République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Saad Dahlab Blida1

Faculté des sciences

Département d'informatique

Mémoire Présenté par :

Mouzai Meriem

Tarabet Chahrazed

En vue d'obtenir le diplôme de master

Domaine : Mathématique et informatique
Filière : Informatique
Spécialité : Informatique
Option : Ingénierie de logiciel

Sujet :

Vérification de modèle de processus métier

Soutenu le :

Devant le jury :

M.	Président
M.	Examinateur
M.	Examinateur
Mlle. Boustia Narhimene	Promotrice
Mme. Semar Kahina	Encadrante

Promotion
2014 / 2015

Dédicaces :

A mes chers parents pour leur soutien moral, encouragements et sacrifices

A mes adorables frères : Noureddine, Amine, Lotfi

A ma sœur Khadidja

Pour votre affection et tendresse

A mon binôme Chahrazed pour sa bonne collaboration

A tous mes amis

Au bonheur des plus chers

Je dédie cet humble travail

MOUZAI Meriem

Dédicaces :

Je dédie ce travail à :

A mes très chers parents

Je vous dois ce que je suis aujourd'hui grâce à votre amour, à votre patience et vos innombrables sacrifices.

Que ce modeste travail, soit pour vous une petite compensation et reconnaissance envers ce que vous avez fait d'incroyable pour moi.

Que dieu, le tout puissant, vous préserve et vous procure santé et longue vie afin que je puisse à mon tour vous combler.

A mes très chers sœurs, frères (Samira, Lyes, Zakia, Lila, Mohamed, Yasmina) et à ma belle-sœur Aicha.

Aucune dédicace ne serait exprimée assez profondément ce que je ressens envers vous.

Je vous dirais tout simplement, un grand merci, je vous aime.

A mes deux copines Asma et Imen ainsi qu'à mes très chers ami(e)s Hadjer et Amel et d'autres.

Et particulièrement à ma chère binôme Meriem.

En témoignage de l'amitié sincère qui nous a liées et des bons moments

passés ensemble. Je vous dédie ce travail en vous souhaitant un avenir radieux et plein de bonnes promesses.

J'espère de tout mon cœur que notre amitié durera éternellement.

A mes neveux et nièce car ils sont notre avenir.

A ma grand mère que dieu la protège, a mes cousins et cousines, oncles et tantes.

A tous mes camarades de promotion 2014-2015

Je le dédie à toutes personnes qui me portent leur amitié, leur amour, leur sympathie.

A tous ceux qui m'ont encouragé à aller de l'avant et avaient espoir en ma personne malgré les embuches que j'ai connu.

TARABET Chahrazed

Remerciements :

On tient à remercier dieu tout puissant de nous avoir permis de mener à bien notre mission

On remercie également notre encadreuse Mme. SEMAR Kahina pour l'orientation, la confiance et la patience qui ont constitué un apport considérable sans lequel ce travail n'aurait pas pu être mené à bon escient

Nos remerciements s'étendent également à Mlle. BOUSTIA Narhimene, notre promotrice pour ses bonnes explications qui nous ont éclairé le chemin de la recherche et sa collaboration avec nous dans l'accomplissement de ce modeste travail

Nous tenons à exprimer nos sincères remerciements à tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce travail

Pour conclure, Nous adressons nos remerciements les plus respectueux au jury qui ont accepté d'évaluer notre travail.

Résumé :

La vérification de modèle de processus métier est basée sur des techniques formelles nécessitant en entrée un modèle qui formalise le processus. Un modèle de processus métier consiste en un ensemble de modèles d'activités et de contraintes d'exécution. Ce dernier fera par la suite l'objet d'une vérification. Dans ce cadre, notre procédure de modélisation consiste à transformer les modèles de processus métier, décrit en BPMN, en réseaux de Petri.

Le modèle résultant de l'étape de modélisation consiste en un graphe (réseau de Petri) qui exprime le comportement global du système. Evidemment, les représentations graphiques constituent des spécifications semi-formelles. Or, les techniques de vérification reposent sur le besoin d'une spécification formelle en entrée.

Partant du principe que nous avons, comme donnée, une spécification formelle qui exprime le réseau de Petri représentant le système à vérifier, notre travail consiste à appliquer des techniques de traitement automatique afin d'extraire les données nécessaires du réseau et les exploiter. Et ce, en respectant le principe général de la méthode du model-checking qui prend en entrée la spécification formelle ainsi que les propriétés comportementales du réseau que l'on souhaite vérifier. Ainsi, en explorant exhaustivement l'ensemble des états du système, elle fournit les résultats de la vérification plus les contre-exemples dans le cas où la propriété en question est violée.

Mots-clefs : BPMN, RdP, model-checking, PM, modèle de processus métier, vérification de processus métier, processus collaboratif.

Abstract:

The business process model checking is based on formal techniques requiring, as an input, a model that formalizes the process. A business process model consists of a set of activity models and execution constraints which will then be checked. In this context, our modeling procedure is to transform business process models, described in BPMN, into Petri nets.

The resulting model from the modeling step consists of a graph (Petri net) which expresses the global behavior of the system. Obviously, the graphical representations constitute semi-formal specifications. However, verification techniques are based on the need of a formal specification input.

Assuming that we have, as an input, a formal specification that expresses the Petri net representing the system to be checked, our job is to apply automatic analyses techniques to extract the necessary data from the Petri net and explore them. And this, while respecting the general principle of model-checking method that takes as inputs the formal specification and behavioral properties of the Petri net that we want to check. Thus, by exploring exhaustively all the states of the system, provides the results of the verification as well as counter-examples where the desired property is violated.

Keywords: BPMN, Petri nets, model-checking, business process, business process model, business process verification, collaborative processes.

ملخص:

يستند فحص نموذج العملية المهنية إلى تقنيات رسمية تتطلب كمدخل نموذج يجعل العملية المهنية رسمية.

إن نموذج العملية المهنية يشكل مجموعة من نماذج الأنشطة و قيود التنفيذ. هذا الأخير، يكون محل الفحص فيما بعد. في هذا الإطار، تتوقف الإجراءات التي نقوم بها للنمذجة على تحويل نماذج العمليات المهنية المبينة ب BPMN إلى شبكات بيتري.

يتمثل النموذج الناتج عن مرحلة النمذجة في رسم بياني الذي يعبر عن السلوك العام للنظام. بطبيعة الحال تشكل الرسومات البيانية مواصفات شبه رسمية، لكن تستند تقنيات الفحص على الحاجة إلى وجود مواصفات رسمية.

انطلاقا من المبدأ أن لدينا كمعطى مواصفة رسمية التي تعبر عن شبكة بيتري و التي تمثل النظام الواجب فحصه، عملنا يتمثل في تطبيق تقنيات المعالجة التلقائية من أجل استخراج معطيات الشبكة اللازمة و استغلالها، و هذا مع احترام المبدأ العام لطريقة model-cheking (الفحص النموذجي) التي تتخذ كمدخل المواصفة الرسمية و كذا الخصائص السلوكية للشبكة التي نريد فحصها. و هكذا و باستكشاف شامل لجميع حالات النظام،فهي تقوم بتزويد نتائج الفحص مع ضد الأمثلة في حالة ما إذا كانت الخصائص المعنية مخترقة.

الكلمات المفتاحية: BPMN، شبكات بيتري، الفحص النموذجي، نمج العمليات المهنية، فحص العمليات المهنية، العمليات الإشتراكية.

Table des matières

Chapitre 3 : Choix de la méthode à utiliser

Liste des figures

Liste des tableaux

Introduction générale

1 - Introduction :

Les entreprises actuelles connaissent des changements multiples aussi bien dans les formes d'organisation que dans leurs façons de concevoir et de produire. Ceci est dû aux nouveaux critères de compétitivité imposés par un marché en évolution continue conduisant à une concurrence de plus en plus rude. La mondialisation du marché, les cycles de vie des produits de plus en plus courts, le besoin accru de flexibilité et des changements fréquents de techniques et de technologies ont obligé les entreprises à se préoccuper plus sérieusement de leur modélisation afin de faire face à tous ces facteurs complexes. Ainsi, la naissance des nouvelles technologies de l'information, a poussé les entreprises à informatiser l'ensemble des activités au tour de leur processus métier. En effet, un fonctionnement efficace des organisations, impose de s'appuyer sur des processus métier robustes, et adaptés à leurs activités. La définition et l'exécution de ces processus nécessitent respectivement un modèle et des outils pour permettre la collaboration, la définition, le déploiement, l'exécution, et le contrôle des processus. Toutefois, la modélisation des processus métier est une tâche complexe qui concerne la représentation et la spécification des différents aspects des opérations des entreprises, fonctionnels, informationnels, ressources et organisationnels.

Aujourd'hui, les différentes organisations collaborent entre elles via des processus métier complexes. A la détection d'ambiguïté dans cette interaction, il est nécessaire de passer par une vérification de cette dernières. La détection des erreurs à l'étape de la modélisation est très importante dans le sens où elle détermine à l'avance si un modèle de processus présente certains comportements non souhaitables, c'est pourquoi la vérification de modèle est primordiale.

2 - Contexte et terminologie :

Un processus métier (ou un processus d'affaires - en anglais *Business Process* -) est défini comme un enchaînement partiellement ordonné, d'exécution d'activités qui s'appuient sur un ensemble de tâches et du savoir faire d'une entreprise pour produire une valeur ajoutée aux clients. Le Workflow Management Coalition (WfMC) définit un processus métier comme: « *un ensemble d'une ou plusieurs procédures ou activités liées entre elles pour réaliser collectivement un objectif ou une politique métier en définissant les rôles et les interactions fonctionnelles au sein d'une structure organisationnelle* ». Ces processus sont des processus opérationnels qui décrivent l'ordre d'exécution des activités principales de l'entreprise en transformant, à l'aide de moyens techniques et humains, les éléments d'entrée

en éléments de rendement pour atteindre un objectif métier ou stratégique. Les processus métier sont le patrimoine de l'entreprise ; autrement dit, un processus métier est un enchaînement d'activités qui prend un input (de n'importe quelle forme) lui rajoute de la valeur à l'aide de ressources et fournit un output (produit /service) répondant aux objectifs de l'entreprise. Il est déclenché par des événements internes ou externes de l'entreprise. Il peut être décomposé en sous-processus et communiquer avec d'autres processus. Un processus métier est donc considéré comme un ensemble de relations logiques entre un groupe d'activités incluant des interactions entre partenaires sous la forme d'échange de données pour fournir une valeur ajoutée aux clients.

2.1- ***Modèle de processus métier :***

La modélisation des processus métiers et de leur organisation permet d'obtenir un modèle de processus métier qui est aujourd'hui considérée comme un préalable nécessaire à la conception d'un système d'information organisationnel. Les concepteurs y définissent, d'une manière abstraite ou détaillée, les processus métier ou redéfinissent un processus existant dans le but de l'améliorer. Elle permet de formaliser le fonctionnement précis d'une organisation en utilisant un langage standard et aisément compréhensible [1].

3 - ***Présentation du sujet :***

Le travail présenté dans ce mémoire, a été menée dans le cadre du projet SCIO-Web Social (Service de Collaboration Inter organisationnelle basée web social) du CDTA.

Ce projet se devise en trois principales parties:

• ***Première partie :*** L'exploitation des réseaux sociaux professionnels pour la recherche et la sélection des meilleurs partenaires de collaboration ainsi que la modélisation des contrats de cette collaboration.

• ***Deuxième partie:*** Modélisation des processus de collaboration inter-organisationnelle, en utilisant l'ingénierie des connaissances.

• ***Troisième partie:*** La mise en œuvre de ce processus de collaboration sur le Cloud.

Notre travail se situe dans la deuxième partie. Le but de cette partie est d'exploiter les connaissances extraites du réseau social et des partenaires de la collaboration afin de

concevoir un modèle de processus collaboratif inter-organisationnel pour qu'il soit exécuté par la suite dans la plateforme Cloud de la collaboration.

Les erreurs lors de l'exécution dans le Cloud sont très coûteuses et difficiles à réparer, le modèle conçu doit donc être vérifié, à la phase de modélisation, afin de les éviter. C'est pour cela que la vérification des premières décisions, lors de la construction d'un système, sont très importantes. En effet, cette vérification permet de minimiser les coûts et gagner du temps permettant ainsi à l'entreprise d'en bénéficier.

3.1 - ***Organisme d'accueil :***

Le centre de développement des technologies avancées (CDTA) est un établissement public à caractère scientifique et technologique créé en 1982 (au sein du commissariat aux énergies nouvelles). Il a comme mission :

- Mener à bien des actions de recherche scientifique et d'innovation technologique.

- Participer à valorisation et la formation dans les domaines scientifiques et technologiques particulièrement la technologie de l'information, technologies industrielles, la robotique, application et technologie des lasers

Dans le cadre de ses missions, le CDTA participe et organise plusieurs évènements scientifiques à travers le monde. Offre des sujets de stage pour divers étudiants à divers niveaux, offre des sujets de master de recherche et doctorat. Tout en participant à la production de logiciels et de produits technologiques.

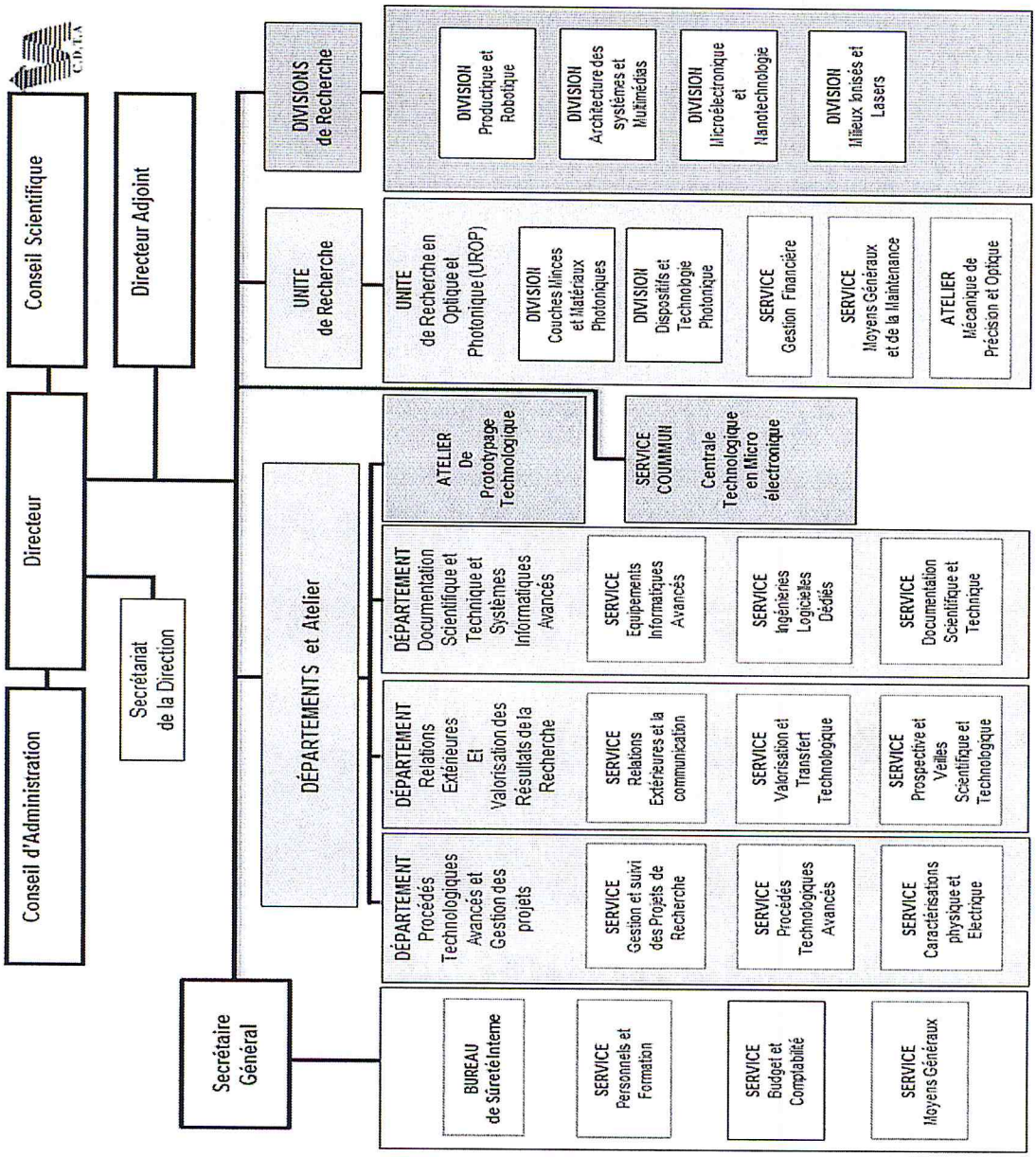


Figure 1 : Organigramme de l'organisme d'accueil.

3.2 - ***Problématique :***

Une entreprise est une organisation qui a besoin de processus métier corrects pour assurer son bon fonctionnement. La collaboration est une nécessité pour l'entreprise, que ce soit en interne, entre ses collaborateurs, ou en externe, avec d'autres entreprises ou des clients. Compte tenu du fait qu'une collaboration se concrétise généralement par le partage de documents, de ressources, de services ou de connaissances. Evidemment, les entreprises communiquent entre elles via des processus métier qui permettent ce partage, ces processus peuvent être contraints par des ressources partagées ou encore par des durées de traitements,

ces contraintes de ressources partagées et de temps augmentent la complexité des processus métier.

De nos jours, ces processus métier, en particulier les processus métier collaboratifs, sont devenus de plus en plus complexes et les modèles qui les modélisent sont désormais de taille très importante, ce qui rend leur vérification une tâche très difficile, voire impossible. Ceci constitue le besoin primordial d'une vérification automatique.

3.3 - ***Objectifs :***

Le travail à réaliser consiste à vérifier des modèles de processus métier dans le cadre d'une collaboration inter-organisationnelle, c'est-à-dire des modèles d'interactions entre plusieurs processus complexes.

La vérification de leur comportement global au moment de la conception, prévoit si un modèle de processus énonce certaines incohérences. En effectuant cette vérification au moment de la conception, il est possible d'identifier des problèmes potentiels et, le cas échéant, le modèle peut être modifié avant son exécution. Ainsi, une analyse des modèles de processus lors de la conception peut améliorer la fiabilité des systèmes.

La procédure à accomplir consiste à transformer les processus métier en un modèle. Le modèle résultant devra, par la suite, être analysé en vérifiant le comportement global et ainsi certaines propriétés comportementales du processus en question. Ceci devra être fait-en suivant un certain enchainement d'étapes qu'on peut organiser en deux principales parties :

- ***Transformer les processus métier en réseaux de pétri :*** ce qui va nous permettre d'obtenir le modèle de réseau de pétri, en effet, beaucoup de travaux ont montré l'intérêt d'utilisation des réseaux de pétri pour la modélisation des processus métier. Cette transformation devra être faite manuellement, car il n'existe malheureusement pas d'outil permettant de transformer d'une manière fiable les processus métier (complexes surtout) en réseaux de pétri.
- ***Vérification du comportement global du modèle du processus en interaction :*** une fois le modèle obtenu, on peut alors vérifier ses propriétés comportementales. Il est bien évident qu'avant tout, on doit faire un certain choix portant sur la technique de vérification de modèle, et ce en se basant sur une étude (état de l'art sur la vérification des processus) afin d'aboutir à l'outil souhaité.

4 - ***Organisation du mémoire :***

Ce mémoire est composé d'une introduction générale suivie de quatre chapitres.

Le premier chapitre porte sur la modélisation des processus métier. Nous y présentons les méthodes de modélisation utilisées au cours de ce travail. Nous verrons, dans un premier temps, une brève présentation sur BPMN. Par la suite, nous exposons les modèles de réseau de Petri et leurs avantages sur BPMN. Enfin, nous aborderons un travail qui traite la transformation d'un modèle BPMN vers un réseau de Petri.

Le deuxième chapitre est consacré à une étude comparative des différentes méthodes et techniques de vérification de modèle de processus métier ainsi que leurs principes de fonctionnement afin de choisir la méthode la plus adéquate.

Le troisième chapitre est essentiellement dédié au choix de l'approche de vérification à utiliser qui consiste à la combinaison de la méthode du model-checking et du traitement automatique de la spécification formelle.

Dans le dernier chapitre, nous passons à la mise en œuvre de la solution retenue.

Puis une conclusion générale viendra clôturer ce mémoire.

Enfin, deux annexes sont associées :

- Aperçu du standard BPMN.
- Concept général des réseaux de Petri.

Chapitre I
Modélisation des processus métier

V.3.4 - ***Problèmes de transformation due à la spécification de BPMM :***

A - ***Modèle avec plusieurs évènements de début :*** la spécification de BPMN déclare que :

1. Chaque évènement de début est indépendant, une instance de processus devra être générée lorsque ce dernier est déclenché.
2. Si les attributs de l'activité succédant les évènements de début précisent que l'activité doit attendre toutes les entrées, alors tous les évènements de départ devront être déclenchés avant le début du processus.

Cependant, un examen plus approfondi des attributs associés aux activités montre qu'aucun d'entre eux permet de modéliser qu'une activité doit attendre toutes les entrées. Il est donc suggéré que la norme BPMN devrait clarifier les combinaisons autorisées des événements de début qui peuvent survenir dans le contexte d'une exécution d'un modèle de processus avec plusieurs événements déclencheurs. La transformation vue précédemment, appliquée sur un tel modèle résultera un réseau de Petri avec plusieurs places sources, il peut toujours être vérifié en utilisant des techniques de réseau de Petri. Or, il faut établir le marquage initial à partir duquel l'analyse qui doit être réalisée.

B - ***Achèvement d'instance de processus :*** La spécification BPMN n'indique pas clairement quand l'exécution d'un modèle de processus doit être considérée comme étant achevée. Cela est problématique surtout pour les modèles ayant plusieurs évènements de fin. Est-ce que l'achèvement indique qu'un seul évènement de fin s'est produit ou bien tous ou bien qu'aucune activité n'est active désormais? D'après les études, une instance d'un modèle de processus est achevée lorsqu'au moins une de ses tâches de fin a été exécutée au moins une fois, et il n'y a pas d'autre tâche active pour ce processus. La transformation vue précédemment résultera un réseau de pétri avec plusieurs places puits, il peut toujours être vérifié en utilisant des techniques de réseau de pétri. Cependant, si le réseau de pétri est 1-sauf on peut le traduire en un réseau de pétri avec une seule place puits.

C - ***Exception pour un sous-processus à plusieurs instances*** : est-ce que l'exception interrompt l'instance du sous-processus en question ou bien toutes les instances ? Si la première option est adaptée, on doit résoudre une autre ambigüité, l'instance interrompue, est-elle considérée comme achevée ? On sait seulement qu'une activité multi-instance qui invoque un sous-processus est achevée si toutes les instances qu'elle produit sont achevées. Dans le cas où l'on n'a pas un réseau 1-sauf, il peut arriver que deux instances d'un sous-

processus s'exécutent simultanément, si l'activité qui les invoque s'exécute une seconde fois au cours de l'exécution de la première instance, alors un appel au sous-processus qui va exécuter une seconde instance simultanément sera découlé. Aussi, si une exception est lancée par une instance et est capturée par un gestionnaire d'exception attaché à l'activité d'invocation, il est difficile de savoir si cette exception n'affecterait que cette instance de sous-processus, ou toutes les instances de sous-processus engendrées par l'activité d'invocation.

