaque jeton qui est créé dans une place donnée possède le temps d'exploitation associé avec lui. Ces deux techniques sont parfois combinées et permettent aux durées d'exploitation d'être assignées aux places et aux transitions, comme dans [16].

La raison pour laquelle le temps est souvent associé avec les transitions est que les transitions représentent des événements dans un modèle et c'est plus naturel de considérer des événements à contraintes temporelles plutôt que de considérer le temps comme associé aux conditions (places). Cependant, il y a des situations où ce n'est pas le cas. Sifakis [96] et Baccelli et al. [16] expriment que les faits d'associer le temps

avec les places ou avec les transitions ont un pouvoir de modélisation équivalent. C'est donc au modélisateur de décider quelle technique est meilleure pour le système qu'elle modélise.

– Durées d'activation

La dernière façon d'incorporation du temps dans les RdPs utilise des durées d'activation. Cette technique était introduite par Merlin pour modéliser des protocoles de communications reconfigurables [85]. En utilisant des durées d'activation, le tir de la transition est fait immédiatement ; les jetons sont enlevés et créés en même instant, et les délais du temps sont représentés en forçant des transitions à être activées pour une période spécifiée de temps avant qu'elles soient tirées.

3.2.2 Sous-classes des RdPs augmentés avec le temps

Dans la littérature, il existe trois façons différentes d'assignation d'un temps à un événement : déterministe, stochastique et à intervalle. Chacune de celles-ci a été appliquée aux différents types de représentations du temps mentionnés dans la section précédente. La méthode choisie dépend du système à modéliser et à analyser.

Dans le cas déterministe, les délais sont sélectionnés dans des sous-ensembles de nombres réels non négatifs ou d'entiers naturels. Pour l'assignation des valeurs temporelles d'une façon stochastique, des lois de distribution ont été utilisées. Dans les modèles à intervalles sont associées une valeur minimale et une valeur maximale à chaque événement. La valeur effective relative au franchissement ou à l'activation de chaque transition est considéré prendre une valeur de cet intervalle.

Nous nous intéressons par la suite aux modèles déterministes et à intervalles. Pour ces modèles, deux grandes familles de RdPs incorporant le temps ont immergé :

- Les RdPs temporisés : les contraintes temporelles sont représentées par des durées (des nombres rationnels positifs ou nuls) qui sont associées soit aux transitions [91] soit aux places [97].
- Les RdPs temporels : pour ces sous classes de RdPs, sont associés aux places [73], aux transitions [85] ou aux arcs [46] des intervalles $[a_i, b_i]$ spécifiant une durée minimale et une durée maximale.

Les RdPs temporisés sont en général associés avec les durées de franchissement et les durées d'exploitation, et les RdPs temporels sont associés avec les durées d'activation. Nous résumons dans les sous sections qui suivent les principales sous classes des RdPs à contraintes temporelles ainsi que leurs sémantiques.

3.2.2.1 Les RdPs temporisés

Les RdPs temporisés ont été proposé en vu d'augmenter les RdPs par les informations relatives aux durées d'exécution. Le temps peut être associé soit aux transitions (RdP t-temporisé) [91] soit aux places (RdP p-temporisé) [97].

Associée à une transition, une durée temporelle mesure la durée de franchissement de cette transition. Par contre, la sémantique d'une valeur temporelle associée à une place

indique le temps de séjour d'un jeton dans cette place. Nous rappelons dans ce qui suit les modèles temporisés proposés selon leur ordre d'apparition.

Ramchandani [91] fut le premier à introduire les RdPs temporisés. Plus particulièrement, il a associé une durée à chaque franchissement de transition, conduisant au modèle t-temporisé. Puis, Sifakis [97] décida d'associer aux différentes places du réseau une durée traduisant le temps de séjour des jetons contenus dans ces places. Il a été démontré que son modèle, le modèle p-temporisé et celui de Ramchandani sont équivalents.

Nous n'allons pas entrer dans les détails profonds des différents modèles énoncés. Toutefois, nous pouvons remarquer qu'un jeton peut séjourner indéfiniment dans une place. En effet, si la décision de tirer une transition n'est pas imposée alors il n'y aura aucune obligation de franchissement de cette dernière. Ainsi, ce type de modèle ne peut garantir qu'une durée de séjour minimale dans une place, et de ce fait, ne permet pas de modéliser efficacement les systèmes à contraintes de temps de séjour, où des durées de séjour minimale et maximale sont imposées.

Plus récemment, dans [69, 70, 71] a été proposé un modèle associant les temporisations aussi bien aux places qu'aux transitions du réseau : le modèle p-t-temporisé. Cependant, même avec ce modèle, si un fonctionnement au plus tôt n'est pas imposé, une fois la temporisation d'un jeton dans une place écoulée, la décision de franchir une transition de sortie de cette dernière peut être reportée à l'infini. Il n'y a donc pas d'obligation de franchissement. C'est en partie pour les raisons précédentes que les auteurs préfèrent utiliser par la suite un modèle p-temporel t-temporisé [69, 70].

Dans [67], après avoir présenté les comportements concurrents dans les RdPs, L. Jenner a ajouté une notion quantitative de temps au comportement opérationnel et par ailleurs a étendu les RdPs en introduisant les durées dans les transitions. La différence principale par rapport au modèle t-temporisé réside au niveau sémantique. En effet, la durée associée à chaque transition ne modélise pas une durée à respecter avant le tir de cette transition mais plutôt une durée de tir. Ainsi, lors du franchissement d'une transition t active, on détruit directement les jetons requis et on ne crée les jetons dans les places de sortie qu'après écoulement de la durée associée à t .

3.2.2.2 Les réseaux de Petri temporels

Les RdPs temporels sont des RdPs contenant des attributs prenant la forme d'intervalles de temps pouvant traduire des plages de faisabilité aussi bien que des incertitudes sur le processus. On distingue deux formes de RdPs temporels : les RdPs t-temporels [85] associant un intervalle de temps à chaque transition et les RdPs p-temporels [72] associant des intervalles de temps aux places.

Le modèle de Merlin [85], communément appelé RdP t-temporel, a été conçu pour l'étude des problèmes de recouvrement pour les protocoles de communication. Dans ce modèle, à chaque transition est attachée une contrainte temporelle de type intervalle, et non plus ponctuelle comme dans le modèle t-temporisé. L'intervalle associé à la transition t est relatif au moment où la transition devient validée, soit $[a, b]$ cet intervalle. Supposons que t soit validée à l'instant τ , alors elle peut être franchie dans l'intervalle matérialisé par les quantités $(a + \tau)$ et $(b + \tau)$, sauf si elle est désensibilisée à cause du franchissement d'une

autre transition avec laquelle elle était en conflit. Dans le modèle de Merlin est soulignée l'ambiguïté dans le cas de multi-sensibilisation (une transition t est *multi-sensibilisée* par un marquage M s'il existe un entier $k > 1$ tel que $M(p) \geq k \forall p \in \bullet t$).

Le modèle p-temporel, proposé par Khansa [72], est caractérisé principalement par une synchronisation forcée. L'intervalle qu'on peut associer à chaque place p_i est un intervalle statique $[a_i, b_i]$ désignant les durées de séjour minimale et maximale du jeton dans la place p_i . Dans ce modèle, il ne suffit pas de considérer que les jetons contenus dans les places en amont de la transition dont le tir est considéré mais aussi tout le réseau pour s'assurer qu'il n'y a pas de jetons morts.

Un autre modèle, présentant des contraintes temporelles de type intervalle a été proposé dans la littérature, il permet aussi de prendre en compte des durées de traitement à valeurs incertaines ; c'est le modèle de Van der Aalst [10]. La principale caractéristique de ce modèle consiste à associer les attributs temporels aux différents jetons du réseau. C'est de plus un modèle coloré, où chaque jeton se voit attaché à une empreinte temporelle représentant l'instant où celui-ci devient disponible pour le franchissement d'une transition. Ce modèle est qualifié de ITCPN (Interval Timed Colored Petri Net). Dans le ITCPN, un jeton possède alors trois attributs : une position (la place dans laquelle il se trouve), une valeur (ou couleur), et une empreinte temporelle. Ainsi, l'instant de sensibilisation d'une transition correspond à la valeur maximale des empreintes des jetons qui seront consommés par le tir considéré. Les transitions sont alors franchies dès que cela est possible et parmi plusieurs transitions candidates au tir, celle possédant l'instant de sensibilisation le plus petit sera tirée la première. Le franchissement d'une transition est une opération ne consommant pas de temps (immédiate) et les jetons créés ont une nouvelle empreinte d'une quantité égale au moins à l'instant du tir.

Puis, Diaz et Senac définissent des RdPs à flux temporels TSPN (Time Stream Petri Net) [95]. Leur idée est de placer les intervalles sur les arcs et d'assigner à chaque transition une règle de déclenchement. Les intervalles, dits de validité temporelle, sont spécifiés selon leurs durées minimale et maximale. Les règles de déclenchement se basent sur trois stratégies de base : au plus tôt des arcs, au plus tard des arcs ou selon les arcs maîtres. La combinaison de ces stratégies produit neuf règles qui précisent les priorités des différents arcs dans le déclenchement de la transition. Les TSPNs permettent de modéliser des scénarios indéterministes qui prennent en compte des délais sur les intervalles de manière complexe. Donc si $[\alpha_i, \beta_i]$ est un intervalle de validité temporelle à l'arc reliant une place p et une transition t . Cet intervalle signifie que la transition t peut et doit être tirée à une date comprise dans l'intervalle $[\tau + \alpha_i, \tau + \beta_i]$ où τ représente l'instant d'arrivée du jeton dans la place p . La principale particularité du modèle TSPN réside dans la représentation des différents mécanismes de synchronisation. En effet, les TSPNs offrent la possibilité au concepteur, de choisir entre différents modes de synchronisation entre flux.

3.3 Réseaux de workflow temporisés (TWF-nets)

En général, les informations temporelles sont incorporées aux RdPs en ajoutant un intervalle de temps et/ou une durée à chaque place et/ou transition du réseau, plus d'autres propositions d'association de temps aux arcs ou aux marques, voir [10, 73, 90, 91, 96] pour

plus de détails.

En se basant sur ces résultats, différentes manières d'incorporation d'informations temporelles dans les WF-nets ont été proposées dans la littérature. La plupart de ces propositions sont basées sur les modèles de RdP modélisant le temps discutés plus haut. Par exemple, dans [78], un intervalle délimitant les durées minimale et maximale est associé à chaque place et à chaque transition du WF-net. Cette information représente la résidence d'une marque une fois associée à une place et la durée d'exécution d'une tâche une fois associée à une transition. Dans [79], un intervalle temporel (temps de début et temps de fin) est assigné à chaque tâche de workflow représentant sa durée d'exécution. Dans [116], les auteurs associent à chaque transition t un intervalle de temps représentant le temps minimal et le temps maximal pour lesquels la transition peut être franchie plus le temps du début de franchissement jusqu'à la fin de la transition t .

Un bon modèle de workflow peut non seulement modéliser les structures de contrôle logiques dans les workflows, mais aussi décrire et analyser les informations de temps et les comportements temporels des ressources et activités.

Vue qu'une transition dans les RdPs représente un événement, il est naturel que les délais doivent être associés avec les transitions, et donc assignant à chaque transition la durée que prend la production de l'événement associé.

De même, dans les processus workflow, le plus simple de considérer les informations temporelles est d'associer avec chaque tâche la durée qu'elle prend pour s'exécuter. Cette information peut être incorporée sous forme d'une étiquette sur la transition en question représentant une valeur naturelle. Ceci a pour but de simplifier la gestion de l'avancement du temps ainsi que le calcul des états accessibles du système.

La seule extension des RdPs associant des durées avec les tâches représentées par les transitions est celle de L. Jenner [67], c'est le modèle de TPN (Timed Petri Net ou RdP temporisé).

Les RdPs Temporisés sont obtenus des RdPs en ajoutant des durées aux transitions. Formellement, un TPN est un tuple $N = (P, T, F, \delta)$ où $N = (P, T, F)$ est un RdP et $\delta : T \rightarrow \mathbb{N}$ est la fonction durée de transition.

Dans les TPNs, chaque transition t consomme un délai positif pour franchir. Quand une transition t est active, un franchissement peut être initialisé par l'enlèvement d'un jeton de chaque place d'entrée de t . Après cet enlèvement, la transition t consomme une durée de tir bien déterminée. Quand cette durée s'est écoulée, le franchissement se termine par ajout d'un jeton à chaque place de sortie de t . Si une transition est activée pendant qu'elle est en exécution, un nouveau marquage indépendant peut être initié.

Vue cette information quantitative de temps associée à chaque tâche, un comportement particulier du TPN lors de l'avancement du temps doit être fourni. Dans [67] et [114], trois disciplines différentes de temps ont été considérées :

1. Temporisation Libérale (liberal timing) : permet un passage arbitraire du temps dans n'importe quel état et requiert uniquement qu'une activité s'exécute au moins pour sa durée, c'est à dire l'activité peut être retardée et prolongée arbitrairement. Ceci correspond au comportement que l'auteur appelle de type L.

3.3. RÉSEAUX DE WORKFLOW TEMPORISÉS (TWF-NETS)

2. Temporalisation Stricte (strict timing) : exige le début immédiat des actions possibles et interdit d'excéder une durée fixe. Ceci correspond au comportement de type S.
3. Temporalisation Mixte (mixed timing) : déclare que seulement le début mais pas la fin d'une certaine action peut être retardé arbitrairement. Ceci correspond au comportement de type E.

Donc notre proposition consiste en l'application du modèle de L. Jenner dans le contexte de processus workflow. Nous étendons le modèle de WF-nets simple présentés par van der Aalst [8] par l'association à chaque transition d'une quantité de temps représentant la durée de la tâche qu'elle modélise. La discipline de temporalisation adoptée dans le modèle proposé annonce que chaque transition activée doit commencer son exécution immédiatement, sinon elle sera désactivée, et une fois commencée, cette transition ne peut pas être retardée, i.e. sa durée doit être respectée.

Dans le reste de cette section, nous étudions l'impact de cette extension sur les WF-nets en termes de structure et de comportement.

3.3.1 Définition d'un TWF-net

Définition 3.1 *Un réseau de workflow temporisé (TWF-net) N est un tuple (P, T, F, δ) où (P, T, F) est un WF-net et $\delta : T \rightarrow \mathbb{N}$ associe une durée à chaque transition.*

Exemple 3.1

Le processus de livraison d'une commande d'achat des produits par une entreprise peut être modélisé par le TWF-net de la figure 3.1.

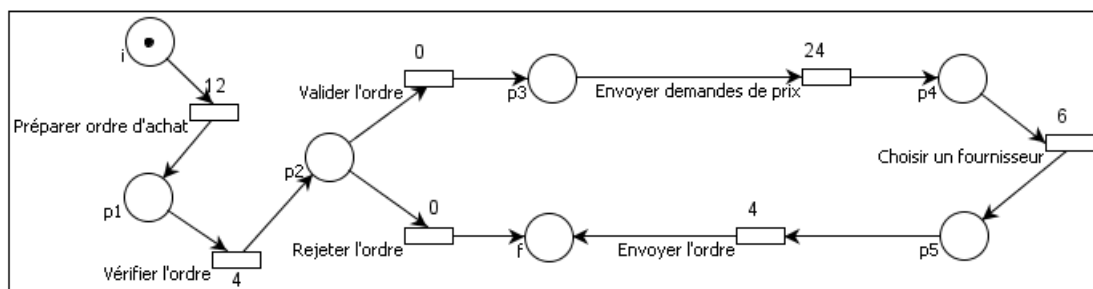


FIGURE 3.1 – Livraison d'un ordre d'achat

D'abord, un ordre peut être préparé en indiquant les produits à acheter, les quantités, etc. Puis, une vérification de l'ordre est nécessaire et celui-ci peut être validé ou rejeté. S'il est validé, un processus de choix du fournisseur est lancé après envoi des demandes de prix à plusieurs fournisseurs. Enfin, la commande d'achat est délivrée au fournisseur approprié. Chaque tâche dans ce processus a besoin d'une quantité de temps pour son exécution. Par exemple, la préparation de l'ordre prend 12 unités de temps pour s'accomplir, la validation ou le rejet de l'ordre est instantané i.e. ces tâches ne prennent pas de temps pour être exécutées. Notons que les tâches, une fois activées, doivent être commencées et elles ne peuvent pas être retardées.

3.3.2 Sémantique des états et leur évolution dans les TWF-nets

Dans cette sous-section, nous définissons un état d'un TWF-net, puis les conditions ainsi que les règles de franchissement des transitions et enfin l'évolution des états.

Soit $T^\pm = \{t^+, t^- / t \in T\}$ les parties des transitions représentant les débuts de franchissement des transitions t^+ (i.e. la consommation des jetons) et les fins des transitions t^- (i.e. la production des jetons).

Etat : Un état d'un TWF-net représente l'état du processus modélisé par le TWF-net après l'occurrence d'un événement, par exemple, le commencement d'une activité.

Un état d'un TWF-net est un triplet (M, C, ρ) où M est un marquage, $C \subseteq T$ est l'ensemble des transitions courantes et $\rho : C \rightarrow \mathbb{N}$ donne le temps restant pour les transitions courantes.

Etant donné que dans un WF-net, seule la place initiale i contient initialement un jeton, l'état initial d'un TWF-net N est $(M_i, \emptyset, \emptyset)$.

L'état évolue selon une règle de transition composée de deux clauses de franchissabilité (conditions de franchissement) et de franchissement.

Franchissabilité : Une transition t est franchissable si chacune de ces places d'entrées contient au moins un jeton. Une transition franchissable est en progression si et seulement si chaque ressource qu'elle requiert n'est pas requise par une autre transition ayant une priorité supérieure. Les transitions qui sont franchissables mais pas en progression sont dites suspendues. Eu égard à la discipline stricte de temporisation que nous avons adoptée, une transition en progression doit être franchie immédiatement.

Franchissement : Quand une transition t est franchie, elle passe par trois phases :

- t commence par franchir t^+ , consommant un jeton de chaque place de $\bullet t$.
- Le temps de franchissement de t doit être égal à sa durée $\delta(t)$. Entre le temps d'activation τ et $\tau + \delta(t)$, les jetons peuvent être considérés comme "gelés" dans t . Les jetons et la transition ne peuvent être impliqués dans un autre franchissement entre τ et $\tau + \delta(t)$. Après $\delta(t)$ unités de temps du temps d'activation de t , $\delta(t)$ va valoir 0 et puis t peut passer à sa phase de fin.
- t finit par franchir t^- , déposant un jeton dans chaque place de t^\bullet .

Remarque : Les sémantiques de t^+ et t^- nous ramènent à confirmer qu'un WF-net T-Temporisé (une durée est associée à chaque transition) est équivalent à un WF-net P-Temporisé (une durée est associée à chaque place) selon cette transformation : Une transition t avec une durée δ est décomposée en deux transitions t^+ et t^- représentant le début et la fin de t . Puis, une place p est ajoutée connectant t^+ et t^- et ayant un taux égale à δ . Ce taux représente la durée de séjour que le jeton, quand arrivé à p (après le franchissement de t^+), va passer avant le franchissement de t^- . Ceci est illustré par la figure 3.2 montrant la transformation d'un WF-net T-Temporisé (a) à un WF-net P-Temporisé (b).

Evolution d'états : Sémantiquement, un état (M, C, ρ) est remplacé par un nouvel état

(M', C', ρ') . Nous écrivons $(M, C, \rho)[\epsilon](M', C', \rho')$ si l'un des cas suivants est rencontré :

1. $\epsilon = t^+$ et $(M, C)[t^+](M', C')$ et $\rho'(t) = \delta(t)$ et $\rho'|_C = \rho$.
2. $\epsilon = t^-$ et $(M, C)[t^-](M', C')$ et $\rho(t) = 0$ et $\rho' = \rho|_C$.

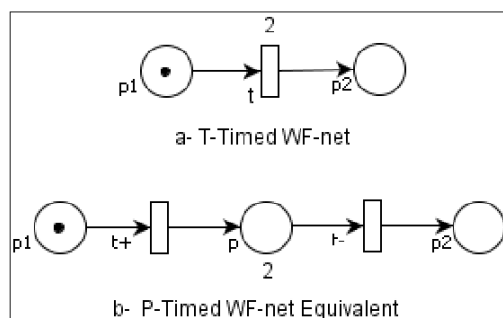


FIGURE 3.2 – Les WF-nets T-Temporisé et P-Temporisé sont équivalents

3. $\epsilon = (r)$ pour $r \in \mathbb{N} \setminus \{0\}$ tel que $(M', C') = (M, C)$ et $\rho' = \rho - r = \min(\rho - r, 0)$, et pour tout $t \in C : r \leq \rho(t)$.

Par la règle 2, une transition t commencée possède un temps résiduel $\delta(t)$ (sa durée), et par les règles 2 et 3, elle doit franchir au moins pour sa durée. La dernière permet l'écoulement du temps, où le marquage et les transitions courantes ne changent pas, mais la période résiduelle des transitions courantes est mise à jour selon l'écoulement du temps. En plus, le temps ne peut passer que si la durée de toute transition courante ne sera pas excédée, ou bien si aucune autre transition ne peut commencer dans l'état actuel, i.e. une transition activée commence aussitôt que possible ou bien elle sera désactivée.

Dans les règles 1 et 2 nous trouvons la règle de franchissement : par 1, franchissement de t^+ et par 2, franchissement de t^- ; il n'y a pas de règle de franchissement dans le cas $\epsilon = (r)$. Dans les règles 1, 2 et 3, nous trouvons la règle de calcul de l'état suivant.

Après franchissement de t^+ :

$$\begin{aligned} M'(p) &= M(p) - W(p, t) \quad \forall p, \\ C' &= C \cup \{t\}, \\ \rho'(t) &= \rho(t) \text{ et } \rho'|_C = \rho. \end{aligned}$$

Après franchissement de t^- :

$$\begin{aligned} M'(p) &= M(p) + W(t, p) \quad \forall p, \\ C' &= C \setminus \{t\}, \\ \text{et } \rho' &= \rho|_C. \end{aligned}$$

Quand t^+ est franchie, des jetons sont éliminés de $\bullet t$ et après $\delta(t)$, t^- va franchir et des jetons vont être ajoutés à t^\bullet .

3.3.3 Espace d'états

Par analogie avec le graphe des marquages accessibles défini dans le cadre d'un RdP ordinaire, nous définissons un graphe comportant tous les états accessibles d'un TWF-net et nous l'appelons espace d'états.

Le comportement d'un TWF-net est caractérisé par son espace d'états défini comme suit :

Définition 3.2 L'espace d'états d'un TWF-net est la structure : $SS = (S, \xrightarrow{a}, S_0)$, où $S = (M, C, \rho)$ est l'ensemble des nœuds représentant les états accessibles de l'état initial $S_0 = (M_i, \emptyset, \emptyset)$, et $a \in T^\pm \cup \mathbb{N}$ représente les annotations des arcs reliant entre les états et définissant l'évolution d'un état à l'autre.

Exemple 3.2

La figure 3.3 (b) représente l'espace d'états accessibles du TWF-net de la figure 3.3 (a).

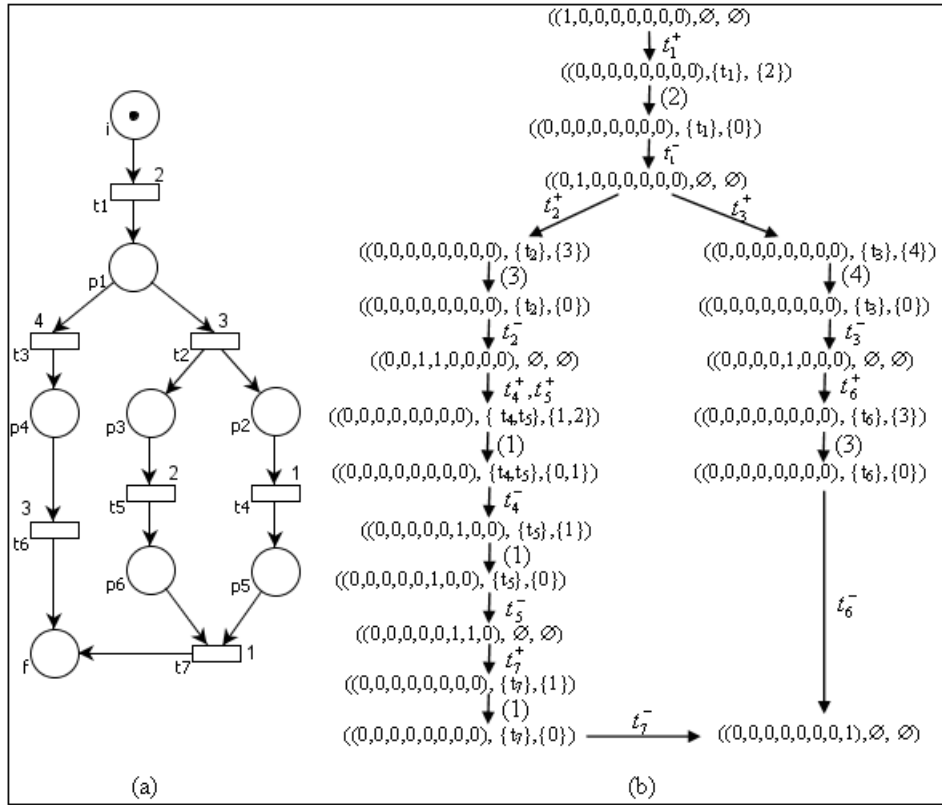


FIGURE 3.3 – Exemple de TWF-net et son espace d'états accessibles

3.4 Analyse des TWF-nets

Dans cette section, nous allons définir les propriétés d'exactitude du modèle proposé qui peuvent être vérifiées en analysant l'espace des états accessibles d'un TWF-net.

Propriété de Cohérence :

Par équivalence à la propriété de cohérence définie pour les WF-nets simples [4, 7, 8], nous définissons la propriété de cohérence d'un TWF-net comme suit.

Définition 3.3 Un TWF-net $N = (P, T, F, \delta)$ est cohérent si et seulement si :

3.4. ANALYSE DES TWF-NETS

1. Pour chaque état (M, C, ρ) accessible de l'état initial $(M_i, \emptyset, \emptyset)$, il existe une séquence de franchissement ramenant de l'état (M, C, ρ) à l'état final $(M_f, \emptyset, \emptyset)$.
Formellement : $\forall (M, C, \rho) \in [(M_i, \emptyset, \emptyset)], (M_f, \emptyset, \emptyset) \in [(M, C, \rho)]$.
2. L'état final est l'unique état accessible de l'état initial avec au moins 1 jeton dans la place f . Formellement : $\forall (M, C, \rho) \in [(M_i, \emptyset, \emptyset)] : M(f) \geq 1 \Rightarrow M = M_f$;
3. Il n'y a pas de transitions mortes dans N .
Formellement : $\forall t \in T, \exists (M, C, \rho) \in [(M_i, \emptyset, \emptyset)] : (M, C, \rho)[t^+]$.

Deux propriétés principales liées à la cohérence d'un WF-net sont la propriété de finitude et la propriété de vivacité. Définissons ces propriétés pour la classe de TWF-nets.

La propriété de finitude :

Définition 3.4 Un TWF-net N est fini ou borné pour un état (M, C, ρ) s'il existe un nombre entier k tel que : pour chaque état accessible (M', C', ρ') de (M, C, ρ) , $M'(p) \leq k$, $\forall p \in P$.

La propriété de vivacité : La propriété de vivacité dénote que chaque transition peut devenir activée par franchissement d'un nombre arbitraire de transitions. Contrairement à ceci, un état mort dénote un état du réseau dans lequel aucune transition n'est active. En ce qui concerne le modèle présenté, le seul état mort acceptable, i.e. un état dans lequel aucune autre action n'est possible, serait l'état final où le processus est complètement fini.

Définition 3.5 Un TWF-net est vivant pour un état (M, C, ρ) si pour chaque état accessible (M', C', ρ') de (M, C, ρ) et chaque transition $t \in T$, il existe un état (M'', C'', ρ'') accessible de (M', C', ρ') dans lequel t^+ peut être franchie.

3.4.1 Vérification de la cohérence d'un TWF-net

Nous définissons d'abord une structure importante qui est la fermeture du TWF-net.

Définition 3.6 Soit N un TWF-net, nous notons par $N^* = (P^*, T^*, F^*, \delta^*)$ le réseau obtenu à partir de N par ajout d'une transition t^* et défini comme suit :

- $P^* = P$,
- $T^* = T \cup \{t^*\}$,
- $F^* = F \cup \{(f, t^*), (t^*, i)\}$,
- $\delta^*(t) = \delta(t)$ pour $t \in T$, et $\delta^*(t^*) = 0$.

N^* est dit la fermeture du TWF-net N et t^* transition de fermeture.

Théorème 3.1 Un TWF-net est cohérent si et seulement si son réseau fermeture est vivant et borné pour l'état $(M_i, \emptyset, \emptyset)$.

Preuve : Soit N un TWF-net et N^* sa fermeture.

(\Leftarrow) Supposons que N^* est vivant et borné. N^* est vivant, i.e. pour chaque état accessible (M, C, ρ) il existe une séquence de franchissements qui mène à un état dans lequel

t^{*+} est active. Puisque f est la place d'entrée de t^* , nous trouvons que pour chaque état (M, C, ρ) accessible de $(M_i, \emptyset, \emptyset)$ il est possible d'atteindre un état avec au moins un jeton dans la place f . Soit (M', C', ρ') cet état, t^{*+} est activée dans cet état, Clairement $M' \geq M_f$, supposons que $M' > M_f$ alors $\exists M''$ tel que $M' = M'' + M_f$. Si t^{*+} et puis t^{*-} franchissent, alors l'état $M'' + i$ est atteint. Ceci contredit le fait que N^* est borné, alors M' doit être égal à M_f . Aussi, pour l'état (M', C', ρ') , $C' = \emptyset$ et alors $\rho' = \emptyset$, car sinon, nous allons obtenir un marquage plus grand que M' . Par conséquent, les conditions (1) et (2) de la définition 3.3 sont vérifiées et la terminaison propre est garantie. La condition (3) suit directement le fait que N^* est vivant. Ainsi, N est un TWF-net cohérent.

(\Rightarrow) Supposons maintenant que N est cohérent et prouvons que N^* est vivant et borné pour $(M_i, \emptyset, \emptyset)$.

Finitude : Puisque N est cohérent, si nous prouvons que N est borné pour $(M_i, \emptyset, \emptyset)$ alors N^* est borné pour $(M_i, \emptyset, \emptyset)$ aussi, car le franchissement de t^* mène à l'état initial $(M_i, \emptyset, \emptyset)$.

Supposons que N est borné pour $(M_i, \emptyset, \emptyset)$, alors il existe deux états (M, C, ρ) et (M', C', ρ') accessible à partir de l'état initial tel que $M' > M$ et $C' = C$. Cependant, puisque N est cohérent, il existe une séquence de franchissement σ tel que $(M, C, \rho)[\sigma](M_f, \emptyset, \emptyset)$. Par conséquent, il existe un état (M'', C'', ρ'') tel que $(M', C', \rho')[\sigma](M'', C'', \rho'')$ et $M'' > M_f$. Ceci contredit l'hypothèse de la cohérence de N . Alors N est borné pour $(M_i, \emptyset, \emptyset)$ et donc N^* est borné pour $(M_i, \emptyset, \emptyset)$.

Vivacité : N est cohérent alors N^* est borné avec respect au $(M_i, \emptyset, \emptyset)$. Dans N^* , pour tout état (M', C', ρ') accessible de $(M_i, \emptyset, \emptyset)$, il est possible de retourner à l'état $(M_i, \emptyset, \emptyset)$. Dans le réseau original N , il est possible de franchir une transition arbitraire t (condition (3) de la définition 3.3). Ceci est aussi le cas dans le réseau modifié N^* . Par conséquent N^* est vivant car pour chaque état (M', C', ρ') accessible à partir de $(M_i, \emptyset, \emptyset)$, il est possible d'atteindre un état qui active une transition arbitraire t .

3.4.2 Exemple de modélisation et d'analyse par TWF-net

Nous décrivons dans la figure 3.4 une procédure workflow pour une "demande de crédit" en indiquant les acteurs participant à l'achèvement de cette procédure. Nous déterminons pour chaque tâche le temps nécessaire pour son exécution.

Il s'agit d'un commercial qui va saisir la demande de crédit. Il va ensuite envoyer cette demande à l'analyste financier. L'analyste financier va recevoir la demande. Il va ensuite vérifier s'il existe un contrat. Il peut notifier au commercial par un refus (tout de suite) ou bien demander au service des contrats l'enregistrement ou mise à jour du contrat. Le service des contrats va recevoir la demande. Ce service va créer un nouveau contrat ou mettre à jour le contrat. Il va ensuite notifier l'analyste financier par le contrat résultat. L'analyste financier pourra ainsi vérifier le contrat puis envoyer son accord au commercial. Le commercial va recevoir le refus ou l'accord pour le crédit.

Notons que les activités sont attribuées selon des rôles (le service des contrats n'a pas les mêmes activités que l'analyste financier). L'ensemble de ces activités ordonnées permet de remplir l'objectif global de la procédure qui est de satisfaire ou rejeter la demande de

crédit.

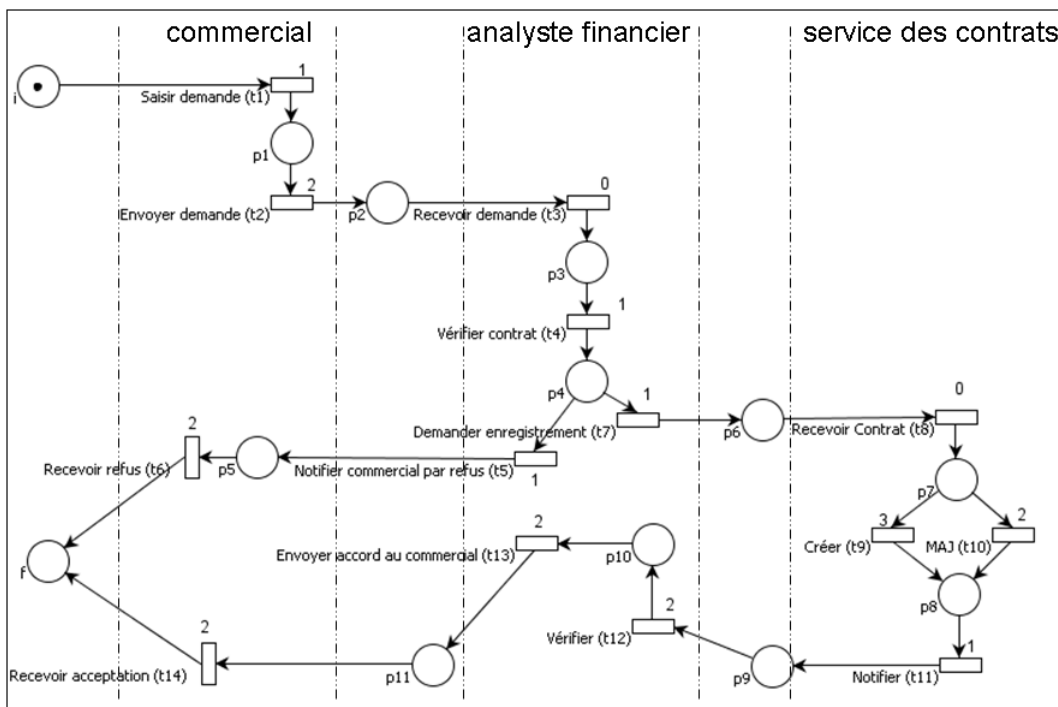


FIGURE 3.4 – Processus workflow temporisé pour une demande de crédit

Pour vérifier la cohérence de ce processus, nous calculons l'espace de ses états accessible (figure 3.5).

D'après la figure 3.5, il n'y a pas de transitions mortes, donc le réseau est vivant. En plus, nous pouvons aisément montrer que le TWF-net est borné. Nous concluons alors que le processus workflow correspondant à la demande de crédit est cohérent.

3.4.3 TWF-nets modélisant k instances

Les réseaux workflow, initialement introduits dans [8] modélisent l'exécution d'un seul cas ; i.e. le marquage initial du WF-net consiste en un seul jeton dans la place source. Dans cette section, nous considérons la modélisation de plusieurs instances du même processus dans les TWF-nets. Ceci est assuré par la présence d'un nombre arbitraire de jetons dans la place initiale.

Notre but est d'étudier l'impact de ce changement sur la structure des TWF-nets ainsi que les critères de cohérence de ces réseaux.

Soit k le nombre d'instances du processus modélisé.

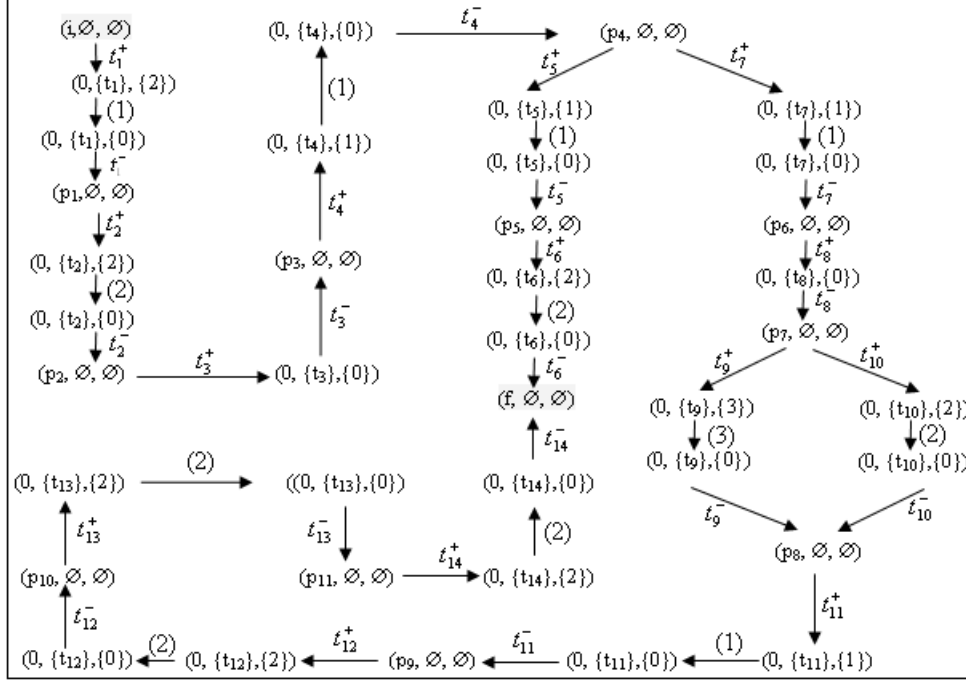


FIGURE 3.5 – Espace des états accessibles du TWF-net de la demande de crédit

3.4.4 Etats des TWF-nets modélisant k instances

En modélisant plusieurs instances de processus workflow, la première chose qui sera automatiquement influencée est l'état du système. On distingue entre : 1) Etat de l'instance où nous nous focalisons uniquement sur les transitions courantes reliées à une instance spécifique du processus ; ceci correspond à la définition d'état de la section 3.3.2 ; 2) Etat du processus où nous pouvons reconnaître les transitions courantes en faisant abstraction aux instances auxquelles elles sont reliées. Nous définissons ici l'état de tout le système : second type.

Un état est représenté par un tuple (M, C, n, ρ) où M est un marquage, $C \subseteq T$ est l'ensemble des transitions courantes, $n : C \rightarrow \mathbb{N}$ est le nombre d'instances courantes de telles transitions courantes et ρ est le temps restant pour les transitions courantes $\rho : C \rightarrow \mathbb{N}$. L'état initial d'un TWF-net N est $(M_{ki}, \emptyset, \emptyset, \emptyset)$, étant donné que dans un TWF-net, uniquement la place initiale i contient initialement k jetons.

Un état $S = (M, C, n, \rho)$ évolue suite à l'occurrence d'un événement résultant en un nouvel état $S' = (M', C', n', \rho')$. Soit ϵ cet événement. Nous écrivons $S[\epsilon]S'$ et nous calculons S' par l'une des règles suivantes :

1. si $\epsilon = t^+$ alors $(M, C, n)[t^+](M', C', n')$ et $\rho'(t) = \delta(t)$ et $\rho'|_C = \rho$ et $n'(t) = n(t) + 1$. (ajouter t à l'ensemble des transitions courantes)
2. Si $\epsilon = t^-$ alors $(M, C, n)[t^-](M', C', n')$ et $\rho(t) = 0$ et $\rho' = \rho|_C$ et $n'(t) = n(t) - 1$. (éliminer t de l'ensemble des transitions courantes)
3. Si $\epsilon = (r)$ pour $r \in \mathbb{N} \setminus \{0\}$ alors $M' = M$ et $C' = C$ (car les jetons consommés des

3.4. ANALYSE DES TWF-NETS

pré-places de la transition courante t ne seront produits dans les post-places que si la durée de t expire) et $\rho' = \rho - r = \min(\rho - r, 0)$, et pour tout $t \in C : r \leq \rho(t)$. (mettre à jour le temps résiduel du temps d'exécution de l'activité)

Examinons l'ensemble des transitions courantes après franchissement de t^+ et t^- . Soit C un tel ensemble.

$$C \subseteq T ; \text{ donc } C = \{t_1, t_2, t_3, \dots\} = \{t_i, 1 \leq i \leq |C|\}.$$

Pour chaque transition courante t_i , nous avons un nombre d'instances : $n(t_i)$. Soit $t_{11}, t_{12}, \dots, t_{1n(t_i)}$ ces instances. Alors nous pouvons définir un ensemble d'instances courantes par :

$$CI = \{t_{ij}, 1 \leq i \leq |C|, 1 \leq j \leq n(t_i)\}.$$

En se focalisant sur les détails des instances, le domaine de la fonction ρ devient CI ($\rho : CI \rightarrow \mathbb{N}$ au lieu de $\rho : C \rightarrow \mathbb{N}$)

Soit t_c une transition activée. t_c commence par franchir t_c^+ et finit par franchir t_c^- .

- Etat après franchissement de t_c^+ :
 - Si $t_c \in C$ alors :
 - $M'(p) = M(p) - W(p, t_c) \forall p$,
 - $C' = C$ et $CI' = CI \cup \{t_{c(n(t_c)+1)}\}$,
 - $\rho'(t_{c(n(t_c)+1)}) = \delta(t_c)$ et $\forall t_{ij} \in CI, \rho'(t_{ij}) = \rho(t_{ij})$;
 - et finalement $n'(t_c) = n(t_c) + 1$.
 - Si $t_c \notin C$ alors :
 - $M'(p) = M(p) - W(p, t_c) \forall p$,
 - $C' = C \cup \{t_c\}$ et $CI' = CI \cup \{t_{c1}\}$,
 - $\rho'(t_{c1}) = \delta(t_c)$ et $\forall t_{ij} \in CI, \rho'(t_{ij}) = \rho(t_{ij})$;
 - et finalement $n'(t_c) = 1$.
- Etat après franchissement de t_c^- :
 - $M'(p) = M(p) + W(t_c, p) \forall p$,
 - $C' = C \setminus \{t_c\}$ et $CI' = CI \setminus \{t_{cn(t_c)}\}$,
 - $\rho'(t_{cn(t_c)}) = 0$ et $\rho'(t_{ij}) = \rho(t_{ij})$ pour tout $i \neq c$ et $j \neq n(t_c)$,
 - et finalement $n'(t_c) = n(t_c) - 1$.

3.4.5 k-cohérence des TWF-nets

En ce qui suit, nous nous focalisons sur la propriété de k-cohérence qui annonce la propre terminaison de l'exécution des k instances et l'absence de tâches ou de conditions ne contribuant pas à l'avancement des cas (ou instances). La propriété de k-cohérence d'un TWF-net est définie comme suit.

Définition 3.7 *Un TWF-net $N = (P, T, F, \delta)$ est k-cohérent si et seulement si :*

1. *Pour tout état (M, C, n, ρ) accessible de l'état initial $(ki, \emptyset, \emptyset, \emptyset)$, il existe une séquence de franchissement menant de l'état (M, C, n, ρ) à l'état final $(kf, \emptyset, \emptyset, \emptyset)$.
Formellement : $\forall (M, C, n, \rho) \in [(ki, \emptyset, \emptyset, \emptyset)], (kf, \emptyset, \emptyset, \emptyset) \in [(M, C, n, \rho)]$.*
2. *L'état final est le seul état accessible de l'état initial avec au moins k jetons dans la place f. Formellement : $\forall (M, C, n, \rho) \in [(ki, \emptyset, \emptyset, \emptyset)] : M(f) \geq k \Rightarrow M = kf$;*

3.4. ANALYSE DES TWF-NETS

3. *Pas de transitions mortes dans N .*

Formellement : $\forall t \in T, \exists (M, C, n, \rho) \in [(ki, \emptyset, \emptyset, \emptyset)] : (M, C, n, \rho)[t^+]$.

Les propriétés de finitude et de vivacité pour les TWF-nets modélisant k instances sont définies par les définitions 3.8 et 3.9 respectivement.

Définition 3.8 *Un TWF-net N est borné pour l'état (M, C, n, ρ) s'il existe un entier naturel k tel que : pour tout état accessible (M', C', n', ρ') de (M, C, n, ρ) , $M'(p) \leq k$, $\forall p \in P$ et $n(t) \leq k$, $\forall t \in T$.*

N est place-borné s'il n'existe pas de limite sur la composante marquage M de l'état (M, C, n, ρ) .

N est transition-borné s'il n'existe pas de limite sur la composante n .

Définition 3.9 *Un TWF-net est vivant pour l'état (M, C, n, ρ) si pour tout état accessible (M', C', n', ρ') de (M, C, n, ρ) et toute transition $t \in T$, il existe un état (M'', C'', n'', ρ'') accessible de (M', C', n', ρ') qui active t^+ .*

De même pour la définition du réseau fermeture d'un TWF-net N , la seule différence est que pour la transition t^* ajoutée : $W(f, t^*) = W(t^*, i) = k$ et $\delta^*(t^*) = 0$

Théorème 3.2 *Un TWF-net est k -cohérent si et seulement si son réseau fermeture est vivant et borné pour $(ki, \emptyset, \emptyset, \emptyset)$.*

Preuve : Soit N un TWF-net et N^ sa fermeture.*

(\Leftarrow) Supposons que N^* est vivant et borné. N^* est vivant, i.e. pour chaque état accessible (M, C, n, ρ) il existe une séquence de franchissements qui mène à un état dans lequel t^{*+} est active. Puisque f est la place d'entrée de t^* , nous trouvons que pour chaque état (M, C, n, ρ) accessible de $(M_{ki}, \emptyset, \emptyset, \emptyset)$ il est possible d'atteindre un état avec au moins k jetons dans la place f . Soit (M', C', n', ρ') cet état, t^{*+} est activée dans cet état, Clairement $M' \geq kf$, supposons que $M' > kf$ alors $\exists M''$ tel que $M' = M'' + kf$. Si t^{*+} et puis t^{*-} franchissent, alors l'état $M'' + ki$ est atteint. Ceci contredit le fait que N^* est borné, alors M' doit être égal à M_{kf} . Aussi, pour l'état (M', C', n', ρ') , $C' = \emptyset$ et alors $n' = \rho' = \emptyset$, car sinon, nous allons obtenir un marquage plus grand que M' . Par conséquent, les conditions (1) et (2) de la définition 3.7 sont vérifiées. La condition (3) suit directement le fait que N^* est vivant. Ainsi, N est un TWF-net cohérent.

(\Rightarrow) Supposons maintenant que N est cohérent et prouvons que N^* est vivant et borné pour $(ki, \emptyset, \emptyset, \emptyset)$.

Le caractère Borné : Puisque N est cohérent, si on prouve que N est borné pour $(ki, \emptyset, \emptyset, \emptyset)$ alors N^* est borné pour $(ki, \emptyset, \emptyset, \emptyset)$ aussi, car le franchissement de t^* mène à l'état initial $(ki, \emptyset, \emptyset, \emptyset)$. Supposons que N est borné pour $(ki, \emptyset, \emptyset, \emptyset)$, alors il existe deux états (M, C, n, ρ) et (M', C', n', ρ') accessibles à partir de l'état initial tel que $M' > M$ et $C' = C$ (alors $n' = n$). Cependant, puisque N est cohérent, il existe une séquence de franchissement σ tel que $(M, C, n, \rho)[\sigma](kf, \emptyset, \emptyset, \emptyset)$. Par conséquent, il existe un état (M'', C'', n'', ρ'') telle

que

$(M', C', n', \rho')[\sigma](M'', C'', n'', \rho'')$ et $M'' > kf$. Ceci contredit l'hypothèse de la cohérence de N . Alors N est borné pour $(ki, \emptyset, \emptyset, \emptyset)$ et donc N^* est borné pour $(ki, \emptyset, \emptyset, \emptyset)$.

La Vivacité : N est cohérent alors N^* est borné pour $(ki, \emptyset, \emptyset, \emptyset)$. Dans N^* , pour tout état (M', C', n', ρ') accessible de $(ki, \emptyset, \emptyset, \emptyset)$, il est possible de retourner à l'état $(ki, \emptyset, \emptyset, \emptyset)$. Dans le réseau original N , il est possible de franchir une transition arbitraire t (condition (3) de la définition 3.7). Ceci est aussi possible dans le cas du réseau modifié N^* . Par conséquent N^* est vivant car pour chaque état (M', C', n', ρ') accessible à partir de $(ki, \emptyset, \emptyset, \emptyset)$, il est possible d'atteindre un état qui active une transition arbitraire t .

3.5 Critiques des TWF-nets

Nous avons présenté dans les sections précédentes un modèle de workflow temporisé basé sur l'ajout de contraintes temporelles strictes. Ce modèle possède l'avantage du calcul aisé des différents états du processus ainsi que la gestion simple de l'écoulement du temps. En plus, il permet de modéliser les processus dont les contraintes de durées des tâches prédéfinies doivent être respectées. Dans cette section, nous présentons dans une première partie deux exemples d'application de ce modèle. Dans une deuxième partie, nous évoquons les limites des TWF-nets à travers des contre-exemples.

3.5.1 Applications des TWF-nets

Dans cette section, nous montrons l'intérêt des TWF-nets à travers des exemples de processus workflow contraints par des durées fixes.

– **Application 1 :**

Considérons un exemple simple de suivi du déroulement d'un cours (figure 3.6). L'unité de temps est en nombre de séances ou bien de jours si en un jour donné, une seule séance peut être assurée.

Après déclenchement du cours, un enseignant doit donner 3 séances de cours avant de lancer les Travaux Pratiques et les Travaux Dirigés. Après avoir assuré 2 séances de TD, un test peut être fait pendant une séance. Ensuite, 2 séances de cours doivent être assurées après le test et en parallèle de 4 séances de TP, un examen doit être organisé (il va durer une séance aussi). Le test et l'examen doivent subir chacun une correction. Les corrections de ces deux épreuves aboutissent à la fin du processus.

– **Application 2 :**

Comme deuxième exemple de processus à contraintes de durées fixes, nous proposons l'organisation d'un concours de recrutement. Les tâches correspondantes sont :

1. Diffusion de l'appel d'offre
2. Réception des dossiers
3. Sélection sur dossier des candidats
4. Examen des candidats choisis (écrit)
5. Correction
6. Diffusion des résultats

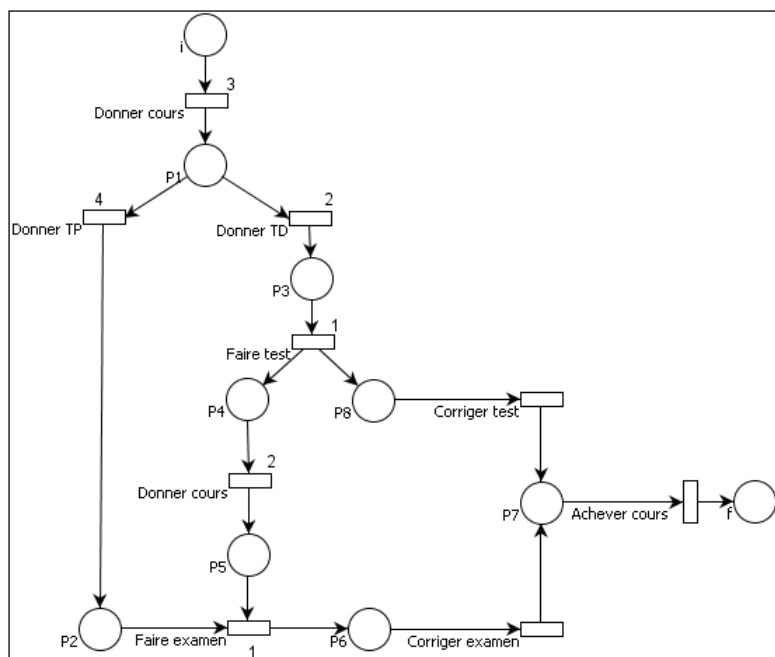


FIGURE 3.6 – TWF-net d'un processus de déroulement d'un cours

Nous pouvons imaginer que ces tâches consomment chacune une durée fixe. Bien que chaque action puisse consommer moins que le temps prévu, nous pouvons imaginer que le directeur doit recevoir l'accomplissement de tâches dans les durées prédéterminées.

3.5.2 Limites des TWF-nets

Malgré qu'en général les entreprises veulent fixer des durées strictes pour la réalisation de chaque tâche de leurs processus workflow, nous nous sommes rendu compte qu'il est plus expressif de déterminer une durée minimale et une durée maximale pour chaque tâche. De cette manière, un acteur peut accélérer son travail quand la durée minimale est passée ; il peut même être averti par le système. En plus, il peut y avoir un changement de la méthode d'exécution d'une tâche, par exemple, pour le processus de demande de crédit (figure 3.4), la durée de la tâche "Envoyer demande" peut différer selon la méthode d'envoi qu'elle soit directe par courrier postal ou bien à travers une société).

Il est donc nécessaire de relaxer la façon d'introduire les contraintes temporelles dans les processus workflow pour se rapprocher plus du monde réel et pour couvrir une large gamme de processus.

Notons aussi que les TWF-nets ne considèrent pas la notion de disponibilité des ressources. L'intérêt est de pouvoir avoir une ressource disponible mais dont on ne peut pas utiliser immédiatement. Par exemple, une imprimante libérée par une autre tâche mais qui doit rester disponible pour un certain temps en raison d'alimentation par papier.

Nous proposons dans la suite de pallier ces limites en étendant les processus workflow

par ajout des délais pour les conditions avant franchissement.

3.6 Vers un modèle temporel plus général de processus workflow

Dans cette section, nous présentons un nouveau modèle, plus général que les TWF-nets, modélisant les processus workflow à contraintes temporelles. Nous définissons d'abord formellement notre modèle que nous avons baptisé ITWF-net et donnons ensuite sa sémantique en termes d'états, de conditions de franchissement et de conditions d'évolution. Enfin, nous abordons quelques problèmes liés à la vérification de ce modèle.

3.6.1 Définition des Interval Timed WorkFlow-nets (ITWF-nets)

Nous proposons dans cette section les WF-nets à intervalles de temps (ITWF-nets) qui adoptent des contraintes temporelles modélisées par les intervalles de temps statiques associés aux jetons. Cette façon d'incorporation des contraintes temporelles a été adopté de [9]. Dans le modèle ITWF-net, nous attachons avec chaque jeton une horloge locale et un intervalle de disponibilité. L'horloge locale mesure le temps écoulé entre la production et la consommation du jeton. Pour chaque jeton, nous retenons aussi le temps de sa production appelé "timestamp". Le temps d'activation d'une transition est le maximum des timestamps des jetons à être consommés (i.e. servant au franchissement de cette transition).