D - ***Passerelle complexe (OR-join)*** : la spécification BPMN déclare qu'une passerelle complexe doit attendre tous les jetons qui ont été produit en amont. Le flux continuera quand les jetons arrivent de tous les arcs entrant. Cependant, si cette passerelle est dans un cycle, ceci peut nous mener à un cycle vicieux.

VI - ***Conclusion :***

Ce premier chapitre énonce la modélisation des processus métier. Nous avons présenté les concepts fondamentaux de BPMN ainsi que ceux des réseaux de Petri. Ces derniers, sont un formalisme de modélisation des systèmes. Nous avons également décrit un travail qui porte sur la transformation d'un modèle BPMN en un réseau de Petri.

D'après l'étude faite, BPMN est une notation conçue pour la modélisation des processus métier, elle constitue l'une des notations les plus utilisées actuellement pour modéliser les processus métier. Cependant, pour une fin de vérification de ces derniers, la norme a besoin d'être accompagnée d'une méthode de modélisation. Nous avons constaté que les réseaux de Petri constituent un meilleur candidat du fait qu'ils comblent le manque de BPMN. Ainsi, la transformation d'un modèle BPMN en un réseau de Petri nous permettra de bénéficier de la forte expressivité de BPMN et de la fiabilité du modèle offert par un réseau de Petri.

Toutefois, et afin d'entamer la vérification dont il s'agit, il est nécessaire d'étudier les différentes méthodes et techniques de vérification de modèle de processus métier ultérieurement (voir chapitre 2 ci-dessous).

I - *Introduction :*

Un processus métier, étant une unité fondamentale pour une organisation, a besoin d'être formellement défini d'une manière correcte, exposant ainsi le fonctionnement exact de l'organisation. La modélisation des processus métier peut effectivement être très utile dans différents domaines, notamment, dans le cadre d'une réorganisation ou amélioration du système ou, plus important encore, dans le cadre de vérification du fonctionnement correct des processus métier.

Une telle définition se fait à travers un langage (généralement standard) aisément compréhensible par tout type d'utilisateur. En ce qui suit, nous aborderons les méthodes de modélisation qu'on va utiliser au cours de notre travail.

II - *Modélisation de processus métier :*

Il existe en effet, plusieurs méthodes pour modéliser un processus métier. Ces dernières se diversifient dans la manière de définir les différents aspects du processus métier qu'elles modélisent, tels que le comportement, la sémantique . . . etc.

III - *Critères de modélisation :*

Un ensemble de critères a été développé pour évaluer les modèles des processus métier, les modèles doivent être capables de modéliser toutes les complexités des PM, à savoir : le séquencement, le choix, les boucles, les constructeurs de concurrences (fork et join), les timeouts, les deadlines, les fautes et les agrégations. Les modèles doivent avoir aussi [1] :

- Un moyen pour distinguer les rôles et de les affecter aux différentes tâches.
- Une représentation graphique non ambiguë du langage.
- Un model de transaction qui permet la description du comment.
- Une spécification définissant comment les instances du processus vont être déclenchées et identifiées durant leur exécution.
- Un moyen de spécifier les caractéristiques du PM qui peuvent intéresser les utilisateurs externes, telles que la qualité de service et le prix.
- Le langage ne doit pas s'embrouiller dans les détails des protocoles de communication.

Les trois premiers critères sont importants pour la modélisation, les deux suivants sont essentiels pour automatiser l'exécution des processus entre les organisations séparées et qui collaborent entre elles. Le dernier garde le modèle dans le niveau d'abstraction adéquat [1].

IV - *Objectifs de modélisation :*

La modélisation des processus a beaucoup d'avantages pour les entreprises qui cherchent à améliorer leurs performances. En effet elle permet une meilleure compréhension de l'existant en fournissant une documentation du PM et facilite ainsi la communication en utilisant un langage commun. Elle aide à améliorer la situation actuelle en expérimentant et simulant de nouveaux concepts et aussi à automatiser des processus [1].

V - *Modèles de processus métier :*

Nous allons voir dans ce qui suit (V.1 et V.2), les principales méthodes de modélisation de processus métier appropriées pour notre travail en mettant l'accent sur leur définition, le modèle que chacune d'entre elles offre ainsi que les avantages et les inconvénients.

V.1 - *Business Process Modeling Notation (BPMN) :*

Initiée par Business Process Management Initiative (BPMI) qui fusionna en 2005 avec l'Object Management Group (OMG), Business Process Modeling Notation (BPMN) est une représentation graphique pour la spécification des processus métier, cette notation est donc utilisée pour représenter et modéliser un processus métier [2]. BPMN définit un BPD (diagramme de processus métier) basé sur une technique d'organigrammes adaptée pour créer des modèles graphiques d'opérations de processus métier [3].

Un BPD est constitué principalement de quatre éléments graphiques qui se composent eux-mêmes de sous-éléments graphiques, notamment : [3][4][5]

- Les objets de flux (Flow Objects) : évènements, activités et passerelles.
- Les objets de liaison (Connecting Objects) : flux de séquence, flux de message et associations.
- Couloirs (Swimlanes) : pools et lanes.
- Artefacts (Artifacts) : objets de données, groupes et annotations.

V.1.1 - *Avantages de BPMN :*

Cette notation dispose en fait de plusieurs avantages. Pour commencer BPMN est un standard n'appartenant à aucune entreprise donnée, mais à l'institution OMG. La norme est supportée par de nombreux logiciels [6]. En utilisant BPMN, les processus métier sont modélisés d'une manière identique (l'avantage de l'uniformité). Bien entendu, les membres d'une équipe d'un projet ont souvent travaillé ensemble d'une façon à avoir leur propre norme de représentation, et si plusieurs équipes échangent entre elles des modèles de processus métier conçus de cette façon, il est presque impossible que les membres d'une équipe comprennent les modèles de toutes les autres équipes car la modélisation des processus métier a muri considérablement, mais cela a aussi rendu aujourd'hui la standardisation possible [7]. BPMN a donc contribué dans la réduction des différences de notation entre les divers outils du marché. En effet, il en existe beaucoup, c'est pourquoi une notation commune permettra une interopérabilité entre différentes applications, de la modélisation à l'exécution des processus [3]. De plus la notation BPMN est très simple et facile à apprendre car elle se constitue d'un nombre réduit d'éléments graphiques qui permet une puissance d'expression infinie [7]. Ainsi, BPMN a été principalement développé pour soutenir l'implémentation technique des processus, donc, Plus la technologie de l'information est importante dans une société, plus l'utilisation de BPMN devient utile [6].

Cependant, la notation BPMN, tant ventée, reste plus ou moins incomplète pour une parfaite modélisation de processus métier vis-à-vis du but de notre travail à réaliser, soit, la vérification de modèle de processus métier. Nous allons, justement, voir en quoi consistent les limites de BPMN.

V.1.2 - *Insuffisance de BPMN :*

L'arrivée d'une notation standard et partagée constitue en soi une progression considérable et significative. Cependant une notation ne suffit jamais à réussir une parfaite modélisation des processus métier complexes. L'astuce réside dans le fait qu'elle doit être accompagnée d'une méthode de modélisation et d'une politique organisationnelle pour une mise en œuvre réussie. Il est évident que la notation achemine le travail, mais avec la définition d'une méthode de modélisation le résultat est garanti. Une méthode de modélisation joue le rôle de « garant » et n'est pas à négliger dans cet exercice [7].

V.1.3 - *Inconvénients, limites et perspectives de BPMN :*

On ne peut pas tout attendre de l'adoption de BPMN car elle couvre un périmètre assez limité et restreint aux activités du processus [8], donc BPMN n'est pas considérée comme norme unique au sein des organisations [6]. Il y a donc souvent besoin de coexistence de la notation BPMN avec d'autres modes de représentation au sein d'une même entreprise ou organisation [7]. En plus, afin de passer de la description d'un processus à son exécution, il y a un certain niveau de précision exigé assez élevé [6].

De cette section, on peut conclure que BPMN est l'une des méthodes les plus utilisée. Cette notation aux avantages multiples offre une grande simplicité pour des travaux plutôt complexes, facilitant ainsi la tâche aux modélisateurs de processus métier. Cependant son usage est parfois limité à un domaine d'application précis, c'est pourquoi il y a besoin d'avoir recours à une méthode de modélisation des processus, notamment « Les réseaux de pétri ».

V.2 - *Les Réseaux de Petri (RdP) :*

V.2.1 - *Choix de réseau de Petri comme modèle :*

Le réseau de Petri est un outil de modélisation d'une grande puissance. Il devient nécessaire de pouvoir concevoir des systèmes toujours plus complexes, et surtout de s'assurer de leur bon fonctionnement en termes de sureté et de vivacité. Notamment la modélisation d'un système n'est utile que si elle permet d'analyser ses propriétés, donc la théorie des réseaux de Petri offre des techniques d'analyse puissantes pour valider des modèles de comportement de systèmes à événements discrets (analyser les propriétés d'un réseau de Petri) mais aussi pour décrire et vérifier des modèles de processus métier. En effet, plusieurs travaux de recherche ont exploité les points forts de ces réseaux afin de vérifier les erreurs que peut contenir un processus métier [9][10][11].

V.2.2 - *Avantages et atouts des réseaux de Petri :*

L'un des principaux avantages des réseaux de Petri est le fait qu'ils possèdent beaucoup de techniques et de logiciels d'analyse tout en étant simple et très expressifs (due à la représentation graphique). En effet, un réseau de Petri constitue un modèle sémantique (opérationnel et axiomatique) sûr du fait qu'il se base sur des fondations mathématiques fortes et peut s'appliquer dans de nombreux domaines à différentes étapes de construction de logiciels. Ainsi, les réseau de Petri fournissent une description précise mais non formelle

servant de support graphique de conception qui permet de décrire le système étape par étape et de passer à une description formelle permettant l'analyse mathématique du système [12][13][14].

En outre, les réseaux de Petri sont avantageux dans le domaine de la vérification car ils proposent une simulation des fonctionnements permettant d'avoir une dualité entre le système et son modèle, ils contribuent donc dans le domaine de la flexibilité. En effet, un tel modèle est tout à fait apte à évoluer en fonction des événements et des contextes et ainsi répondre en temps réel aux contraintes imposées en termes de flexibilité. Ainsi, les réseaux de Petri sont particulièrement bien adaptés à la représentation du partage de ressources qui est très fréquent dans les systèmes de production et dans divers domaines d'application [13].

De là nous avons conclu que le bon choix de la méthode de modélisation de processus métier consiste à transformer le modèle résultant de la notation BPMN en un réseau de pétri, bénéficiant ainsi des atouts de BPMN et de son modèle graphique et formel, ainsi, des avantages qu'offrent les réseaux de pétri à l'égard de la vérification des processus modélisés.

V.3 - ***Transformation de BPMN vers RdP :***

Nous avons étudié cette transformation en se basant principalement sur le travail décrit dans [15].

V.3.1 - ***Contraintes de Transformation :***

Selon le travail décrit en [15], afin de pouvoir transformer un modèle BPMN en un réseau de Petri, il faut que le modèle BPMN à transformer soit « un processus BPMN bien structuré ». On appelle un processus BPMN « processus BPMN bien structuré » s'il vérifie et respecte les conditions suivantes :

1. Un évènement de début ou d'exception a un seul flux sortant et aucun flux entrant.
2. Un évènement de fin a un seul flux entrant et aucun flux sortant.
3. Les activités et les évènements intermédiaires ont un flux entrant et un flux sortant.
4. Les passerelles de fork ou de décision ont un seul flux entrant et plusieurs flux sortant.
5. Les passerelles de fusion ou de jointure ont un seul flux sortant et plusieurs flux entrant.

Si le modèle BPMN ne respecte pas ces conditions, il faut le réécrire en suivant les règles suivantes afin d'obtenir un processus BPMN bien structuré :

1. Transformer les flux entrant multiples d'un évènement ou d'une activité en un seul flux entrant en faisant précéder l'objet correspondant par une passerelle XOR join qui a tous les flux entrant.
2. Transformer les flux sortant multiples d'un évènement ou d'une activité en un seul flux sortant en faisant suivre l'objet correspondant par une passerelle AND split qui a tous les flux sortant.
3. Décomposer une AND (ou XOR) passerelle avec plusieurs flux entrants et sortants en une AND (ou XOR) join passerelle suivie d'une AND (ou XOR) split passerelles, où la passerelle join a tous les flux entrant et la passerelle split a tous les flux sortant.
4. Transformer une tâche avec un flux de message en entrée et un flux de message en sortie en deux tâches reliées, une avec un flux de message en entrée et l'autre avec un flux de message en sortie.
5. Transformer un processus qui n'a pas d'évènement de début et de fin en un processus qui possède ces évènements, en faisant précéder chaque tâche n'ayant aucun flux entrant par un évènement de début et en faisant suivre chaque tâche n'ayant aucun flux sortant par un évènement de fin.

Un modèle BPMN respectant les cinq (5) conditions précédemment vues, devrait être prêt pour une transformation vers un réseau de Petri, excepté deux (2) cas :

1. ***Activité en boucle :***

(a)

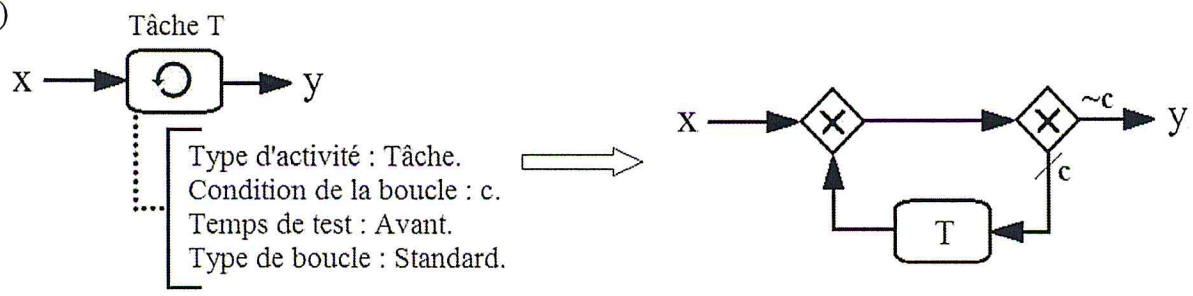


Figure 2 : Activité en boucle de type « while – do » [15].

(b)

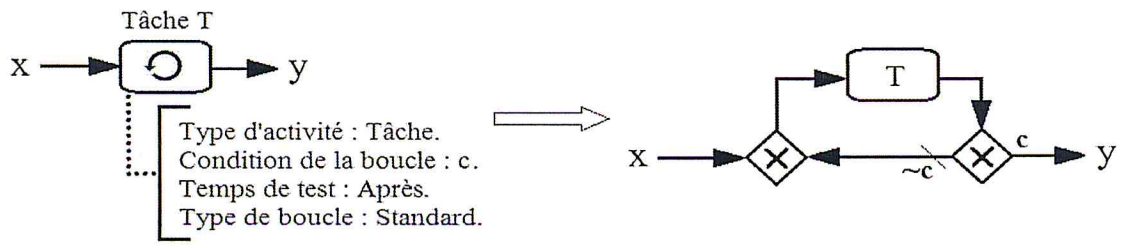


Figure 3 : Activité en boucle de type « do – until » [15].

2. ***Activité en instances multiples :***

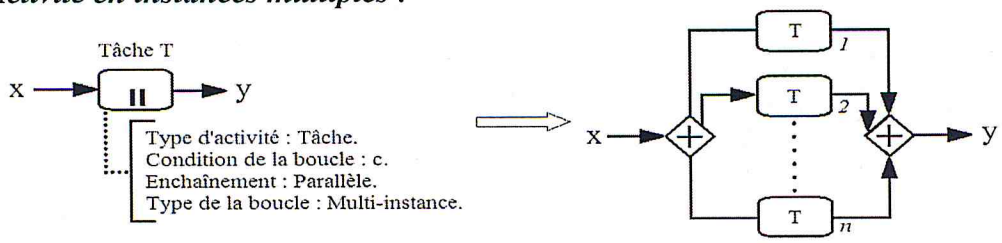


Figure 4 : Activité en instances multiples [15].

V.3.2 - ***Transformation d'un modèle BPMN en un RdP :***

D'après ce travail, la transformation est assez simple, chaque symbole BPMN a sa traduction, de façon à ce qu'à la fin, les deux modèles aient la même sémantique.

A - ***Transformation des objets de flux :***

1. Les évènements :

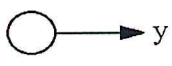
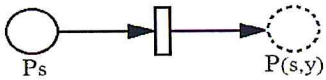
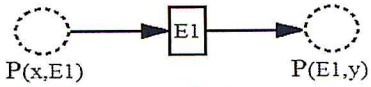
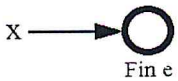
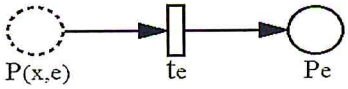
	Objet BPMN :	Module RdP :
Evènement de début :		
Evènement intermédiaire :		
Evènement de fin :		

Tableau 1 : transformation des évènements [15].

Le tableau 1 montre la transformation des évènements BPMN en modules RdP. Un évènement intermédiaire est transformé en une transition avec une place d'entrée et une place de sortie. Un évènement de début ou de fin est transformé en un module RdP similaire, sauf qu'une transition silencieuse est utilisée pour signaler le début ou la fin du processus.

2. Les activités :

a- Activité de type « tâche » :

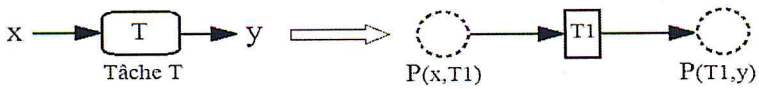


Figure 5 : Transformation de tâche [15].

De la même manière qu'un évènement intermédiaire, l'activité de type tâche se transforme en une transition avec une place d'entrée et une place de sortie.

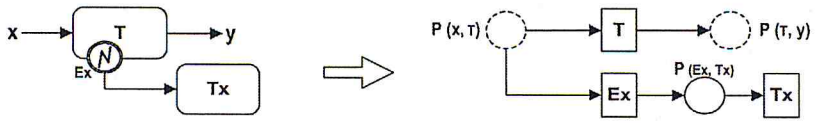


Figure 6 : Transformation d'une activité avec une exception [15].

Dans le cas d'une tâche atomique, l'apparition de l'exception Ex peut « voler » les jetons qui, normalement, devront être consommés par la transition correspondant à la tâche T.

Dans le cas d'un sous processus, l'apparition de l'exception annulera tout le sous-processus, ce qui complique la transformation. Ceci voudra dire que, quand la transition correspondant à l'évènement d'exception est franchissable, tous les jetons restants dans le réseau de pétri devront être retirés. Or, il est encombrant de modéliser un aspirateur qui retire les jetons d'un certain fragment du RdP. Comme solution, il est intéressant de contourner les tâches et les évènements à l'intérieur d'un sous-processus, en attachant un statut au sous-processus qui aura comme valeur « OK » permettant le flux normal de continuer ou « NOK » signalant l'occurrence de l'exception, entrainant l'arrêt du flux normal mais permettant de contourner le reste des évènements et des tâches jusqu'à la fin du sous-processus.

b- Activité de type « sous-processus » :

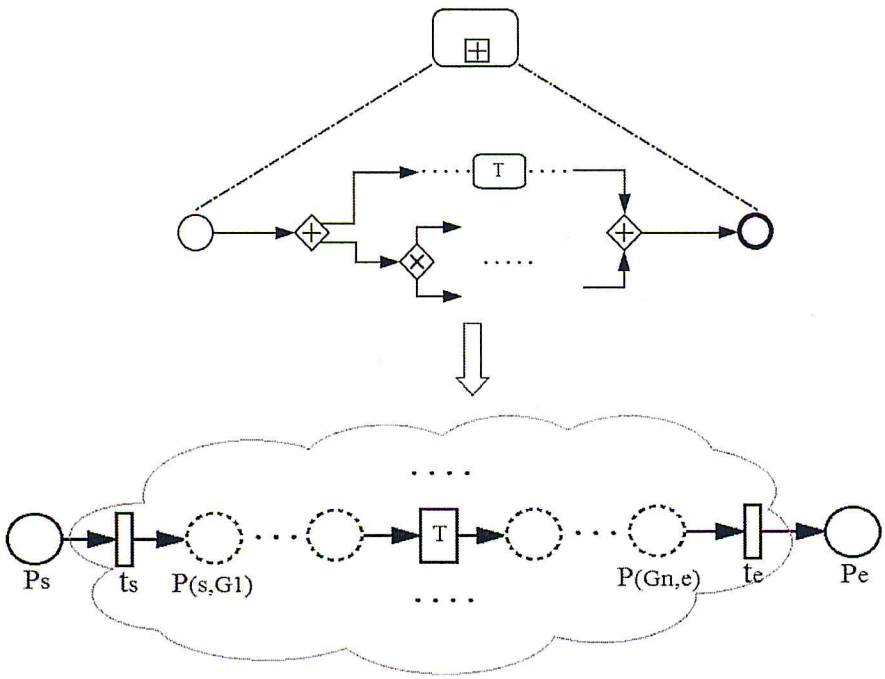


Figure 7 : Transformation de sous-processus sans traitement d'exceptions [15].

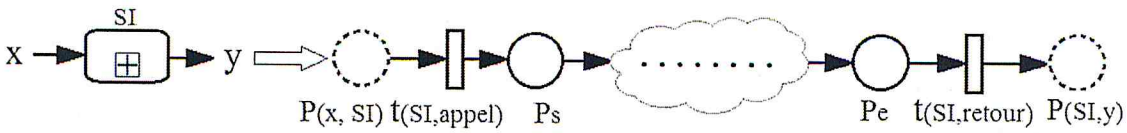


Figure 8 : Transformation d'une activité d'invocation de sous-processus (SI) appelant le sous-processus (P) [15].

3. Les passerelles :

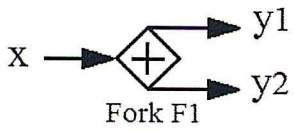
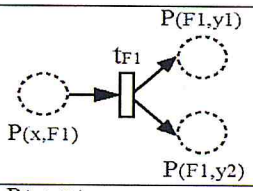
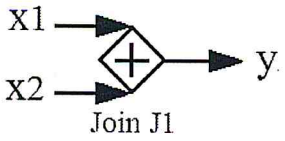
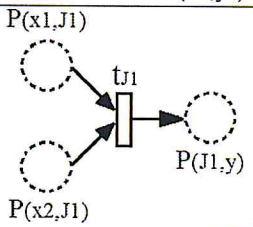
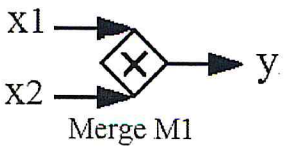
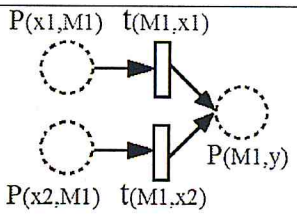
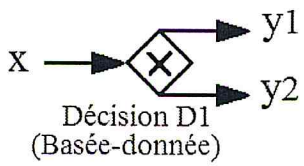
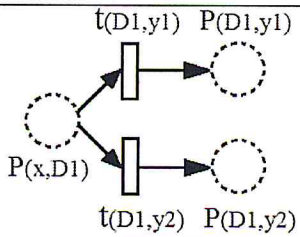
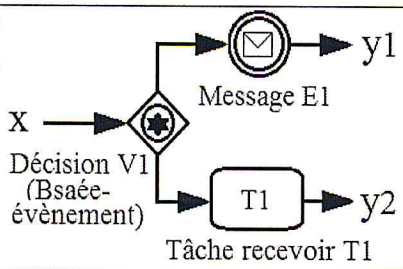
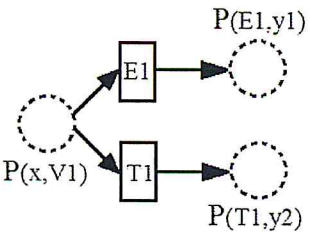
	Objet BPMN :	Module RdP :
Parallèle Fork (AND Split) :		
Parallèle Join (AND Join) :		
Exclusive (XOR Join) :		
Décision XOR (Basée-donnée) :		
Décision XOR (Basée-évènement) :		

Tableau 2 : Transformation des passerelles [15].