Le franchissement d'une transition active aura lieu aussitôt que possible, donc la transition qui a un temps d'activation plus petit va être franchie en premier. Le tir, qui est une action atomique, va ainsi être effectué en produisant des jetons avec un timestamp égale au moins au temps de tir. On appelle la différence entre le temps de tir et le timestamp du jeton produit le délai de franchissement. Ce délai de tir est spécifié par un intervalle représentant une borne inférieure et une borne supérieure.

Avant de donner la définition formelle des ITWF-nets, rappelons quelques définitions liées à la notion de Multi-Ensemble.

En mathématiques, un multi-ensemble (ou sac) [30] est une généralisation d'un ensemble. Comme un ensemble, un multi-ensemble est une collection d'éléments sur le même sous-ensemble d'un certain univers. Cependant, contrairement à un ensemble, un multi-ensemble autorise des occurrences multiples du même élément.

Formellement, un multi-ensemble b , sur un ensemble A , est une fonction de A dans \mathbb{N} , i.e. $b : A \rightarrow \mathbb{N}$. Si $a \in A$ alors $b(a)$ est le nombre d'occurrences de a dans le multi-ensemble b . A_{MS} est l'ensemble de tous les multi-ensembles sur A . Le multi-ensemble vide est noté par \emptyset_A ou bien \emptyset . On représente souvent un multi-ensemble $b \in A_{MS}$ par la somme $\sum_{a \in A} b(a)a$.

Considérons par exemple l'ensemble $A = \{a, b, c, \dots\}$, les sous-ensembles $3a$, $a+b+c+d$, $1a+2b+3c+4d$ et \emptyset_A sont des membres de A_{MS} . Les opérateurs \leq , $=$, $+$, $-$, \in peuvent être appliqués comme les opérateurs usuels.

3.6. VERS UN MODÈLE TEMPOREL PLUS GÉNÉRAL DE PROCESSUS WORKFLOW

Notation 3.1

- TS définit le temps, $TS = \{x \in \mathbb{R} | x \geq 0\}$, i.e. l'ensemble de tous les réels non-négatifs.
- $INT = \{[y, z] \in TS \times TS / y \leq z\}$, représente l'ensemble de tous les intervalles fermés. Si $x \in TS$ et $[y, z] \in INT$, alors $x \in [y, z]$ si et seulement si $y \leq x \leq z$.

Dans ce qui suit, nous définissons un jeton par un tuple $(p, c, [a, b])$ avec $p \in P$ la place qui le contient, c son horloge local (qui vaut initialement 0) et $[a, b]$ son intervalle de validité.

Notons par Tok l'ensemble de tous les jetons possibles.

Définition 3.10 Un ITWF-net est défini par un quadruplet (P, T, F, TF) vérifiant les conditions suivantes :

- (P, T, F) est un WF-net
- TF est la fonction transition sur T .
 $TF(t) : Dom(TF(t)) \rightarrow (Tok \times INT)_{MS}$ où $Dom(TF(t)) \subseteq (Tok)_{MS}$

La fonction transition spécifie chaque transition dans le modèle de ITWF-net. Pour une transition t , $TF(t)$ spécifie la relation entre le multi-ensemble des jetons consommés et le multi-ensemble des jetons produits. Le domaine de $F(t)$ décrit la condition pour laquelle la transition est franchissable. Notons que les jetons produits possèdent un délai spécifié par un intervalle. Dans ce modèle, nous exigeons que le multi-ensemble des jetons consommés et celui des jetons produits soient finis.

Exemple 3.3

Considérons le processus workflow de relecture des papiers dans une conférence scientifique. Le ITWF-net de ce processus est donné dans la figure 3.7. Ce ITWF-net décrit le cycle de vie d'un seul papier dès sa soumission par un de ses auteurs jusqu'à son acceptation ou son refus par les organisateurs de la conférence (conference chairs). Les paramètres temporels sont associés (syntaxiquement) avec les arcs sortant des transitions puisqu'ils déterminent les intervalles de validité des jetons produits.

Le processus de relecture des papiers de la figure 3.7 représente une instance du processus prête à l'exécution. Une instance réfère à un traitement d'un papier. Le lancement de la conférence est noté par une diffusion d'un appel à soumission (ou Call For Papers noté CFP). Cette action résulte en un CFP disponible pour au plus 60 jours avant l'ouverture du site de la soumission.

Notons que si nous avons décidé de lancer l'exécution de l'instance du processus, la première action (ici la Diffusion du CFP) doit être exécutée directement puisque le jeton de la place initiale i ne possède pas d'intervalle de disponibilité.

Ensuite, les organisateurs peuvent ouvrir le site pour les éventuelles soumissions. Dès que le site de la soumission soit ouvert avec une durée maximale de 30 jours, un auteur peut soumettre un papier dans cette période (on limite notre exemple ici au cas où un seul auteur peut soumettre un papier unique par soumission).

3.6. VERS UN MODÈLE TEMPOREL PLUS GÉNÉRAL DE PROCESSUS WORKFLOW

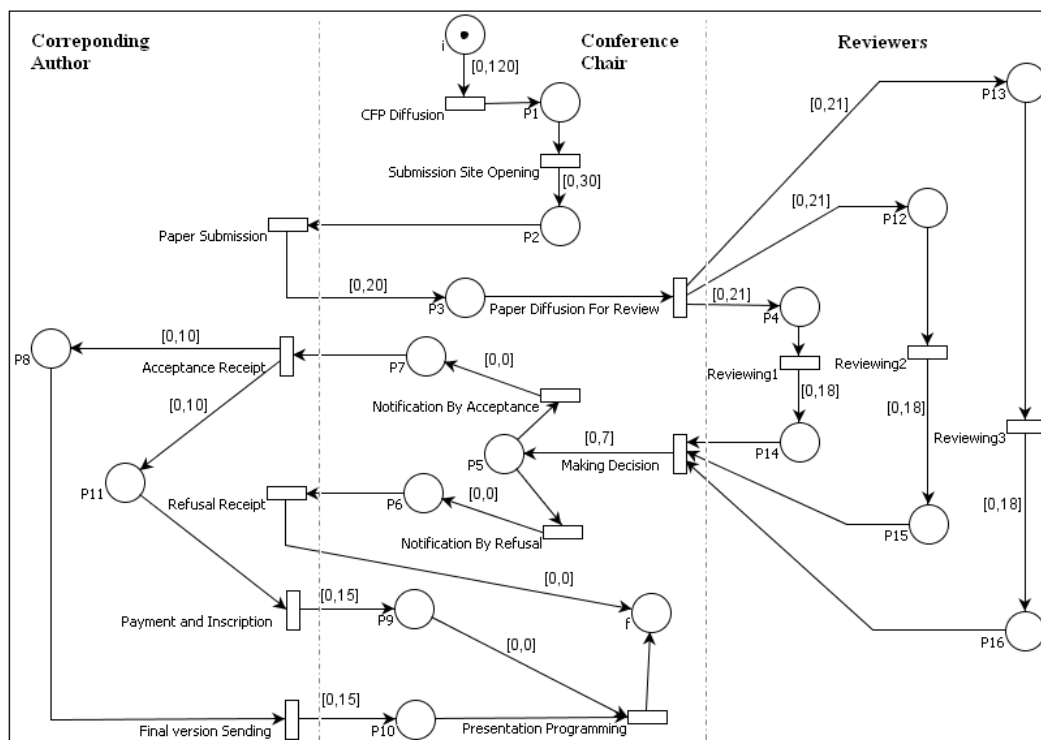


FIGURE 3.7 – Un ITWF-net du processus de relecture d’un papier dans une conférence

Le papier soumis peut être directement passé à l’étape de relecture (examen par le comité de lecture), mais il peut rester disponible dans le système de conférence pour au plus 20 jours avant de l’examiner.

On suppose maintenant qu’un organisateur de la conférence décide (après avoir fermé le site de soumission et après tout un processus de choix des examinateurs des papiers soumis) de diffuser le papier à trois examinateurs, en leur précisant la durée minimale et la durée maximale avant sa réception de leur relectures (dans notre cas, on précise que chaque relecteur possède le papier pour au plus 21 jours). Pour cette période, chaque relecteur doit envoyer une réponse (contenant ses commentaires sur le contenu du papier). Cette réponse peut rester dans le système de conférence pour 18 jours.

Après la réception des rapports des trois relecteurs, l’organisateur de la conférence peut diffuser la décision à l’auteur concerné. Cette décision peut être un refus qui mène à la fin du processus, ou bien une acceptation du papier. Cette acceptation exige de l’auteur à confirmer sa participation (au plus dans 10 jours) par envoi de la version finale de son papier en plus de son inscription et le paiement des frais de sa participation. 15 jours au maximum après l’inscription de l’auteur et l’envoi de la version finale de son papier, l’organisateur de la conférence va programmer officiellement le papier dans la conférence.

3.6.2 Comportement dynamique d'un ITWF-net

Le quadruplet (P, T, F, TF) spécifie la structure statique d'un ITWF-net. Dans cette partie, nous définissons le comportement d'un ITWF-net, i.e. la sémantique du modèle ITWF-net.

3.6.2.1 Les états d'un ITWF-net

Définition 3.11 – *Un état est défini par un multi-ensemble de jetons où chacun possède une valeur de son horloge local.*

- On note par S son espace des états accessibles, i.e. l'ensemble de tous les états possibles : $S = (Tok \times TS)_{MS}$.
- Le marquage d'un ITWF-net dans l'état $s \in S$ est la distribution non temporisée de jetons : $M(s) \in Tok_{MS}$ et $M(s) = \sum_{(p,c) \in Tok \times TS} s(p,c)(p)$

Un état est donc un multi-ensemble de tuples (p, c) ($p \in P$ et $c \in TS$). Le marquage est obtenu à partir d'un état par élimination des paramètres temporels.

Par exemple l'état (initial) du processus de la figure 3.7 est $(i, 0)$ car initialement il n'y a qu'un seul jeton dans la place i avec une valeur d'horloge qui vaut 0 puisqu'il représente une instance prête à l'exécution.

3.6.2.2 Evolution d'états

Etant donné que les transitions sont les composants actifs d'un ITWF-net, nous devons définir les conditions de tir, le tir d'une transition et donc l'évolution des états.

Une transition t est franchissable (ou active) dès que toutes les ressources qu'elle exige soient disponibles, i.e. il y a des jetons dans $\bullet t$ et pour chacun de ces jetons, sa valeur de l'horloge est incluse dans son intervalle de validité.

Etant donnés une transition franchissable et l'ensemble des jetons requis pour son exécution, nous donnons en ce qui suit une définition formelle d'un événement.

Définition 3.12 *Un événement est un triplet (t, b_{in}, b_{out}) , qui représente les franchissements possibles de la transition t par enlèvement des jetons spécifiés par le multi-ensemble b_{in} et ajout des jetons spécifiés par le multi-ensemble b_{out} .*

Un événement $e = (t, b_{in}, b_{out})$ représente le tir d'une transition franchissable t en consommant les jetons spécifiés par b_{in} et en produisant les jetons spécifiés par b_{out} .

Si $(p, c, [a, b]) \in b_{in}$, alors e consomme un jeton de p . Si $(p', c', [a', b']) \in b_{out}$, alors e produit un jeton dans p' avec une valeur initiale d'horloge $c' = 0$ et un délai $[a', b']$.

Notons que les intervalles de temps sont spécifiés par $TF(t)(b_{in})$.

L'ensemble de tous les événements est noté par $E = T \times (Tok \times TS)_{MS} \times (Tok \times TS)_{MS}$.

Soit s un état et $(p, c, [a, b])$ un jeton de s . Il est clair que ce jeton est disponible si $c \leq b$ et il va rester disponible pour une période de temps spécifiée par un intervalle qui

3.6. VERS UN MODÈLE TEMPOREL PLUS GÉNÉRAL DE PROCESSUS WORKFLOW

est : $[max(0, a - c), b - c]$.

Soit $e = (t, b_{in}, b_{out})$ un événement.

e est possible dans l'état s si et seulement si les jetons de b_{in} sont aussi présents dans s et sont valides.

L'intervalle de franchissement de e est l'intersection des intervalles de validité de ses jetons. Il est défini par un délai minimal ($MinD$) et un délai maximal ($MaxD$) comme suit :

- $MinD = max(0, max_{(p,c,[a,b]) \in b_{in}}(a - c))$
- $MaxD = min_{(p,c,[a,b]) \in b_{in}}(b - c)$.

Le franchissement d'un événement provoque une évolution d'état. Notons que les horloges (des jetons) accroissent avec le temps et mènent donc aussi à une évolution d'état. Nous présentons dans ce qui suit comment un état évolue par un événement de franchissement et par une progression de temps.

- *Événement de franchissement*

Un événement possible peut avoir lieu et mène ainsi au franchissement d'une transition. Dans ce cas, le modèle va passer d'un état à un autre en consommant les jetons (valides) nécessaires et en produisant d'autres. L'occurrence d'un événement est instantanée.

Si (t, b_{in}, b_{out}) se produit dans un état s_1 , alors le réseau passe dans l'état s_2 défini par : $s_2 = (s_1 - b_{in}) + b_{out}$.

- *Progression de temps*

A partir d'un état donné, le modèle peut évoluer avec progression de temps sans aucun franchissement. Une progression de temps de x unités de temps peut se produire d'un état s si et seulement si x est inférieur ou égal au maximum des délais de tous les événements possibles.

Après cette progression de temps, les horloges locales des jetons de s vont être augmentées par x unités de temps. Par exemple, prenons l'état s du processus de la figure 3.7 après 'Ouverture Site Soumission'; $s = (p_2, 0)$ montre qu'il y a un jeton créé dans la place p_2 avec une valeur d'horloge initiale 0. Après trois unités de temps, on peut ne pas avoir de papiers soumis bien que l'événement correspondant au franchissement de la transition 'Soumission Papier' est permis, mais l'état change puisque la valeur de l'horloge du jeton est changée. Le nouvel état est $s' = (p_2, 3)$.

Notons que la progression du temps peut mener à des jetons non valides (morts) qui ne peuvent pas être considérés au futur, et donc qui ne participeront pas au franchissement de transitions.

Notons maintenant qu'un système évolue par événement de franchissement ou par progression de temps. Considérons chacun de ces cas comme une occurrence d'un événement.

L'occurrence d'un événement $e = (t, b_{in}, b_{out})$ permet d'accéder d'un état s_1 à un état s_2 , ceci est aussi noté par $s_1[e]s_2$.

De plus, $s_1 \} s_2$ note l'existence d'un événement permis e tel que $s_1[e]s_2$.

Définition 3.13 Une séquence de tir est une séquence d'états et d'événements notée :

$$s_1[e_1]s_2[e_2]s_3[e_3]s_4[e_4]..$$

L'état s_n est accessible de s_1 si et seulement il existe une séquence de franchissement d'une longueur finie commençant en s_1 et finissant en s_n : $s_1[e_1]s_2[e_2]s_3..[e_{n-1}]s_n$

3.6.2.3 Graphe d'accessibilité

Nous définissons ici le graphe d'accessibilité ou simplement l'espace d'états représentant tous les états accessibles à partir de l'état initial.

Définition 3.14 Un graphe d'accessibilité est défini par un tuple (s, \rightarrow, s_0) où :

- s représente l'ensemble des nœuds spécifiant les états accessibles à partir de s_0 .
- s_0 est l'état initial.
- \rightarrow définit la relation transition (spécifiée soit par événement de franchissement soit par progression de temps).

Exemple 3.4

Pour l'ITWF-net de la figure 3.7, l'état initial est $s_0 = (i, 0)$. A partir de cet état, il y a un seul événement de tir possible qui est :

$$e_0 = ('CFP Diffusion', ((i, 0, [0, \infty]), ((P_1, 0, [0, 120])))).$$

Cet événement va se produire lorsqu'un lancement d'une instance du processus devrait avoir lieu. Le franchissement de cet événement provoque la production d'un jeton dans la place P_1 avec un intervalle de validité qui vaut $[0, 120]$, menant ainsi à l'état $s_1 = (P_1, 0)$.

A partir de cet état, un événement de franchissement aussi bien qu'une progression de temps sont possibles. L'événement de franchissement possible est

$e_1 = ('Ouverture Site Soumission', ((P_1, 0, [0, 120]), ((P_2, 0, [0, 30]))))$ et le temps peut progresser de 1 jusqu'à 120 unités de temps. Supposons que 5 unités de temps sont passées sans franchissement de la transition '*Ouverture Site Soumission*', alors le système est devenu dans l'état $s_1 = (P_1, 5)$ et il y a un événement de franchissement possible qui est :

$$e_2 = ('Ouverture Site Soumission', ((P_1, 5, [0, 120]), ((P_2, 0, [0, 30]))))$$

en ayant toujours la possibilité de progression de temps sans franchir la transition.

Supposons maintenant que l'organisateur de la conférence décide d'ouvrir le site de soumission, la production de l'événement e_2 mène à l'état $(P_2, 0)$ représentant un jeton dans la place P_2 avec une valeur d'horloge initiale 0 et un intervalle de validité $[0, 30]$. Ceci indique que le site sera ouvert uniquement pour 30 jours. Donc, un auteur peut soumettre un papier dans cette période. Supposons qu'il n'y a pas eu de soumission après 10 jours, alors le système est dans un nouvel état $(P_2, 10)$. A partir de cet état, l'événement de tir

$(\text{'Soumission Papier', } ((P_2, 10, [0, 30]), ((P_3, 0, [0, 20])))$ est possible. L'état résultant de cet événement de franchissement à cet instant est $(P_3, 0)$, et ainsi de suite, le système évolue de la même manière.

3.6.3 Analyse des ITWF-nets

Nous présentons dans le reste de cette section des résultats utiles dans l'analyse de l'accessibilité dans les ITWF-nets. L'analyse d'accessibilité est une technique qui construit le graphe d'accessibilité, appelé aussi l'arbre d'accessibilité. Un tel graphe contient un nœud pour chaque état possible et un arc pour chaque changement d'état. L'analyse d'accessibilité est une méthode très puissante dans le sens où elle peut être utilisée pour prouver tous les types de propriétés.

En général, le graphe d'accessibilité peut, même pour des réseaux de petites tailles, devenir très large voire infini. Ceci est essentiellement causée par le fait que nous utilisons les intervalles de temps et que nous avons intégré les événements de progression de temps. Donc, ce graphe ne peut être utilisé pour vérifier le fonctionnement correct d'un processus. Et c'est pour cette raison que nous passons à chercher une méthode de réduction pour le graphe d'accessibilité des ITWF-nets. Cette réduction se base sur la notion de classes d'états.

Définition 3.15 *Une classe d'états est définie par un multi-ensemble de jetons chacun admet un intervalle de validité. $SCS = (Tok \times INT)_{MS}$ est l'espace de toutes les classes d'états.*

Un événement est un triplet (t, b_{in}, b_{out}) , qui représente le franchissement possible de la transition t en éliminant les jetons spécifiés par le multi-ensemble b_{in} et en ajoutant les jetons spécifiés par le multi-ensemble b_{out} .

Dans ce cas, la condition de franchissement est définie comme suit :

Définition 3.16 *Un événement (t, b_{in}, b_{out}) est possible dans la classe d'état $sc \in SCS$ si et seulement si les trois conditions suivantes sont vérifiées :*

- $b_{in} \leq sc$
- $M(b_{in}) \in dom(F(t))$
- $b_{out} = F(t)(M(b_{in}))$

Le temps pour lequel un jeton devient disponible est spécifié par un intervalle (nous faisons abstraction aux valeurs des horloges locales des jetons), par conséquent il est impossible de spécifier le temps pour lequel un événement est permis. Cependant, il est possible de donner une borne inférieure et une borne supérieure de ce temps de possibilité d'occurrence d'un événement de franchissement e .

Définition 3.17 *La valeur minimale et la valeur maximale du temps d'occurrence d'un événement (t, b_{in}, b_{out}) sont définies comme suit :*

- $ET_{min}((t, b_{in}, b_{out})) = \max_{(p, [a, b]) \in b_{in}} a$
- $ET_{max}((t, b_{in}, b_{out})) = \max_{(p, [a, b]) \in b_{in}} b$

Un événement possible e va se produire si et seulement si le temps minimal d'occurrence de e est inférieur au temps maximal d'occurrence de tout autre événement possible.

3.7 Simulation des ITWF-nets

Le fait d'associer des durées minimales et des durées maximales aux actions des ITWF-nets offre la possibilité de modéliser plus fidèlement les processus d'entreprise du monde réel. Ainsi, ceci nous a permis de fournir une certaine souplesse dans l'instant de commencement d'exécution des différentes tâches.

Pour mettre en vigueur l'utilité des ITWF-nets, nous avons fait des simulations de ce modèle sur des exemples concrets. Etant donnée l'absence d'outils disponibles qui permettent la simulation de notre modèle ni des RdPs à arcs temporels, nous nous sommes fixés comme objectif l'implémentation d'un module, sous notre plateforme jPnet, dédié à la modélisation et à la simulation des ITWF-nets ainsi qu'à la génération de leurs graphes d'accessibilité.

3.7.1 Implémentation d'un module de simulation des ITWF-nets

Pour augmenter les processus workflow modélisés avec notre plateforme jPnet (sous forme de WF-nets) par les informations relatives aux durées des activités, nous avons dû étendre jPnet par un module de modélisation des ITWF-nets. Ce module s'intéresse donc en premier lieu aux changements à introduire dans le modèle de WF-net actuel, à savoir l'ajout des informations temporelles uniquement sur les arcs sortants des transitions du réseau. En deuxième lieu, le module doit assurer la simulation et la génération du graphe d'accessibilité des ITWF-nets modélisés correctement.

Le module que nous avons implémenté et que nous avons baptisé *Temporal Workflow Environment* assure donc les fonctionnalités détaillées dans les sous-sections qui suivent.

3.7.1.1 Modélisation d'un ITWF-net

La gestion du temps est d'une importance fondamentale pour les processus d'entreprise et en même temps très complexe. Vu cette importance, nous avons proposé de permettre à un développeur de processus de modéliser ses procédures d'entreprise contraintes par le temps à travers un sous-module implémenté sous jPnet. Ce sous-module met à disposition du développeur les constructions nécessaires à l'édition de son processus. Ces constructions sont : place initiale i , place finale f , places intermédiaires (place modélisant une ressource partagée ou non), transition et arc reliant une place à une transition ou une transition à une place.

Le but de cette phase étant d'assurer la conformité de la description du processus à la définition d'un ITWF-net. Elle doit permettre à l'utilisateur d'affecter un intervalle de temps à chaque arc reliant une transition à une place. Pour la garantir, nous avons inclus certaines modifications à la classe Arc qui permet de définir les attributs intrinsèques de l'arc avant de l'ajouter.

Ces modifications ont été manifestées par le fait que si un arc reliant une transition à une place a été ajouté par l'utilisateur, nous y associons directement un intervalle qui vaut initialement $[0, \infty[$. Ces bornes peuvent être modifiées après en sélectionnant l'arc correspondant, en cliquant sur le bouton droit de la souris et en choisissant 'Open Specification'

puis 'Time' du menu affiché. Pour les arcs sortant des places, ils n'auront pas d'attributs temps et l'utilisateur n'aura pas l'onglet relatif à la définition des attributs temporels.

La figure 3.8 représente un exemple de modélisation d'un processus workflow simple. L'opération d'ajout du délai à l'arc reliant $t1$ à $p1$ a échoué vu que la borne supérieure de l'intervalle est inférieure à la borne inférieure. Donc la détection et le traitement des éventuelles erreurs commises lors de la saisie des attributs temporels sont réalisés le plus tôt possible.

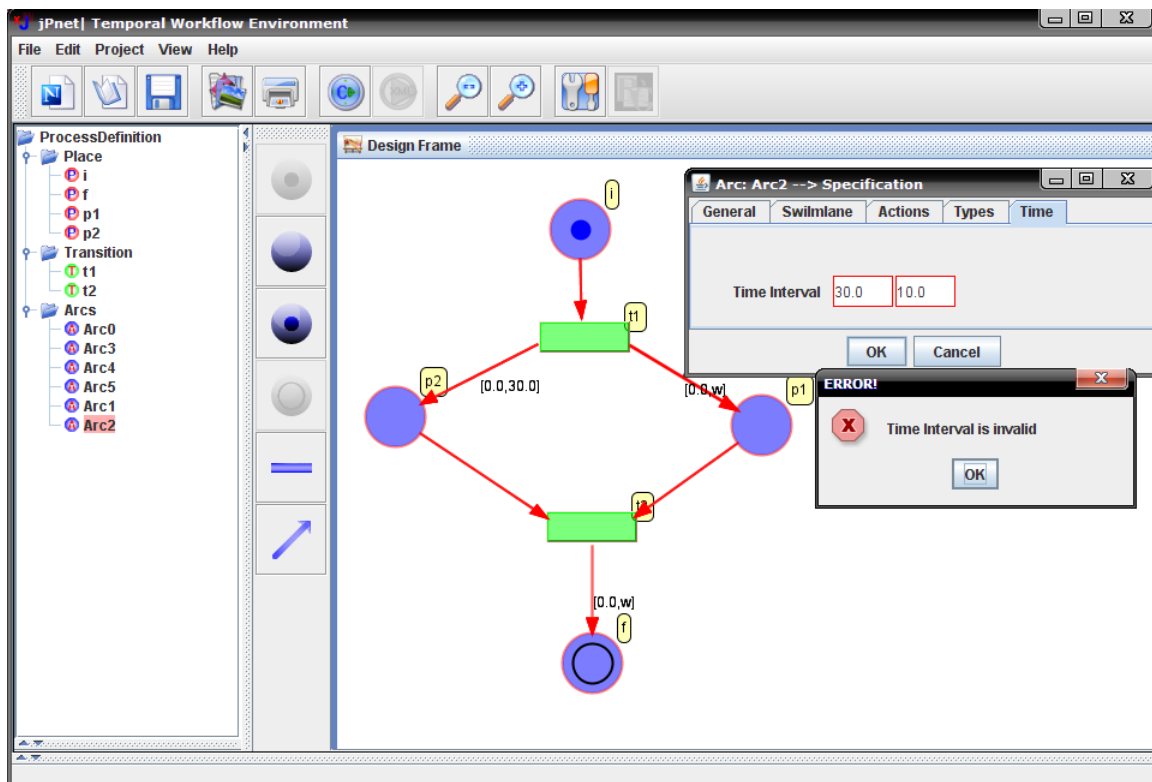


FIGURE 3.8 – Modélisation d'un processus workflow temporel sous jPnet

3.7.1.2 Compilation d'un processus modélisé

Le processus workflow édité par un développeur de processus sous jPnet doit être compilé avant d'être utilisé. La compilation est une tâche basique puisqu'elle vérifie si le processus édité est conforme avec la définition originale d'un WF-net ou non. Cette vérification est en fait commencée au fur et à mesure de l'ajout des constructions nécessaires par l'utilisateur. En effet, le fait qu'il doit y avoir une seule place source et une seule place puit est déjà assuré puisqu'il y a désactivation du bouton correspondant après la première utilisation. Reste donc à vérifier la condition suivante : pour chaque nœud x (place ou transition), il existe un chemin reliant i à x et un chemin reliant x à f . Outre les vérifications que le processus édité correspond à un WF-net, nous vérifions que les propriétés temporelles ajoutées sont conformes.

3.7. SIMULATION DES ITWF-NETS

Pour les processus workflow dont la compilation n'a pas réussi, il n'y aura pas de possibilité de simulation ni de génération du graphe d'accessibilité.

3.7.1.3 Simulation du processus

Après réussite de la compilation du processus workflow édité, le sous-menu 'Simulate Process' du menu 'Project' sera activé. En cliquant sur ce sous-menu, une fenêtre de simulation va apparaître. Cette fenêtre donnera à l'utilisateur la possibilité de commencer, mettre en pause, arrêter ou recommencer une simulation du processus en cours. Un exemple de telle fenêtre est donné dans la figure 3.9.

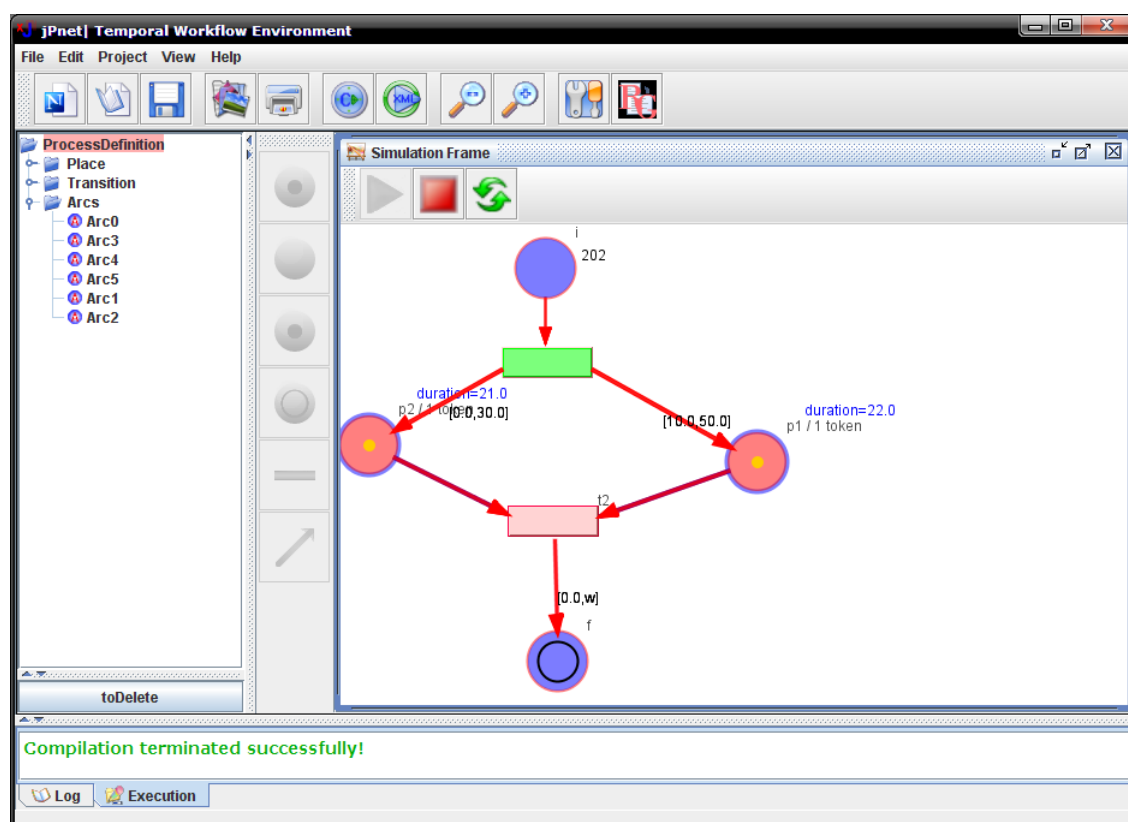


FIGURE 3.9 – Simulation d'un processus workflow temporel sous jPnet

Au début, seules les transitions qui suivent la place i sont franchissables, donc ces transitions commencent à clignoter dès le lancement de la simulation. Puis, un clic sur une des transitions franchissables provoque la consommation des jetons requis pour ce franchissement et la production des jetons dans les places en sortie de cette transition. Dans le cas des transitions sortantes de i , il n'y a pas de vérification de la disponibilité du jeton à consommer puisque son intervalle de validité vaut $[0, \infty[$. Cependant, les jetons créés vont avoir chacun un intervalle de validité calculé à partir de l'arc reliant la transition en cours avec la place dans laquelle le jeton va être produit. En plus, à ce jeton va être associée une horloge locale qui mesure la durée de son appartenance à cette place et qui

vaut initialement 0 et s'incrémente de 1 après chaque unité de temps.

Ensuite, les transitions franchissables ne sont pas reconnues uniquement par la présence de jetons dans ses places d'entrée, mais aussi par la validité de ces jetons. Ainsi, une transition n'est ajoutée dans l'ensemble des transitions franchissables (qui sont donc mis en clignotement) que si pour chacun des jetons à consommer, la valeur de son horloge locale est incluse dans son intervalle de validité. L'ensemble des transitions franchissables est recalculé à chaque unité de temps pour désensibiliser les transitions pour lesquelles un des jetons à consommer n'est plus valide (jeton mort à cause du dépassement de son intervalle de disponibilité).

L'exécution du processus, qui pourrait être suivie au fur et à mesure du choix de la transition à franchir et de la consommation et la production de jetons, va montrer à l'utilisateur les éventuels blocages qui pourraient avoir lieu. La figure 3.9 représente une capture d'écran de la simulation du processus workflow de la figure 3.8 après avoir modifié les attributs temporels. Le processus est dans un état avec deux places marquées ($p1$ et $p2$) et une transition franchissable ($t2$). La transition est franchissable (ceci est modélisé par clignotement de la transition) parce qu'elle requiert pour son exécution un jeton valide dans la place $p1$ et un jeton valide dans la place $p2$. La figure montre que ces jetons sont valides puisque les valeurs de leurs horloges locales sont incluses (respectivement) dans leurs intervalles de validité (les valeurs d'horloges sont désignées sur la figure par *duration*).

3.7.1.4 Génération du graphe d'accessibilité

La génération du graphe d'accessibilité d'un processus workflow modélisé et compilé avec succès est l'une des tâches les plus importantes de notre outil 'jPnet|Temporal-Workflow-Environment'. En effet, pour un tel processus, nous devons dresser tous les états accessibles à partir de l'état initial.

Pour les ITWF-nets, le calcul de tous les états accessibles est une tâche très difficile voire impossible puisqu'il faut donner l'état du système après passage de chaque unité de temps, vu que les états progressent soit par événement de franchissement soit par progression de temps. C'est pour cela que nous nous sommes intéressés uniquement par les événements de franchissement en faisant abstraction des événements de progression de temps qui symbolisent un écoulement de temps sans aucun franchissement de transition.

Le calcul des états accessibles est rendu donc très proche du calcul dans le cas des réseaux de Petri ordinaires puisque nous nous intéressons pour chaque état accessible aux événements possibles de tir. Ceci est équivalent dans les RdPs ordinaires à la recherche des transitions franchissables, mais dans notre cas, nous devons calculer avec chaque événement de tir possible son délai, c'est à dire la date au plus tôt et la date au plus tard de l'occurrence de l'événement de tir correspondant. Les étapes de calcul des états accessibles sont les suivantes :

– **Etape 1 :**

i) Marquer l'état initial du système, qui est connu pour tous les ITWF-nets :

$$\boxed{\textit{State } 0 : ((i, [0.0, \infty]))}$$

ii) Ajouter cet état initial à l'ensemble *ReachableStates* qui est initialement vide.

- iii*) Pour chaque état de l'ensemble *ReachableStates* Faire
- *a*) Chercher toutes les transitions franchissables (ressources requises disponibles) à partir de cet état et les ajouter dans l'ensemble *FirableTransitions* qui est initialement vide.
 - *b*) Supprimer l'état visité de l'ensemble *ReachableStates*.

– **Etape 2 :**

- i*) Pour chaque transition *t* de l'ensemble *FirableTransitions* Faire
- **a)** Calculer l'état accessible après franchissement de cette transition dans le délai requis :

Ce franchissement est représenté par l'événement (t, b_{in}, b_{out}) , donc puisque *t* est connue, et l'ensemble b_{in} est aussi connu (il représente les jetons à consommer par *t*), nous devons calculer l'ensemble b_{out} , c'est à dire l'ensemble des jetons à produire. En d'autres termes, il s'agit d'examiner les arcs sortants de *t* et de placer dans chacune des places destinataires de ces arcs un jeton avec comme intervalle de validité celui associé avec l'arc.

L'affichage concerne toutes les informations sur le changement d'état du système et du franchissement établi. En effet, il montre avant de donner la valeur d'un état donné, l'état qui le précède, la transition à franchir et la condition de son franchissement. Pour l'exemple de la figure 3.10, le franchissement de *t1* mène à l'état 'State1' affiché comme suit :

<i>State 1</i> <i>From : State 0</i> <i>Firable Transition : t1</i> <i>Firing Interval : [0.0, ∞[</i> <i>State : ((p1, [0.0, 20.0]), (p2, [0.0, 20.0]))</i>

- **b)** Ajouter l'état trouvé (caractérisé par b_{out}) dans l'ensemble *ReachableStates*.
 - **c)** Supprimer *t* de l'ensemble *FirableTransitions*
- ii*) Si $ReachableStates \neq \emptyset$ Alors Aller vers *Etape 1, iii*)

Si l'état final : $((f, [0.0, \infty[))$ est atteint, le calcul des états accessibles de cet état ne va pas ajouter d'éléments dans *FirableTransitions* et donc la recherche des états accessibles est terminée.

Notre outil de modélisation des ITWF-nets donne au développeur de processus la possibilité d'enregistrer le projet de modélisation de son processus workflow pour l'ouvrir ultérieurement en vue de modification, de simulation ou d'autres actions.

En outre, le graphe d'accessibilité généré peut être sauvegardé dans un fichier html pour une consultation ultérieure plus simple. L'affichage dans le fichier html se fait de la même manière que dans la fenêtre de présentation du graphe d'accessibilité dans jPnet.

3.7. SIMULATION DES ITWF-NETS

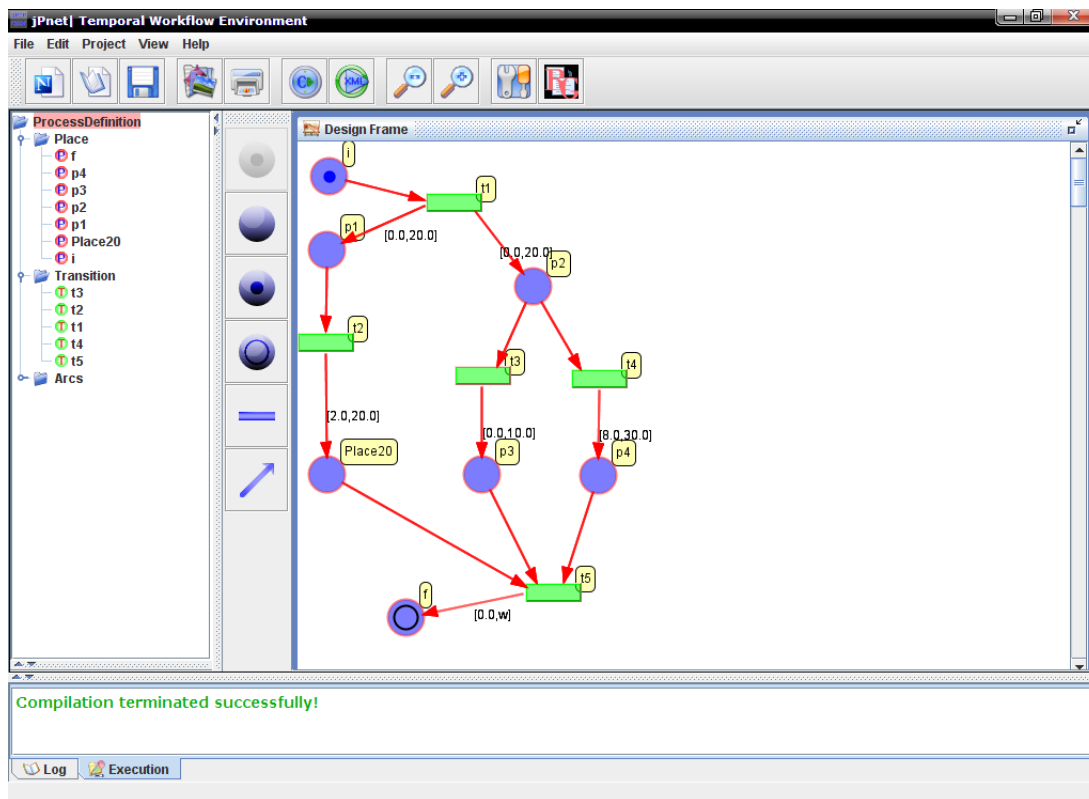


FIGURE 3.10 – Exemple de modélisation de processus workflow sous jPnet

3.7.2 Etudes de cas

3.7.2.1 Processus de relecture d'un papier dans une conférence

La figure 3.11 schématise une capture d'écran de la modélisation du processus de relecture d'un papier dans une conférence présenté dans la section 3.6.1.

Le processus modélisé a été compilé avec succès et l'espace de ses états accessibles a été généré donnant lieu à 15 états accessibles à partir de l'état initial $((i, [0.0, \infty[))$ et menant à l'état final $((f, [0.0, \infty[))$.

Le graphe d'accessibilité de ce processus est donné en deux parties dans les figures 3.12 et 3.13.

3.7. SIMULATION DES ITWF-NETS

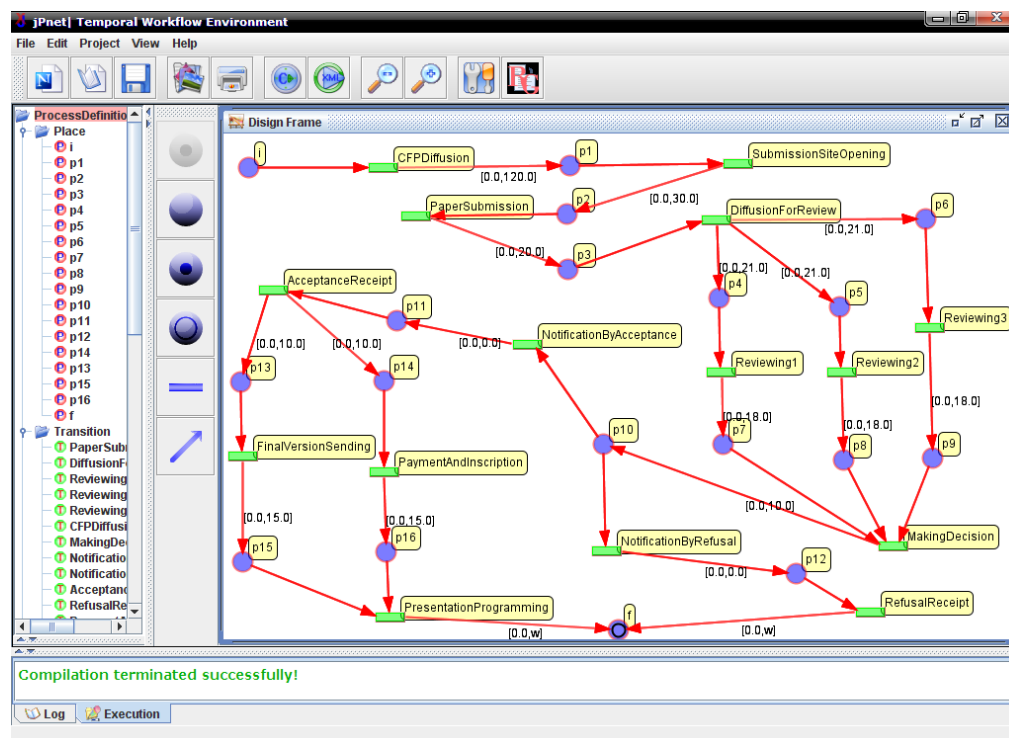


FIGURE 3.11 – Modélisation du processus de conférence sous jPnet

3.7.2.2 Processus de déroulement d'un Projet de Fin d'Année à l'ENIT

On considère le processus workflow de la réalisation d'un Projet de Fin de la 2^{ème} Année (PFAII) à l'ENIT (ce processus est donné dans la figure 3.14). Nous supposons que le processus commence par "le choix du sujet". Après le choix, l'élève-ingénieur doit "faire des réunions avec l'encadreur" dans 10 jours (c'est un sous processus que nous ne traitons pas dans notre exemple) et en même temps il doit faire de la "documentation" sur le sujet choisi.

Puis, l'élève-ingénieur, après 15 jours, doit "proposer un plan de travail" qui doit être vérifié et "validé par l'encadreur" dans 5 jours au plus tard. La réalisation de l'application et du rapport se feront ensuite en parallèle et chacune de ces tâches ne doit pas dépasser 30 jours. Il faut que l'encadreur vérifie les travaux de l'élève-ingénieur dans 15 jours au plus pour que ce dernier puisse déposer son rapport. Le rapport reste 2 jours chez l'administration puis le rapporteur désigné pourra l'examiner dans au plus 5 jours. Dans cette période, l'élève-ingénieur est en train de réaliser sa présentation orale. La tâche de présentation aura lieu dans 7 jours après et enfin l'évaluation du PFAII ne doit pas dépasser un jour.

La figure 3.14 présente une capture d'écran de la modélisation du processus de déroulement d'un PFAII à l'ENIT. La génération du graphe d'accessibilité correspondant a résulté en 14 états accessibles de l'état initial et aboutissant à l'état final ($(f, [0.0, \infty[)$) après exécution de la tâche d'évaluation du projet par les membres du jury.

3.7. SIMULATION DES ITWF-NETS

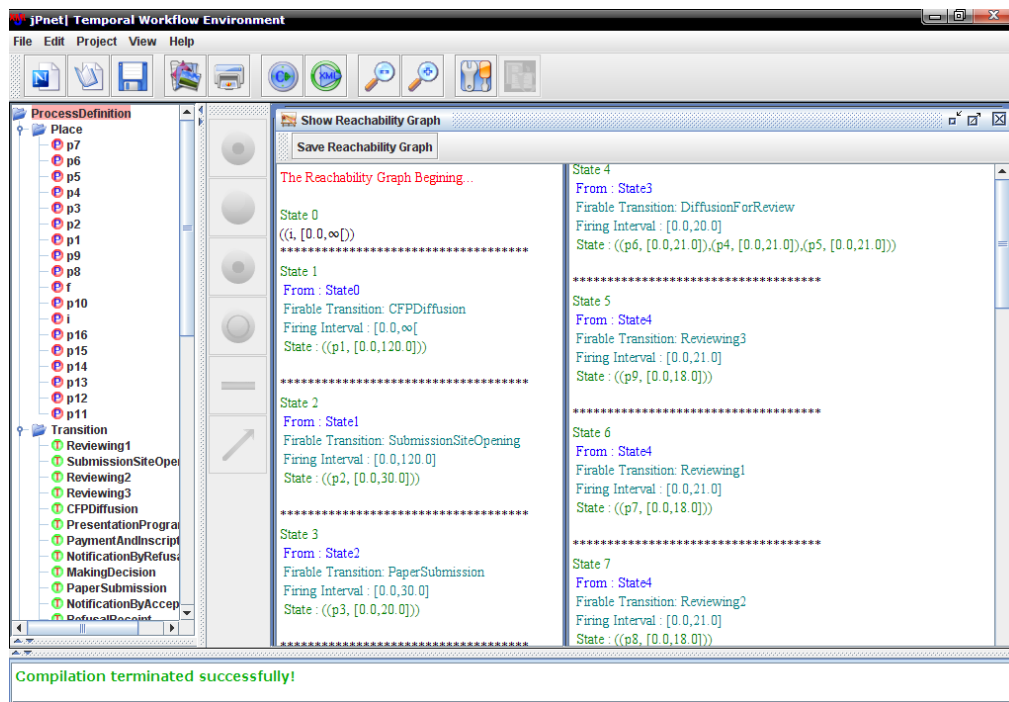


FIGURE 3.12 – Le graphe d’accessibilité du processus de la conférence généré - Partie1

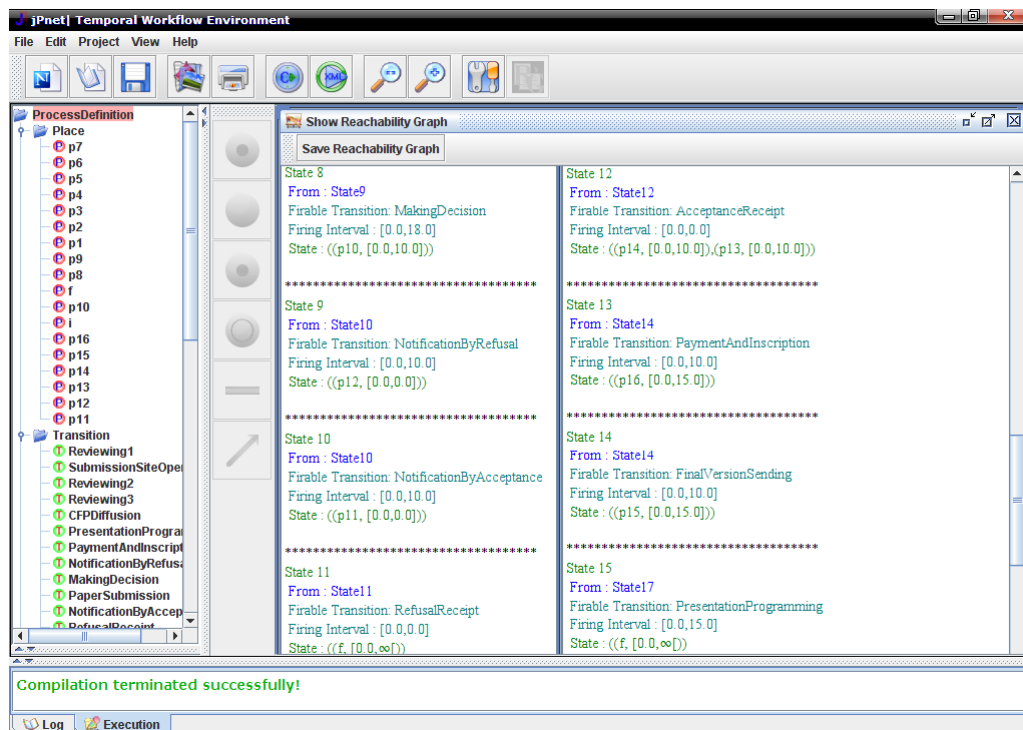


FIGURE 3.13 – Le graphe d’accessibilité du processus de la conférence généré - Partie2

3.8. CONCLUSION

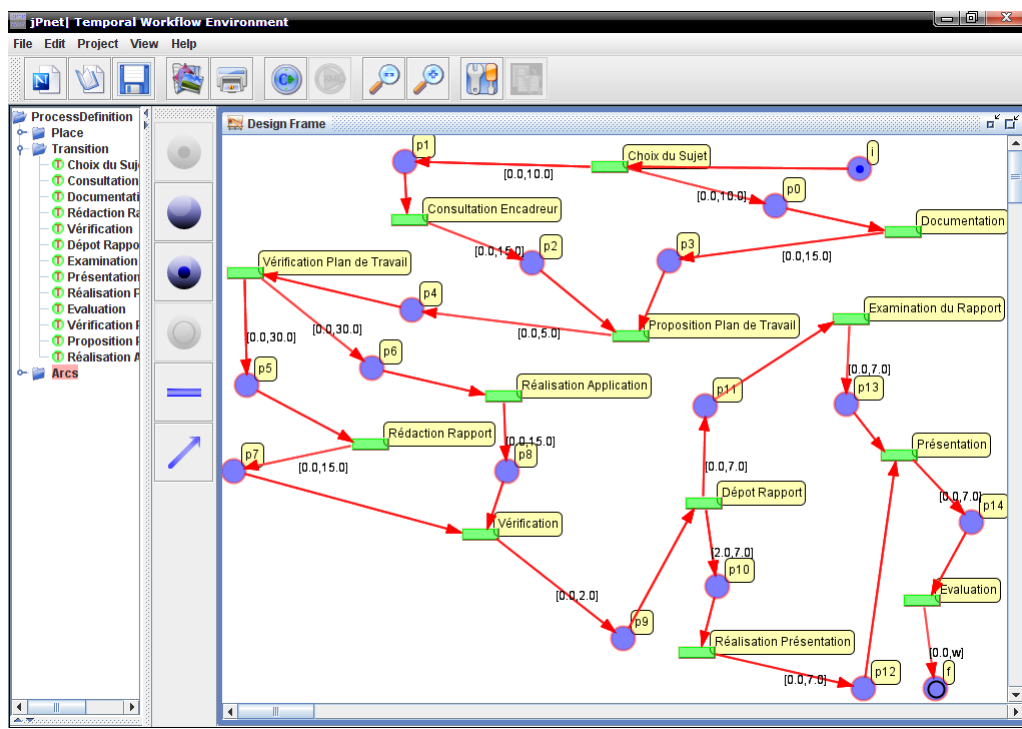


FIGURE 3.14 – Modélisation du processus de PFAI sous jPnet

3.8 Conclusion

Visant les caractéristiques des systèmes de workflow et en se focalisant sur les processus métier complexes du monde réel qui contiennent de grande quantité d'informations de données et de temps, nous avons proposé en premier lieu un modèle de workflow basé sur les réseaux de Petri temporisés : le modèle de TWF-net. En deuxième lieu, nous avons présenté un modèle temporel de processus workflow -baptisé ITWF-net- basé sur les réseaux de Petri à arcs temporels.

Nous avons défini dans la première partie de ce chapitre le modèle de TWF-net et ses sémantiques en termes de franchissabilité, de règles de franchissement et de l'évolution des états du modèle. Ensuite, nous avons présenté trois propriétés des TWF-nets qui peuvent être vérifiées lors de l'analyse de l'espace d'état : la cohérence, la finitude et la vivacité. Nous avons montré que pour un TWF-net N , la vivacité et la finitude du réseau fermeture N^* sont des conditions nécessaires et suffisantes pour la propriété de la cohérence de N . Ce résultat est trouvé en se basant sur la caractérisation de la cohérence des WF-nets simples donnés dans [4, 7] et l'étude de l'effet des contraintes temporelles sur les états du modèle. Puis, nous avons étudié l'impact de la gestion d'un nombre arbitraire d'instances du même processus dans un TWF-net en termes d'états du système et de propriétés vérifiables.

La deuxième partie de ce chapitre a été réservée à la proposition d'un nouveau modèle -plus général- de workflow à contraintes temporelles. Après la définition formelle de ce modèle de ITWF-net, nous avons étudié son comportement dynamique et par conséquent

3.8. CONCLUSION

nous avons défini ses états ainsi que leurs évolutions. Ensuite, nous avons présenté une méthode d'analyse des ITWF-nets basée sur le graphe d'accessibilité. Puis, vu que ce graphe pourrait être très large et même infini, nous avons donné une méthode de réduction du graphe d'accessibilité. Cette méthode basée sur la notion de classes d'états (regroupement des états de même marquage) a nécessité d'introduire quelques modifications dans les conditions de franchissement d'une transition et dans le temps d'activation et de production d'un événement. Enfin, nous avons présenté notre module de simulation des ITWF-nets que nous avons implémenté sous jPnet et baptisé 'Temporal Workflow Environment'.

Chapitre 4

Vérification de processus workflow par Model Checking

4.1 Introduction

Les tests et la simulation avaient été, depuis longtemps, utilisés dans la vérification de systèmes. Cependant, ces méthodes de vérification ne permettent d'explorer qu'une partie des comportements possibles. Pour cela, elles diffèrent des techniques de vérification formelle qui garantissent qu'une propriété est vérifiée par toutes les exécutions possibles du système.

Les deux principales techniques de vérification formelle sont la preuve et le model checking. Dans la première technique, un modèle du système à vérifier est vu comme un ensemble d'axiomes qui servent à prouver une propriété. Cette technique n'est pas automatique ; ils existent des outils d'aide à la preuve mais demandent une forte expertise. Le model checking est constitué de plusieurs techniques automatiques dans lesquelles la propriété à vérifier est testée exhaustivement sur l'ensemble des exécutions possibles du système.

L'objectif de ce chapitre est d'exposer notre utilisation du model checking pour vérifier si un processus workflow satisfait ou non une spécification représentant son comportement. Nous présentons notre approche qui permet d'abord de traduire un processus workflow en modèle Promela, par la suite, de formuler la propriété désirée en Logique Temporelle Linéaire (LTL) et enfin de lancer la vérification en utilisant l'outil SPIN.

Nous commençons par présenter dans la section 4.2 les notions fondamentales du model checking ainsi que le principe du model checker SPIN et de la logique temporelle adoptée LTL. Ensuite, nous détaillons dans la section 4.3 la description en langage Promela du processus workflow à vérifier par SPIN. Puis, nous traitons, dans la section 4.4 de la vérification des propriétés d'un processus workflow et nous analysons divers aspects de la cohérence d'un processus et ce en les formulant en LTL. Enfin, nous étendons les formules proposées pour vérifier les processus workflow modélisant plusieurs instances et partageant des ressources (section 4.5).