Les passerelles, excepté la passerelle de décision XOR basée-évènement et celle de décision OR inclusive, sont transformées en modules RdP avec des transitions silencieuses captant leur comportement de routage. Comme le montre le tableau 2. La passerelle de

décision XOR basée-évènement est transformée de la même manière, mais sans transition silencieuse. Il est évident qu'on a du mal à voir la différence entre les deux dernières transformations. En effet la différence réside dans la manière d'utiliser les modules RdP résultant, chose qu'on ne peut pas voir dans le tableau ci-dessus (tableau 2). Dans le cas de la passerelle de décision XOR basée-donnée, les transitions silencieuses, qui modélisent les conditions booléennes et qui ont une place d'entrée commune, vont concourir pour un jeton, et le choix quant à celle qui sera franchissable est non-déterministe. Ici, on ne modélise pas les conditions, mais le fait qu'une des conditions demeurera vraie. Par contre, dans le cas de passerelle de décision XOR basée-évènement, la concurrence entre l'évènement ou la tâche recevoir est illustrée par le fait que la transition correspondante concourra pour des jetons dans la place qui correspond au flux d'entrée de la passerelle.

B - ***Transformation des objets de liaison :***

- Flux de message :

1. Tâche vers tâche :

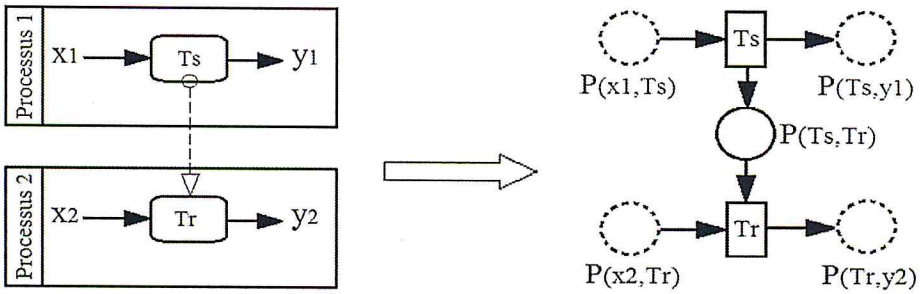


Figure 9 : Transformation de flux de message « tâche vers tâche » [15].

2. Evènement de fin vers tâche :

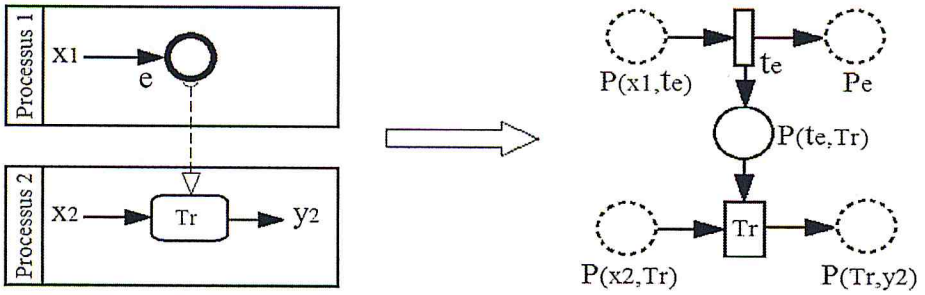


Figure 10 : Transformation de flux de message « évènement de fin vers tâche » [15].

3. Tâche vers évènement de début :

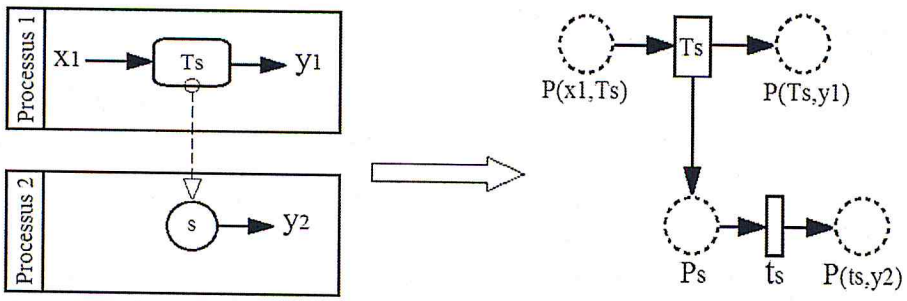


Figure 11 : Transformation de flux de message « tâche vers évènement de début » [15].

4. Evènement de fin vers évènement de début :

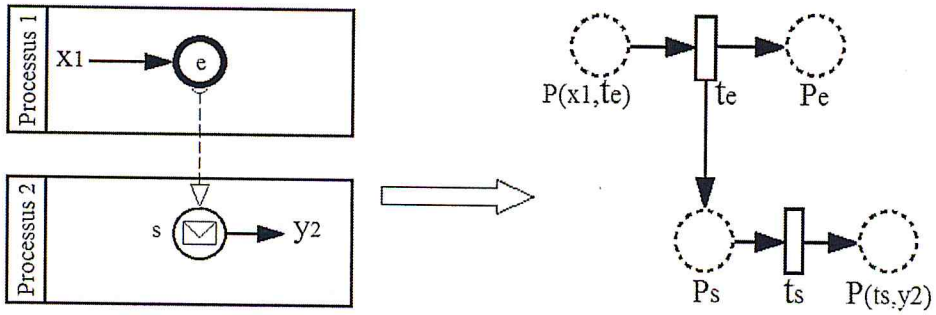


Figure 12 : Transformation de flux de message « évènement de fin vers évènement de début » [15].

Les figure précédentes (figure 9, 10, 11 et 12) montrent quatre règles de transformation, capturant chacune un cas d'un message envoyé par une tâche ou un événement de fin et reçue par une tâche ou un événement de début. Notons qu'une tâche peut être remplacée par un événement de message intermédiaire sans modifier les règles. Les transformations ci-dessus sont limitées à des tâches qui soit envoient ou reçoivent des messages mais pas les deux à la fois. Cette restriction ne limite pas la puissance expressive de BPMN, car l'envoi et la réception d'un message d'une manière successive peuvent être représentés par deux tâches comme un envoi suivi d'une réception.

C - ***Transformation du marquage initial :***

L'état initial d'un modèle BPMN peut être spécifié par le marquage initial du modèle réseau de Petri correspondant. L'idée de base pour la configuration du marquage initial est de marquer les places de déclenchement pour chacun des événements de début qui n'ont pas de flux message entrant et que les processus auxquels ils appartiennent sont des processus de haut niveau.

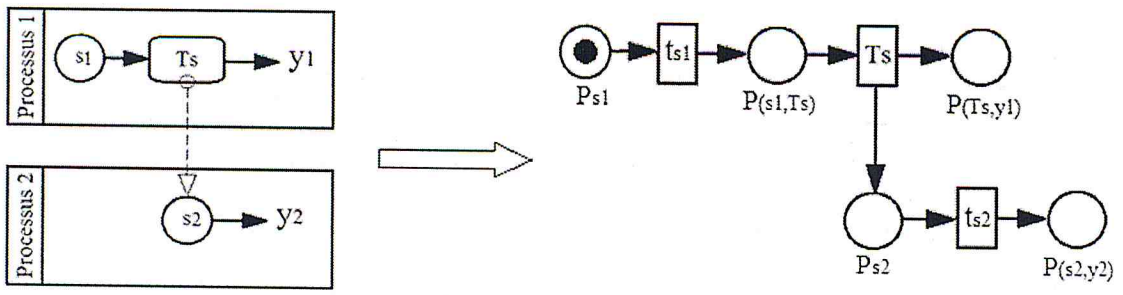


Figure 13 : Une seule dépendance d'initiation de processus [15].

Un flux de messages qui a comme cible l'événement de début d'un processus, va créer une instance du processus lors de la livraison de message. Ainsi, l'application doit s'assurer que la place de déclenchement de chaque événement de démarrage avec un flux de message entrant ne contient pas de jeton dans le marquage initial, car le processus peut seulement être instancié à la suite de cet événement quand un message est arrivé.

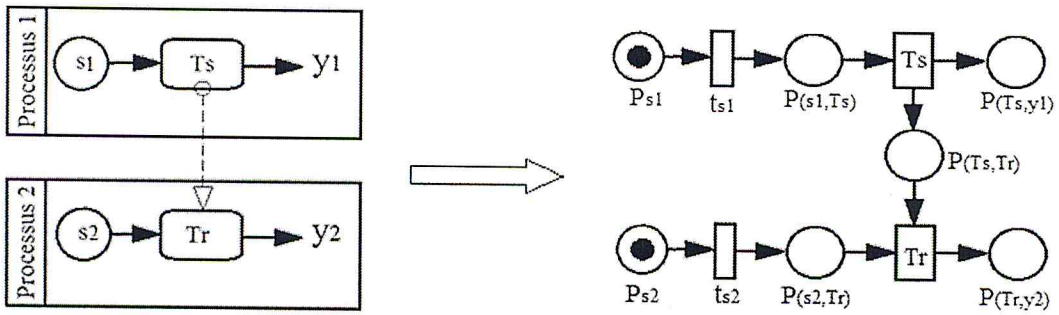


Figure 14 : Aucune dépendance d'initiation de processus [15].

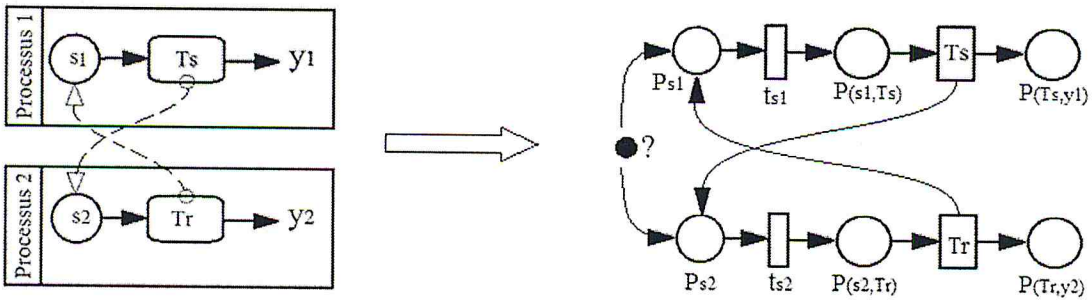


Figure 15 : Dépendance mutuelle d'initiation de processus [15].

Un cas particulier est que chaque processus de haut niveau est instancié par un autre processus par un message de flux entrant à son événement de début, le concepteur devra alors déterminer lequel des deux événements de début des processus de haut niveau est déclenché initialement.

Pour toutes les figures de transformation, les places dessinées en pointillés indiquent que leur usage n'est pas propre à un seul module. Elles sont utilisées pour relier deux objets BPMN et donc elles sont identifiées par les deux objets. x, x1 ou x2 désigne un objet d'entrée, tandis que, y, y1 ou y2 désigne un objet de sortie.

V.3.3 - *Exemple de transformation d'un processus BPMN en un réseau de Petri :*

La figure ci-dessous (figure 16) représente un processus métier décrit en BPMN.

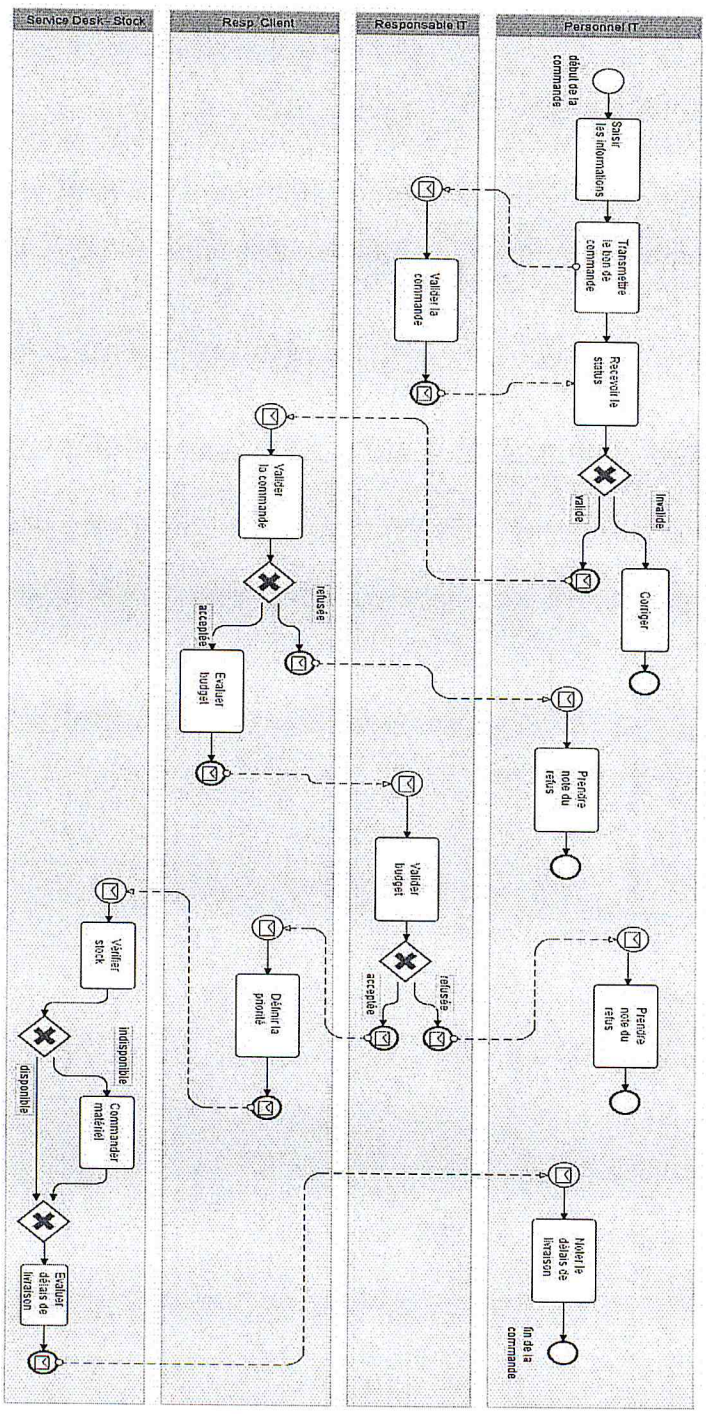


Figure 16 : Processus BPMN.

Après avoir appliqué les règles de transformation sur ce modèle, nous obtenons le réseau de Petri de la figure 17.

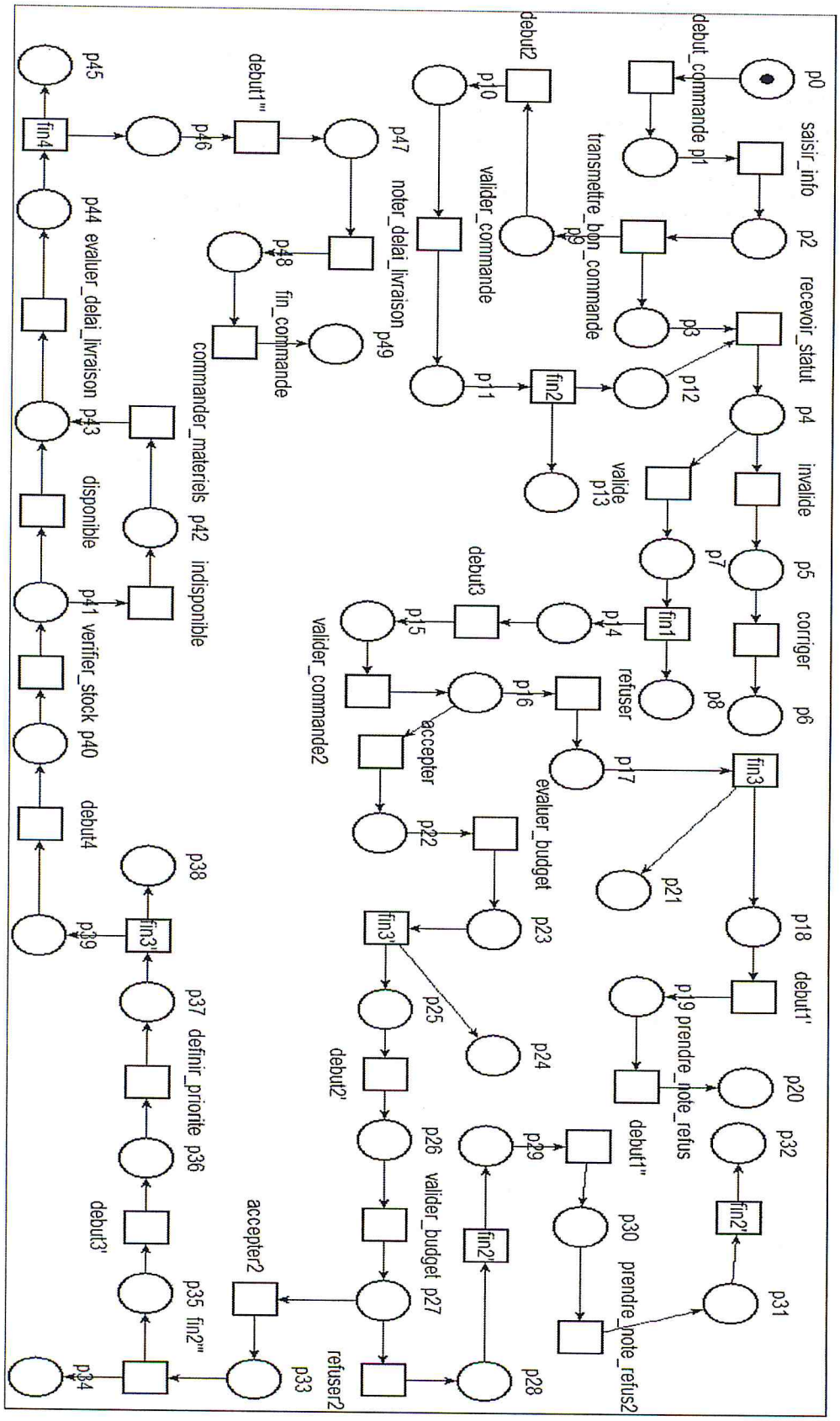


Figure 17 : Résultat de la transformation – réseau de Petri.

V.3.4 - ***Problèmes de transformation due à la spécification de BPMM :***

A - ***Modèle avec plusieurs évènements de début :*** la spécification de BPMN déclare que :

1. Chaque évènement de début est indépendant, une instance de processus devra être générée lorsque ce dernier est déclenché.
2. Si les attributs de l'activité succédant les évènements de début précisent que l'activité doit attendre toutes les entrées, alors tous les évènements de départ devront être déclenchés avant le début du processus.

Cependant, un examen plus approfondi des attributs associés aux activités montre qu'aucun d'entre eux permet de modéliser qu'une activité doit attendre toutes les entrées. Il est donc suggéré que la norme BPMN devrait clarifier les combinaisons autorisées des événements de début qui peuvent survenir dans le contexte d'une exécution d'un modèle de processus avec plusieurs événements déclencheurs. La transformation vue précédemment, appliquée sur un tel modèle résultera un réseau de Petri avec plusieurs places sources, il peut toujours être vérifié en utilisant des techniques de réseau de Petri. Or, il faut établir le marquage initial à partir duquel l'analyse qui doit être réalisée.

B - ***Achèvement d'instance de processus :*** La spécification BPMN n'indique pas clairement quand l'exécution d'un modèle de processus doit être considérée comme étant achevée. Cela est problématique surtout pour les modèles ayant plusieurs évènements de fin. Est-ce que l'achèvement indique qu'un seul évènement de fin s'est produit ou bien tous ou bien qu'aucune activité n'est active désormais? D'après les études, une instance d'un modèle de processus est achevée lorsqu'au moins une de ses tâches de fin a été exécutée au moins une fois, et il n'y a pas d'autre tâche active pour ce processus. La transformation vue précédemment résultera un réseau de pétri avec plusieurs places puits, il peut toujours être vérifié en utilisant des techniques de réseau de pétri. Cependant, si le réseau de pétri est 1-sauf on peut le traduire en un réseau de pétri avec une seule place puits.

C - ***Exception pour un sous-processus à plusieurs instances*** : est-ce que l'exception interrompt l'instance du sous-processus en question ou bien toutes les instances ? Si la première option est adaptée, on doit résoudre une autre ambigüité, l'instance interrompue, est-elle considérée comme achevée ? On sait seulement qu'une activité multi-instance qui invoque un sous-processus est achevée si toutes les instances qu'elle produit sont achevées. Dans le cas où l'on n'a pas un réseau 1-sauf, il peut arriver que deux instances d'un sous-

processus s'exécutent simultanément, si l'activité qui les invoque s'exécute une seconde fois au cours de l'exécution de la première instance, alors un appel au sous-processus qui va exécuter une seconde instance simultanément sera découlé. Aussi, si une exception est lancée par une instance et est capturée par un gestionnaire d'exception attaché à l'activité d'invocation, il est difficile de savoir si cette exception n'affecterait que cette instance de sous-processus, ou toutes les instances de sous-processus engendrées par l'activité d'invocation.

D - ***Passerelle complexe (OR-join)*** : la spécification BPMN déclare qu'une passerelle complexe doit attendre tous les jetons qui ont été produit en amont. Le flux continuera quand les jetons arrivent de tous les arcs entrant. Cependant, si cette passerelle est dans un cycle, ceci peut nous mener à un cycle vicieux.

VI - ***Conclusion :***

Ce premier chapitre énonce la modélisation des processus métier. Nous avons présenté les concepts fondamentaux de BPMN ainsi que ceux des réseaux de Petri. Ces derniers, sont un formalisme de modélisation des systèmes. Nous avons également décrit un travail qui porte sur la transformation d'un modèle BPMN en un réseau de Petri.

D'après l'étude faite, BPMN est une notation conçue pour la modélisation des processus métier, elle constitue l'une des notations les plus utilisées actuellement pour modéliser les processus métier. Cependant, pour une fin de vérification de ces derniers, la norme a besoin d'être accompagnée d'une méthode de modélisation. Nous avons constaté que les réseaux de Petri constituent un meilleur candidat du fait qu'ils comblent le manque de BPMN. Ainsi, la transformation d'un modèle BPMN en un réseau de Petri nous permettra de bénéficier de la forte expressivité de BPMN et de la fiabilité du modèle offert par un réseau de Petri.