4.2 Model checking

4.2.1 Intérêt des méthodes formelles

Contrairement à la spécification en langage naturel pouvant donner lieu à différentes interprétations, la spécification formelle d'un système se base sur un langage formel avec une sémantique bien définie. Cette description formelle peut être utilisée afin de vérifier que la réalisation finale du système respecte les fonctionnalités prévues.

Pour prouver des propriétés sur un système, on se base sur la spécification qui forme une représentation abstraite du système ne contenant que les détails nécessaires pour la description du comportement du système. Par ailleurs, débarrassé des détails inutiles, il est généralement plus simple de vérifier des propriétés sur la spécification du système que sur sa description complète.

Les méthodes formelles sont des outils puissants pour les développeurs des logiciels devant fonctionner correctement. On distingue deux grandes catégories de méthodes formelles permettant la vérification de propriétés sur des systèmes spécifiés formellement : la preuve de théorème et le model checking.

- La preuve de théorème consiste à demander de l'ordinateur de prouver automatiquement des propriétés à partir d'une description du système, d'un ensemble d'axiomes et d'un ensemble de règles d'inférences. Cette recherche de preuve est désormais connue comme un problème non décidable [126], c'est à dire il n'existe pas d'algorithme qui permet de vérifier, en temps fini, si une certaine propriété est satisfaite ou non. Il est à noter l'existence de cas pour lesquels le problème est décidable et donc existent des algorithmes pouvant le résoudre. Cependant, le temps et les ressources requis pour la vérification de propriétés avec ces algorithmes peuvent dépasser les ressources que dispose l'ordinateur.
- Le model checking consiste en un ensemble de techniques permettant une analyse automatique des systèmes réactifs. Il permet de vérifier des propriétés sur un système en effectuant une énumération exhaustive de tous les états accessibles. Etant données une spécification formelle du système et une spécification d'une propriété à vérifier, le model checking répond par vrai si la propriété est satisfaite par toutes les exécutions possibles du système et répond par faux s'il existe une exécution qui ne satisfait pas la propriété. Cette exécution peut être retournée comme un contre-exemple.

4.2.2 Principe du model checking

Le model checking ou la vérification de modèle est l'une des méthodes formelles les plus puissantes. Le model checking est une méthode de vérification exhaustive des systèmes. Au lieu de tester des prototypes physiques, il s'agit de vérifier exhaustivement un modèle mathématique du système à vérifier et donc en utilisant un algorithme qui explore toutes les configurations possibles du système. Le nombre de ces configurations peut être énormément grand, les outils de model checking actuels peuvent énumérer explicitement des espaces d'états contenant 10^{20} états [36].

Le principe du model checking est de générer toutes les exécutions possibles d'un programme et de s'assurer que les propriétés spécifiées sont vérifiées dans chaque exécution.

En outre, la production des exécutions et le contrôle des spécifications peuvent être faits automatiquement par un outil logiciel dit model checker ou vérificateur de modèle.

On distingue deux différentes techniques de model checking qui se différencient par la façon de représenter l'ensemble des exécutions qui peut être infini : le model checking symbolique [36] basé sur une représentation ensembliste des configurations que peut avoir un système, et le model checking basé sur l'approche automate [112] pour lequel les exécutions du système et la propriété sont représentés par des automates. Dans le reste de cette section, nous nous intéressons à l'approche automate du model checking.

Dans l'approche automate du model checking, illustrée par la figure 4.1 [104], le modèle M d'un système à vérifier est vu comme un automate A_M (dit w-automate) reconnaissant des mots de longueur infinie (dits w-mots). Chaque lettre de ces w-mots correspond à une configuration du système. Ainsi, un w-mot représente une séquence d'exécution du système traversant chacune de ces configurations. Le langage de l'automate (noté $\mathcal{L}(A_M)$) est l'ensemble des w-mots qu'il reconnaît ; il représente tous les comportements possibles du système.

De même, la propriété comportementale φ à vérifier sur M est exprimée par un w-automate $A_{\neg\varphi}$ dont le langage est l'ensemble des comportements qui ne satisfont pas la propriété φ . Pour déterminer si $\mathcal{L}(A_M)$ et $\mathcal{L}(A_{\neg\varphi})$ partagent un w-mot, i.e. s'il existe une exécution de M qui ne vérifie pas φ , le produit synchronisé des deux automates est d'abord construit, il s'agit d'un w-automate reconnaissant l'intersection des deux langages. Puis, un "test du vide" de ce produit est effectué, cette opération détermine si le langage reconnu par un automate est vide. Ceci est vérifié lorsqu'il n'existe aucune séquence d'exécution de M pour laquelle φ est invalide. Si par contre ce test n'est pas vérifié, un contre-exemple peut être retourné. Le contre-exemple représente un w-mot traçant une exécution du système qui ne satisfait pas φ .

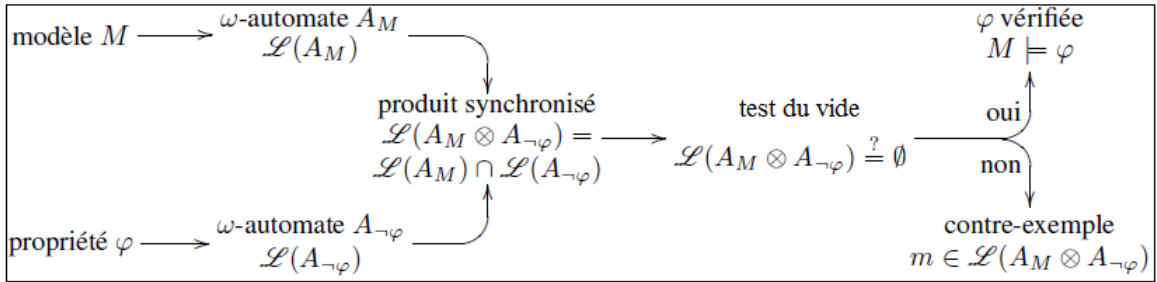


FIGURE 4.1 – Approche automate du model checking.

A travers cette approche, on utilise des structures finies qui sont les w-automates pour représenter et manipuler des ensembles infinis à savoir les langages. Ils existent plusieurs types de w-automates se distinguant par leurs conditions d'acceptation, c'est à dire leur manière d'indiquer quels chemins infinis de l'automate correspondent à des w-mots acceptés. Les w-automates les plus utilisés sont appelés automates de Büchi.

Le model checking basé sur l'approche automate permet de vérifier toute propriété qu'on peut exprimer par un automate de Büchi. Cependant, il n'est pas toujours aisé de spécifier des propriétés directement sous cette forme. En contre partie, on peut utiliser des

logiques telle que la Logique Temporelle Linéaire (LTL) [112] dont la sémantique est simple et dont les énoncés peuvent être transformés en automates.

La vérification de modèle peut être effectuée, en pratique, en utilisant des algorithmes sophistiqués basés sur la logique et la théorie des automates.

4.2.3 SPIN Model Checker

Un model checker, l'outil logiciel qui effectue la vérification de modèle, examine tous les scénarios possibles du système de manière systématique. De cette façon, il peut être démontré qu'un modèle de système donné répond véritablement à une certaine propriété. Il existe beaucoup d'outils puissants de model checking comme NuSMV [39], BLAST [60] et SPIN [61, 62]. Dans ce travail nous adoptons SPIN.

SPIN est un model checker basé sur l'approche automate développé par Gerard J. Holzmann pour vérifier les protocoles de communication. Il a été largement répandu dans les industries qui construisent des systèmes critiques. Les modèles devant être vérifiés par SPIN sont écrits en Promela, un langage qui décrit le comportement du système. Les propriétés à vérifier doivent être formulées en Logique Temporelle Linéaire (LTL).

Etant données la spécification formelle du comportement du système et la spécification d'une propriété en logique temporelle, SPIN model checker vérifie si cette propriété est satisfaite. En effet, le mode de vérification de SPIN détermine si le modèle proposé en Promela satisfait ou non une propriété LTL. Ceci est effectué en menant, si nécessaire, une vérification exhaustive i.e. en parcourant l'ensemble des exécutions possibles du système.

Le principe de vérification de SPIN est qu'il transforme, à partir des propriétés spécifiées, leur négation en automates de Büchi [112]. SPIN calcule ensuite le produit synchronisé de cet automate avec celui du système. Le résultat est encore un automate de Büchi dont on vérifie si le langage est vide ou non. S'il est vide, alors il n'existe aucune exécution violant la formule de départ, sinon il accepte au moins une exécution (ou mot du langage de l'automate, chaque lettre du mot étant représentée par une action du système ou de la propriété).

Un automate de Büchi accepte un mot (et donc son langage est non-vidé) si et seulement si l'exécution du système passe infiniment souvent par un ou plusieurs des états acceptants de l'automate. I.e. qu'il existe un cycle accessible depuis l'état initial qui comporte un état acceptant. Pour prouver que toutes les exécutions du système sont valides vis-à-vis de la propriété à vérifier, il suffit de prouver qu'il n'existe pas de cycle acceptant accessible depuis l'état initial. Si ce mot existe, il va permettre de trouver un ensemble de transitions des automates du système qui viole la propriété. Un mot qui viole la propriété s'appelle un contre-exemple.

L'ensemble des transitions pour lesquelles la propriété n'est pas vérifiée forment les contre-exemples qui peuvent être générés par SPIN. Ces contre-exemples sont censés permettre à l'utilisateur de modifier par la suite son modèle (ou la propriété si elle a été mal exprimée), afin de corriger les erreurs éventuelles.

La méthode utilisée, pour trouver un contre-exemple, est une recherche en profondeur imbriquée (Nested Depth-First Search [62, 63]) : un premier DFS est lancé de façon à trouver tous les états acceptants accessibles à partir de l'état initial. Ensuite, l'algorithme

cherche, en relançant un DFS (dit nested, ou imbriqué) à partir de ces états.

4.2.4 Logique Temporelle

La Logique Linéaire Temporelle (LTL) est la logique formelle dans laquelle sont écrites les formules à vérifier par SPIN. LTL est basée sur le calcul propositionnel. Les formules du calcul propositionnel sont composées de propositions atomiques et des opérateurs. Ces opérateurs sont donnés dans la table 4.1 en utilisant la notation mathématique en plus de la syntaxe utilisée en SPIN pour écrire ces formules.

Opérateur	Math	SPIN
non	\neg	!
et	\wedge	&&
ou	\vee	
implique	\rightarrow	- >
équivalent	\leftrightarrow	< - >

TABLE 4.1 – Opérateurs propositionnels de LTL utilisés en SPIN

Outre les opérateurs dérivés du calcul propositionnel, sont utilisés dans les formules LTL les opérateurs temporels donnés dans le tableau 4.2.

Opérateur	Math	SPIN
éventuellement	\diamond	<>
toujours	\square	[]
jusqu'à	\mathcal{U}	\mathcal{U}

TABLE 4.2 – Opérateurs temporels de LTL utilisés en SPIN

les opérateurs \diamond et \square sont unaires et l'opérateur \mathcal{U} est binaire. Les opérateurs temporels et propositionnels peuvent être combinés dans la même formule LTL.

La sémantique d'une formule syntaxiquement correcte est définie en lui donnant une interprétation : Une assignation des valeurs de vérité, T (vrai) ou F (faux), à ses propositions atomiques et l'extension de l'assignation à une interprétation de la formule entière en tenant compte des règles des opérateurs. Pour le calcul propositionnel, ces règles sont définies par les tables de vérité usuelles. Pour la logique temporelle, la sémantique d'une formule est donnée en termes d'exécutions (calculs) et les états d'une exécution. Les propositions atomiques de la logique temporelle sont des expressions booléennes qui peuvent être évaluées dans un seul état indépendamment d'une exécution.

4.3 Un modèle Promela des processus workflow

Dans ce travail, nous présentons une méthode, basée sur SPIN model checker, pour la vérification des propriétés de cohérence d'un workflow net. Avant d'interroger SPIN, une

préparation du modèle à vérifier est nécessaire et se fait en deux étapes. La première étape consiste à traduire le WF-net en langage Promela. Dans la deuxième étape, les propriétés sont exprimées par des formules LTL (Logique Temporelle Linéaire). La première étape fait l'objet de la section en cours alors que la deuxième étape fera l'objet de la section 4.4.

Une méthode de modélisation des réseaux de Petri avec le langage Promela a été introduite par Holzmann dans [61, 62]. Dans cette méthode, un réseau de Petri est représenté par un processus unique. La représentation du réseau de Petri donnée dans [61, 62] a été réécrite dans [121] représentant l'ensemble des places P par un tableau de $|P|$ éléments plutôt que $|P|$ variables.

La spécification Promela d'un réseau de Petri donnée dans [61] ne décrit que le marquage des places et ne prend pas en compte le nombre de tirs des transitions. Toutefois, il est nécessaire d'enregistrer le nombre de franchissement de chaque transition, tout en progressant dans l'exécution du processus. Ces informations seront utilisées lors de la vérification des critères de cohérence.

Dans le reste de cette section, nous présentons une spécification du workflow net en langage Promela.

Le comportement d'un WF-net peut être décrit en termes d'états du système et de leur évolution. Un marquage est initialisé à M_i et il est modifié en fonction de la règle de transition qui suit : Une transition t est franchissable si chaque place d'entrée de t est marquée. En outre, une transition franchissable t peut être franchie ou pas. Un franchissement de t enlève un jeton de chaque place d'entrée $p \in \bullet t$ et ajoute un jeton à chaque place de sortie de t .

Le comportement dynamique d'un processus peut être décrit en Promela dans le processus *init*. Ainsi, le mot-clé *init* est utilisé pour déclarer le comportement du processus qui est actif à l'état initial du système. Dans ce processus, nous proposons de décrire l'exécution des tâches par une boucle *do* dans laquelle chaque ligne spécifie un franchissement d'une transition (i.e. une exécution d'une tâche). Par conséquent, chaque ligne décrit les instructions à exécuter pour assurer le franchissement de la transition : enlèvement des jetons des places d'entrée et production des jetons dans les places de sortie. La séquence de ces instructions relatives à un franchissement d'une transition doit être exécutée comme une unité atomique, sans entrelacement avec d'autres processus, et donc nous précédonz la séquence des instructions du franchissement par *atomic*.

Nous proposons de présenter la spécification Promela d'un WF-net en BNF (Backus Naur Form) dans laquelle $\langle Process \rangle$ est l'axiome (symbole initial) :

$$\begin{array}{l}
 \hline
 \langle Process \rangle ::= \textit{init} \{ \\
 \quad \langle Initial_Marking \rangle \\
 \quad \textit{do} \\
 \quad \quad \langle Firings \rangle \\
 \quad \textit{od} \\
 \quad \} \\
 \langle Firings \rangle ::= \langle Firing \rangle \langle Firings \rangle \\
 \langle Firing \rangle ::= \textit{atomic} \{ \\
 \quad \langle Enabled_Tk \rangle \rightarrow \langle Fire_Tk \rangle \\
 \quad \} \\
 \hline
 \end{array}$$

Notons que dans la dernière ligne $\langle Enabled_Tk \rangle \rightarrow \langle Fire_Tk \rangle$, $\langle Enabled_Tk \rangle$ est une condition et $\langle Fire_Tk \rangle$ est une action qui va être exécutée si et seulement si $\langle Enabled_Tk \rangle$ est vraie. L'exécution de la boucle *do* continue d'une façon non déterministe dans une branche dont la condition est vraie.

Pour simplifier la spécification Promela d'un WF-net, nous proposons de décrire le WF-net en termes de marquage des places et de nombre de tirs des transitions. Par conséquent, nous utilisons les conventions suivantes :

- Les places sont représentées par un tableau PL de variables de types *byte* de longueur égale au nombre de places. Ce tableau contient initialement zéro dans chaque élément.
- Les transitions sont modélisées par un tableau TR de variables de types *byte* (initialisées à zéro) dont la longueur est égale au nombre de transitions.

Les concepts liés au comportement dynamique d'un processus workflow sont traduits dans la description Promela du WF-net correspondant comme suit : Le tir d'une transition consiste d'une part en la décrémentation de 1 de la valeur correspondante à chaque place $p \in \bullet t$ dans le tableau PL (destruction de jetons) et d'autre part en l'incrémementation de 1 des éléments de PL correspondant aux places de sortie ($p \in t \bullet$) (c'est ce qui correspond à la production de jetons). De plus, ce tir de t va être enregistré en augmentant de 1 l'élément de TR correspondant à la transition t .

Ces modifications sont assurées par les macros *fire*, *addi*, i de 1 à S et *removej*, j allant de 1 à K , K et S sont respectivement le nombre maximum de places d'entrée possibles et le nombre maximum de places de sortie possibles.

Pour franchir une transition t qui a I places d'entrée et J places de sortie, nous appelons ces macros avec les arguments appropriés :

1. *removeI*(p_1, p_2, \dots, p_I) détruit un jeton de chaque place d'entrée. Ce macro vérifie si ces places d'entrée sont marquées ($PL[indice_de_p_j] > 0$, $1 \leq j \leq I$) et une fois cette condition est satisfaite, il enlève un jeton de chacun de ses paramètres.
2. *fire*(t) incrémente l'élément correspondant à t dans TR .
3. Et finalement *addJ*(p_1, p_2, \dots, p_J) crée un jeton dans chacune des places passées en arguments de ce macro.

Ainsi, en utilisant ces macros, la séquence atomique $\langle Enabled_Tk \rangle \rightarrow \langle Fire_Tk \rangle$ va être réécrite comme suit :

4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

$$\frac{\text{remove}N(PL[I_1], PL[I_2], \dots, PL[I_N])}{\text{fire}(TR[indice_de_Tk]); \text{add}M(PL[O_1], PL[O_2], \dots, PL[O_M])}$$

Dans cette spécification Promela d'un franchissement d'une transition Tk , nous avons noté par N le nombre des places d'entrées de Tk et par M le nombre de ses places de sortie. Nous avons aussi supposé que les indices des places d'entrées dans PL sont I_1, I_2, \dots, I_N et les indices des places de sortie sont O_1, O_2, \dots, O_M .

Cette description Promela d'un WF-net et son comportement est illustrée par un exemple.

La description Promela du WF-net de la figure 4.2 est présentée dans la figure 4.3.

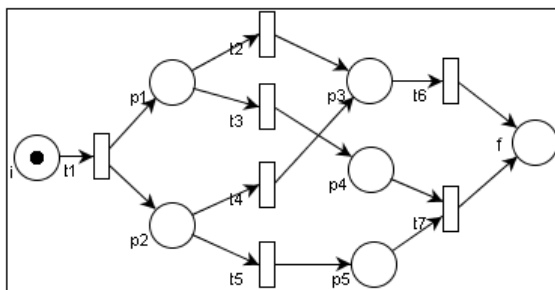


FIGURE 4.2 – Un exemple de WF-net

4.4 Formulation des propriétés de cohérence d'un WF-net en LTL

Une propriété est exprimée en SPIN par un automate fini appelé "never claim" qui est exécuté en même temps avec l'automate fini qui représente le programme Promela. Spécifier une propriété d'exactitude directement comme un never claim est difficile, en contre partie une formule écrite en Logique Temporelle Linéaire sera traduite par SPIN en un never claim, utilisé plus tard pour la vérification.

Le model checking des propriétés de la Logique Temporelle Linéaire (LTL) peut être fait à l'aide de l'approche théorique des automates. Cette méthode de vérification de modèle adopte une traduction de propriétés exprimées en LTL en automates finis sur des mots infinis (automates de Büchi), qui seront ensuite utilisés pour déterminer si un modèle de système vérifie une propriété LTL donnée.

SPIN fournit un algorithme pour convertir les formules LTL en automates de Büchi équivalents. never claim est le modèle Promela de l'automate de Büchi correspondant à la formule LTL. Ainsi, les never claims sont utilisés pour spécifier le comportement du système qui ne devrait jamais se produire.

Lors de la génération d'un vérificateur, SPIN crée un processus pour le never claim déclaré. Le vérificateur exécute le processus du never claim entre toutes les exécutions des autres processus et reporte une erreur si le processus du never claim se termine.

4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

```

#define PLACE 7
#define TRANSITION 7

byte TR[TRANSITION]
byte PL[PLACE]

#define remove1(x)      (x>0) -> x—
#define remove2(x,y)  (x>0 && y>0) -> x--; y—

#define add1(x)        x++
#define add2(x,y)     x++; y++

#define fire(x)       x++

init
{  PL[0]=1;
  do
  :: atomic { remove1(PL[0])      -> fire(TR[0]); add2(PL[1],PL[2]) }
  :: atomic { remove1(PL[1])     -> fire(TR[1]); add1(PL[3])      }
  :: atomic { remove1(PL[1])     -> fire(TR[2]); add1(PL[4])      }
  :: atomic { remove1(PL[2])     -> fire(TR[3]); add1(PL[3])      }
  :: atomic { remove1(PL[2])     -> fire(TR[4]); add1(PL[5])      }
  :: atomic { remove1(PL[3])     -> fire(TR[5]); add1(PL[6])      }
  :: atomic { remove2(PL[4],PL[5]) -> fire(TR[6]); add1(PL[6]) }
  od
}

```

FIGURE 4.3 – Modèle Promela du WF-net de la figure 4.2

Supposons que nous voulions exprimer la propriété *Prop* qui indique que la place *f* du processus workflow présenté dans la figure 4.2 va éventuellement être toujours marquée. Alors, nous avons à définir dans le programme Promela une proposition *p* qui précise que la place *f* est marquée :

```

#define p (PL[6] >= 1)

```

La formule LTL de *Prop* est $F : \langle \rangle [] p$ (selon la syntaxe de SPIN). Elle correspond à l'automate de Büchi de la figure 4.4.

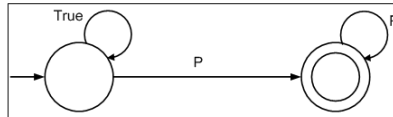


FIGURE 4.4 – Un automate de büchi correspondant à F

Pour transformer une propriété en un never claim, il suffit de l'inverser. On obtient alors la formule LTL suivante :

```

!<> [ ] p = [ ] ! [ ] p = [ ] <> !p

```


4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

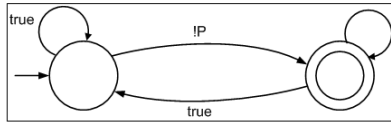


FIGURE 4.5 – Un automate de büchi correspondant à la négation de F

Cette formule (négation de F) correspond à l'automate de Büchi de la figure 4.5.

Le never claim (généré par SPIN) correspondant à la formule F est donné dans la figure 4.6.

```

never {      /* !( <> [!p] ) */
T0_init :
    if
    :: (! ((p))) -> goto accept_S9
    :: (1) -> goto T0_init
    fi ;
accept_S9 :
    if
    :: (1) -> goto T0_init
    fi ;
}

```

FIGURE 4.6 – Le never claim correspondant à F

Dans le reste de cette section, nous décrivons notre méthode de vérification des propriétés de cohérence d'un processus workflow par LTL et Promela.

4.4.1 Formules LTL des sous propriétés de la cohérence d'un processus workflow

Avant de vérifier une description Promela donnée d'un WF-net, nous devons contrôler si cette description présente une spécification correcte d'un WF-net. Ici, la propriété correcte concerne la structure du processus workflow donné. En d'autres termes, nous devons vérifier les trois affirmations mentionnées dans la définition 2.1 d'un WF-net. Cette définition est aisément reformulée pour définir un réseau de Petri structurellement correct comme suit.

Un réseau de Petri N est structurellement correct si et seulement si :

1. Il y a exactement un nœud initial qui est sans arcs entrants.
2. Il comporte un nœud final, qui est le seul nœud qui ne possède pas des arcs sortants.
3. Chaque nœud du modèle est sur un chemin du nœud initial vers le nœud final.

Nous pouvons citer comme erreur structurelle la présence de transitions sans places d'entrée ou places de sortie par exemple.

Nous proposons dans la suite que nous traitons des WF-nets structurellement corrects.

Une des principales propriétés qu'un processus workflow doit satisfaire est la propriété de cohérence introduite dans la section 2.2.3 (définition 2.2).

4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

Cette définition exprime qu'un WF-net est cohérent si et seulement si ces trois conditions sont satisfaites : terminaison, terminaison propre et absence d'inter-blocages (pas de transitions mortes).

Pour vérifier la cohérence et en se référant à cette définition, nous allons tout d'abord traduire ces trois sous-propriétés de base.

4.4.1.1 Terminaison

La propriété de terminaison permet de s'assurer que, partant de l'état initial consistant en un seul jeton dans la place i , il est toujours possible d'atteindre l'état pour lequel il y a au moins un jeton dans la place f . Formellement : $M_f \in [M_i]$

Les techniques de model checking requièrent une expression précise et non ambiguë des propriétés, décrites en formules logiques, à examiner.

En utilisant LTL, la terminaison implique que l'état M_f est éventuellement atteint. Par conséquent, en définissant une proposition nommée *term*, cette condition peut être traduite en la formule LTL suivante :

$$\boxed{\diamond term}$$

Notons que la proposition *term* devrait être définie dans la spécification Promela du WF-net considéré comme suit :

$$\boxed{\#define term (PL[indice_de_f] >= 1)}$$

4.4.1.2 Terminaison propre

Un processus workflow consiste en un ensemble d'activités ordonnées. Il possède un début défini et termine après l'exécution d'un nombre fini d'activités. La terminaison propre assume qu'au moment un jeton est produit dans la place f , toutes les autres places devraient être vides.

Formellement : $\forall M \in [M_i] : M(f) \geq 1 \Rightarrow M = M_f$;

Raisonnant en LTL, la terminaison propre signifie qu'à chaque état de chaque exécution possible, "si f est marquée" alors "toutes les places sauf f sont vides et f contient exactement un jeton". Ce raisonnement nous a permis de formuler la terminaison propre en LTL comme suit :

$$\boxed{\square(term \rightarrow prop)}$$

Les propositions *term* et *prop* sont définies en Promela comme suit :

$$\boxed{\begin{array}{l} \#define term (PL[indice_de_f] >= 1) \\ \#define prop \quad (\bigwedge_{i=0}^{i=|P|-2} (PL[i] == 0) \ \&\& \ PL[indice_de_f] == 1) \end{array}}$$

4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

Nous convenons que le marquage de la place f est enregistré dans le dernier élément du tableau PL , alors $indice_de_f$ est égal à $|P| - 1$.

4.4.1.3 Absence d'inter-blocages

La troisième sous-propriété de la cohérence assume qu'il n'y aura aucun inter-blocage lors de l'exécution du processus. Commenant avec un jeton dans la place initiale i , il est possible d'exécuter n'importe quelle tâche en suivant un chemin approprié par le WF-net, c'est à dire il n'y aura pas de transitions mortes.

Formellement : $\forall t \in T, \exists M \in [M_i] : M[t]$.

Cette propriété exige que toute transition doit être franchie au moins pour une exécution du workflow. En d'autres termes, pour prouver la présence d'inter-blocage, il suffit de montrer l'existence d'une transition qui n'a jamais été franchie dans toutes les exécutions possibles du processus. Le problème rencontré ici est que SPIN model checker ne permet pas d'exprimer ce genre de propriété. Cependant, nous pouvons ramener la vérification de l'absence d'inter-blocage d'un WF-net (qui correspond à la quasi-vivacité) à la vérification de la vivacité de la fermeture de ce WF-net (le WF-net augmenté d'une transition qui relie f à i). Ceci revient à vérifier que toutes les transitions du réseau fermeture ont été franchies au moins une fois. Ainsi, il suffit de vérifier la formule LTL suivante :

$$\diamond live$$

avec $live$ une proposition définie en Promela comme suit :

$$\#define live (\&\&_{j=0}^{j=|T|} (TR[j] >= 1))$$

Ici, le compteur j varie de 0 à $|T|$ puisque le nombre de transitions du WF-net a été augmenté de 1 pour pouvoir vérifier que éventuellement toutes les transitions du réseau fermeture ont été franchies.

4.4.1.4 Options à adopter en SPIN

– *Équité faible (Weak Fairness)*

L'équité est généralement nécessaire pour prouver la vivacité (les propriétés de la forme $\diamond p$). En fait, pour prouver une certaine forme de progrès, la progression doit être possible. L'équité est concerné avec une résolution équitable du non-déterminisme. Elle annonce que chaque instruction est exécutée infiniment souvent dans chaque exécution.

Il existe deux notions bien connues de l'équité : équité forte et équité faible. L'équité forte exige que si une activité est infiniment souvent activée (pas nécessairement toujours), elle doit être exécutée infiniment souvent. L'équité faible annonce que si une activité est continuellement activée (pas de mise hors service temporaire), elle doit être exécutée infiniment souvent.

4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

Le model checker SPIN ne garantit pas la vérification des propriétés en vertu de l'hypothèse d'équité forte. Equité forte signifie que dans chaque séquence infinie de franchissement chaque transition est éventuellement souvent franchie. Avant de vérifier une propriété, nous devons donc préciser que nous considérons l'équité faible. En fait, si nous n'avons pas spécifié que nous adoptons l'équité faible, il y'aura une exécution non terminée dans laquelle le code est exécuté indéfiniment. Ainsi, la propriété citant que "le programme se termine toujours" est vérifiée si et seulement si l'hypothèse de l'équité faible est adoptée.

– *Acceptation (Acceptance)*

La vérification des formules temporelles de la forme $\diamond p$ est également réalisée sous l'hypothèse de "l'équité faible". De plus, elle doit être lancée dans un mode dit "recherche des cycles acceptants".

Par exemple, pour vérifier efficacement $\diamond live$, SPIN génère un never claim pour la négation de cette formule qui est $\neg(\diamond live)$ comme suit :

```
never { /* !(<>live) */
accept_init:
T0_init:
    if
    :: (! ((live))) -> goto T0_init
    fi;
}
```

Pour ce never claim, il y a deux possibilités : Si *live* est toujours vraie, SPIN est bloqué dans l'instruction *if* car il n'y a pas d'autres alternatives à $!(live)$. Une exécution qui contient un état dans lequel *live* est vraie a été trouvée, et donc cette exécution ne peut pas rendre $!live$ fausse. Ici le vérificateur ne retourne aucune erreur.

Si, par contre, *live* ne devient jamais vraie, SPIN va boucler toujours dans le never claim ; il exécute d'une façon répétitive l'instruction *if* étiquetée par *accept_init*. Une exécution d'un never claim qui passe infiniment souvent par une instruction étiquetée par *accept* est dite "cycle acceptant". Si une vérification trouve un cycle acceptant, ce cycle acceptant montre l'existence d'une exécution infinie dans laquelle *live* est toujours vraie, donc $!live$ est vraie. Par dualité, c'est l'équivalent de $! \langle \rangle live$, alors $\langle \rangle live$ est fausse. Ce cycle acceptant est enregistré dans un fichier de trace dit *trail* afin de pouvoir examiner le contre-exemple pour trouver l'erreur.

4.4.2 Formule LTL de la cohérence d'un processus workflow

Après avoir étudié les trois sous-propriétés qu'un WF-net correct doit satisfaire à savoir la terminaison, la terminaison propre et l'absence d'inter-blocages, nous résumons ci-dessous notre méthode de caractérisation de la cohérence.

Etant donné un WF-net $N = (P, T, F)$, nous devons d'abord le traduire en un modèle

4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

Promela comme expliqué dans la section 4.3. Après, nous devons définir trois propositions reliées respectivement aux terminaison, terminaison propre et absence d'inter-blocages

```
#define term (PL[|P| - 1] >= 1)
#define prop ( &&_{i=0}^{|P|-2} (PL[i] == 0) && PL[|P| - 1] == 1)
#define live ( &&_{j=0}^{|T|} (TR[j] >= 1)) (pour N*)
```

Finalement, nous caractérisons la propriété de cohérence d'un WF-net en LTL par la définition suivante :

Définition 4.1 *Un WF-net N est cohérent si et seulement si les trois formules LTL qui suivent sont satisfaites :*

1. $\diamond term$
2. $\Box(term \rightarrow prop)$
3. $\diamond live$ (pour N^*)

Exemple 4.1 *Gestion de la chaîne logistique*

Nous considérons dans cet exemple le processus workflow représentant la gestion de la chaîne logistique donné dans la figure 4.7.

La spécification Promela du WF-net de la figure 4.7 est donnée dans la figure 4.8.

Pour vérifier la cohérence de ce processus de gestion de la chaîne logistique, nous exprimons cette propriété en LTL par les deux formules suivantes :

$$\diamond term \ \&\& \ \Box(term \rightarrow prop)$$

$$\diamond live \text{ (signifie que } N^* \text{ doit être vivant)}$$

Ces formules sont satisfaites et par conséquent le WF-net de la figure 4.7 est cohérent.

Exemple 4.2 *Demande d'un crédit*

Nous considérons dans cet exemple un processus workflow représentant le traitement d'une demande de crédit (figure 4.9).

Nous avons spécifié en Promela le processus workflow de la figure 4.9 et vérifié les sous-propriétés de la cohérence relatives au même processus. Les imprimés écrans des figures 4.10 et 4.11 montrent le modèle Promela de ce processus ainsi que les résultats de vérification de ces sous-propriétés.

4.4.3 Cohérence faible

Le concept de la cohérence faible est introduit par Martens [83] dans le contexte de processus d'entreprise distribué. On peut permettre à un processus de se comporter proprement sans avoir à exécuter toutes les tâches. Le WF-net possède une cohérence faible,

4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

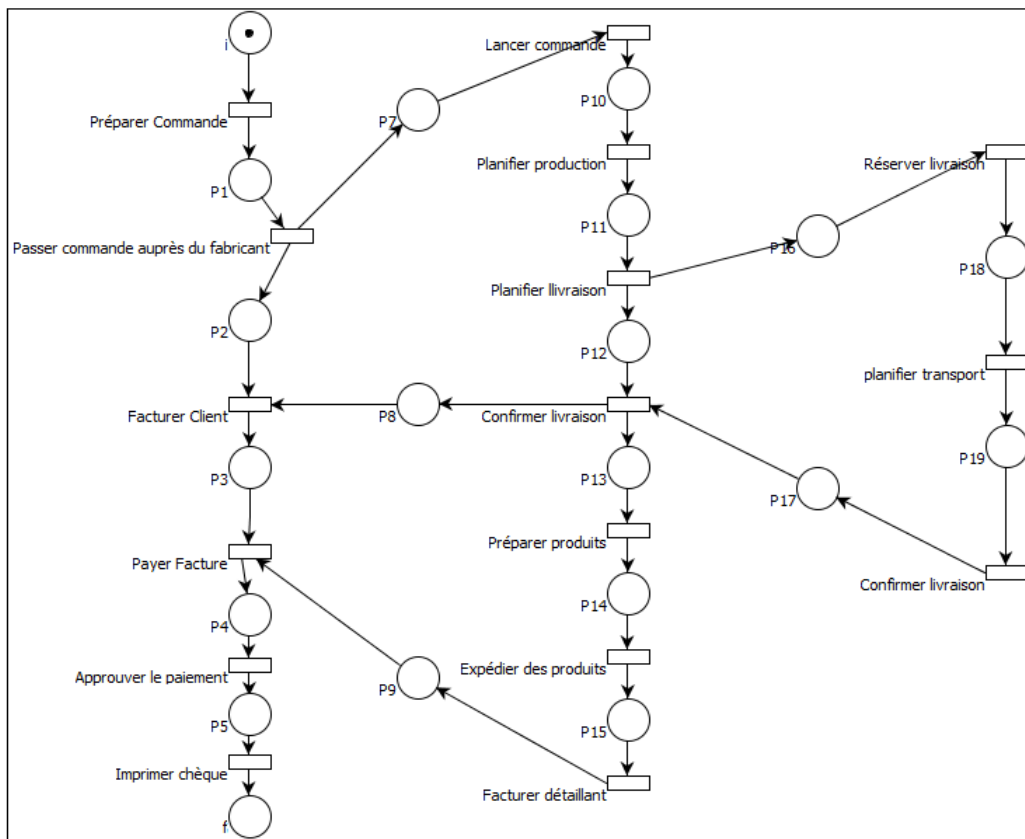


FIGURE 4.7 – Un WF-net résumant la gestion de la chaîne logistique

car l'état final f peut être atteint à partir de l'état initial i , et quand f est atteint, il ne reste aucun jeton dans les autres places.

4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

```

#define P 20
#define T 16

#define remove1(x)      (x>0) -> x--
#define remove2(x,y)    (x>0&& y>0) -> x--; y--
#define add1(x)         x++
#define add2(x,y)       x++; y++
#define fire(x)         x++

byte PL[P];
byte TR[T];

init
{
  PL[0]=1;
  do
  :: atomic{remove1(PL[0])      -> fire(TR[0]) ; add1(PL[1])      }
  :: atomic{remove1(PL[1])      -> fire(TR[1]) ; add2(PL[2], PL[7]) }
  :: atomic{remove2(PL[2], PL[8]) -> fire(TR[2]) ; add1(PL[3])      }
  :: atomic{remove2(PL[3], PL[9]) -> fire(TR[3]) ; add1(PL[4])      }
  :: atomic{remove1(PL[4])      -> fire(TR[4]) ; add1(PL[5])      }
  :: atomic{remove1(PL[5])      -> fire(TR[5]) ; add1(PL[19])     }
  :: atomic{remove1(PL[7])      -> fire(TR[6]) ; add1(PL[10])     }
  :: atomic{remove1(PL[10])     -> fire(TR[7]) ; add1(PL[11])     }
  :: atomic{remove1(PL[11])     -> fire(TR[8]) ; add2(PL[12], PL[16]) }
  :: atomic{remove2(PL[12], PL[17]) -> fire(TR[9]) ; add2(PL[8], PL[13]) }
  :: atomic{remove1(PL[13])     -> fire(TR[10]) ; add1(PL[14])     }
  :: atomic{remove1(PL[14])     -> fire(TR[11]) ; add1(PL[15])     }
  :: atomic{remove1(PL[15])     -> fire(TR[12]) ; add1(PL[9])      }
  :: atomic{remove1(PL[16])     -> fire(TR[13]) ; add1(PL[18])     }
  :: atomic{remove1(PL[18])     -> fire(TR[14]) ; add1(PL[6])      }
  :: atomic{remove1(PL[6])      -> fire(TR[15]) ; add1(PL[17])     }
  od
}

```

FIGURE 4.8 – Spécification Promela d'un processus de gestion de la chaîne logistique

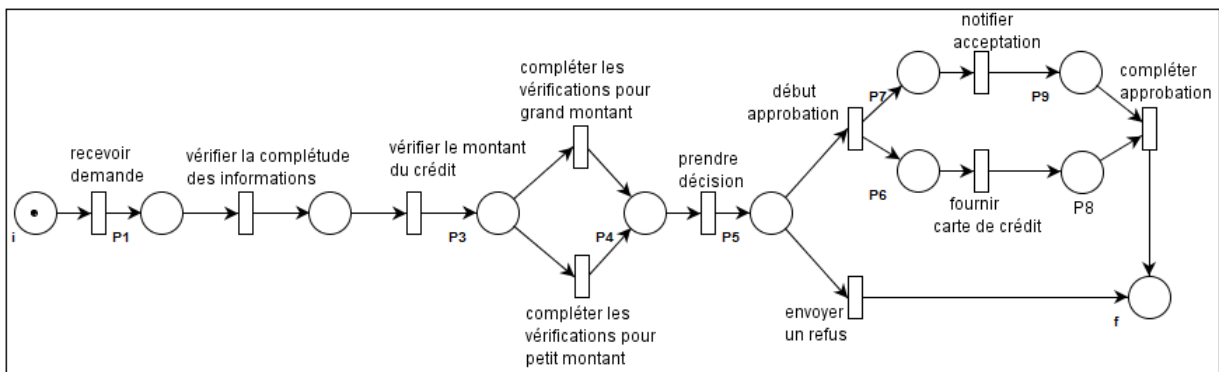


FIGURE 4.9 – Un WF-net résumant une demande de prêt d'un crédit

4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

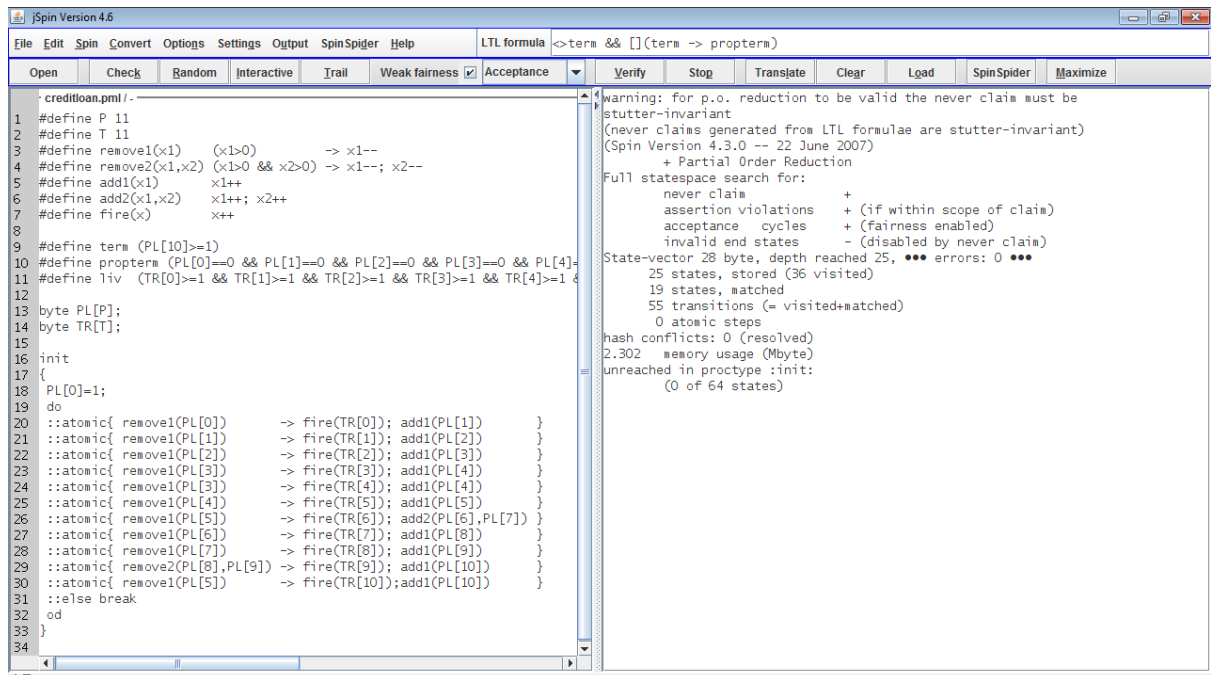


FIGURE 4.10 – Vérification d'un processus de demande de crédit (1)

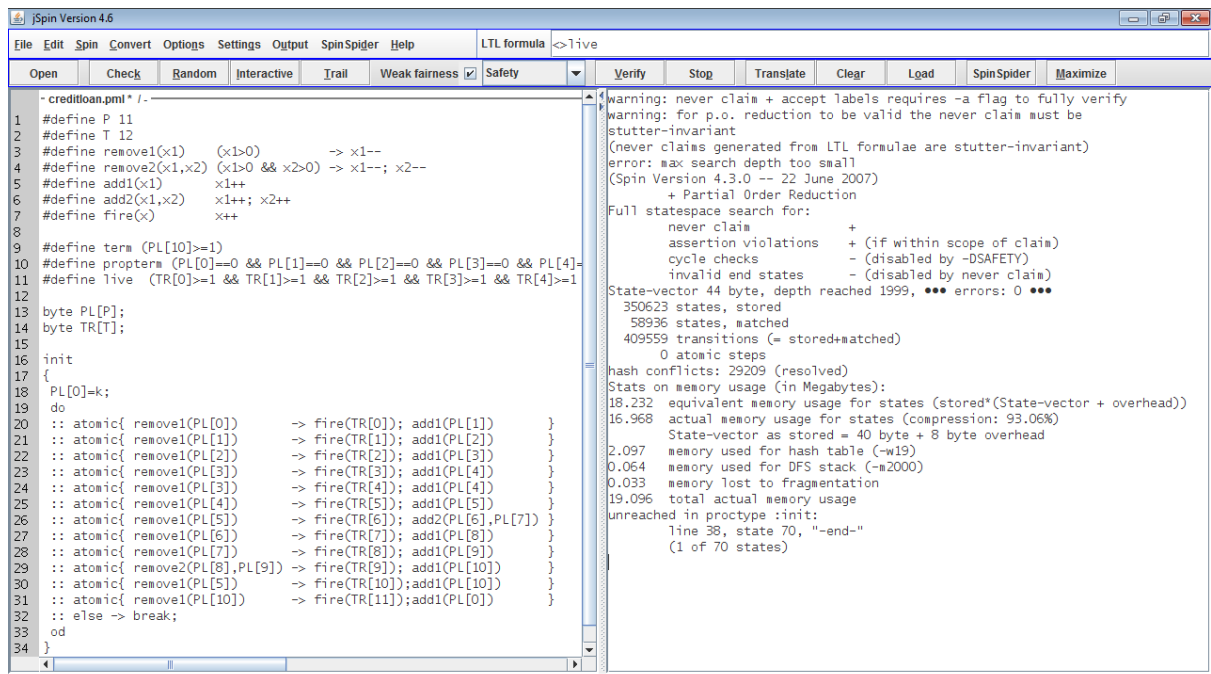


FIGURE 4.11 – Vérification d'un processus de demande de crédit (2)

En utilisant les réseaux de Petri, un workflow net $N = (P, T, F)$ satisfait la cohérence faible si et seulement si :

4.4. FORMULATION DES PROPRIÉTÉS DE COHÉRENCE D'UN WF-NET EN LTL

1. Pour chaque marquage M , accessible de M_i , le marquage final M_f est atteint.
2. Pour chaque marquage accessible M , si $M \geq M_f$ alors $M = M_f$.

A partir des formules LTL définies dans la section 4.4.1, nous pouvons définir la cohérence faible comme suit :

Définition 4.2 *Un WF-net satisfait la cohérence faible si et seulement si les deux formules LTL qui suivent sont satisfaites :*

1. $\Diamond term$
2. $\Box(term \rightarrow prop)$

Exemple 4.3

Considérons le WF-net de la figure 4.12. Il est trivial de noter que le jeton présent dans la place i peut toujours atteindre la place finale et au moment où un jeton est ajouté dans f , toutes les autres places sont vides. Par conséquent, les deux conditions de la cohérence faible sont satisfaites.

La capture d'écran imprimée dans la figure 4.13 présente la spécification Promela de ce WF-net ainsi que le résultat de la vérification de la cohérence faible. "**●●● errors :0 ●●●**" montre que ce WF-net satisfait la cohérence faible.

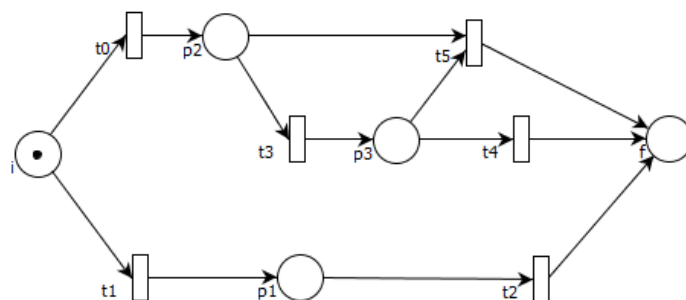


FIGURE 4.12 – Un exemple d'un WF-net non cohérent qui satisfait la cohérence faible

La figure 4.14 montre que le WF-net de la figure 4.12 n'est pas cohérent. Un avantage principal de la vérification par model checking est la possibilité de visualiser un contre-exemple quand une formule est violée.

Pour notre exemple, la figure 4.15 montre la première exécution dans laquelle l'état final ne garantit pas la cohérence du WF-net. Dans cette exécution, affichant les différentes valeurs des variables manipulées (tableaux PL et TR), nous pouvons examiner la dernière ligne donnant les valeurs finales des éléments du tableau. Les valeurs de $TR[1]$, $TR[2]$ et $TR[5]$ sont nulles ; ceci prouve que, dans cette exécution, t_1 , t_2 et t_5 n'ont pas été franchies lors de l'exécution de la fermeture du WF-net correspondant. Par conséquent, le WF-net n'est pas cohérent.

4.5. FORMULES LTL DE LA COHÉRENCE DES PROCESSUS WORKFLOW COMPLEXES

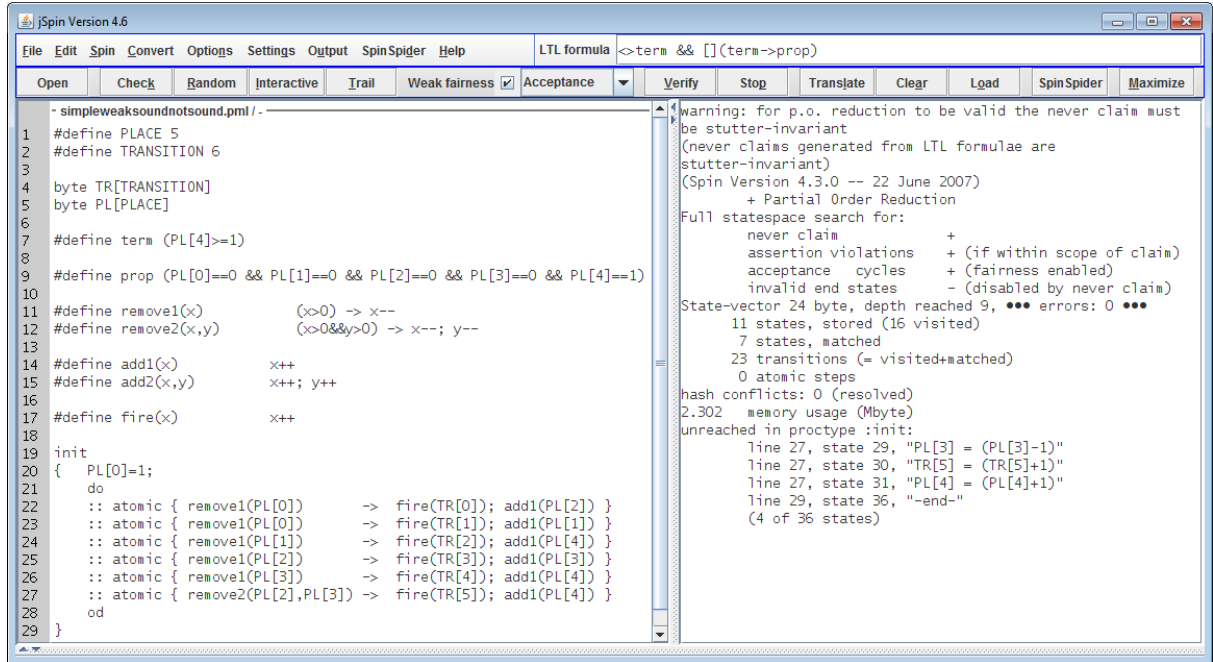


FIGURE 4.13 – Vérification de la cohérence faible d'un exemple de WF-net

4.5 Formules LTL de la cohérence des processus workflow complexes

Les WF-nets traités dans la section précédente traitent un seul cas (instance) de la procédure modélisée (i.e. un seul jeton existe au début dans la place i). Dans cette section, nous considérons les processus workflow modélisant k instances.

4.5.1 k -cohérence des WF-nets modélisant des processus concurrents

La propriété de cohérence des k instances du processus workflow correspond à la propriété de k -cohérence. Cette propriété est une extension de la cohérence qui a été prouvé décidable dans [22, 119]. La valeur k indique le nombre de jetons dans la place d'entrée à l'état initial et donc le nombre d'instances du workflow prêtes à l'exécution. Par conséquent, la k -cohérence consiste à s'assurer de la terminaison propre des k instances avec absence de transitions mortes (voir définition 2.3.1).

Pour vérifier la k -cohérence en utilisant notre méthode basée sur SPIN model checker, nous avons à modifier le tableau PL , son premier élément va être initialisé à k qui doit être déclaré comme constante dans le programme Promela. La définition de la k -cohérence basée sur LTL est la suivante :

4.5. FORMULES LTL DE LA COHÉRENCE DES PROCESSUS WORKFLOW COMPLEXES

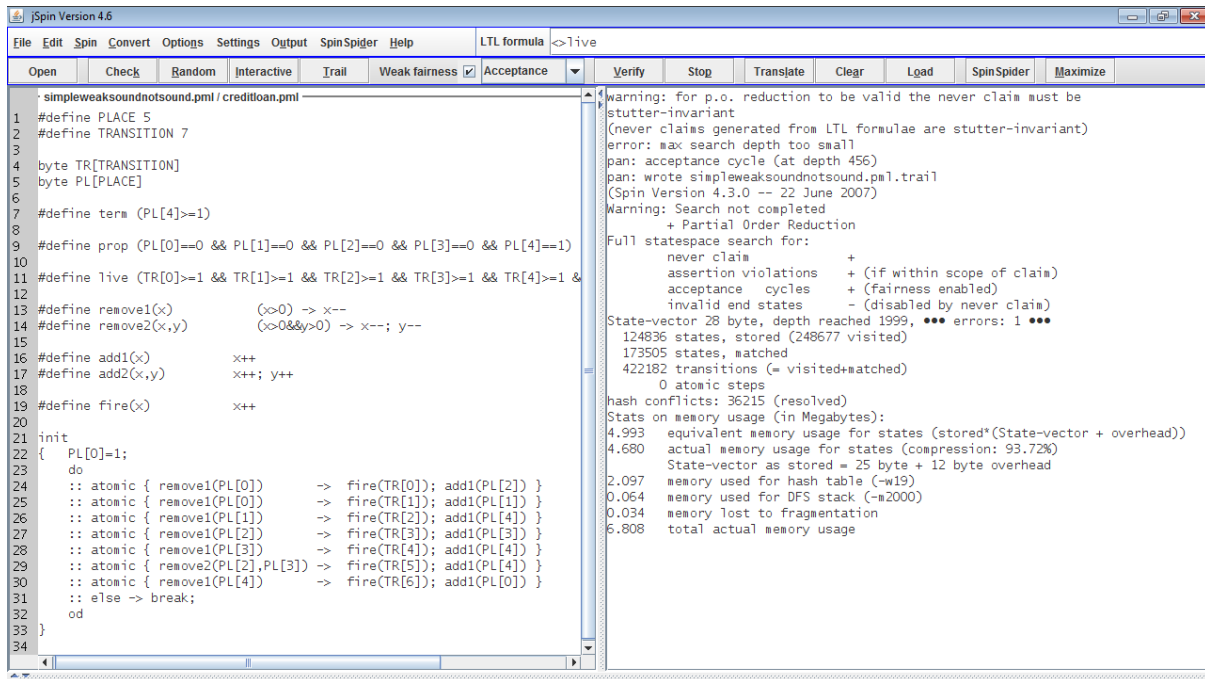


FIGURE 4.14 – Un imprimé écran vérifiant la non cohérence d’un WF-net

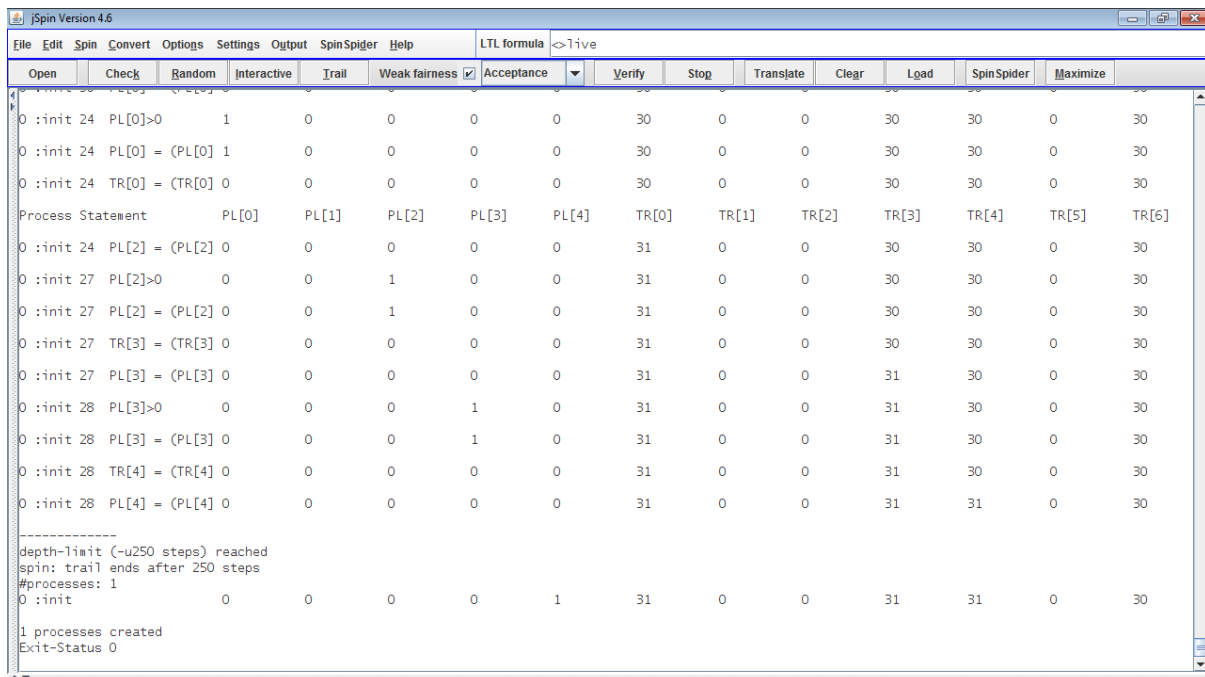


FIGURE 4.15 – Un contre-exemple montrant la non cohérence d’un WF-net

4.5. FORMULES LTL DE LA COHÉRENCE DES PROCESSUS WORKFLOW COMPLEXES

Définition 4.3 *Un WF-net N est k -cohérent si et seulement si les trois formules LTL qui suivent sont satisfaites :*

1. $\diamond kterm$
2. $\square(kterm \rightarrow kprop)$
3. $\diamond live$ (pour N^*)

Avec $kterm$, $kprop$ et $live$ sont des propositions à définir comme suit :

```
#define kterm (PL[|P| - 1] >= k)
#define kprop ( &&_{i=|P|-2}^{i=0} (PL[i] == 0) && PL[|P| - 1] == k)
#define live ( &&_{j=0}^{j=|T|} (TR[j] >= 1) ) (vivacité du réseau fermeture)
```

Exemple 4.4 *Un exemple d'un WF-net non 2-cohérent.*

Considérons l'exemple de la figure 4.16. La description Promela de ce WF-net est donnée dans la figure 4.17. Dans cette description, nous avons montré que l'état initial consiste en 2 jetons dans la place i spécifiant 2 instances modélisées.