Toutefois, et afin d'entamer la vérification dont il s'agit, il est nécessaire d'étudier les différentes méthodes et techniques de vérification de modèle de processus métier ultérieurement (voir chapitre 2 ci-dessous).

Chapitre II

Méthodes et techniques de vérification de modèles de processus

I - *Introduction :*

La fiabilité des processus d'une part et la maitrise des couts de maintenance d'autre part conduisent à accorder une attention plus grande aux questions de vérification des processus. En effet, la vérification des processus métiers a pour but, dans le cadre de notre travail, de montrer que les activités du processus s'exécutent en conformité avec son plan de réalisation et qu'elles n'ont pas introduit des erreurs dans les résultats.

Nous avons vu précédemment comment modéliser nos processus métier. Dans ce chapitre on abordera les différentes méthodes et techniques de vérification par modèles formels ainsi que leur principe de fonctionnement afin de pouvoir choisir le modèle le plus approprié qui fera ensuite l'objet de la vérification.

II - *Vérification de processus métier :*

La vérification est l'ensemble des différents moyens, ou autres techniques appropriées, permettant d'établir et de documenter une certaine conformité du développement par rapport aux critères qui sont préétablis. (ISO 8402) définit la vérification comme étant « *la confirmation par examen et apport de preuves tangibles (informations dont la véracité peut être démontrée, fondée sur des faits obtenus par observation, mesures, essais ou autres moyens) que les exigences spécifiées ont été satisfaites* ».

On peut classifier les techniques de vérification en trois critères notamment :

La technique est-elle considérée comme une technique de vérification ou plutôt comme une technique de validation ?
La focalisation de la technique: est-ce que la technique se focalise sur le côté statique et descriptif, ou bien sur la dynamique du système ?
Le niveau de formalisation de la technique: est-ce que la technique est faite sur une base formelle, semi- formelle ou alors non formelle ? Ceci nous conduit aux types de technique ci-dessous :

- **Technique non formelle :** ce type de technique examine le modèle manuellement afin de détecter des erreurs ou ce qui y ressemble. On en déduit que c'est plutôt une techniques de validation.
- **Technique semi-formelle :** il s'agit de techniques dont le résultat peut être vu comme indépendant d'une expertise humaine mais sans pour autant revêtir une exhaustivité. Statiquement, il peut s'agir de vérifier si le modèle est conforme à un standard

existant. Dynamiquement, les techniques usuelles sont la simulation, l'émulation ou le test. Leur inconvénient par contre est le fait qu'elles ne permettent pas une exhaustivité des cas envisagés, il est alors impossible de garantir l'absence de fautes. La qualité du modèle de simulation ainsi que l'expérience des utilisateurs déterminent la pertinence et la qualité du modèle. Ceci dit, les techniques semi-formelles sont beaucoup plus adaptées pour une vérification ou validation pour, à titre d'exemple, un débogage.

- **Technique formelle :** ce sont des techniques basées sur des méthodes formelles. Leur particularité réside dans l'utilisation des langages et des concepts relevant du domaine des mathématiques. [13]

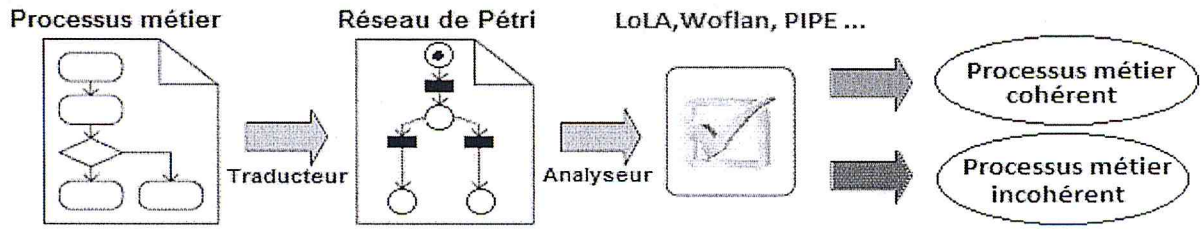


Figure 18 : Exemple du processus général de vérification. [16]

III - *Méthodes et techniques formelles de vérification :*

Dans un logiciel, les méthodes formelles constituent la base mathématique. Le formalisme d'une méthode réside dans la solitude de sa base mathématique donnée en langage de spécification formel. Ce dernier est défini comme ayant une sémantique basée sur des règles d'interprétation qui garantissent la non ambiguïté, ainsi que des règles de déduction permettant un certain raisonnement sur les spécifications, et ce, dans le but de détecter des anomalies, notamment : incomplétudes, inconsistances ou alors afin de prouver des propriétés. Ces méthodes peuvent aussi aider à découvrir des failles, des contradictions et des incohérences aussi bien qu'à déterminer la correction de l'implantation d'un système.

III.1 - *Preuve de théorème :*

III.1.1 - *Définition :*

La preuve de théorème automatique consiste à démontrer des théorèmes mathématiques à l'aide d'un programme informatique. L'outil de preuve de théorème nécessite deux entrées distinctes. La première correspond à la description du système de manière formelle. Dans cette description, on trouve des axiomes et / ou des hypothèses servant à

modéliser le système. Tandis que la deuxième correspond à une propriété que l'on souhaite vérifier, cette propriété est également spécifiée formellement. [17]

Elle constitue le processus qui sert à prouver formellement la propriété à vérifier via un mécanisme déductif à partir de la description du système. Cette technique peut très bien s'appliquer sur des espaces d'état mêmes infinis, notamment : les réseaux de Petri. Cependant, le temps que le processus de vérification prend pour prouver une propriété à partir des exigences du système formalisées est potentiellement lent, ainsi, la preuve de théorème demande une grande expertise et des utilisateurs spécialisés [18].

III.1.2 - ***Avantages :*** [17][18]

- La preuve de théorème, basée sur les mathématiques permet de mettre en œuvre des outils très puissant.
- Le résultat est fiable.
- Cette méthode peut s'appliquer à des réseaux de Petri, voire des espaces d'état infinis, et donc, elle est bien adaptée à notre travail.

III.1.3 - ***Inconvénients :*** [17][18]

- Les outils de preuves de théorème sont seulement utilisés par des personnels spécialement formés.
- La construction d'une preuve peut prendre beaucoup de temps, voire des semaines.

III.2 - ***WoFlAn (WOrkFLow ANalyzer) :***

III.2.1 - ***Besoin d'un outil de vérification de workflow :***

Woflan est un outil d'analyse de workflow qui a été conçu au cours des années 90s, qui vérifie si un réseau de Petri est conforme à certaines restrictions de base pour un workflow. Woflan ne gère que les réseaux de Petri non-colorés sans synchronisation.

De toute évidence, il ya un besoin d'un tel outil de vérification, parce que les systèmes de gestion de workflow d'aujourd'hui ne supportent pas les techniques avancées pour vérifier l'exactitude des définitions de processus de workflow. Ces systèmes se limitent généralement à un certain nombre (trivial) de vérifications syntaxiques. Par conséquent, les erreurs graves, tels que des blocages et des boucles infinies peuvent rester non-détectées. Cela signifie qu'un

workflow erroné peut être produit, provoquant ainsi des problèmes dramatiques pour l'organisation. Un workflow erroné peut alors conduire à un travail supplémentaire. [19]

III.2.2 - *Architecture de Woflan :*

Woflan se compose d'un analyseur qui permet d'analyser les définitions de processus de workflow spécifiées en termes de réseau de Petri. En outre, chaque nœud du réseau de Petri devrait être mis sur un chemin allant de la place d'entrée à la place de sortie. L'analyseur lit le réseau de Petri à partir d'un fichier d'entrée et construit une structure de données pour chaque définition de processus de workflow qui doit être analysée. Si le réseau de Petri ne satisfait pas les exigences mentionnées, Woflan avertit l'utilisateur. Ainsi, il dispose d'une routine d'analyse qui est une structure de données créée par l'analyseur utilisée comme point de départ pour toutes sortes d'analyses. Woflan vérifiera la propriété de la cohérence. Enfin Woflan est équipé d'une interface utilisateur graphique qui présente les résultats d'analyse à l'utilisateur [19].

III.2.3 - *Les techniques d'analyse prises en charge par Woflan :*

Deux des concepts clés de Woflan, sont la définition d'un workflow-net et la propriété cohérence.

Un réseau de Petri qui modélise une définition de processus de workflow est appelé un workflow-net (WF-net). Notons que la définition d'un workflow-net est une définition syntaxique, les exigences peuvent être vérifiées statiquement, car ils ne portent que sur la structure du réseau de Petri.

La propriété de cohérence concerne la partie dynamique de la définition de processus de workflow. Un workflow-net est cohérent si les conditions suivantes sont vérifiées : [19]

1. Pour tous les cas, il est possible que le processus se termine, c'est à dire, il est possible d'atteindre un état d'au moins un jeton dans la place de sortie.
2. Quand l'un des cas se termine (c'est à dire un jeton apparaît dans la place de sortie), il n'y a pas de jetons laissés dans le workflow-net.
3. Absence de tâches mortes, i.e., en commençant avec un jeton dans la place d'entrée, il devrait être possible d'exécuter une tâche arbitraire en suivant la voie appropriée à travers le WF-net.

La cohérence est la propriété minimale que toute définition de processus de workflow doit satisfaire. Notons que la cohérence implique l'absence de blocages et de boucles infinies. A cet effet Woflan utilise une relation intéressante entre la cohérence d'une part et de la vivacité et la bornitude d'autre part. Un workflow-net est cohérent, si et seulement si, le réseau obtenu en connectant la place d'entrée et la place de sortie via une supplémentaire transition t* est vivant et borné. Bien que la cohérence puisse être décidée en temps polynomial pour certaines sous-classes, Woflan construit le graphe d'accessibilité pour vérifier si le workflow-net est vivant et borné. [19]

Si la définition de processus de workflow n'est pas cohérente, Woflan guide l'utilisateur à trouver et à corriger l'erreur. Pour aider l'utilisateur dans la réparation de l'erreur, Woflan offre une fonction d'aide en ligne. L'aide en ligne est basée sur une approche par étapes pour localiser et supprimer les constructions qui violent la propriété de la cohérence. Cela permet aux utilisateurs sans expérience dans les réseaux de Petri de faire fonctionner l'outil et réparer une définition erronée de processus de workflow. [19]

III.2.4 - ***Avantages :*** [19]

- Woflan est basé sur les réseaux de Petri, ce fait constitue un avantage pour nous car notre modèle d'entrée va être présenté par un réseau de Petri.

III.2.5 - ***Inconvénients :*** [19]

- Du fait que nous visons à vérifier le maximum de propriétés que possible, Woflan ne constitue pas un bon candidat puisqu'il ne permet de vérifier que la propriété de cohérence.

III.3 - ***WofBPEL (WOrkFlow Business Process Execution Language) :***

III.3.1 - ***Définition :***

WofBPEL utilise l'outil BPEL2PNML (outil d'accompagnement) qui prend en entrée le code BPEL (un langage de définition de processus adapté pour l'informatique orientée services) et produit un fichier conforme à la syntaxe Petri Nets Markup Language (PNML). Ce fichier peut être fourni comme entrée pour WofBPEL qui, selon les options choisies, applique un certain nombre de méthodes d'analyse et produit un fichier XML décrivant les résultats d'analyse. Actuellement, il prend en charge trois types d'analyse : [20]

A - ***Analyses d'accessibilité :*** WofBPEL détecte les activités inaccessibles en s'appuyant sur deux méthodes différentes, à savoir la cohérence détendue et l'invariant de transition. La première est complète mais coûteuse car elle prend en compte tous les cas possibles à partir d'un état initial à l'état final désiré. Chaque transition qui n'est pas couverte par l'une de ces pistes est dite non-cohérente détendue et indique clairement une erreur (Si le but du RdP est de passer de l'état initial à l'état final souhaité). Pour pallier le problème d'espace d'état, on remplace la cohérence détendue par l'invariant de transition. Cependant, ses résultats ne sont pas nécessairement complets. Ils assument plus ou moins que les jetons suffisants existent pour que chaque transition s'exécute autant de fois que nécessaire. [20]

B - ***Détection des activités contradictoires de consommation de message :*** cette analyse consiste à détecter les activités qui consomment le même type de message et qui sont activées simultanément, où le type de message est identifié par une combinaison d'un partnerLink, un portType, une opération, et un ensemble de corrélation optionnel [20]. Par exemple :

```
<invoke name="E1" partnerLink="pl1" portType="pt1" operation="op2">

<receive name="R1" partnerLink="pl2" portType="pt2" operation="op2">

<receive name="R2" partnerLink="pl2" portType="pt2" operation="op3">

<receive name="R3" partnerLink="pl2" portType="pt2" operation="op2">
```

R1 et R3 sont des activités de réception contradictoires car elles ont le même partnerLink (pl2), le même portType (pt2), et la même opération (op2). Nous pouvons résoudre l'éventuel problème de l'espace d'état à l'aide des règles de réduction des RdP bien connues [20].

C - ***Collection de corbeille pour les messages en attente :*** La sortie de cette analyse serait un processus BPEL annoté où chaque activité de base est associée à un ensemble de types de messages (identifié par un partnerLink, un portType, une opération, et éventuellement un ensemble de corrélation). Après l'exécution d'une activité « a », le moteur BPEL pourrait comparer l'ensemble des types de messages dans « MTa » associés à « a » avec l'ensemble des messages dans la file d'attente « Mq » et jetez tous les messages appartenant à « Mq » et n'appartenant pas à « MTa ». [20]

III.3.2 - ***Avantages :*** [20][21][22]

- WofBPEL peut s'adapter à des espaces d'état, notamment, aux réseaux de Petri.
- Cet outil dispose de trois différentes analyses pour les processus.

III.3.3 - ***Inconvénients :*** [20][21][22]

- WofBPEL est mieux adapté pour les processus BPEL décrivant principalement des services web.
- Malgré les trois analyses qu'il fournit, il ne vérifie pas beaucoup de propriété.

III.4 - ***PIPE (Platform Independent Petri net Editor) :***

III.4.1 - ***Définition :***

PIPE (Platform Independent Petri éditeur Net) est un outil de plateforme-indépendant open source pour créer, simuler et analyser des réseaux de Petri stochastiques généralisés (GSPN) (un type de réseau de Petri qui permet la « spécification temporelle »). [23]

PIPE est mis en œuvre entièrement en Java pour garantir l'indépendance de plateforme et fournit une interface utilisateur graphique facile à utiliser qui permet la création, l'enregistrement, le chargement et l'analyse des réseaux de Petri conformes au format d'échange PNML (Petri Nets Markup Language). [23]

PIPE offre également une gamme complète de « modules d'analyse » qui sont : la classification, comparaison, l'interface DNAmaca, l'analyse GSPN, analyse de l'invariant, incidence et marquage, graphe d'accessibilité, simulation et l'analyse de l'espace d'état qui permettent de vérifier les propriétés comportementales, produire des statistiques de performance, et certaines fonctions peu communes telles que la comparaison des réseaux de Petri et leur classification. [23]

III.4.2 - ***L'interface utilisateur graphique :***

PIPE a été conçu avec l'objectif de fournir un outil intuitif, facile à utiliser pour l'édition des réseaux de Petri d'une manière facile, rapide et efficace. Quiconque qui est familier avec une interface standard de design peut utiliser PIPE sans connaissances spécifiques de l'application. PIPE utilise une représentation standard pour les différents éléments qui constituent un réseau de Petri. [23]

L'interface graphique est conforme aux standards PNML / XML afin qu'il puisse ouvrir et travailler avec des réseaux de Petri PNML existants. Il fournit une interface de documents multiple pour que l'on puisse travailler avec plusieurs réseaux de Petri en même temps,

chacun d'entre eux situé dans un onglet. Les réseaux de Petri peuvent être imprimés ou exportés en deux (2) formats graphiques: Postscript et PNG. [23]

III.4.3 - ***Mode animation :***

PIPE offre un animateur afin de fournir à l'utilisateur la possibilité de tirer une des transitions sensibilisées par chaque état. L'ensemble des transitions actives est mise en valeur graphiquement pour que l'utilisateur puisse clairement les voir et en choisir une qui doit être tirée. L'historique de l'animation est enregistré, c'est à dire toutes les transitions tirées peut être vu sur le côté de l'écran, afin que depuis l'état actuel, l'animation peut aller en avant ou en arrière. L'exécution automatique d'une transition aléatoire en mode d'animation est également possible, pour cette fonction, l'utilisateur spécifie le délai de tir et le nombre de tirs. [23]

III.4.4 - ***Avantages :*** [23]

- PIPE peut être appliqué sur des espaces d'état notamment, les réseaux de Petri.
- Il dispose d'une multitude de modules permettant différentes analyses.

III.4.5 - ***Inconvénients :*** [23]

- Cet outil est mieux adapté pour les modèles markoviens et les réseaux de Petri stochastiques.
- Il ne permet pas la vérification de toutes les propriétés comportementales des réseaux de Petri.

III.5 - ***TiNA (TIme petri Net Analyzer) :***

III.5.1 - ***Définition :***

TINA sert à éditer (graphiquement) et analyser des RdPs classiques et temporisés, et supporte la construction des graphes d'accessibilité permettant l'analyse [24]. Afin de vérifier, le modèle TINA exige, en entrée, un modèle formel de RdP ainsi que les propriétés. Il effectue des abstractions sur le modèle et vérifie par la suite les propriétés souhaitées [25].

Les différents outils constituant l'environnement peuvent être utilisés de façon combinée ou indépendante. Parmi eux on cite : nd (NetDraw : l'éditeur des RdPs temporels et des automates), tina (construction des représentations de l'espace d'états d'un RdP, temporel ou non) selt (vérificateur de modèle pour les formules de Stat/Event LTL) play (simulateur pas à pas), ndrio (outil de conversion pour RdPs temporels), ktzio (outil de conversion pour

les systèmes de transition Kripke), frac(compilateur de la description Fiacre dans les systèmes de transition), muse (vérificateur des systèmes de transition Kripke), sift (construction et la vérification des graphes d'accessibilité), struct (détermine les propriétés d'invariance et de consistance), plan (analyseur de chemin). En plus, Tina vérifie des propriétés générales d'accessibilité (caractère borné, présence de blocage, pseudo-vivacité et vivacité).

III.5.2 - ***Avantages :*** [24][25][26]

- Elle permet de calculer les chemins les plus lents et les plus rapides.
- Permet de vérifier pas mal de propriétés : d'accessibilité (caractère borné, présence de blocage, pseudo-vivacité et vivacité) et de cohérence.
- Elle permet de fournir une séquence contre-exemple en clair ou sous un format exploitable par le simulateur de TINA ainsi les contre-exemples générés peuvent être sauvegardés dans un format chargeable dans le deuxième simulateur, pour la relecture.
- Une langue riche permettant notamment de déclarer de nouveaux opérateurs ou redéfinir les opérateurs existants.
- Construction d'espaces d'états abstraits, basés sur les techniques d'ordre partiel.

III.5.3 - ***Inconvénients :*** [24][25][26]

- L'inconvénient majeur est que cet outil est plus adapté pour la vérification des modèles basés sur des réseaux de Petri temporisés.
- Il ne permet de vérifier qu'une minorité de propriétés comportementales.

III.6 - ***LoLA (LOw Level petri net Analyzer) :***

III.6.1 - ***Définition :***

LoLA est un outil développé pour la validation des techniques de réduction pour les graphes d'accessibilité des réseaux de Petri. C'est un analyseur de bas niveau [24] vérifiant si un modèle d'un système donné en réseau de Petri satisfait une certaine propriété. On note que la propriété à vérifier est donnée en une formule de logique temporelle. [27]

Il n'est pas conçu pour supporter la modélisation. Cependant, l'interface est faite de telle sorte que l'intégration de LoLA dans un cadre de modélisation est très facile. Les entrées et sorties sont organisées sous forme de flux de données (qui peuvent être des fichiers). LoLA lit le réseau de Petri à partir d'un fichier ASCII et obtient la propriété d'un autre fichier ou

comme paramètre par ligne de commande. La propriété (donnée comme une formule en CTL*) est analysée et reportée à une routine d'analyse appropriée pouvant être un model-checker CTL, un model-checker LTL, un vérificateur d'accessibilité, ou un détecteur d'interbloquage). Une formule CTL* qui n'est pas dans un des fragments mentionnés, ne peut être analysée. Lola délivre alors si la propriété est vraie ou pas. Dans certains cas, il peut en outre produire un témoignage ou un chemin vers un contre-exemple. L'application des techniques de réduction peut être utilisée par ligne de commande. [27]

L'évaluation de la propriété se fait par exploration exhaustive (de tous les cas possibles de l'ensemble de l'espace d'état) et explicite d'un espace d'état réduit. Cela signifie qu'on évalue la propriété par l'exploration de tous les cas possibles afin de révéler la véracité ou non de la propriété. On note que la propriété à vérifier est donnée en une formule de logique temporelle. [27][28]

III.6.2 - ***Les propriétés que LoLA permet de vérifier :***

LoLA peut analyser et vérifier, en utilisant la logique temporelle CTL, l'accessibilité d'un état, la bornitude du réseau ou une place, l'existence ou l'absence d'interbloquage, les transitions mortes, la réversibilité du réseau, l'existence d'état d'accueil, la vivacité ainsi que la quasi-vivacité d'une transition donnée [27][29]. Ainsi LoLA répond à des requêtes en logique temporelle : [29]

- Une certaine transition est-elle morte ?
- Le réseau est-il réversible ?
- Y a-t-il un état d'accueil ?
- Y a-t-il un état d'interbloquage ?
- Un certain état est-il borné ?
- Le réseau est-il borné ?

LoLA effectue donc la vérification de deux manières différentes, soit en affichant directement les résultats d'une analyse générale du réseau en question, ou bien en donnant la main à l'utilisateur pour faire entrer dans une zone de texte la formule de la propriété qu'il veut vérifier en CTL. [29]

III.6.3 - ***Avantages :*** [27][28][29]

- LoLA s'applique sur les réseaux de Petri.

- La propriété en entrée est analysée et reportée à une routine d'analyse appropriée.
- En cas ou le modèle ne satisfait pas la propriété, il se peut qu'un contre-exemple soit donné.
- LoLA a participé plusieurs fois dans des concours de vérification de modèle qui a lieu avec les conférences de réseaux de Petri et a montré une compétitivité importante.
- LoLA permet la vérification de beaucoup de propriétés comportementales.
- LoLA est rapide : 33,000 - 100,000 états par seconde, 42,000 - 120,000 transitions par seconde, 450 - 800 MB par minute ainsi, la propriété de cohérence est vérifiée en 4 ms par processus.

III.6.4 - ***Inconvénients :*** [27][28][29]

- La propriété donnée qui n'est pas dans le model-checker LTL, le model-checker CTL, le vérificateur d'accessibilité ou le détecteur d'interbloquage ne peut pas être vérifiée.
- LoLA repose principalement sur les techniques de réduction de graphe et n'analyse que les espaces d'état réduits.

III.7 - ***Model-checking :***

III.7.1 - ***Définition :***

Le model-checking, est reconnu comme étant la méthode formelle la plus utilisée et des plus puissantes en termes d'efficacité et d'automaticité qui existe de nos jours. En effet, c'est une technique basée sur des modèles mathématiques qui permettent d'analyser des systèmes réactifs. Le model-checking est une méthode de vérification exhaustive. L'exhaustivité réside dans son algorithme explorant exhaustivement l'ensemble de toutes les exécutions et les configurations possibles du système (exploration par force brute) [30], et ce, en utilisant un des algorithmes astucieux mis en œuvre qui incluent généralement une construction de l'espace d'état du système puis un parcours de cet espace à la recherche d'erreurs. Cet espace d'état est un graphe dirigé qui décrit toutes les évolutions possibles du système. Les nœuds de ce graphe sont les états du système et les arcs représentent les transitions entre ces états. Cependant, le nombre des états générés par ce modèle peut être exponentiel, ceci peut rarement être considéré comme un problème car, les outils du model-checking actuels peuvent énumérer explicitement des espaces d'états à taille importante [31]. Le model-checker (l'outil qui performe l'algorithme du model-checking) prend deux entrées, la première est une abstraction du comportement du système réactif (le modèle formalisant le

comportement du système réel en structure de kripke, RdP, automates, etc.) et une formule en logique temporelle (le quoi du système en LTL, CTL, CTL*, ect.) [16].Ainsi, il vérifie si l'abstraction du système réel satisfait la formule de la propriété [31]. Le model-checking permet de vérifier beaucoup de propriétés comportementales et durant une exécution, est en mesure de fournir un cotre exemple si une propriété n'est pas satisfaite [16][32]. Les deux outils pionniers du model-checking actuels sont Spin et SMV [16].

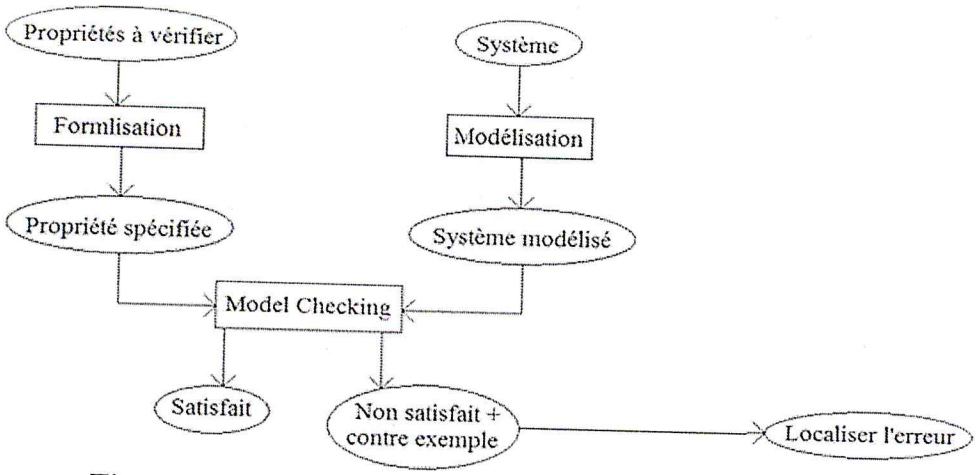


Figure 19 : Illustration du principe du model Checking. [30]

III.7.2 - ***Le processus du model-checking :*** [33]

1. Modélisation du système en question (S) et expression formelle des propriétés (Q)
2. Analyse des résultats :
    a) (S) satisfait (Q) ➔ le modèle (S) est sûr. Fin. Vérifier quand même que le système réel est sûr aussi.
    b) (S) ne satisfait pas (Q) ➔ retour d'un contre exemple.
        i. Vrai bug ➔ Avertir et attendre correction. Fin
        ii. Erreur de modélisation ➔ revenir à (1) et raffiner le modèle.

III.7.3 - ***Avantages :*** [29][33][34]

- Approche de vérification générale appliquée à une large gamme d'applications.
- Support de vérification partielle (chaque propriété peut être vérifiée individuellement), ainsi, il n'est pas nécessaire de spécifier complètement les exigences.
- Le model-checking fournit un diagnostic en cas où la propriété est non valide. En effet, une propriété appréciable d'un model-checker est qu'il fournit un contre exemple à l'utilisateur en cas d'erreur dans le modèle. Ce contre-exemple est une

séquence de transitions dans l'espace d'état qui infirme la propriété spécifiée. Cette trace d'exécution est un outil précieux pour corriger le problème.

- C'est une technologie qui ne requiert ni un degré élevé d'interaction avec l'utilisateur ni un degré élevé d'expertise.
- Il peut facilement être appris.
- Il a un fondement mathématique (basé sur les théories des graphes … etc.).
- Ainsi, un avantage majeur du model-checking réside dans la précision de la réponse obtenue. Un model-checker répondra en indiquant si la propriété est vérifiée ou non alors que les techniques de test et d'interprétation abstraite auront des réponses plus floues et ne permettront pas d'affirmer avec certitude que le système est correct.
- De plus, cette procédure est totalement automatique : la tache de l'utilisateur se résume à fournir au model-checker un modèle ainsi que la propriété à vérifier. Notons toutefois que certains algorithmes ou optimisations peuvent nécessiter des informations fournies par l'utilisateur.

III.7.4 - ***Limites du model-checking :*** [30][34]

- Le nombre d'états nécessaire pour modéliser le système risque de dépasser la mémoire de l'ordinateur, le modèle en question ne peut donc pas être exploré par les algorithmes faute de ressources (temps et mémoire).
- La plupart des algorithmes de model-checking supposent que l'espace d'état du système étudié est fini.
- Si l'on souhaite vérifier un programme à l'aide d'un model-checker il faudra s'assurer que celui-ci consomme une quantité de mémoire finie. Par conséquent, le model-checking est théoriquement inapplicable à des programmes qui allouent dynamiquement de la mémoire ou qui contiennent des appels récursifs.

IV - *Conclusion :*

Le but de ce chapitre, est de donner une idée détaillée sur la vérification des modèles des processus métiers et les évolutions qu'elle a subies au fil du temps.

Les résultats présentés dans ce chapitre sont le fruit d'une étude détaillée sur l'état de l'art portant sur les techniques et les méthodes de vérification de modèle de processus métier.

Nous avons synthétisé les résultats obtenus dans le tableau suivant :

	Exhaustivité :	Vérification de RdP ordinaire :	Propriétés comportementales :	Apport d'un contre exemple :	Localisation de l'erreur :	Correction d'erreur :	Structure du RdP :	Total (+) :
Preuve de théorème :	-	+	Absence de blocage. (+)	+	-	-	Logique temporelle	3
WoFlAn :	-	+	Absence de blocage. (+)	-	+	-	PNML	3
WoFBPEL :	-	- (processus BPEL)	Atteignabilité. (+)	-	-	-	BPEL traduit en PNML	1
PIPE :	-	- (GSPN)	Bornitude, sûreté, absence de blocage. (+++)	-	- (seulement pour le blocage)	-	RdP graphique	3
TiNA :	-	- (temporisé)	Atteignabilité, bornitude, vivacité, absence de blocage. (++++)	+	+	-	RdP graphique ou textuel	6
LoLA :	+	- (réduit)	Absence de blocage, atteignabilité, vivacité, bornitude, réversibilité, état d'accueil. (++++++)	+	+	-	Fichier ASCII	9
Model-checking :	+	+	Absence de blocage, atteignabilité, vivacité, quasi-vivacité, sûreté, bornitude, réversibilité, état d'accueil, réseau sauf. (+++++++++)	+	+	-	Kripke Promela …	13

Tableau 3 : Comparaison des différentes techniques.

Après avoir étudié dans la littérature, les approches permettant la vérification de modèles de processus, Nous déduisons que le Model-checking est potentiellement avantageux par rapport aux autres techniques de vérification, vu son exploration exhaustive prenant en compte la vérification de toute exécution possible. Chose qui permet au model-checking de vérifier le maximum de propriétés comportementales.

Cette étude comparative nous a également permis de choisir qu'elle approche utiliser afin de vérifier nos modèles de processus métier.

Notre approche va être présentée en détail tout le long du prochain chapitre (chapitre 3).

Chapitre III

Choix de l'approche à utiliser

I - Introduction :

Il existe, comme vu dans le chapitre précédant, une multitude de techniques de vérification automatiques, la technique du model-checking qui est l'une des plus connues et des plus utilisées s'avère être l'une des techniques les plus efficaces. L'efficacité de cette technique réside dans les diverses méthodes qu'elle possède, chacune de ses méthodes contribue dans l'amélioration de la technique.

Durant ce chapitre, nous allons aborder la vérification par model-checking. Nous y verrons plus en détail le fonctionnement de cette technique en y exposant les différentes méthodes qu'elle utilise pour assurer une vérification exhaustive fiable. Nous citons à titre d'exemple : les méthodes résolvant l'explosion combinatoire de la taille de l'espace d'état (la réduction d'ordre partielle, le hash compacting . . .), la mise en œuvre (les algorithmes, les techniques utilisés, les outils . . .) etc. Nous verrons également l'approche du traitement automatique des spécifications formelles et son principe de fonctionnement, présentant ainsi le choix de la méthode à utiliser.

II - Définition générale de la technique du model-checking :

L'application des méthodes formelles, y compris la méthode du model-checking, dans différents domaines a vu un grand succès. Ces méthodes permettent, à partir d'une description appelée spécification du système, de vérifier les propriétés souhaitées de manière automatique avant la phase d'implémentation. Le model-checking dispose d'algorithmes qui vérifient la totalité du système en question en explorant exhaustivement tous les états du système qui est généralement modélisé par un graphe où l'exploration consiste à vérifier si chaque chemin d'exécution satisfait ou non la ou les propriétés désirées. Afin d'éviter des réexplorations d'état, les états explorés sont gardés dans la mémoire. Le graphe qui modélise le système peut être soit de taille raisonnable ou au contraire de très grande taille comme l'exige beaucoup de systèmes logiciels réels. Dans le premier cas, les méthodes de model-checking sont très efficaces, cependant, dans le deuxième cas, l'explosion combinatoire de la taille de l'espace d'état constitue un obstacle considérable pour la vérification automatique [35].

III - Model-checking explicite et symbolique :

En informatique, la vérification des modèles ou la vérification des propriétés se réfère au problème suivant : Etant donné le modèle du système, il faut vérifier automatiquement et de manière exhaustive si ce dernier répond à un cahier des charges donné. Typiquement, on a

des systèmes matériels ou logiciels à l'esprit, alors que la spécification contient des exigences de sécurité telles que l'absence de blocages et états critiques similaires qui peuvent causer un crash du système. La vérification par model-checking est une technique permettant de vérifier automatiquement les propriétés de régularité des systèmes à états finis [34][35].

Donc il existe deux grandes approches au problème du model-checking : l'approche explicite et l'approche symbolique :

III.1 - ***Approche explicite :***

L'approche explicite est basée sur une représentation explicite de l'espace d'état : chaque état est présent explicitement en mémoire dans une structure de données utilisée pour stocker l'espace d'état du système, p.ex., une table de hachage. La construction de cet espace s'effectue par un parcours de graphe [34].

III.2 - ***Approche symbolique :***

L'approche symbolique, répandue pour la vérification de propriétés exprimées dans la logique temporelle arborescente (CTL), est fondée sur la représentation des états et des transitions du système par des ensembles [34], où un état / transition symbolique représente un ensemble d'états / transitions explicites [35]. Elle est apparue plus tard avec les travaux de Kenneth McMillan et Ed Clarke, et se distingue par les structures de données qu'elle utilise et les algorithmes qu'elle met en œuvre. De nombreuses méthodes de représentation d'ensembles d'états ont vu le jour. La plus connue utilise les diagrammes de décision binaire (BDD) est une structure de données compactes sous forme d'arbre binaire qui permet de représenter efficacement des fonctions booléennes en permettant un partage fort entre les nœuds de l'arbre [34].

Le fonctionnement d'un model-checker symbolique consiste en l'obtention de points fixes pour déterminer les états accessibles ainsi que ceux qui satisfont une ou plusieurs propriétés. Ces points fixes sont calculés à l'aide de deux transformateurs de prédicats associés au franchissement des transitions : Le premier (Post) détermine, pour un ensemble d'états donnés, l'ensemble des états successeurs tandis que le second (Pré) est l'opération réciproque du premier. Ces techniques sont souvent plus efficaces pour les systèmes présentant un fort degré de concurrence mais font intervenir des mécanismes trop lourds pour des systèmes séquentiels [34].

IV - *Approche de fonctionnement par automates :*

Cette approche consiste, généralement, à transformer, en premier lieu, la spécification formelle du système en question en une structure de kripke, puis de considérer la négation de la formule à vérifier (propriété exprimée en logique temporelle) et de la transformer en un automate de Büchi. L'opération qui suit ces deux transformations consiste à calculer le produit synchronisé des deux structures (la structure de kripke représentant la spécification formelle du comportement du système et l'automate de Büchi qui reflète la négation de la propriété à vérifier). Le processus de vérification est traduit par le langage de l'automate résultant d'un tel produit. Ainsi, le fait de déterminer si le système satisfait ou non la propriété souhaitée consiste à vérifier si le langage est vide ou non. Dans le premier cas, la propriété est bien vérifiée, sinon, la propriété est violée et les constituants du langage représentent des contre-exemples [16][35].

Bien que cette approche soit efficace en terme de vérification, elle nécessite beaucoup d'espace mémoire et conduit très souvent au problème de l'explosion combinatoire [16].

Remarques :

A - ***Automate de Büchi*** : c'est un quintuplet : $A = \{\Sigma, Q, \delta, Q_0, F\}$ où : [36]

- Σ est un ensemble fini de lettres.
- Q un ensemble fini d'états, δ la relation de transition.
- Q_0 l'ensemble des états initiaux.
- F l'ensemble des états d'acceptation.

B - ***Structure de kripke*** : c'est une variante du système de transition [37]. Elle est représentée pas le triplet suivant : M = (UM,IM,RM) où : [16][38][39][32]

- UM est un ensemble où les éléments sont appelés selon le domaine d'application (ensemble d'états par exemple).
- IM une fonction définie comme suit : IM : Variables → P(UM) où IM(P) est l'ensemble des états où la variable P est satisfaite.
- RM est un ensemble de relations (appelées relations d'accessibilité), si u(Ri)v alors l'état v est accessible à partir de u pour i.

C - La propriété à vérifier est introduite en logique temporelle, il y a donc le choix entre LTL (Logique Temporelle Linéaire), CTL (Logique Temporelle Arborescente) et CTL* (Logique Temporelle combinant LTL et CTL).

V - *Problème de l'explosion combinatoire de la taille de l'espace d'états :*

L'explosion combinatoire de la taille de l'espace d'états se traduit par le fait que le nombre des états du système augmente exponentiellement en fonction du nombre de ses composants [35].

Cependant, on peut pallier ce problème, comme nous le verrons par la suite, soit en utilisant des techniques spéciales visant principalement à combattre l'explosion combinatoire, soit en utilisant une approche différente de celle des automates.

V.1 - *Combattre l'explosion combinatoire :*

Le model-checking a longtemps été considéré comme une technique non adaptée pour les systèmes industriels et assez immature à cause du problème d'explosion combinatoire. Cela fut la principale raison de concevoir des méthodes et des techniques pouvant combattre ce phénomène de manière efficace. Le model-checking s'est désormais largement répandu au sien de l'industrie [34].

Afin de remédier à ce problème, plusieurs méthodes ont été développées. Nous allons ci-dessous présenter les principales méthodes utilisées pour combattre l'explosion combinatoire.

V.1.1 - *Les méthodes déterministes :*

V.1.1.1 - *Réduction de la mémoire nécessaire pour le stockage des états :*

A - ***Hash compaction :*** Cette méthode vise à réduire la mémoire nécessaire pour le stockage des descripteurs d'état. Notons qu'un descripteur d'état contient les valeurs des variables dans l'état en question. Contrairement à la vérification par énumération explicite où on sauvegarde en mémoire tout le descripteur de l'état, la méthode du hash compaction sauvegarde, au lieu du descripteur entier, un descripteur d'état compressé que l'on calcule à travers une fonction de hachage [35].

B - ***Bit-state hashing :*** Cette méthode sauvegarde en mémoire un ensemble de bits initiés à zéro (0) au lieu de garder l'ensemble de descripteurs d'état, ce qui minimise considérablement

la mémoire nécessaire pour le stockage des états. La fonction de hachage s'applique sur le descripteur de chaque état lors de sa visite en donnant deux valeurs de hachage, les bits dont les indices correspondent à ces valeurs sont mis à un (1). Ceci facilite l'exploration sans redondance, car à chaque nouvel état il suffit de tester si les deux bites sont à un (1) ou à zéro (0), s'ils sont à un (1) alors cet état est déjà visité. Cependant, cette fonction de hachage est généralement non injective, donc deux descripteurs différents qui ont les mêmes valeurs de hachage peuvent avoir un ou deux bits en commun, en conséquence de la visite d'un état, il se peut que les deux bits d'un autre état non visité soit mis à un (1) résultant ainsi la non exploration d'un état non visité et donc l'omission de la vérification de tous les états du système [35].

V.1.1.2 - ***Réduction de la taille de l'espace d''etats :***

A - ***Réduction d'ordre partiel :*** Cette technique utilise un algorithme de model-checking afin de réduire la taille de l'espace d'état à explorer résultant des graphes d'états à traiter de taille considérablement réduite. L'idée principale de cette technique est qu'elle explore seulement un espace d'état réduit et suffisant pour la vérification des propriétés, et ce, en exploitant la commutativité des transitions qui donnent le même état lors de leurs exécutions [35]. La réduction d'ordre partiel vient pour résoudre l'une des principales causes du problème, notamment l'exécution concurrente de plusieurs processus. Cette technique réduit l'explosion d'états en n'explorant qu'une ou quelques-unes des séquences d'exécutions ayant le même effet sur le système [34].

B - ***Réduction symétrique :*** Dans les systèmes d'états finis, on trouve généralement beaucoup de composants identiques dits symétriques [35]. Ces symétries peuvent, en effet, être utilisées pour réduire le nombre d'états visités [34]. On dit que deux états sont symétriques s'ils ont sémantiquement les mêmes variables d'états. Considérons les trios (3) états symétriques suivants : (X,Y,Y), (Y,X,Y) et (Y,Y,X) ces trois états, ayant deux états locaux possibles, forment une classe d'équivalence qui peut être représentée comme suit (X = 1 , Y = 2). Dans cette technique, la réduction de l'espace d'états réside dans l'exploitation des symétries. Cela se fait par la décomposition de l'espace d'état en classes d'équivalence d'états symétriques. Donc, l'exploration du model-checking ne va pas passer par touts les états mais par un état de chaque classe d'équivalence. Cependant, l'identification et le calcul de ces classes d'équivalence peut constituer une tâche très difficile, surtout pour les systèmes complexes [35].

V.1.2 - *Les techniques de remplacement :*

V.1.2.1 - *Principe des techniques de remplacement :*

L'idée principale des techniques de remplacement est la gestion de la sauvegarde des états en mémoire. Ces techniques tranchent vers une solution adéquate qui vise à résoudre les problèmes de stockage, car si l'on garde tous les états en mémoire, cette dernière sera insuffisante et cela résultera un abandon de l'exploration, si, par contre, on ne garde aucun état en mémoire, on tombera dans le problème de redondance. C'est pourquoi les techniques de remplacement sont indispensables pour pallier le problème de stockage. Il s'agit en effet de garder le plus d'états possibles en mémoire et d'effectuer des remplacements pour le but d'optimiser le gain d'espace mémoire. Pour ce faire il existe deux différentes formes. La première est de vider toute la mémoire après que celle-ci soit pleine et de réinitialiser l'algorithme d'exploration puis de relancer. La deuxième consiste à remplacer classiquement état par état, c.à.d. chaque nouvel état à explorer prend la place d'un ancien état en mémoire. La stratégie de remplacement se charge de choisir lequel des états sacrifier [35].

V.1.2.2 - *Le remplacement des états en mémoire :*

L'exploration exhaustive du model-checking vise à parcourir état par état en vérifiant à chaque nouvel état si celui-ci n'a pas déjà été visité en recherchant son existence dans la mémoire parmi les états visités. S'il n'y existe pas, alors on l'explore et on l'ajoute dans la mémoire. Cependant, il se peut que la mémoire soit pleine et qu'il n'y aura pas de place pour les nouveaux états à explorer. La technique de remplacement ne garde qu'un sous-ensemble d'états en mémoire tout en réussissant l'exploration exhaustive. Si on suit le fondement de cette dernière, on peut vite constater que le fait de remplacer un état par un autre dans la mémoire ne va pas fournir un résultat correct à 100% quant au test des états visités, car il se peut qu'on explore un nouvel état suite au résultat négatif de la recherche de son existence en mémoire alors qu'il a déjà été exploré et supprimé de la mémoire (remplacé par un autre état). Ceci va causer, bien évidemment, une redondance d'exploration, néanmoins, selon l'expérience, ça ne donne pas de résultat incorrect et malgré la redondance l'exploration totale a été obtenue dans plusieurs cas [35].

V.1.2.3 - *Stratégies de remplacement :*

Il faut disposer d'une stratégie de remplacement qui va guider le choix de l'ancien état à remplacer. Pour cela, il existe plusieurs politiques dont : [35]

1. Les états visités le plus fréquemment.
2. Les états visités le moins fréquemment.
3. Les états faisant partie de la classe d'états la plus large actuellement (classement par le nombre de visites).
4. Les états les plus anciens dans la mémoire.
5. Les états faisant partie de la moitié inférieure de l'arbre de recherche.
6. Le remplacement aléatoire, constituant la stratégie la plus utilisée dans la littérature, consiste à choisir, de manière aléatoire dans la mémoire, un nœud et de le remplacer par le nouvel état.

V.1.2.4 - ***Modélisation :***

Pour suivre le principe de l'exploration en utilisant la technique de remplacement, l'algorithme d'exploration devra avoir trois paramètres, le premier est la condition d'arrêt de l'exploration, le deuxième est la fonction qui sélectionne l'état suivant à explorer et le troisième est une fonction définissant la stratégie de remplacement en actualisant l'espace d'état en mémoire [35].

V.1.2.5 - ***Algorithme d'exploration :***

A - ***Critères recherchés :*** Les performances des algorithmes d'exploration qui sont destinés à la vérification se mesurent généralement par deux principaux critères notamment : la couverture et l'atteignabilité [35].

A.1 - ***La couverture :*** La couverture se traduit par la capacité d'explorer l'espace d'état. Une bonne couverture donne une exploration moins redondante. Le taux de redondance se traduit par la formule suivante :

$$\tau = \frac{n-k}{n}.$$

Où « k » est le nombre de nœuds dans le graphe et « n » est le nombre d'étapes de son exécution. On sait que dans un algorithme d'exploration, on peut soit visiter un nouvel état soit revisiter un état, et ce à chaque étape. Dans le premier cas, on incrémente « n » et « k » à la fois, tandis qu'en deuxième cas, on incrémente seulement le nombre « n » ce qui augmente la redondance [35].

A.2 - ***L'atteignabilité :*** L'atteignabilité représente la capacité d'atteindre les états du graphe par l'algorithme, en mettant l'accent sur l'atteinte des états qui présentent une erreur [35].