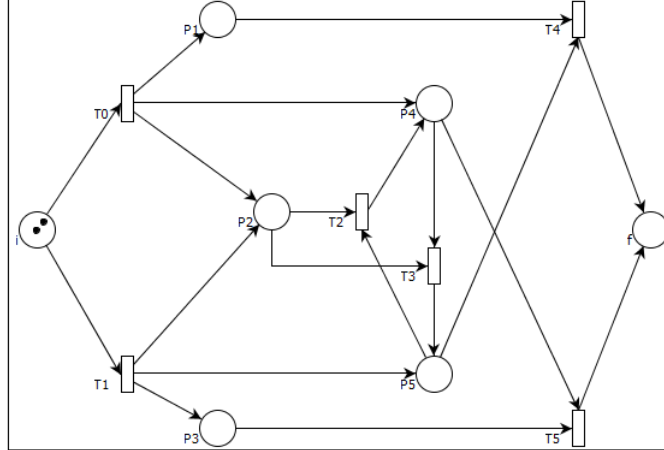


FIGURE 4.16 – Un exemple d'un WF-net non 2-cohérent

Pour vérifier la 2-cohérence du WF-net N de la figure 4.16, nous avons intégré dans le fichier Promela de N les deux propositions suivantes. La troisième proposition sert pour vérifier la vivacité du WF-net N^* :

```
1)#define kterm (PL[6] >= 2)
2)#define kprop (PL[0] == 0 && PL[1] == 0 && PL[2] == 0 && PL[3] == 0
&& PL[4] == 0 && PL[5] == 0 && PL[6] == 2)
3)#define live (TR[0] >= 1 && TR[1] >= 1 && TR[2] >= 1 && TR[3] >= 1
&& TR[4] >= 1 && TR[5] >= 1 && TR[6] >= 1)
```

4.5. FORMULES LTL DE LA COHÉRENCE DES PROCESSUS WORKFLOW COMPLEXES

```

#define PLACE 7
#define TRANSITION 6

byte TR[TRANSITION]
byte PL[PLACE]

#define remove1(x)      (x>0) -> x—
#define remove2(x,y)  (x>0 && y>0) -> x--; y—

#define add1(x)         x++
#define add3(x,y,z)    x++; y++; z++

#define fire(x)        x++

init
{  PL[0]=2;
  do
  :: atomic { remove1(PL[0]) -> fire(TR[0]); add3(PL[1],PL[2],PL[4]) }
  :: atomic { remove1(PL[0]) -> fire(TR[1]); add3(PL[2],PL[3],PL[5]) }
  :: atomic { remove2(PL[2],PL[5]) -> fire(TR[2]); add1(PL[4]) }
  :: atomic { remove2(PL[2],PL[4]) -> fire(TR[3]); add1(PL[5]) }
  :: atomic { remove2(PL[1],PL[5]) -> fire(TR[4]); add1(PL[6]) }
  :: atomic { remove2(PL[3],PL[4]) -> fire(TR[5]); add1(PL[6]) }
  od
}

```

FIGURE 4.17 – La description Promela d’un WF-net non 2-cohérent

Ensuite, nous avons formulé la propriété de 2-cohérence en LTL comme suit :

$$\boxed{\diamond kterm \ \&\& \ \square(kterm \rightarrow kprop)}$$

$$\boxed{\diamond live} \quad (\text{pour } N^*)$$

SPIN retourne une erreur et donc la 2-cohérence est violée. Notons que la 2-cohérence est violée dès la vérification de la k -terminaison (en testant $kterm$ uniquement, SPIN retourne une erreur).

4.5.2 (k,R)-cohérence des WF-nets avec ressources partagées

Introduire des ressources partagées dans la modélisation des processus workflow concurrents est d’une grande importance dans la gestion des systèmes actuels de processus workflow. C’est pourquoi nous proposons d’étendre les résultats trouvés dans les sous-sections précédentes à couvrir la cohérence des processus workflow partageant des ressources nécessaires pour l’exécution de leurs k instances ($k \in \mathbb{N}$).

Pour modéliser proprement l’information sur les ressources dans le marquage initial (WF-net), seules les places ressources (une place par type de ressource) vont être ajoutées. Il est important de noter que le modèle obtenu limite le comportement du WF-net initial.

4.5. FORMULES LTL DE LA COHÉRENCE DES PROCESSUS WORKFLOW COMPLEXES

Un processus workflow sous contraintes de ressources peut être représenté à l'aide d'un RdP appelé WFR-net défini dans la section 2.4.

Les ressources sont durables tout au long de l'exécution du workflow. Une ressource acquise sera par la suite libérée et une ressource libérée a été précédemment requise. Et donc, les ressources doivent être préservées. Les ressources disponibles font partie des jetons prêts à consommer par les instances du workflow.

On désigne par $ki + R$ le marquage initial du WFR-net où R est le marquage initial des places ressources. Une fois toutes les instances du processus ont terminé proprement leur exécution, le marquage final du WF-net souligné (ne considérant pas les ressources) est atteint et les ressources sont libérées. C'est à dire le marquage final d'un WFR-net qui a terminé proprement est $kf + R$.

La propriété de cohérence des WFR-nets modélisant k instances partageant des ressources disponibles avec le marquage R s'appelle (k,R)-cohérence et a été introduite dans la section 2.4. Partant d'un marquage initial qui vaut $ki + R$, le marquage $kf + R$ est atteint tout en s'assurant de l'absence de transitions mortes.

Pour vérifier la (k,R)-cohérence avec SPIN, nous considérons le modèle Promela des WF-nets que nous avons introduit dans la section 4.3 et nous le modifions par modélisation des places ressources et initialisation de leur marquage. Pour cette raison, nous proposons d'ajouter un nouveau tableau qui nous servira pour enregistrer le marquage des places ajoutées. Dans la suite, nous allons illustrer notre approche de vérification de la (k,R)-cohérence des WFR-nets par un exemple.

Exemple 4.5

Nous considérons dans la figure 4.18 un exemple d'un WFR-net modélisant un processus workflow possédant 2 instances prêtes à l'exécution et partageant 6 ressources de type r_1 et une ressource de type r_2 .

Notons que dans l'exemple de la figure 4.18, nous avons des arcs dont la valuation est supérieure à 1. Donc nous avons à étendre les macros *remove* et *add* pour qu'ils puissent être utilisés pour la destruction et la production d'un nombre quelconque de jetons à la fois.

Nous proposons donc de les redéfinir comme suit :

- *removeI*($p_1, p_2, \dots, p_I, n_1, n_2, \dots, n_I$) détruit n_j jeton de chaque place d'entrée repérée par p_j , $j = 1..I$. Ce macro vérifie si ces places d'entrée sont suffisamment marquées ($PL[indice_de_p_j] \geq n_j$, $1 \leq j \leq I$) et une fois cette condition est satisfaite, elle enlève n_j jeton(s) de chaque paramètre p_j .
- *addJ*($p_1, p_2, \dots, p_J, n_1, n_2, \dots, n_J$) crée n_1 jeton(s) dans la place p_1 , n_2 jeton(s) dans la place p_2 , ..., n_J jeton(s) dans la place p_J .

Le modèle Promela correspondant à notre exemple est donné dans la figure 4.19, dans ce code Promela, nous avons initialisé le marquage à $2i + 6r_1 + r_2$ comme suit :

4.5. FORMULES LTL DE LA COHÉRENCE DES PROCESSUS WORKFLOW COMPLEXES

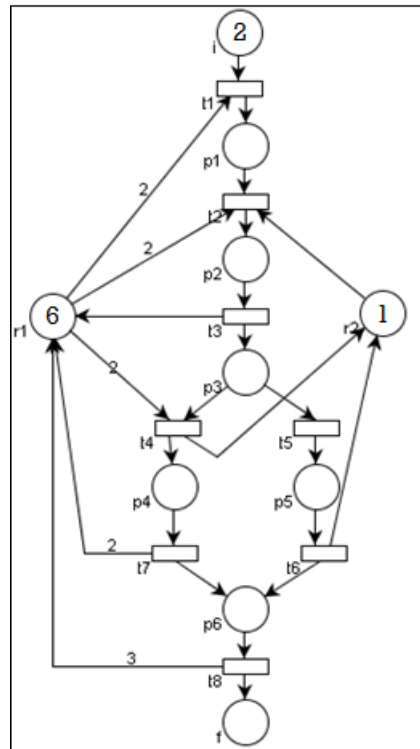


FIGURE 4.18 – Un exemple d'un WFR-net

$$\begin{aligned} PL[0] &= 2; \\ RPL[0] &= 6; \\ RPL[1] &= 1; \end{aligned}$$

Maintenant, pour vérifier la (k,R) -cohérence de ce WFR-net, avec $k = 2$ et $R = 6r_1 + r_2$, il suffit de vérifier d'abord que dans toutes les exécutions possibles, le marquage final est $2f + R : 2f$ pour montrer que les 2 instances ont fini leur exécution et R pour montrer que les ressources ont été libérées.

Pour ce faire, nous avons défini les propositions *term* et *prop* qui vont nous servir pour la vérification de la terminaison et de la terminaison propre respectivement.

$$\begin{aligned} \#define \textit{term} & (PL[7] \geq 2 \ \&\& \ RPL[0] \geq 6 \ \&\& \ RPL[1] \geq 1) \\ \#define \textit{prop} & (\bigwedge_{i=0}^{i=6} (PL[i] == 0) \ \&\& \ PL[7] == 2 \ \&\& \ RPL[0] == 6 \ \&\& \ RPL[1] == 1) \end{aligned}$$

La figure 4.19 montre que la formule LTL $\diamond \textit{term} \ \&\& \ \square (\textit{term} \rightarrow \textit{prop})$ est satisfaite.

4.5. FORMULES LTL DE LA COHÉRENCE DES PROCESSUS WORKFLOW COMPLEXES

Reste donc à vérifier si la troisième propriété de la (k,R) -cohérence est satisfaite ou non. Pour ce faire, nous avons ajouté la proposition *live* dans le fichier Promela correspondant à la fermeture du WF-net de la figure 4.18. Cette proposition est définie en Promela comme suit :

$$\#define\ live\ (\bigwedge_{j=0}^{j=8} (TR[j] \geq 1))$$

La figure 4.20 montre que la propriété LTL $\Diamond live$ est satisfaite et par conséquent le WFR-net de la figure 4.18 est $(2, 6r_1 + r_2)$ -cohérent.

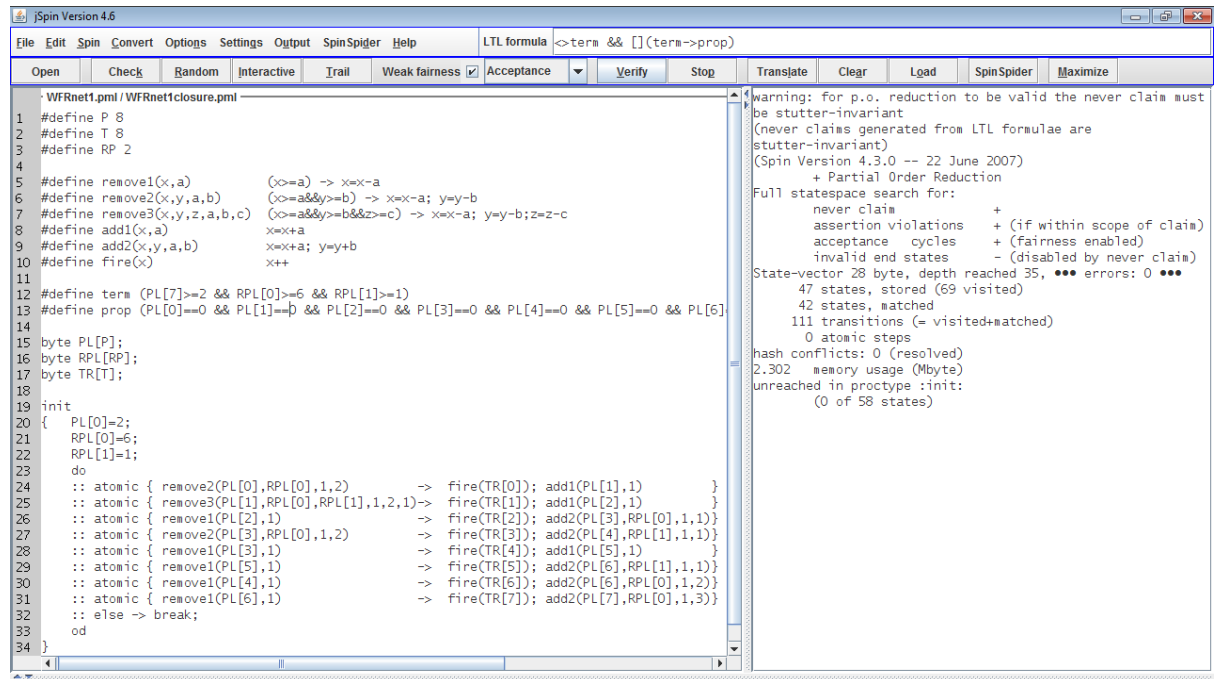


FIGURE 4.19 – Une capture d’écran de la vérification par SPIN d’un WFR-net(1)

4.6. CONCLUSION

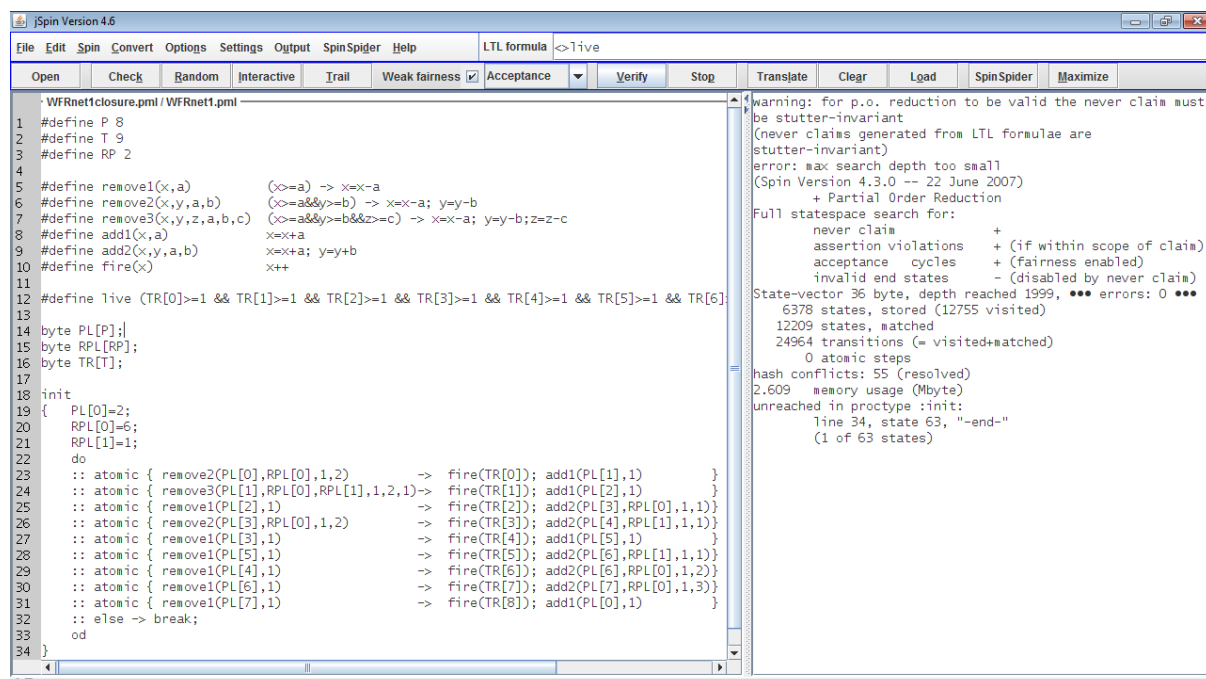


FIGURE 4.20 – Une capture d’écran de la vérification par SPIN d’un WFR-net(2)

4.6 Conclusion

Les processus métier typiques sont généralement constitués d’un certain nombre d’activités interdépendantes qui sont déployées pour atteindre les objectifs de l’entreprise. Les technologies de gestion des processus tels que les systèmes de workflow fournissent un moyen d’automatiser la coordination et la planification des activités qui les composent.

L’exactitude des spécifications workflow est essentielle pour l’automatisation des processus métier. Pour cette raison, les éventuelles erreurs dans les spécifications doivent être détectées et corrigées le plus tôt possible : lors de la spécification.

Dans ce chapitre, nous avons présenté une méthode de validation pour les processus workflow en utilisant des techniques de model checking. En effet, nous avons proposé une approche pour la vérification de certaines propriétés de cohérence des processus workflow à l’aide du model checker SPIN [98].

Tout d’abord, nous avons décrit comment traduire n’importe quel processus workflow, présenté par un WF-net, vers un modèle vérifiable par SPIN. Ce modèle, devant être décrit en langage Promela, définit seulement les matières nécessaires pour vérifier la cohérence du processus donné : le franchissement des transitions et l’évolution des états.

Ensuite, nous avons étudié certaines propriétés de cohérence des WF-nets et présenté la traduction de ces propriétés en formules LTL. Ces propriétés ont concerné d’une part la cohérence des processus workflow qui représentent une seule instance en faisant abstraction des ressources partagées. D’autre part, nous avons étendu les résultats trouvés pour examiner la cohérence des processus workflow en présence de k instances prêtes à s’exécuter

4.6. CONCLUSION

et de certaines ressources partagées différenciées par leurs types.

Les simulations faites sur SPIN en testant les propriétés évoquées sur différents WF-nets ont montré l'efficacité de cette méthode de vérification des processus workflow. En particulier, la notion du contre-exemple généré par SPIN est d'une grande importance vu qu'elle nous aide à trouver rapidement la cause de l'erreur et par la suite la corriger.

4.6. CONCLUSION

Conclusion

1 Résumé des contributions

Les systèmes d'information ont traditionnellement utilisé la modélisation des données comme point de départ, ainsi, les approches pilotées par les données ont dominé le domaine des systèmes d'information. Cependant, au cours de la dernière décennie, il est devenu clair que les processus sont également importants et doivent être supportés de manière systématique. En effet, *Un processus est un ensemble d'activités qui, à partir d'une ou plusieurs entrées, produit un résultat représentant une valeur pour un client interne ou externe* [55].

La gestion de processus métier (Business Process Management (BPM)) est un sujet important vu sa pertinence d'un point de vue pratique, de plus il offre de nombreux défis pour les développeurs de logiciels et les scientifiques. Cela a entraîné une vague de systèmes de gestion de workflow dans le milieu des années 90, visant à l'automatisation des processus métier.

Malgré le nombre important des systèmes de workflow existants, chaque système définit sa propre façon de décrire ses processus. En plus, le nombre des études théoriques relatives aux modèles qu'ils proposent est généralement faible. Plus particulièrement, le développement de nombreux systèmes de gestion de workflow s'est basé sur des techniques et outils de modélisation de processus tout en ignorant la vérification de la cohérence de ces processus, la prise en compte de l'allocation des ressources (partagées) nécessaires à l'exécution de leurs activités, des contraintes temporelles et de la sécurité.

Le manque d'outils de vérification de la cohérence dans un moteur de workflow peut permettre la gestion de processus comportant des malveillances comme un inter-blocage qui ne pourrait être détecté et qui pourrait engendrer des défaillances du système dans une phase où on ne pourrait pas revenir et détecter le problème original. En plus, le fait de ne pas prendre en compte des composantes temporelles et/ou des ressources partagées par exemple pour un système de gestion de workflow peut restreindre le nombre d'utilisateurs de ce système vu que ces types de contraintes sont primordiaux dans les spécifications des processus métier.

Nos principales contributions à ce sujet sont :

- La modélisation et la vérification efficace et optimale des processus workflow qui intègrent ou non des ressources partagées.
- La proposition d'un modèle de workflow temporisé qui permet de spécifier des processus à contraintes de durées des activités ainsi que la proposition de méthodes de

- vérification de leur cohérence.
- La proposition d’un modèle de workflow à contraintes temporelles plus général qui adopte les intervalles de temps pour spécifier les délais dans un processus. Vu la possibilité d’explosion des états accessibles du système, nous avons proposé une méthode d’analyse basée sur un graphe d’accessibilité calculé en utilisant la notion des classes d’états.
- L’implémentation d’une plateforme intégrée
 - de modélisation de processus workflow avec ou sans ressources partagées et intégrant ou non des contraintes temporelles,
 - de vérification (structurelle et comportementale) de la cohérence de ces processus,
 - et essentiellement d’export des définitions de processus, cohérents, dans les formats majeurs acceptés par les systèmes de gestion de workflow existants.
- La proposition d’une méthode basée sur le model checking pour la vérification des processus workflow. Plus particulièrement, nous avons utilisé le model checker SPIN pour décrire le modèle du processus workflow et vérifier si une propriété, exprimée en LTL, est satisfaite ou non.

1.1 Modélisation et vérification structurelle des processus workflow

Pour la modélisation des processus workflow, nous sommes partis d’une sous-classe particulière des réseaux de Petri, dite workflow-net (WF-net) et nous nous sommes intéressés à la vérification de la cohérence d’un processus workflow modélisé par un WF-net. La vérification proposée est basée sur la théorie structurelle des réseaux de Petri. Elle consiste à déterminer des structures particulières qui peuvent être vues comme des patrons de flot de contrôle assurant la propriété de cohérence.

Ces structures sont fortement liées à la propriété de siphon contrôlé soit par invariant, soit par trappe. A l’issue de cette caractérisation, nous avons identifié de nouvelles sous classes de WF-net pour lesquelles la cohérence peut être établie structurellement et résolue de manière efficace. Egalement pour les WF-nets avec ressources partagées (WFR-nets), nous avons donné une caractérisation structurelle qui vérifie efficacement leur cohérence. [19], [20]

Pour que ces résultats théoriques relatifs à la vérification soient utiles dans la technologie de workflow, nous avons dû automatiser le calcul des siphons minimaux dans un WF-net. Pour ce faire, nous avons analysé en profondeur les algorithmes proposés dans la littérature et les structures de données judicieuses qui leur sont associées pour proposer un algorithme optimal pour la vérification de la propriété de siphon contrôlé appropriée à la classe des WF-nets.

L’algorithme que nous avons proposé est constitué de deux phases dont la première consiste à trouver tous les circuits alternés et à contracter chaque circuit trouvé dans une seule place. La procédure de contraction, que nous avons proposée, assure le regroupement des places qui vérifient la propriété suivante : si une de ces places appartient à un siphon minimal alors toutes les places du groupe appartiennent au même siphon.

La deuxième phase consiste dans le calcul des siphons minimaux du réseau contracté. Pour ce calcul nous avons adopté l’approche de Cordone et al. [38], [37] à laquelle nous

avons apporté des modifications et optimisation. L'idée de base de cette approche suit la méthode de diviser pour régner et peut se résumer ainsi : Ayant défini comme problème le calcul des siphons minimaux dans un réseau de Petri, il s'agit d'extraire un siphon générique par élimination des places qui ne peuvent évidemment être contenues dans un siphon, puis dans ce siphon générique, on cherche un siphon minimal Sm et enfin pour chaque place de Sm , on relance la recherche dans tout le réseau privé de cette place.

L'algorithme proposé (réduction du réseau en contractant les circuits alternés et calcul des siphons minimaux) a montré en pratique un gain important dans le temps de calcul par rapport aux résultats trouvés par Cordone et al. L'intérêt majeur de cet algorithme est son aptitude à être appliqué à n'importe quelle sous-classe des réseaux de Petri et non seulement aux WF-nets.

1.2 Modélisation des processus workflow à contraintes temporelles

Après la vérification structurelle de la cohérence des processus modélisés par WF-nets, nous nous sommes intéressés à l'intégration des contraintes temporelles dans la modélisation et la vérification des processus workflow. Après avoir effectué une étude et une classification des différentes extensions des RdPs par le facteur temps, nous avons proposé un modèle temporisé de workflow. Ce modèle baptisé TWF-net (Timed WorkFlow-net) consiste en l'extension des WF-nets par association d'une durée à chaque tâche. Nous avons défini sa sémantique en termes d'états et de transitions entre états et par conséquent de l'espace des états accessibles. Outre la définition formelle du modèle, nous avons défini les propriétés de cohérence et de k-cohérence d'un TWF-net et donné une condition nécessaire et suffisante (en fonction d'autres propriétés également définies) pour la vérification de chacune de ces propriétés [101].

Le modèle de TWF-net est simple en termes de vérification et d'analyse des processus workflow mais il peut s'avérer limité dans certains cas puisqu'il ne permet pas au développeur de processus de définir des bornes inférieures et supérieures d'exécution des activités. C'est la raison pour laquelle nous nous sommes orientés vers l'étude des workflows avec délai d'action, c'est à dire contraints par une date au plus tôt et une date au plus tard pour l'exécution de leurs activités. Nous avons proposé un deuxième modèle qui essaye de répondre à cette question. Pour ce modèle, que nous avons appelé ITWF-net (Interval Timed WorkFlow-net), un intervalle de validité a été associé avec chaque jeton, sur le réseau, cet intervalle étant mentionné sur les arcs en sorties des transitions.