B - ***Schéma général des algorithmes d'exploration :*** [35]

Structures de données :

- ⇨ P : Paramètres de l'algorithme (statique) ;
- ⇨ I : Informations sur le graphe (dynamique) ;
- ⇨ V : Ensemble des nœuds gardés en mémoire ;
- ⇨ v : nœud courant ;

Initialisation :

- ⇨ V ← {v0} ;
- ⇨ v ← {v0} ;

Corps de l'algorithme :

- ⇨ Tant que (¬ condition d'arrêt) faire
  - ○ v ← sélectionner (V, P, I) ;
  - ○ visiter (v) ;
  - ○ (V, I) ← actualiser (V, v, P, I) ;
- ⇨ Fait

Où P représente les paramètres d'entrée de l'algorithme, nous citons à titre d'exemple la propriété à vérifier, I possède des informations globales sur la structure du graphe [35].

B.1 - ***La condition d'arrêt :*** Intuitivement, la condition d'arrêt est représentée par l'exploration totale de l'espace d'état. Cependant, on peut ne pas arriver jusque là, dans ce cas la condition d'arrêt peut être, par exemple, l'atteinte du pourcentage de couverture désiré, l'épuisement de la mémoire . . . etc. [35]

B.2 - ***La fonction « sélectionner » :*** Cette fonction prend en charge le choix du prochain nœud à visiter parmi les successeurs des nœuds visités qui sont stockés dans « V ». Un tel choix peut être guidé par les paramètres « P » ou encore par les informations « I » comme le nombre de successeurs . . . etc. [35]

B.3 - ***La fonction « actualiser » :*** La fonction d'actualisation prend en charge la mise à jour de l'ensemble « V » ainsi que les informations I suite à chaque étape de l'exploration, permettant ainsi une évolution dans le sens désiré. La fonction d'actualisation s'intéresse à la stratégie de stockage, c.à.d. au remplacement éventuel des états en mémoire tout respectant l'ordre et le format de stockage dans « V ». Comme nous l'avons cité précédemment, il existe deux solutions pour le problème d'insuffisance de mémoire dans le cas de graphe de taille importante. La première est de remplacer état par état (ancien état par nouvel état) et la deuxième est de réinitialiser l'algorithme quand la mémoire est plaine. On note que la deuxième approche est la plus simple et la plus utilisée pour les algorithmes randomisés d'exploration [35].

V.1.3 - ***Les techniques symboliques :***

Elles visent à représenter l'espace d'états de telle façon à ce que le partage de données soit fort entre les différents états, et ce, à l'aide des structures de données. La plus utilisée de ces dernières est les diagrammes de décisions binaires (Binary Decision Diagram - BDD). Un BDD est une représentation canonique permettant d'accélérer la recherché de manière non négligeable en appliquant des algorithmes qui lui sont efficacement adaptés [34].

A ces techniques, s'ajoute un nombre de techniques d'abstraction pouvant être appliquées directement sur le modèle et pas durant la recherche. Ces techniques utilisent le principe de transformer le modèle initial en un modèle plus simple à l'égard de la vérification, par exemple la réduction de réseau de Petri [34].

La liste des techniques présentées ci-dessus est loin d'être complète. Il est à souligner que chacune de ces méthodes s'avère être efficace pour certains systèmes (qui lui sont adéquats) et pas tous, dans le sens que, d'un côté, elle peut sembler inadaptée pour un certain modèle mais, de l'autre côté, elle réduit un autre système exponentiellement [34].

V.2 - ***L'utilisation d'autres approches :***

Nous pouvons, en effet, penser à utiliser la méthode du model-checking et suivre son principe mais en procédant différemment. Pour ce faire, il faut trouver un moyen d'extraire toutes les données qu'il nous faut de la spécification formelle du système. A partir de cette idée générale, nous pouvons nous inspirer des techniques existantes dans le monde de l'informatique et qui partent d'une telle spécification.

Dans ce qui suit nous allons décrire l'approche que nous avons utilisée.

V.2.1 - *Besoin d'une spécification formelle :*

La modélisation des systèmes via des réseaux de Petri constitue une spécification semi-formelle puisque ces derniers forment une représentation graphique. Bien que cette modélisation apporte une contribution positive à l'analyse du problème conduisant ainsi à un affinement du problème, on ne pourra pas appliquer une analyse formelle sur ce modèle. En effet, il se peut qu'il y ait, dans l'analyse des spécifications semi-formelles, certaines ambigüités menant à des bugs du système. Ceci constitue le besoin d'une spécification formelle, cette dernière exprime et décrit le quoi d'un système dans un langage formel [40].

V.2.2 - *Traitement Automatique de la Langue (TAL) :*

Nous allons, ci-dessous, présenter une technique qui nous a inspiré, soit le traitement automatique de la langue (TAL).

Les premiers travaux importants du traitement automatique de la langue datent de 1957 et portent sur la traduction automatique, plus précisément sur la traduction automatique de la langue russe en anglais [41]. De nos jours le traitement automatique de la langue s'est beaucoup élargi et touche désormais des domaines divers. Ceci a causé une certaine pression sur l'ingénierie linguistique et la demande des logiciels performants de traitement automatique de la langue s'est considérablement accrue, menant ainsi au développement de deux stratégies complémentaires d'analyse syntaxique qui sont : l'analyse de surface et l'analyse profonde [42].

V.2.2.1 - *Analyse de surface (shallow parsing) :*

L'avènement d'une telle analyse vient de la difficulté de la réalisation d'une analyse complète. En effet cette dernière peut s'avérer très compliquée, il est donc préférable d'effectuer une première analyse dite de surface permettant de répondre aux questions suivantes : [42]

- Quelles sont les frontières des unités linguistiques de base ?
- Quelle est la nature syntaxique des constituants ?

Autrement dit, cette analyse vise à étudier la structure générale du document à analyser permettant notamment d'extraire et de filtrer les informations [42].

V.2.2.2 - ***Analyse profonde (deep parsing) :***

Comme mentionné précédemment cette analyse complète l'analyse de surface en approfondissant l'analyse des constituants du document. Cette analyse permet de répondre aux questions suivantes : [42]

- Est-ce que le constituant en cours d'analyse est bien formé ? Ex : la phrase appartient-elle à la grammaire de la langue prise en considération ?
- Quelles structures syntaxiques explicites peut-on attribuer au constituant analysé ?

V.2.3 - ***Traitement automatique et spécification formelle :***

Les techniques du TAL s'étendent et s'appliquent à d'autre projets que ceux du traitement automatique du langage naturel, notamment au domaine d'ontologie, où l'analyse de surface analyse la structure d'un document, donnant ainsi forme et sens au contenu, ce qui résulte un premier noyau d'ontologie. Quant à l'analyse profonde, elle vise à expliciter le contenu afin de le formaliser sous la forme d'ontologie [43]. Ainsi, l'analyse syntaxique a joué un grand rôle dans le développement des compilateurs. En effet, elle réside en l'analyse et la traduction automatiques des programmes (ou suites d'instructions) écrits dans un langage de programmation en langage-machine [42].

VI - ***Choix de l'approche à utiliser :***

Notre choix consiste à suivre le principe général du model-checking, c.à.d. les entrées (le modèle du système à vérifier et les propriétés souhaitées), les sorties (résultats de la vérification plus les contre-exemples si la propriété en question est violée) et le fonctionnement (exploration exhaustive, algorithmes …etc.). Ainsi, partant du principe que l'on a, comme donnée, une spécification formelle qui exprime le réseau de Petri représentant le système à vérifier, nous pouvons y appliquer une analyse syntaxique. Après avoir lu le fichier contenant la spécification formelle, la mise en pratique de l'analyse de surface pourra servir, par exemple, à manipuler la structure du fichier et de détecter, éventuellement, certaines erreurs (puisque c'est une spécification formelle portant sur un seul type d'objet qui est un réseau de Petri, la structure générale des fichiers devra être la même). Ainsi, l'analyse profonde nous fournira toutes les données nécessaires à l'étape de vérification.

La combinaison de la méthode du model-checking et du traitement automatique de spécification formelle constitue notre contribution au sein de la vérification des processus métier.

VII - *Les propriétés vérifiées par le model-checking :*

VII.1 - *Absence de blocage/Pseudo-vivacité « Deadlock-freeness » :*

L'absence de blocage exprime le fait que le système ne peut pas se trouver dans une situation où il est impossible d'évoluer. Cette propriété est importante dans le sens où elle constitue une propriété de correction pour les systèmes supposés ne jamais se terminer [16][44][45]. Autrement dit :

(R,M_0) est pseudo-vivant $\leftrightarrow \forall m \in M(R,M_0), \exists t \in T$ tel que $m[t>$. où R est le réseau, M_0 son marquage initial et T l'ensemble de transitions [46][47].

VII.2 - *Quasi-vivacité :*

Cette propriété exprime le fait qu'au sein du système, chaque fonctionnalité a été exécutée au moins une fois, ainsi, le système ne possède pas de fonctionnalité inutile. On dit qu'un réseau est quasi-vivant si toutes ses transitions le sont [16][46][47].

Transition quasi-vivante : $t \in T$ est quasi-vivante $\leftrightarrow \exists m \in M(R,M_0)$ tel que $m[t>$. [46]

VII.3 - *Vivacité « Liveness » :*

Une propriété de vivacité *"Liveness property"* désigne qu'une situation désirable, sous certaines conditions, finira par avoir lieu [16][46]. Vis-à-vis d'un système, elle signifie qu'à tout moment et quelque soit l'état du système, il dispose encore de toutes ses fonctionnalités. On dit qu'un réseau est vivant si toutes ses transitions le sont [46][47][48].

Transition vivante : $t \in T$ est vivante $\leftrightarrow \forall m \in M(R,M_0)$, t est quasi-vivante pour m. [46]

VII.4 - *Le caractère borné (K-Bornitude) :*

Cette propriété est liée au marquage du réseau et donc au nombre de jetons dans les places. On dit qu'un réseau est borné, si toutes ses places le sont [46][47].

Place bornée : $p \in P$ est K-bornée $\leftrightarrow \forall m \in M(R,M_0)$, $m(p) \leq K$. [46]

VII.5 - ***Réseau sauf :***

Cette propriété est un cas particulier de celle de K-Bornitude. On dit qu'un réseau est sauf, s'il est K-Borné où k=1 (il est donc 1-Borné) [46][47].

VII.6 - ***Atteignabilité « Reachability » :***

La propriété d'atteignabilité ou d'accessibilité *"Reachability property"* représente le fait qu'une certaine situation précise peut être atteinte à partir de l'état initial du système, donc dans un tel cas on devra éprouver un ensemble de mauvais états A qui peut être atteint comme le fait « qu'*il existe un état accessible où x vaut 0* » [16][44].

Donc il s'agit de déterminer s'il existe au moins un chemin menant de l'état initial à un état vérifiant la propriété souhaitée [16][44][45].

VII.7 - ***Sûreté « Safety » :***

La propriété de sûreté *"Safety property"* est une situation indésirable qui ne doit jamais se produire. Elle peut se vérifier par parcours exhaustif de l'ensemble des états. Dans le cas où la propriété de sûreté est violée, on obtient un contre-exemple fini, c.à.d. une exécution finie qui viole la propriété. Ainsi, l'invariance est un cas particulier de sûreté, Par exemple, « une variable n'est jamais égale à 0», « le système n'a pas de deadlock » [16][45][46]. Ce type de propriété doit donc être vérifié et satisfait à chaque étape de l'exécution du système [16].

VII.8 - ***Etat d'accueil :***

Un réseau possède un état d'accueil A s'il existe une séquence S telle que $M_i[S>A$ pour tout M_i, avec M_i un marquage accessible [46]. Autrement dit :

R possède un état d'accueil A $\leftrightarrow \forall m \in M(R,M_0), \exists S \in T^*$ tel que $m[S>A$. [46]

VII.9 - ***Réversibilité :***

La propriété de réversibilité est un cas particulier de celle de l'état d'accueil. On dit qu'un réseau et réversible (ou réinitialisable) si le marquage initial est un état d'accueil. Pour un système cela signifie que quelque soit la séquence d'exécution prise en compte, le système est réinitialisable [46].

VIII - *Conclusion :*

La finalité de ce troisième chapitre est d'étudier de manière plus approfondie la technique du model-checking et ses différentes approches. Les principales composantes qui permettent le bon fonctionnement du model-checking y sont détaillées. La composante la plus importante est l'exploration exhaustive qui est une particularité de cette technique. Nous avons expliqué comment une telle exhaustivité peut fonctionner efficacement face à la très grande taille des systèmes réels. Nous sommes aussi passées par les propriétés que le model-checking permet de vérifier en expliquant comment ces dernières peuvent être exprimées formellement. Nous avons également vu une approche qui porte sur le traitement automatique de spécification formelle. Ainsi nous sommes parvenues à choisir la méthode de vérification à utiliser, soit, la combinaison entre model-checking et traitement de spécification formelle.

A présent, nous avons en main tout ce qu'il faut pour implémenter un outil de vérification (modèle en entrée et méthode de vérification). Nous allons voir, dans le chapitre suivant, comment implémenter et mettre en œuvre un tel outil.

Chapitre IV
Implémentation d'un vérificateur de modèles de processus

I - *Introduction:*

Afin de pouvoir accomplir tout projet, il est nécessaire de réaliser un système répondant à tous les besoins afin de satisfaire le client. Ce chapitre porte sur l'implémentation du vérificateur de modèle de processus métier.

La réalisation de ce projet a nécessité certains outils pour la programmation de l'application et une certaine connaissance du langage de programmation avec lequel l'application a été programmée ainsi que les outils qui ont été utilisés.

Au cours de ce chapitre, nous allons voir comment nous avons pu implémenter notre outil en partant du choix de la spécification formelle à utiliser (représentant le réseau de Petri) jusqu'à la mise en œuvre d'un ensemble d'algorithmes d'exploration et de techniques d'analyse.

Ainsi, nous allons présenter un déroulement d'un exemple d'un réseau de Petri dans le but d'une meilleure visualisation des différentes fonctionnalités dont l'outil dispose.

II - *Outils de développement utilisés :*

Le développement logiciel est une activité dont l'importance ne cesse de croître. Cela est dû au déploiement général de l'informatique. Dans beaucoup de domaines, le processus de développement joue un rôle critique dans le temps de mise sur le marché d'un logiciel. En effet, l'implémentation décrit précisément la façon dont l'application sera paramétrée afin de répondre aux besoins fonctionnels de l'entreprise.

Toute implémentation nécessite un ensemble d'outils de développement et de langage de programmation. Afin de mener à bien la réalisation de notre outil, nous avons utilisé le langage de programmation Java qui nous a permis de mettre en œuvre toutes les fonctionnalités du système. En plus les logiciels programmés avec ce langage doivent être très facilement portables sur plusieurs systèmes d'exploitation tels que UNIX, Windows, Mac OS ou GNU/Linux, avec peu ou pas de modifications. Pour cela, diverses plateformes et frameworks associés visent à guider, sinon garantir, cette portabilité des applications développées en Java.

Les fonctionnalités de notre outil ont été développées en utilisant Eclipse qui est un environnement de développement (IDE) historiquement destiné au langage Java. Ainsi, nous nous sommes servies de Netbeans pour la construction des interfaces graphiques.

III - *Spécification formelle :*

Comme mentionné dans le chapitre précédent, nous avons besoin d'une spécification formelle qui représente le modèle du processus métier à vérifier qui est modélisé par un réseau de Petri.

D'après les recherches effectuées, il existe plusieurs spécifications pouvant formaliser un réseau de Petri. Nous avons pu constater que, selon leurs emplois, certaines spécifications sont plus adéquates que d'autres.

III.1 - *Choix de la spécification formelle :*

Selon les critères que nous avons fixés par rapport à l'emploi de la spécification au sein de notre travail, nous avons abouti à un choix d'une spécification formelle qui fera l'objet d'entrée de l'outil à réaliser. Une telle spécification peut être fournie par l'outil TiNA. En effet, au cours du chapitre 2, nous sommes passées par la présentation de cet outil qui dispose de plusieurs modules, dont l'éditeur graphique nd (netDraw). Cet éditeur permet à l'utilisateur d'éditer graphiquement un réseau de Petri et de l'exporter sous plusieurs formats, dont le format « tpn ». Toutefois, l'utilisateur peut se tromper en éditant le réseau et ainsi l'exporter sans que netDraw l'avertisse. Donc la spécification exportée doit être analysée afin de corriger les éventuelles erreurs. Ainsi, notre outil dispose d'un éditeur textuel permettant à l'utilisateur d'éditer la spécification formelle représentant son réseau de Petri. Cette dernière passe par une analyse plus rigoureuse du fait que l'utilisateur a la liberté d'écrire du texte informel.

Dans le chapitre précédant (chapitre 3), nous avons expliqué que nous allons vérifier les modèles de processus métier complexes en appliquant un traitement automatique sur la spécification qui formalise ces modèles et c'est exactement pour cette raison que nous avons choisi le format « tpn » comme spécification formelle. Cette dernière est en effet très optimale car elle permet d'exprimer un réseau de Petri textuellement formant un fichier texte de taille réduite par rapport à la plupart des autres spécifications, ainsi, la syntaxe de celle-ci est très simple et facile à comprendre et surtout à exploiter.

IV - *Analyse de la spécification formelle :*

Il s'agit ici de s'inspirer des techniques de traitement automatique de la langue et d'en appliquer celles qui sont nécessaires sur notre spécification. En effet, ce ne sont pas toutes les techniques qui seront utilisées car la tâche de traitement est moins compliquée puisqu'il s'agit d'un traitement automatique d'une spécification formelle.

Le traitement automatique de la langue repose sur deux analyses complémentaires : l'analyse de surface et l'analyse profonde. Ces deux analyses visent à exploiter du texte à partir de sa syntaxe.

IV.1 - *Application de l'analyse de surface sur la spécification formelle :*

L'analyse de surface consiste à analyser la structure générale du texte et de reconnaitre la nature des constituants. Puisqu'il s'agit d'une spécification formelle, la structure et la nature des composants du fichier devraient être les mêmes. Cette analyse nous a donc permis, en premier lieu, de repérer l'existence ou pas de certaines erreurs qui portent sur la structure du fichier en entrée, elle nous a permis également d'extraire les composants du graphe comme les places, les transitions, les étiquettes … etc.

IV.2 - *Application de l'analyse profonde sur la spécification formelle :*

Cette analyse, comme son nom l'indique, est plus profonde que l'analyse de surface car elle étudie, non seulement la syntaxe de chaque élément, mais aussi, comment les éléments sont combinés entre eux. Effectivement ceci peut s'avérer très utile, dans le sens où on veut vérifier si un constituant (élément ou ensemble d'éléments) est bien formé et s'il appartient bien à la grammaire de la spécification formelle. L'analyse profonde permet donc de détecter des erreurs éventuelles concernant le graphe, ainsi, elle permet d'extraire plus d'informations nécessaires à l'étape de vérification qui peuvent révéler, par exemple, si une transition est franchissable ou non, le marquage initial du réseau de Petri … etc.

V - *Vérification des propriétés :*

V.1 - *Propriétés :*

Les propriétés à vérifier dans notre travail sont celles qu'on a évoquées au cours du troisième chapitre. Certaines de ces propriétés peuvent ne concerner que le réseau tandis que d'autres peuvent être liées, non seulement au réseau, mais aussi, à des transitions, des places

ou bien des états. Pour une meilleure lisibilité, nous avons organisé les propriétés dans des catégories comme suit :

V.1.1 - ***Propriétés liées aux places :***

- Une place est-elle bornée ?
- Une place est-elle sauve ?

V.1.2 - ***Propriétés liées aux transitions :***

- Une transition est-elle quasi-vivante ?
- Une transition est-elle vivante ?

V.1.3 - ***Propriétés liées aux états :***

- Un état est-il un état d'accueil ?
- Un état est-il sûr ?
- Un état est-il atteignable ?

Remarque :

En ce qui concerne la sûreté et l'atteignabilité d'un état, on peut procéder de deux manières différentes. La première est de considérer un état comme un marquage, donc la propriété en question doit prendre comme entrée un marquage. La deuxième consiste à considérer un état comme une étiquette ou un ensemble d'étiquettes, il s'agit donc d'avoir comme entrée une ou plusieurs étiquettes. La deuxième approche n'est applicable qu'en cas d'un réseau étiqueté.

V.1.4 - ***Propriétés liées au réseau :***

- Absence de blocage.
- Quasi-vivacité.
- Vivacité.
- Bornitude.
- Réseau sauf.
- Réversibilité.
- Existence d'un état d'accueil.

V.2 - ***Principe de vérification :***

Notre principe de vérification suit celui du model-checking. Il s'agit donc de vérifier si la propriété en question est vérifiées ou non et de retourner un contre-exemple dans le cas où cette dernière est violée. Dans notre travail on est en mesure de retourner tous les contre-exemples qui violent la propriété en cours de vérification, et ce, pour apporter une aide à l'utilisateur et mieux le guider pour trouver et corriger la source de son erreur.

V.3 - ***Détection et localisation :***

La vérification des propriétés consiste à chercher le facteur qui les viole. La détection est le fait de décider si la propriété est vérifiée ou violée, tandis que la localisation est le fait de trouver le facteur qui viole la propriété, autrement dit, la localisation est équivalente à l'ensemble des contre-exemples qui violent la propriétés. On cite à titre d'exemple la propriété d'absence de blocage, il s'agit ici de chercher l'existence d'un marquage puits au sien du graphe de marquage correspondant au réseau de Petri. Dans le cas négatif, la propriété est vérifiée, il n'y a donc rien à détecter ni à localiser. Dans le cas contraire, la détection consiste en l'existence du marquage puits et la localisation est le fait de fournir ce marquage comme contre-exemple.

VI - ***Algorithmes utilisés :***

Afin d'extraire les différentes propriétés caractérisant le réseau de Petri, nous avons procédé par exploration exhaustive du graphe de marquage correspondant au réseau. Donc nous construisons d'abord le graphe de marquage correspondant à la spécification en entrée. Remarque étant faite que l'une des particularités du model-checking est que le graphe de marquage doit être fini pour pouvoir effectuer la vérification. En effet, puisqu'il utilise une approche de vérification par graphe de marquage et afin de pouvoir l'explorer intégralement, ce dernier doit être fini. Sinon, il risque de ne jamais terminer l'exécution (l'exploration du graphe).

Dans ce qui suit, nous allons présenter les algorithmes de graphe de marquage qu'on a utilisés :

VI.1 - ***Algorithme 1 :*** [49]

1 - Accessibles := { marquage_initial}

2 - A_explorer := { marquage_initial}

3 - Tantque A_explorer !=Ø faire

3.1 - M := un marquage élément de « A_explorer »

3.2 - A_explorer := A_explorer - {M} ;

3.3 - Pour toutes les transitions t du réseau faire

3.3.1 - Si M > Pre (t) alors

3.3.3.1 - M' := M + Post (t) - Pre (t) ;

3.3.3.2 - Si M' n'appartient pas à Accessibles alors

3.3.3.2.1 - Accessibles := Accessibles union {M'} ;

3.3.3.2.2 - A_explorer := A_explorer union {M'} ;

3.3.3.3 - Fin si ;

3.3.2 - Fin si ;

3.4 - Fin pour ;

4 - Fin tantque ;

Cet algorithme nous a permis de construire le graphe de marquage en utilisant deux ensembles. Le premier (Accessible) contient initialement le marquage initial du réseau, et au fur et mesure on met à jour cet ensemble en lui affectant chaque nouveau marquage qu'on atteint (sans répétition). Le deuxième ensemble (A_explorer) contient également le marquage initial en premier lieu, en suite, il parcourt l'ensemble « Accessible » afin d'explorer tous les marquages et à chaque fois qu'il en explore un, il se met à jour en supprimant cet élément de ses constituants. L'exploration de chaque état (marquage) consiste à vérifier si chaque transition du réseau est franchissable, et pour chaque transition franchissable, on calcule le nouveau marquage obtenu en la franchissant à partir du marquage de l'état actuel. Si ce dernier n'appartient pas à « Accessible », on le rajoute aux deux ensembles (afin de pouvoir l'explorer par la suite).

Il existe en Java plusieurs structures de données pouvant être appliquées sur l'algorithme que nous venons de voir. La structure que nous avons utilisée est la structure des listes chaînées.

VI.2 - ***Algorithme 2 :***

Pour ce deuxième algorithme, nous avons défini une nouvelle structure de données qui exprime mieux les nœuds du graphe de marquage, permettant ainsi de garder le lien entre chaque nœud et ses fils, c'est-à-dire une structure reflétant un vrai graphe. Un nœud est constitué de deux champs notamment : son marquage et ses fils (qui sont du type nœud également).

Algo (Nœud : Racine, Accessible) :

1 - Nœud : Fils

2 - Racine.marquage := marquage initial ;

3 - Pour toutes les transitions t du réseau faire

3.1 - Si Racine.marquage > Pre (t) alors

3.1.1 - Fils.marquage := M + Post (t) - Pre (t) ;

3.1.2 - Racine.fils := Racine.fils U Fils ;

3.2 - Fin si ;

4 - Fin pour ;

5 - Accessible := Accessible U {Racine} ;

6 - Pour tous les fils F de Racine faire

6.1 - Si F n'appartient pas à Accessibles alors

6.1.1 - Algo (F)

6.2 - Fin si ;

7 - Fin pour ;

Fin Algo ;

Cet algorithme illustre une méthode récursive permettant de construire le graphe de marquage en gardant les liens entre les nœuds et leur fils. Il prend comme entrées le nœud Racine ainsi qu'un ensemble « Accessible ». En premier lieu, on affecte au champ marquage

de la Racine le marquage initial, puis, pour chaque transition franchissable à partir du marquage initial, on calcule le nouveau marquage qu'on affecte à un nœud temporaire, ensuite, ce dernier va être lui-même affecté au champ fils de la Racine. Une fois que le nœud Racine est construit (marquage + fils), on l'ajoute à l'ensemble « Accessible ». En dernier, pour chaque fils F de la Racine et s'il n'appartient pas à « Accessible », on réapplique l'algorithme avec les paramètres F et l'ensemble « Accessible » mis à jour.

La raison pour laquelle nous avons utilisé deux algorithmes différents pour la construction du graphe de marquage, est que le premier algorithme est plus optimal mais ne permet d'extraire que certaines propriétés, tandis que le deuxième offre la possibilité d'extraire l'ensemble de toutes les propriétés mais il est plus coûteux vis-à-vis de l'espace mémoire.

VII - ***Architecture du système :***

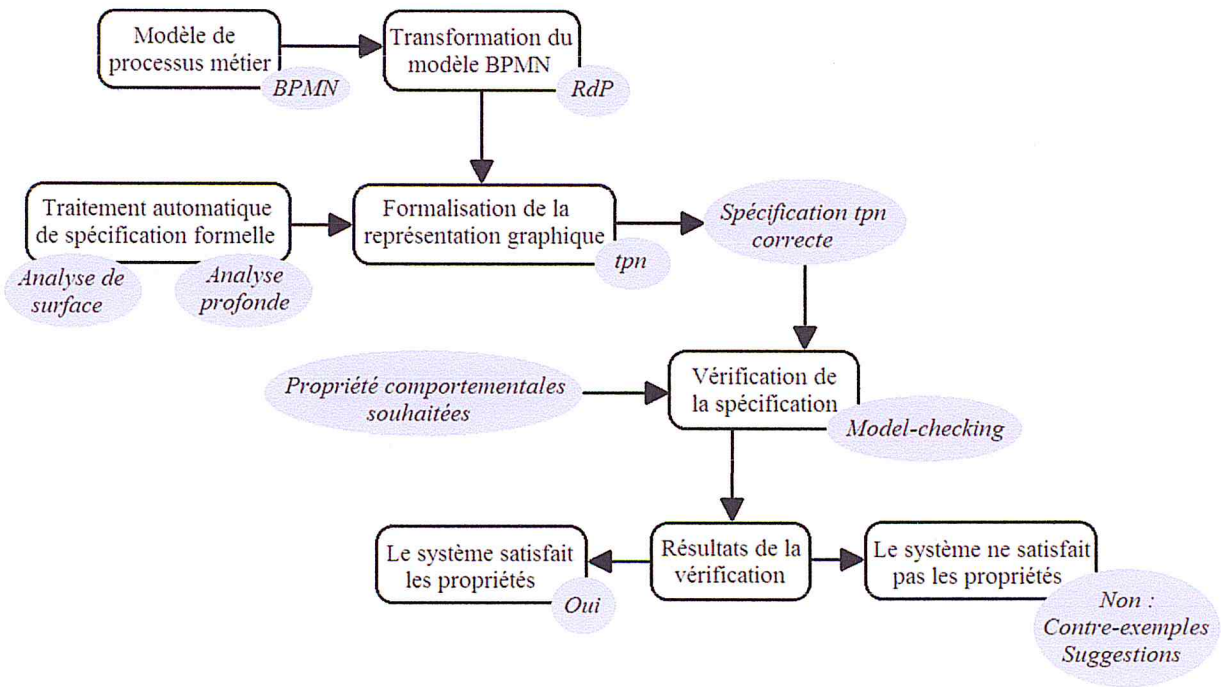


Figure 20 : Organigramme représentant l'architecture du système.

VIII - ***Simulation :***

Dans cette section, nous allons donner un aperçu général sur les différentes fonctionnalités de notre outil.

D'après ce qui a été cité auparavant, notre approche nécessite une spécification formelle reflétant le réseau de Petri à vérifier. Notre outil dispose d'un éditeur textuel

permettant à l'utilisateur d'éditer la spécification formelle sous format « tpn ». Ce dernier est doté d'un compilateur servant à détecter les différentes erreurs que l'utilisateur peut introduire.

L'outil permet également aux utilisateurs non expérimentés, et qui ne savent pas spécifier un réseau de Petri en cette spécification, d'importer des fichiers « tpn » que l'on analyse par le biais du compilateur. En effet, il existe beaucoup d'éditeurs graphiques de réseau de Petri permettant à l'utilisateur d'éditer graphiquement le réseau de Petri et de l'exporter sous format « tpn ». Prenant l'exemple de l'outil TiNA, il dispose d'un tel éditeur graphique.

La figure 21 montre un réseau de Petri édité graphiquement dans l'éditeur netDraw faisant partie des modules de TiNA, tandis que la figure 22 montre comment il permet de l'exporter sous format « tpn » :

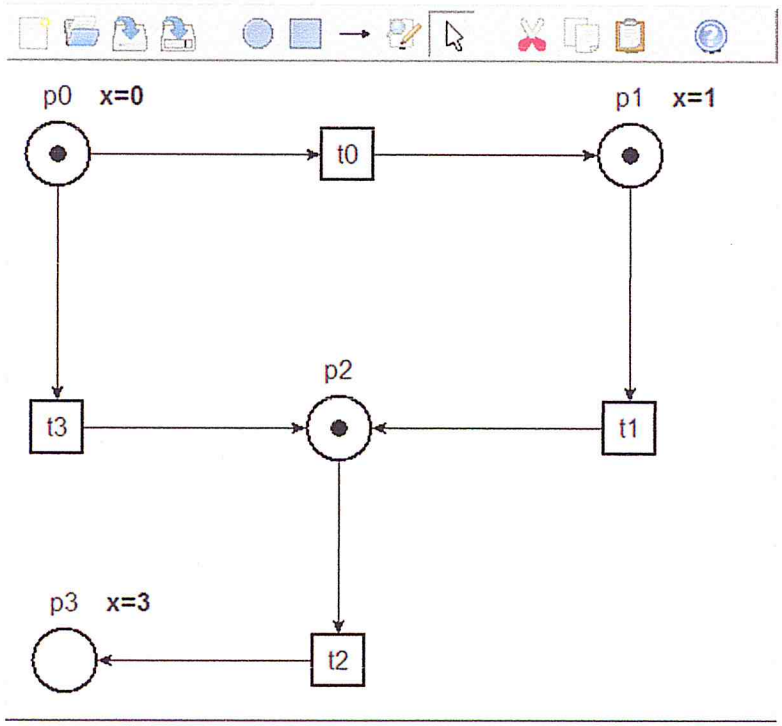


Figure 21 : Edition graphique de réseau de Petri.

```
net reseau1
tr t0 p0 -> p1
tr t1 p1 -> p2
tr t2 p2 -> p3
tr t3 p0 -> p2
pl p0 : {x=0} (1)
pl p1 : {x=1} (1)
pl p2 (1)
pl p3 : {x=3}
```

Figure 22 : Réseau de Petri exporté.

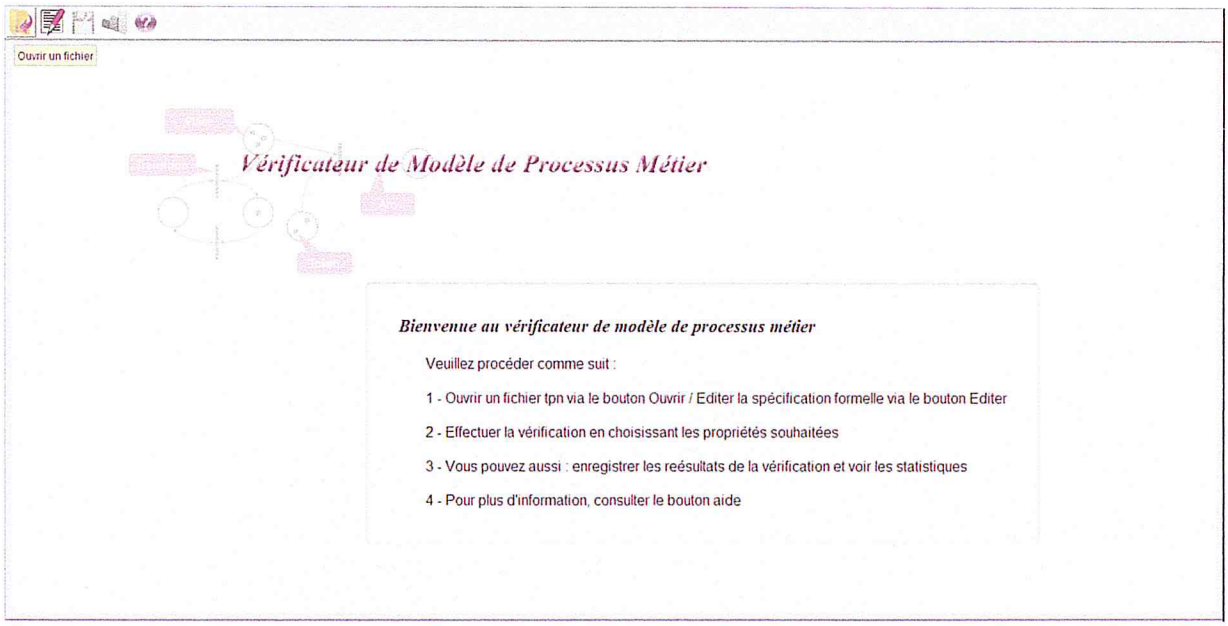


Figure 23 : Page d'accueil – ouverture d'un fichier « tpn ».

Après avoir cliqué sur le bouton « *ouvrir* », l'utilisateur peut choisir un fichier à partir de son disque dur. Le fichier en question passe par une analyse afin de détecter les erreurs éventuelles. Un message indiquant si le fichier est correct ou non lui sera affiché.

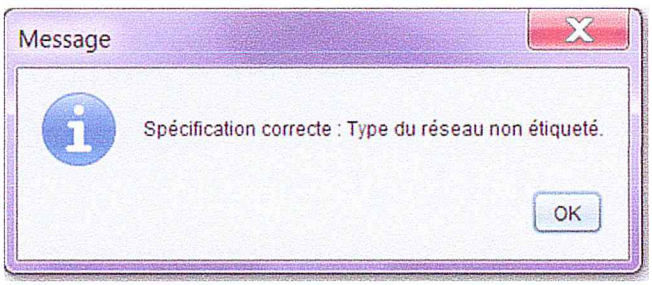


Figure 24 : Message de confirmation.

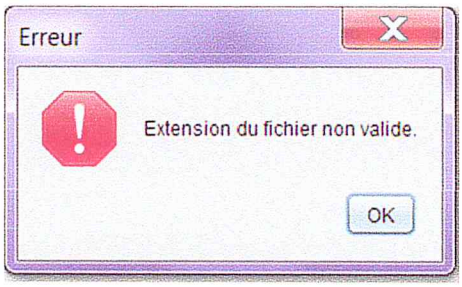


Figure 25 : Message d'erreur – extension non valide.

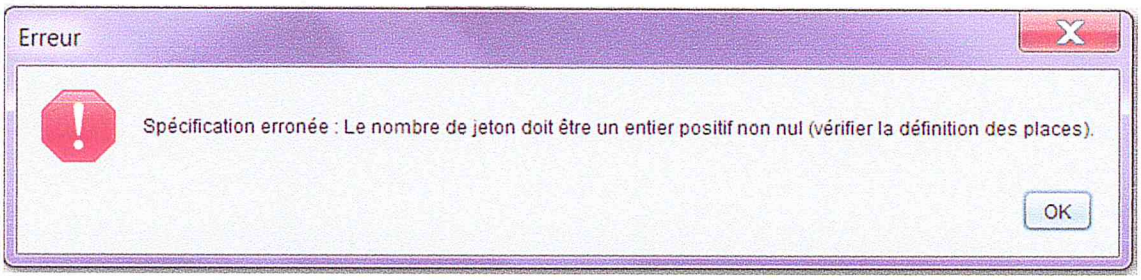


Figure 26 : Message d'erreur – spécification erronée.

Dans le cas où l'utilisateur importe une spécification erronée, l'outil est en mesure de signaler l'erreur, son type et ainsi la localiser. Ceci peut effectivement aider l'utilisateur et mieux le guider pour corriger la source de son erreur.

Dans le contexte où l'utilisateur édite sa spécification formelle, il procède en appuyant sur le bouton « *rédiger* » afin d'accéder à l'éditeur textuel de l'outil.

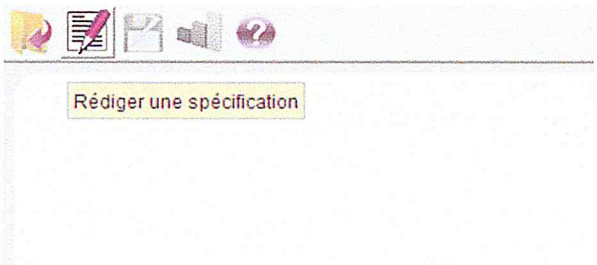


Figure 27 : Rédaction d'une spécification.

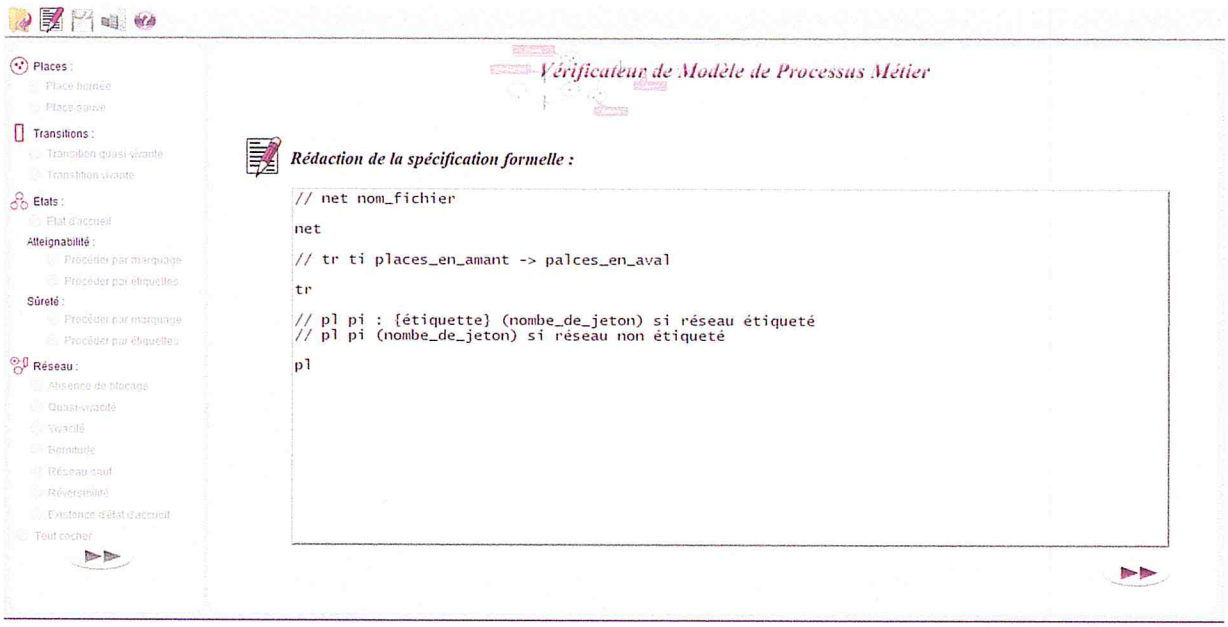


Figure 28 : Page de rédaction.

Une fois que l'utilisateur valide sa spécification, le compilateur de l'outil indique si la spécification est correcte ou non. Dans le cas où elle est erronée, comme dans l'ouverture d'un fichier, l'outil mentionne l'erreur en affichant un message contenant le type d'erreur introduite ainsi que sa localisation.

Nous mentionnons que dans ces cas, nous avons appliqué les techniques de traitement automatique. En effet l'analyse de surface nous a permis de reconnaitre la structure générale de la spécification et en analysant les mots clefs et les types de données elle détermine si la spécification est correcte ou non. Ainsi l'analyse profonde nous a permis d'analyser les différentes déclarations et d'y détecter des erreurs s'il en existe.

Après avoir ouvert / édité une spécification formelle, comme la figure ci-dessous l'indique (figure 29) l'utilisateur peut soit modifier la spécification (en appuyant sur le bouton « *modifier* »), soit se lancer dans la procédure de vérification (en utilisant le panneau à gauche).

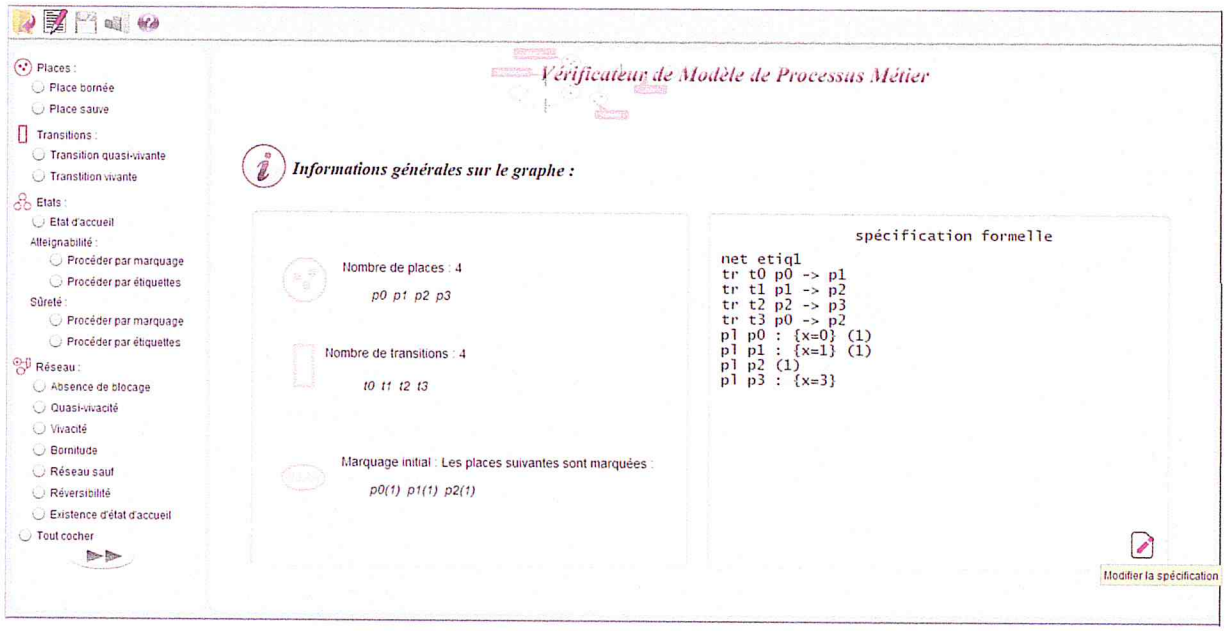


Figure 29 : Page d'information – modification de la spécification.

En choisissant de modifier sa spécification, l'utilisateur est dirigé vers la page de rédaction de spécification formelle (voir figure 28), cette page devra bien sûr contenir le code actuel qu'il souhaite modifier. Après avoir apporté des modifications et les avoir validées, le compilateur de l'outil analyse encore une fois la nouvelle spécification à la recherche d'éventuelles erreurs. Ainsi, dans le cas où l'utilisateur sort de la page de rédaction, on lui propose de sauvegarder sa spécification avant de sortir.

Si, par-contre, l'utilisateur se lance dans la vérification des propriétés, il n'a qu'à cocher toutes les propriétés souhaitées et appuyer sur le bouton « *vérifier* » comme le montre la figure ci-dessous (figure 30) :

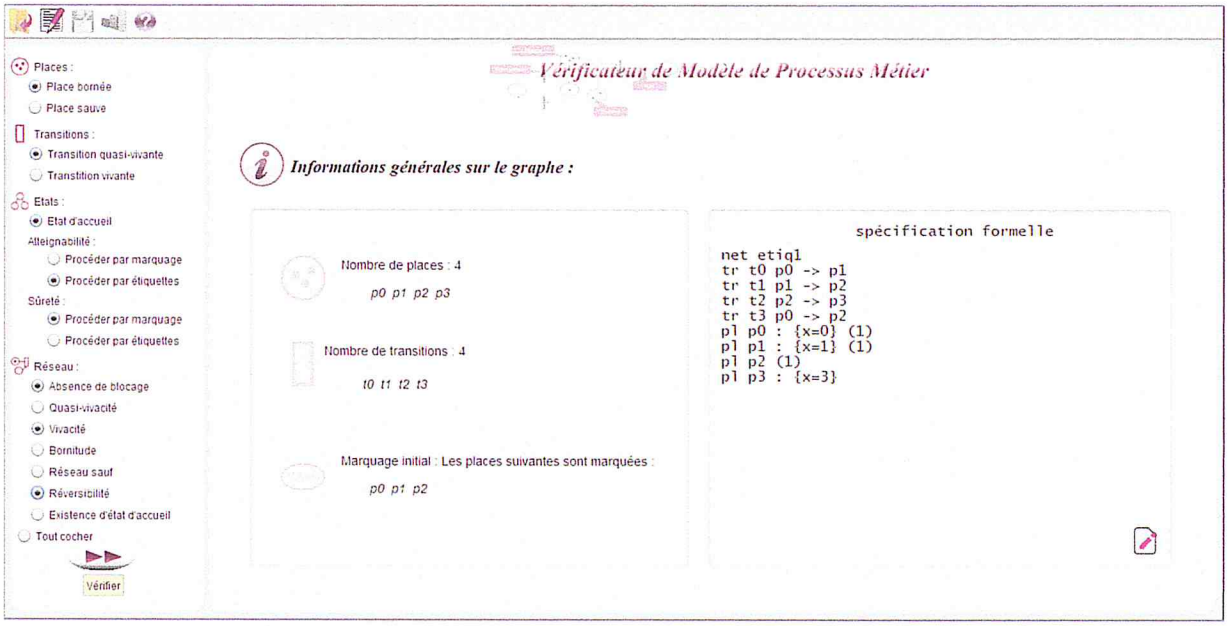


Figure 30 : Page d'information – vérification des propriétés.