Après avoir défini formellement les ITWF-nets et caractérisé leur comportement dynamique à travers la définition des états et leurs évolutions, nous avons présenté une méthode d'analyse des ITWF-nets basée sur le graphe d'accessibilité. Puis, vu que ce graphe pourrait être très large et même infini, nous avons donné une méthode de réduction du graphe d'accessibilité. Cette méthode basée sur la notion de classes d'états (regroupement des états de même marquage) a nécessité d'introduire quelques modifications dans les conditions de franchissement d'une transition et dans le temps d'activation et de production d'un événement [100].

1.3 Plateforme logicielle de modélisation des processus workflow

Pour concrétiser les contributions résumées ci-haut, nous avons mené en parallèle l'élaboration de notre outil de modélisation et de vérification de processus workflow [99] qui offre actuellement plusieurs fonctionnalités telles que :

- Modélisation de processus workflow sous forme de WF-nets (avec ou sans ressources)
- Simulation de WF-nets
- Vérification structurelle de la cohérence si le WF-net en question appartient à une des classes caractérisées dans le chapitre 3
- Vérification comportementale de la cohérence pour n'importe quelle autre classe de WF-nets
- Calcul de tous les siphons minimaux
- Vérification du contrôle de siphons (par trappe et par invariant)
- Environnement de modélisation des processus workflow à intervalles de temps :
 - Modélisation et simulation d'un ITWF-net
 - Génération du graphe d'accessibilité
- Conversion de et vers PNML, jPdl, XPDl et BPEL.

1.4 Vérification de processus workflow à l'aide du model checker SPIN

La dernière contribution de cette thèse a porté sur la vérification formelle des processus workflow par la méthode du model checking. En effet, nous avons proposé une approche qui se base sur l'utilisation du model checker SPIN pour vérifier les propriétés de cohérence des processus workflow.

Nous avons en premier lieu montré comment décrire le modèle du WF-net à vérifier avec le langage Promela adopté par SPIN. En second lieu, nous avons exprimé les propriétés de cohérence en Logique Temporelle Linéaire (LTL).

Enfin, nous avons invoqué SPIN qui, prend en entrée le modèle du WF-net plus la formule LTL de la propriété, et retourne oui ou non selon que la propriété est vérifiée ou non. Dans le cas où la propriété n'est pas satisfaite, SPIN génère un contre-exemple, et donc des corrections dans le processus en question seront possibles.

Les propriétés exprimées et vérifiées ont concerné aussi bien des WF-nets simples que des WF-nets modélisant plusieurs instances et/ou partageant des ressources.

2 Perspectives de recherche

Le domaine de la BPM ou gestion de processus métier est très émergent de nos jours et évolue rapidement, les fondements théoriques proposés dans le cadre de cette thèse ont permis d'ouvrir plusieurs perspectives que nous résumons ci-après.

2.1 Intégration des ressources lors de la gestion des workflows temporels

Nous proposons d'étudier l'intégration des ressources partagées lors de la modélisation des processus workflow à contraintes temporelles. Dans ce cas, le respect des deadlines

n'est plus monotone. Si par exemple, on augmente les ressources (nous avons déjà vu que la cohérence structurelle peut ne plus être préservée) avec les contraintes de temps, on ne peut pas assurer la préservation du deadline global fixe et donc la quantité optimale de ressources partagées devient un problème à résoudre.

2.2 Préservation de la cohérence lors de la composition des processus

Un deuxième point important est l'étude de la préservation des résultats de la cohérence dans le contexte d'un processus inter-organisationnel assurant la composition entre plusieurs processus.

Un processus d'entreprise BPEL exécute un nouveau Service Web composite en spécifiant ses actions réciproques avec les services existants (appelé des partenaires). Il fournit des constructions pour décrire les processus d'entreprise complexes qui peuvent réagir réciproquement d'une façon synchrone ou asynchrone avec leurs partenaires.

Pour évaluer un Service Web composite, un modèle formel est très utile, parce qu'il facilite l'application et l'automatisation de méthodes de génération d'essai. Nous proposons de développer une procédure de transformation de la spécification BPEL dans les ITWF-nets. Ceci va permettre de modéliser les constructions BPEL de temps et donc de gérer les événements, les exceptions et la terminaison. Cette transformation pourrait aussi être intégrée dans notre outil de modélisation et de simulation des ITWF-nets.

2.3 Workflow Mining

Le workflow mining est un concept très utile puisqu'il peut être utilisé pour créer un feedback pour le modèle de workflow aux circonstances de changement et à la détection des conflits du modèle. Il est relié aux nouvelles tendances de la gestion des processus métier telles que la BPR (Business Process Reengineering), la BI (Business Intelligence), la BPA (Business Process Analysis), la CPI (Continuous Process Improvement), et la KM (Knowledge Management). Le workflow mining paraît plus approprié pour la BPR et donc il est très intéressant dans la mesure où la bonne compréhension des processus existants est vitale pour tout effort de redesign.

Dans ce cadre, nous proposons d'analyser les fichiers historiques d'un processus donné pour générer le schéma d'un workflow capable de supporter les promulgations efficaces du processus. Les informations générées à partir de ces fichiers pourraient être profitables dans la résolution de quelques problèmes tels que l'identification des activités critiques, la caractérisation d'Echec/Succès ou l'optimisation du workflow.

2. PERSPECTIVES DE RECHERCHE

Bibliographie

- [1] W.M.P. van der Aalst et K. van Hee. Workflow Management : Models, Methods, and Systems. The MIT Press, ISBN 0-262-01189-1, 2002.
- [2] W.M.P. van der Aalst et A.H.M. ter Hofstede. YAWL : Yet Another Workflow Language W.M.P. FIT-TR-2002-06, université de la technologie de Queensland, Brisbane, 2002
- [3] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, et A.P. Barros. Workflow Patterns. Technical report, Eindhoven University of Technology, 2002.
- [4] W.M.P. van der Aalst. Workflow Verification : Finding Control-Flow Errors using Petri-net based Techniques. Business Process Management, Lecture Notes in Computer Science, Vol. 1806. Springer-Verlag, pp. 161-183, 2000.
- [5] W.M.P. van der Aalst, G.D. Michelis et C.A. Ellis. Proceedings of Workflow Management : Net-based Concepts, Models, Techniques and Tools. C.A. ed., Eindhoven University of Technology, Eindhoven, the Netherlands, 1998.
- [6] W.M.P. van der Aalst, T. Basten, H.M.W. Verbeek, P.A.C. Verkoulen et M. Voorhoeve. Adaptive Workflow On the interplay between flexibility and support. International Conference on Enterprise Information Systems, 1998.
- [7] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, Vol. 8(1)., pp. 21-66, 1998.
- [8] W.M.P. van der Aalst. Verification of workflow nets. International Conference of Application and Theory of Petri Nets 1997, Lecture Notes in Computer Science, Vol. 1248., 1997.
- [9] W.M.P. van der Aalst. Interval Timed Coloured Petri Nets and their Analysis. International Conference of Application and Theory of Petri Nets, 1993.
- [10] W.M.P. van der Aalst. Timed coloured Petri nets and their application to logistics. PhD thesis, Eindhoven University of Technology, Eindhoven, 1992.
- [11] N.R. Adam, V. Atluri, et W.K. Huang. Modeling and Analysis of Workflows using Petri Nets. Journal of Intelligent Information Systems, Vol. 10(2)., pp. 131-158, 1998.
- [12] Akazi. FlowMind The Business Process Management System. www.flowmind.org, 2001.
- [13] N. Alon, R. Yuster et U. Zwick. Finding and Counting Given Length Cycles. In Proc. Eur. Symp. Algorithms, Lecture Notes in Computer Science, Springer-Verlag, 1994.

- [14] G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, et C. Mohan. Advanced Transaction Models in Workflow Contexts. *International Conference on Data Engineering (ICDE'96)*, pp. 574-581, 1996.
- [15] D. Andreu. *Commande et supervision des procédés discontinus : une approche hybride*. Thèse de Doctorat, Université Paul Sabatier, Toulouse III, 1996.
- [16] F.L. Baccelli, G. Cohen, G.J. Olsder et J. Quadrat. *Synchronization and Linearity : An Algebra for Discrete Event Systems*. John Wiley and Sons, 1992.
- [17] T. Baeyens. *The State of Workflow*. www.jboss.com, 2003.
- [18] S. Bassil. *Workflow Technology for Complex Socio-Technical Systems*. Thèse de Doctorat, Université de Montréal, 2004.
- [19] K. Barkaoui, R. Ben Ayed et Z. Sbaï. *Vérification Paramétrée de la Cohérence des Processus Workflow*. Sixième Colloque Francophone sur la Modélisation des Systèmes Réactifs MSR'07, 2007.
- [20] K. Barkaoui, R. Ben Ayed et Z. Sbaï. *Workflow Soundness Verification based on Structure Theory of Petri Nets*. *International Journal of Computing and Information Sciences (IJCIS)*, Vol. 5(1), pp. 51-61. 2006.
- [21] K. Barkaoui, J.M. Couvreur et K. Klai. *On the Equivalence between Liveness and Deadlock-Freeness in Petri Nets*. ICATPN 05, LNCS 3536, Springer Ed, 2005.
- [22] K. Barkaoui et L. Petrucci. *Structural Analysis of Workflow Nets with Shared Resources*. *Proc. Workflow Management : Net-based Concepts, Models, Techniques and Tools (WFM'98)*, *Computing Science Reports*, Vol. 98/7, pp. 82-95. Eindhoven University of Technology, 1998.
- [23] K. Barkaoui et J.F. Peyre. *On liveness and controlled siphons in Petri nets*. ICATPN 96, LNCS 1091. Springer-Verlag, 1996.
- [24] K. Barkaoui, J. M. Couvreur et C. Dutheillet. *On liveness in extended non self-controlling nets*. *Int. Conf. Application and Theory of Petri Nets*, LNCS 935, Springer-Verlag, 1995.
- [25] K. Barkaoui et M. Minoux. *A polynomial-time graph algorithm to decide liveness of some basic classes of bounded Petri nets*. *Proc. Int. Conf. Application and Theory Petri Nets*, Sheffield, U.K., 1992.
- [26] K. Barkaoui et B. Lemaire. *An effective characterization of minimal deadlocks and traps based on graph theory*. *Proc. Int. Conf. Application and Theory of Petri Nets*, Bonn, Germany, 1989.
- [27] B. Berthomieu et F. Vernadat. *Time Petri Nets Analysis with TINA*. *International Conference on the Quantitative Evaluation of Systems (QEST'06)*, 2006.
- [28] B. Berthomieu et M. Diaz. *Modeling and Verification of Time Dependent Systems Using Time Petri Nets*. *IEEE Trans. on Software Engineering*, Vol. 17(3), pp. 259-273, 1991.
- [29] E. Best et C. Fernández. *Nonsequential Processes*. *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.
- [30] Blizard et D. Wayne. *Multiset theory*. *Notre Dame Journal of Formal Logic*, Vol. 30(1), pp. 36-66, 1989.

- [31] J. Billington, S. Christensen, K. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno et M. Weber. The Petri Net Markup Language : Concepts, Technology and Tools. University of South Australia, Computer Systems Engineering Centre.
- [32] M.B. Blake. An Agent-Based Cross-Organizational Workflow Architecture in Support of Web Services. Proceedings of the 11th International Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE'02), 2002.
- [33] V. Bouchitté. Cours de Graphes et algorithmique des graphes rédigé par Brice Goglin. Ecole Normale Supérieure d'Informatique, 1998.
- [34] F. D. J. Bowden. A Brief Survey and Synthesis of the Roles of Time in Petri Nets. Mathematical and Computer Modelling Vol. 31, pp. 55-68, 2000.
- [35] J. R. Büchi. On a decision method in restricted second-order arithmetics. International Congress on Logic, Method and Philosophy of Science, pp. 1-12, 1962.
- [36] J. R. Burch, E. M. Clarke, K. L. Mcmillan, D. L. Dill et L. J. Hwang Symbolic Model Checking : 10^{20} States and Beyond. International Journal of Information and Computation, Vol. 98, pp. 142-170, 1992.
- [37] R. Cordone, L. Ferrarini et L. Piroddi. Enumeration Algorithms for Minimal Siphons in Petri Nets Based on Place Constraints. IEEE Transactions on systems, man and cybernetics-parts : A systems and humans, vol. 35(6), 2005.
- [38] R. Cordone, L. Ferrarini, et L. Piroddi. Some Results on the Computation of Minimal Siphons in Petri Nets. Proceedings of the 42nd IEEE Conference on Decision and Control, 2003
- [39] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani et A. Tacchella. NuSMV 2 : An OpenSource Tool for Symbolic Model Checking. Proceeding of International Conference on Computer-Aided Verification, 2002.
- [40] L. Clément, P.Sébastien. Schema2XMLs. Rapport de projet, 2005.
- [41] Z. Conghua and C. Zhenyu. Model checking workflow net based on Petri net. Wuhen university journal of natural sciences, Vol. 11(5), 2006.
- [42] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, et S. Weerawarana. Unraveling the Web Service Web : An Introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing, Vol. 6(2), pp. 86-93, 2002.
- [43] T.H. Davenport et J.E. Short. The New Industrial Engineering : Information Technology and Business Process Redesign. Sloan Management Review, pp. 11-27, 1990.
- [44] J. Desel. Teaching system modeling, simulation and validation. 2000 Winter Simulation Conference, 2000.
- [45] M. Diaz. Vérification et mise en œuvre des réseaux de Petri. Hermès Sciences Publication, 2003.
- [46] M. Diaz. Les réseaux de Petri - Modèles fondamentaux. Hermès Sciences Publication, 2001.
- [47] K. Donald, E. Vervaet, J. Grelle, S. Andrews, et R. Stoyanchev. Spring Web Flow Reference Guide. <http://static.springframework.org/spring-webflow/docs/2.0.x/reference/html/index.html>, 2009.

BIBLIOGRAPHIE

- [48] J. Eder, E. Panagos, H. Pezewaunig, et M. Rabinovich. Time Management in Workflow Systems. Proceedings of the Third International Conference on Business Information Systems (BIS'99), pp. 265-280. Springer-Verlag, 1999.
- [49] J. Eder, et W. Liebhart. Contributions to Exception Handling in Workflow Management. in EDBT Workshop on Workflow Management Systems, pp. 3-10, 1998.
- [50] D. Fensel. Ontologies : A Silver Bullet for Knowledge Management and Electronic Commerce. Springer Ed., 2003.
- [51] G. C. Gannod et S. Gupta. An automated tool for analysing Petri nets using SPIN. In Proceeding of International Conference on Automated Software Engineering, 2001.
- [52] D. Georgakopoulos, M.F. Hornick et A.P.Sheth. An overview of workflow management from process modeling to workflow automation infrastructure. Distributed and Parallel Databases, Vol. 3(2), pp. 119-153, 1995.
- [53] U. Goltz et W. Reisig. The non-sequential behaviour of Petri nets. Information and Control, Vol. 57, pp. 125-147, 1983.
- [54] M. Guénon. Gestion des Processus Métiers (BPM) et Workflow, 2006.
- [55] M. Hammer et J. CHAMPY. Le reengineering. Dunod, Paris, 1993.
- [56] M. Hammer et J. CHAMPY. Reengineering the Corporation : A Manifesto for Business Revolution. Harper Business Essentials, 2003.
- [57] D. Harel. State Charts : A Visual Formalism for Complex Systems. Science of Computer Programming, Vol. 8(3), pp 231-274, 1987.
- [58] K. van Hee, N. Sidorova et M. Voorhoeve. Resource-Constrained Workflow nets, Fundamenta Informatica XX, pp. 1-15, 2005.
- [59] K. van Hee, N. Sidorova et M. Voorhoeve. Generalized Soundness of Workflow Nets is Decidable. ICATPN 04, LNCS 3099, 2004.
- [60] T.A. Henzinger, R. Jhala, R. Majumdar, et G. Sutre. Software Verification with Blast. In Proceedings of the 10th SPIN Workshop on Model Checking Software (SPIN), Lecture Notes in Computer Science 2648, Springer-Verlag, pp 235-239, 2003.
- [61] G. J. Holzmann. The SPIN Model Checker, Primer and Reference Manual. Addison-Wesley, 2003.
- [62] G. J. Holzmann. The Model Checker SPIN. IEEE Transactions on software engineering, Vol. 23(5), 1997.
- [63] G. J. Holzmann, Doron Peled et Mihalis Yannakakis. On Nested Depth First Search. In The Spin Verification System, pp 23-32. American Mathematical Society, 1996.
- [64] A. Itai et M. Rodeh. Finding a minimum circuit in a graph. In STOC'77 : Proceedings of the ninth annual ACM symposium on Theory of computing, pp. 1-10, 1977.
- [65] S. Jablonski et C. Bussler. Workflow Management - Modeling Concepts, Architecture and Implementation. International Thompson Computer Press, ISBN 1850322228, 1996.
- [66] JBoss. Le manuel de référence de jPdl. www.jboss.com/products/jbpm/docs/jPdl, 2003.

- [67] L. Jenner. Further studies on timed testing of concurrent systems. Technical Report 4, Institute fuer Mathematik, Universitaet Augsburg, 1998.
- [68] S. Joosten et S. Brinkkemper. Fundamental Concepts for Workflow Automation in Practice. International Conference on Information Systems Development, 1996.
- [69] S. Julia , R. Valette et M. A. Fernandes. Scheduling batch systems using a token player algorithm. IEEE International Conference on Systems Man and Cybernetics (SMC'98), 1998.
- [70] S. Julia. Conception et Pilotage de Cellules Flexibles à Fonctionnement Répétitif Modélisées par Réseaux de Petri. Thèse de Doctorat, Université Paul Sabatier, Toulouse, 1997.
- [71] S. Julia, R. Valette et M. Tazza. Computing a feasible schedule under a set of cyclic constraints. International Conference on Industrial Automation, 1995.
- [72] W. Khansa. Réseaux de Petri P-temporels : Contribution à l'Etude des Systèmes à Evénements Discrets. Thèse de Doctorat de l'Université de Savoie, France, 1997.
- [73] W. Khansa, J-P. Denat et S. Collart-Dutilleul. P-Time Petri Nets for manufacturing systems. In International Workshop on Discrete Event Systems, WODES'96, Edinburgh (U.K.), pp. 94-102, 1996.
- [74] E. Kindler. The Petri Net Markup Language : Concepts, Status, and Future Directions. EKA 2006, Department of Computer Science, University of Paderborn Warburger, Germany, 2006.
- [75] E. Kindler. Using the Petri Net Markup Language for Exchanging Business Processes -Potential and Limitations-. Proceedings of the 1st GI Workshop XML4BPM - XML Interchange Formats for Business Process Management at 7th GI Conference Modellierung, 2004.
- [76] E. Kindler et R. Walter. Message passing mutex. Dans J. Desel, editor, Structures in Concurrency Theory, Workshops in Computing, pp. 205-219, Springer-Verlag, 1995.
- [77] F. Kordon, L. Petrucci. Structure of Abstract Syntax trees for Colored Nets in PNML, version 0.2, 2004.
- [78] J. Li, Y. Fan, et M. Zhou. Timing constraint workflow nets for workflow analysis. IEEE Transactions on Systems, Man, and Cybernetics, part A : Systems and Humans, Vol. 33(2), pp. 179-193, 2003.
- [79] S. Ling et H. Schmidt. Time petri nets for workflow modeling and analysis. Proceedings of the IEEE Transactions on Systems, Man, and Cybernetics, Vol. 4, pp. 3039-3044, 2000.
- [80] O. Marjanovic. Methodological Considerations for Time Modeling in Workflows. Proceedings of the 12th Australasian Conference on Information Systems (ACIS'01), 2001.
- [81] O. Marjanovic et M.E. Orłowska. On Modeling and Verification of Temporal Constraints in Production Workflows. Knowledge and Information Systems, Vol. 1(2), pp. 157-192, 1999.
- [82] O. Marjanovic. Dynamic Verification of Temporal Constraints in Production Workflows. Department of Computer Science and Electrical Engineering, The University of Queensland.

- [83] A. Martens. Analyzing web service based business processes. In Proceeding of International Conference on Fundamental Approaches to Software Engineering, Part of the European Joint Conferences on Theory and Practice of Software, Lecture Notes in Computer Science, Vol. 3442, Springer-Verlag, 2005.
- [84] S. McReady. There is more than one kind of Workflow software. Computerworld, Vol. 2, pp. 86-90, 1992.
- [85] P. Merlin. A study of the Recoverability of Communication Protocols. Ph.D. Thesis, Computer Science Dep., University of California, Irvine, 1974.
- [86] S.D. Nikolopoulos et L. Palios. Hole and antihole detection in graphs. In SODA'04 : Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 850-859, 2004.
- [87] P. Wohed, N. Russell, A.H.M. ter Hofstede, B. Andersson et W.M.P. van der Aalst. Patterns-based evaluation of open source BPM systems : The cases of jBPM, OpenWFE, and Enhydra Shark. Information and Software Technology, Vol. 51, pp. 1187-1216, 2009.
- [88] T. Murata. Petri nets : Properties, Analysis and applications, Proceedings of IEEE Vol. 77(4), 1989.
- [89] Petri Nets World. Online Services for the International Petri Nets Community. www.daimi.au.dk/PetriNets, 2004.
- [90] M. Pezzé et M. Young. Time Petri Nets : A Primer Introduction. Tutoriel présenté au Multi-Workshop on Formal Methods in Performance Evaluation and Applications, 1999.
- [91] C. Ramchandani. Analysis of asynchronous concurrent systems by timed Petri nets. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1974.
- [92] O.R. Ribeiro et J.M. Fernandes. Translating Synchronous Petri Nets into PROMELA for Verifying Behavioural Properties. Proceeding of International Symposium on Industrial Embedded Systems, 2007.
- [93] O.R. Ribeiro, J.M. Fernandes et L.F. Pinto. Model Checking Embedded Systems with PROMELA. Proceeding of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2005.
- [94] S.W. Sadiq. Workflows in Dynamic Environments Can they be managed Computer science and Electrical Engineering. pp. 165-176, 1999.
- [95] P. Senac, M. Diaz, A. Léger et P. de Saqui-Sannes. Modelling logical and temporal synchronisation in hypermedias systems. IEEE Journal on selected Areas in Communications, Vol. 14(1), pp. 84-103, 1996.
- [96] J. Sifakis. Performance Evaluation of Systems using Nets. Lecture Notes in Computer Science, Vol. 84. Springer-Verlag, Berlin, pp 307-319, 1979.
- [97] J. Sifakis. Use of Petri Nets for Performance Evaluation. In : Beilner, H. ; Gelenbe, E. : Modelling and Performance Evaluation of Computer Systems, Measuring, Modelling and Evaluating Computer Systems, pp. 75-93 , 1977.
- [98] Z. Sbaï, A. Missaoui, K. Barkaoui et R. Ben Ayed. On the Verification of Business Processes by Model Checking Techniques. IEEE International Conference on Software Technology and Engineering (ICSTE), 2010.

- [99] Z. Sbaï, K. Barkaoui et R. Ben Ayed. Workflow Modeling and Analysis using Petri Nets. International Conference on Information and Communication Systems, ICICS 2009.
- [100] Z. Sbaï, R. Ben Ayed et K. Barkaoui. Modelling and Analysis of Workflow Processes with Timing Constrained Petri Nets. International Conference on Computing And e-Systems, TIGERA'09, 2009.
- [101] Z. Sbaï, K. Barkaoui et R. Ben Ayed. Sur la vérification de la cohérence de processus workflow temporisés. Dans Nouvelles Tendances Technologies en Génie Electrique et Informatique, GEI'08, pp. 451-460, Ed CPU, 2008.
- [102] Z. Sbaï, R. Ben Ayed et K. Barkaoui. Vers l'analyse des processus d'entreprise jPdI fondée sur les réseaux de Petri. Dans Nouvelles Tendances Technologiques en Génie Electrique et Informatique, GEI'06, pp. 171-181, Ed CPU, 2006.
- [103] A.W. Scheer, S. Borowsky, S. Klabunde et A. Traut. Flexible industrial applications through model-based Workflows. International Conference on Enterprise Integration and Modeling Technology, Springer, pp. 439-448, 1997.
- [104] Y. Thierry-Mieg, S. Baarir, A. Duret-Lutz et F. Kordon. Nouvelles techniques de model checking pour la vérification de systèmes complexes. Génie Logiciel, Vol. 69, pp. 17-23, ISSN 1265-1397, 2004.
- [105] OMG. Unified Modeling Language. www.uml.org.
- [106] F.L. Tiplea et G.I. Macovei. E-Timed Workflow Nets. 2nd International Workshop on Petri Nets and their Applications to Workflow Management, 2006.
- [107] F.L. Tiplea et G.I. Macovei. Timed Workflow Nets. 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, 2005.
- [108] TRANSFLOW Nederland BV. Workflow Patterns - Cosa Workflow Software. Workflow Patterns, 1.0 Released Version, 2003.
- [109] A. Tuguchi, S. Tuoku et T. Watanabe. An algorithm GMST For Extracting Minimal siphon-trap And Its Application to Efficient Computation Of Petri Net Invariants. 2003.
- [110] S. Tanirnoto, M. Yamauchi et T. Watanabe. Finding minimal siphons in general Petri nets, IEICE Trans. Fundam., Vol. E79-A, No. 11, pp. 1817-1824, 1996.
- [111] S. Taoka, K. Takano et T. Watanabe. Extracting minimal siphon/traps of Petri nets and its application to computing nonnegative integer-invariants, IEICE Trans. Fundamentals, Vol. E85-A, No.11, pp. 2447-2452, 2002.
- [112] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. Dans Faron Moller and Graham M. Birtwistle, editors, Proceedings of the 8th Banff Higher Order Workshop, Lecture Notes in Computer Science, Vol. 1043, pp. 238-266, Springer-Verlag, 1996.
- [113] H.M.V. Verbeek, T. Basten et W.M.P. van der Aalst. Diagnosing Workflow Processes using Woflan. The Computer Journal, Vol. 44(4), pp. 246-279, 2001.
- [114] W. Vogler. Timed testing of concurrent systems. Information and Computation. Vol. 101, pp. 149-171, 1995.

- [115] M. Voorhoeve et W.M.P. van der Aalst. Ad-hoc Workflow : Problems and Solutions. DEXA Workshop Conference on Database and Expert Systems Applications, pp. 36-41, 1997.
- [116] B-Y. Wang, S-M. Zhang et Y-L. Zhu. A Workflow Model Based on Time-Extended and Hierarchy-Extended Petri Net. 8th IEEE Conference on Computer Supported Cooperative Work in Design Proceedings, 2003.
- [117] J. Weissenfels, P. Muth et G. Weikum. Flexible Worklist Management in a Light-Weight Workflow Management System. Proceedings of the Workshop on Workflow Management Systems at the Sixth International Conference on Extending Database Technology (EDBT'98), pp. 29-38, 1998.
- [118] WFMC. Workflow management coalition terminology and glossary. Technical Report WFMC-TC-1011, Workflow Management Coalition, Brussels, 1999.
- [119] F.L. Tiplea et D.C. Marinescu. Structural soundness of workflow nets is decidable. Information Processing Letters 96, pp. 54-58, 2005.
- [120] J.M.E.M. van der Werf. EPNML an XML format for Petri nets, 2004.
- [121] S. Yamaguchi, M. Yamaguchi et M. Tanaka. A soundness verification tool based on the SPIN model checker for acyclic workflow nets. In the proceeding of ITC-CSCC, 2008.
- [122] R. Yuster et U. Zwick. Finding even cycles even faster. SIAM J. Discret. Math., Vol. 10(2), pp. 209-222, 1997.
- [123] G. Zacharewicz. Un environnement G-DEVS/HLA : Application à la modélisation et simulation distribuée de workflow. Thèse de doctorat de l'université Paul Césanne Aix-marseille III, 2006.
- [124] M. Zur Muehlen. Workflow-Based Process Controlling : Foundation, Design, and application of Workflow - driven Process information system. 2002.
- [125] Dictionnaire en ligne et moteur de recherche des définitions reliées à l'informatique et à l'internet. www.webopedia.com, Mars 2010.
- [126] L'encyclopédie libre Wikipédia. www.wikipedia.org, Mars 2010.