Comme nous pouvons le voir les propriétés sont regroupées dans des catégories (places, transitions, états et réseau) pour une meilleure lisibilité. Certaines de ces propriétés nécessitent des entrées lues à partir du clavier, notamment :

- Place bornée : elle nécessite deux entrées, à savoir : la ou les place(s) et la borne.
- Place sauve : cette propriété nécessite en entrée la ou les place(s) en question.
- Transition quasi-vivante / vivante : elle exige en entrée la ou les transition(s).
- Etat d'accueil : elle a besoin d'un marquage en entrée.
- Atteignabilité / sûreté par marquage : elle nécessite un marquage en entrée.
- Atteignabilité / sûreté par étiquettes : elle nécessite une ou plusieurs étiquettes en entrée.
- Bornitude : cette propriété a besoin de la borne supérieur en entrée.

Si au moins l'une de ces propriétés est cochée, un formulaire s'affiche afin de remplir les informations nécessaires pour la vérification.

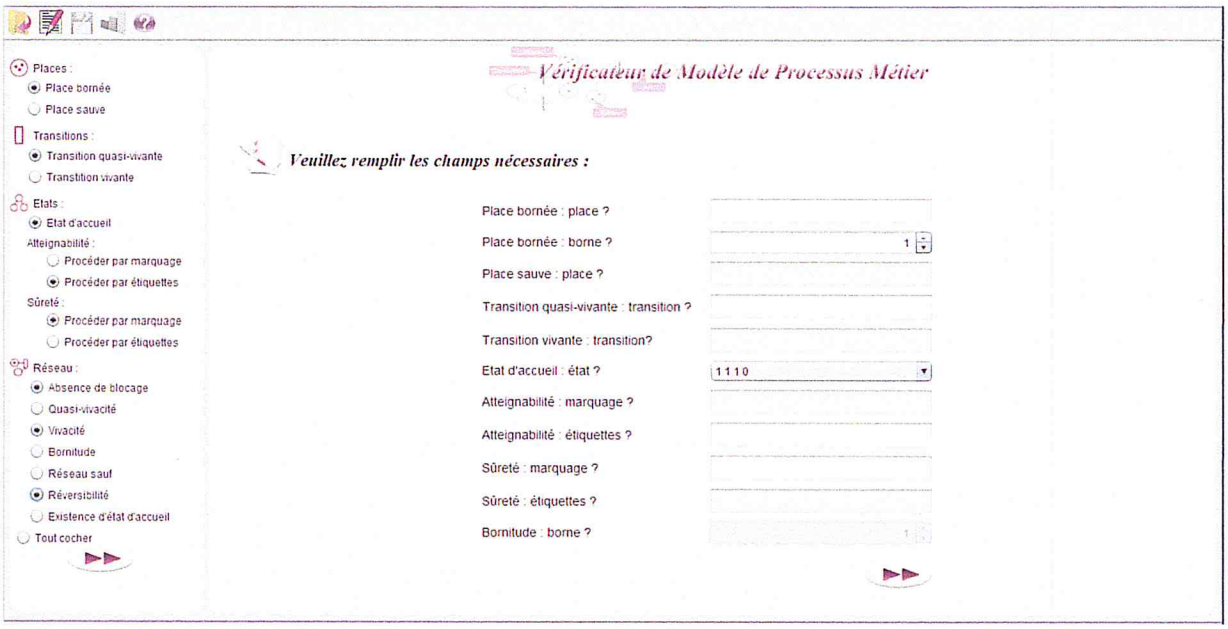


Figure 31 : Page de formulaire.

Ce formulaire ne permet d'éditer que les champs nécessaires (d'après ce qui a été coché), le reste des champs sera grisé et non éditable. On est en mesure de contrôler toutes les entrées du formulaire (si un(e) place / transition / marquage / étiquette appartient au réseau, si une borne est un entier, … etc.). Après avoir validé le formulaire, la page de vérification s'affiche comme le montre la figure ci-dessous (figure 32) :

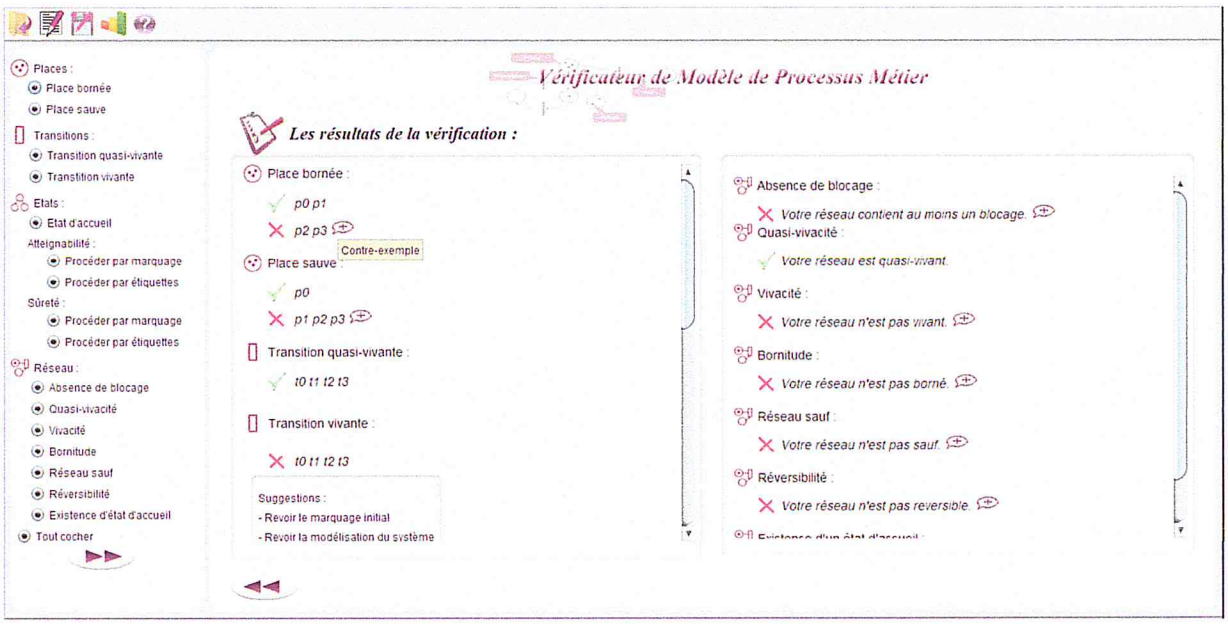


Figure 32 : Page de résultat.

A partir de cette page l'utilisateur peut revenir à la page d'information en appuyant sur le bouton « *précédant* ». En effet ceci peut être utile dans le cas où l'utilisateur veut modifier sa spécification formelle par exemple. Ainsi l'outil offre la possibilité de consulter les contre-exemples des propriétés qui sont violées (certaines propriétés ne disposent pas de contre-exemple). Ainsi, pour chaque propriété violée, nous sommes en mesure de proposer un ensemble de suggestions afin de faciliter la correction.

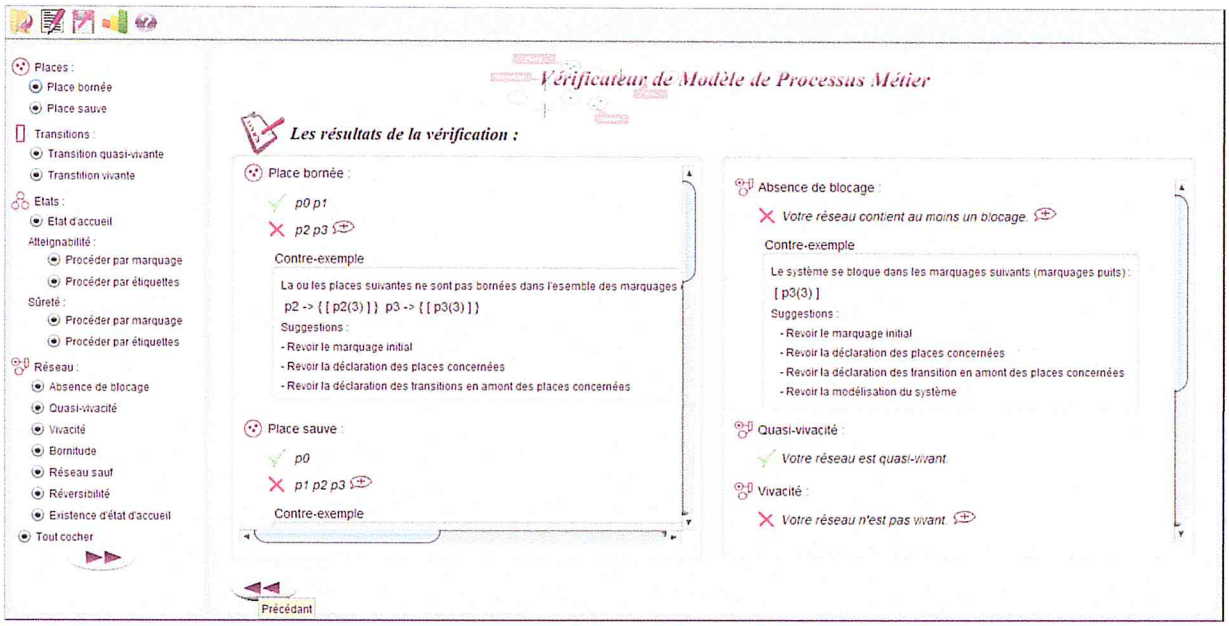


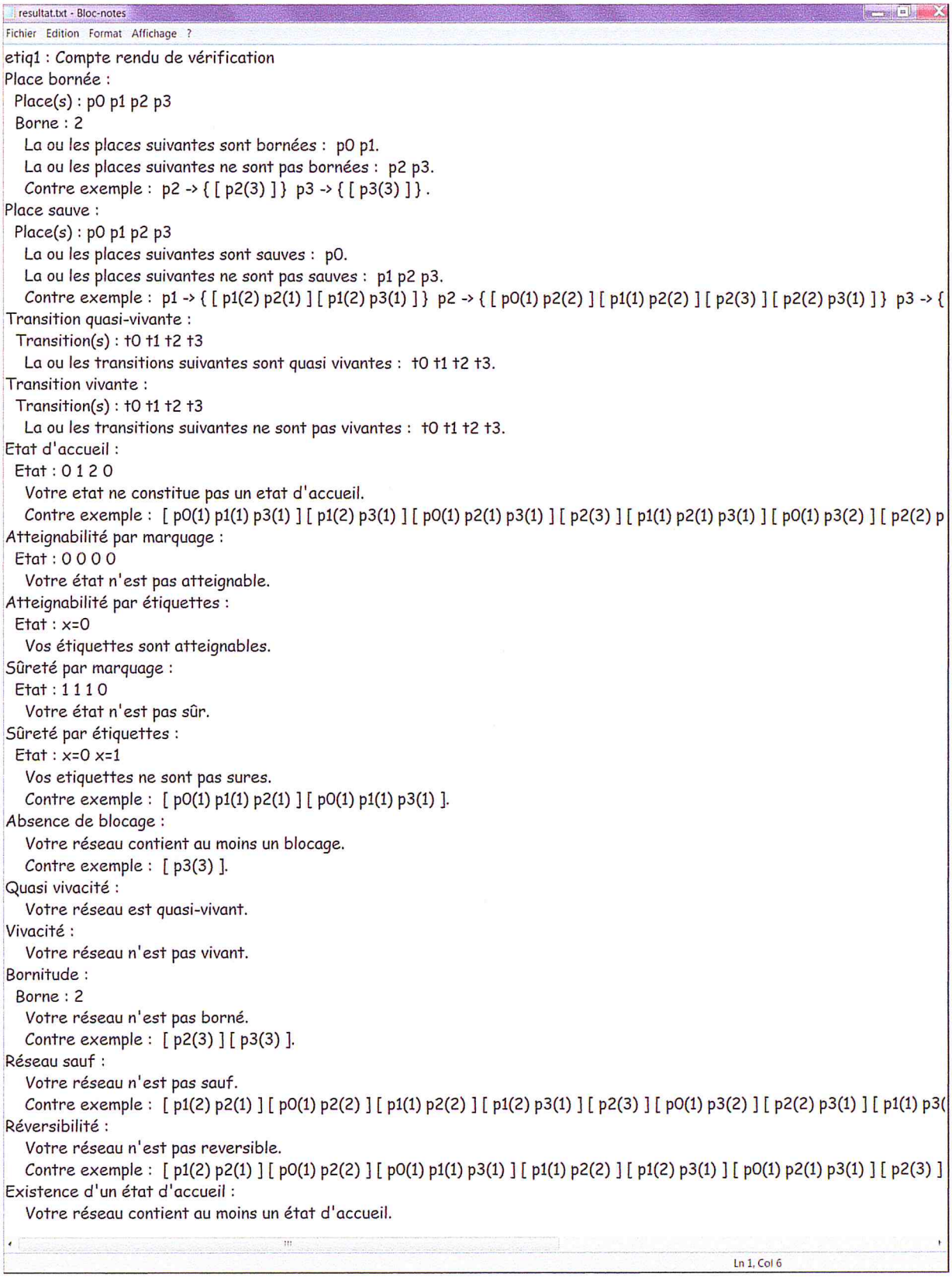
Figure 33 : Page de résultat – contre-exemples.

Comme nous pouvons le voir, dans la page de résultat, deux autres fonctionnalités s'ajoutent dans le menu, à savoir : « *enregistrer les résultats* » et « *statistiques* ».

Figure 34 : bouton enregistrer / statistique fonctionnels.

Après avoir vérifié le réseau de Petri, l'utilisateur pourra enregistrer les résultats de la vérification dans un fichier texte en choisissant son emplacement.

La figure ci-dessous (figure 35) montre le fichier contenant les résultats de la vérification de la figure 33 :



resultat.txt - Bloc-notes

Fichier Edition Format Affichage ?

```
etiq1 : Compte rendu de vérification
Place bornée :
 Place(s) : p0 p1 p2 p3
 Borne : 2
  La ou les places suivantes sont bornées : p0 p1.
  La ou les places suivantes ne sont pas bornées : p2 p3.
  Contre exemple : p2 -> { [ p2(3) ] } p3 -> { [ p3(3) ] } .
Place sauve :
 Place(s) : p0 p1 p2 p3
  La ou les places suivantes sont sauves : p0.
  La ou les places suivantes ne sont pas sauves : p1 p2 p3.
  Contre exemple : p1 -> { [ p1(2) p2(1) ] [ p1(2) p3(1) ] } p2 -> { [ p0(1) p2(2) ] [ p1(1) p2(2) ] [ p2(3) ] [ p2(2) p3(1) ] } p3 -> {
Transition quasi-vivante :
 Transition(s) : t0 t1 t2 t3
  La ou les transitions suivantes sont quasi vivantes : t0 t1 t2 t3.
Transition vivante :
 Transition(s) : t0 t1 t2 t3
  La ou les transitions suivantes ne sont pas vivantes : t0 t1 t2 t3.
Etat d'accueil :
 Etat : 0 1 2 0
  Votre etat ne constitue pas un etat d'accueil.
  Contre exemple : [ p0(1) p1(1) p3(1) ] [ p1(2) p3(1) ] [ p0(1) p2(1) p3(1) ] [ p2(3) ] [ p1(1) p2(1) p3(1) ] [ p0(1) p3(2) ] [ p2(2) p
Atteignabilité par marquage :
 Etat : 0 0 0 0
  Votre état n'est pas atteignable.
Atteignabilité par étiquettes :
 Etat : x=0
  Vos étiquettes sont atteignables.
Sûreté par marquage :
 Etat : 1 1 1 0
  Votre état n'est pas sûr.
Sûreté par étiquettes :
 Etat : x=0 x=1
  Vos etiquettes ne sont pas sures.
  Contre exemple : [ p0(1) p1(1) p2(1) ] [ p0(1) p1(1) p3(1) ].
Absence de blocage :
  Votre réseau contient au moins un blocage.
  Contre exemple : [ p3(3) ].
Quasi vivacité :
  Votre réseau est quasi-vivant.
Vivacité :
  Votre réseau n'est pas vivant.
Bornitude :
 Borne : 2
  Votre réseau n'est pas borné.
  Contre exemple : [ p2(3) ] [ p3(3) ].
Réseau sauf :
  Votre réseau n'est pas sauf.
  Contre exemple : [ p1(2) p2(1) ] [ p0(1) p2(2) ] [ p1(1) p2(2) ] [ p1(2) p3(1) ] [ p2(3) ] [ p0(1) p3(2) ] [ p2(2) p3(1) ] [ p1(1) p3(
Réversibilité :
  Votre réseau n'est pas reversible.
  Contre exemple : [ p1(2) p2(1) ] [ p0(1) p2(2) ] [ p0(1) p1(1) p3(1) ] [ p1(1) p2(2) ] [ p1(2) p3(1) ] [ p0(1) p2(1) p3(1) ] [ p2(3) ]
Existence d'un état d'accueil :
  Votre réseau contient au moins un état d'accueil.
```

Ln 1, Col 6

Figure 35 : Enregistrement des résultats.

Dans le cas où l'utilisateur effectue d'autres vérifications sur le même réseau et qu'il veuille enregistrer les nouveaux résultats, en cliquant sur le bouton enregistrer le fichier sera mis à jour en ajoutant les nouveaux résultats (sans dupliquer des mêmes résultats).

Si l'utilisateur choisi de sortir de la page des résultats en ayant effectué une ou plusieurs vérifications sans les avoir enregistrées, on lui propose de les enregistrer avant de sortir.

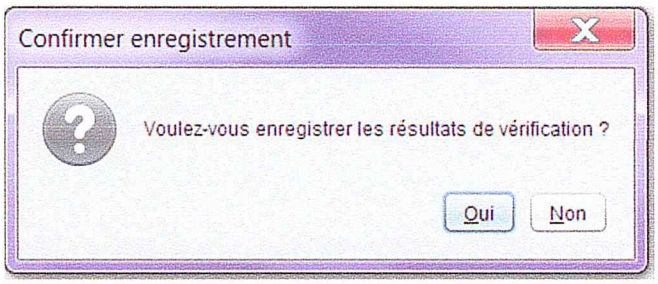


Figure 36 : Demande d'enregistrement.

Concernant le bouton « statistiques », il permet à l'utilisateur, après avoir effectué une vérification, de visualiser la partition des différentes propriétés du réseau qu'il a souhaitées vérifier. En effet, chaque propriété dispose d'un pourcentage ajoutant de la fiabilité au réseau quand celle-ci est vérifiée. Les statistiques ne montrerons pas la répartition de l'ensemble de toutes les propriétés mais seulement celles des propriétés souhaitées car c'est ce qui importe l'utilisateur. Ainsi, ces statistiques peuvent être très utiles à l'utilisateur, hors-mis le fait de mieux lui monter la fiabilité de son système, il peut copier ces statistiques ou encore les imprimer et ainsi les utiliser dans le cadre d'un rapport concernant le processus métier en question.

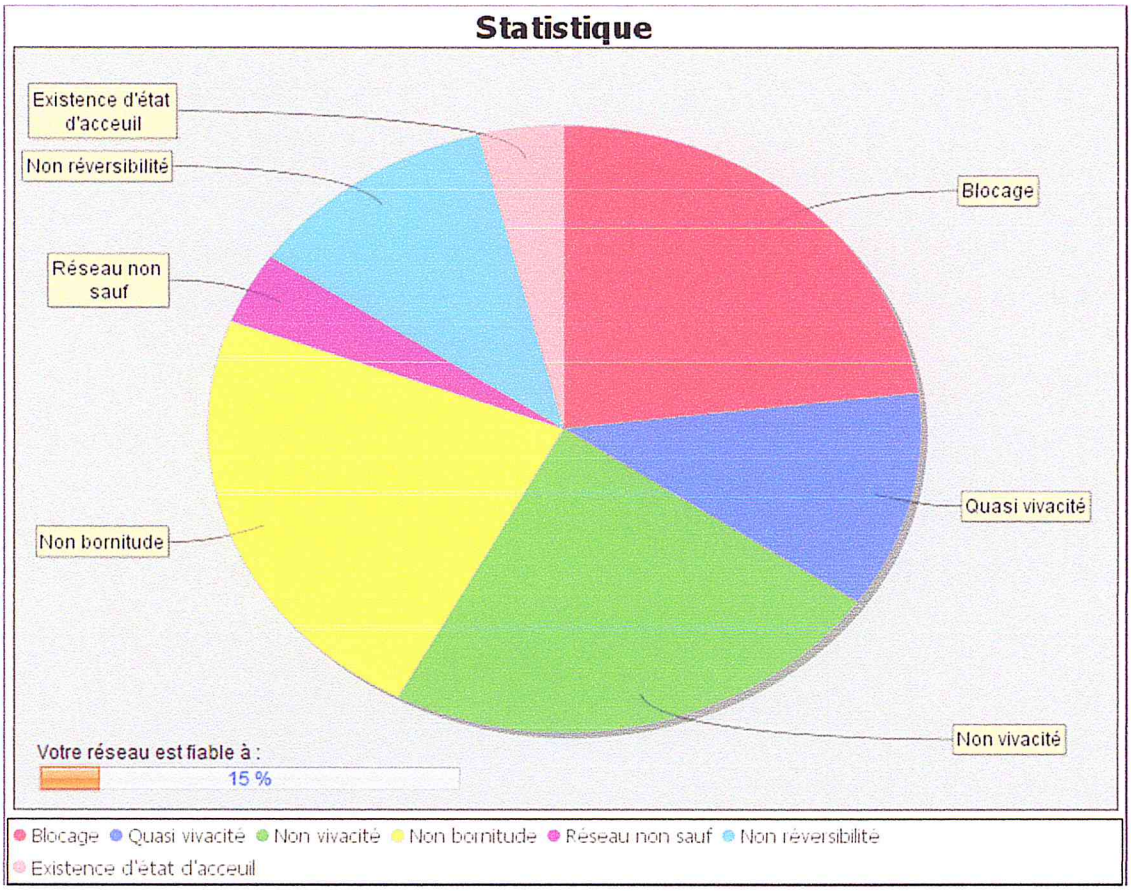


Figure 37 : Statistiques du réseau.

Enfin, l'outil dispose du bouton « *aide* » fournissant plus d'informations sur le fonctionnement et l'utilisation de l'outil.

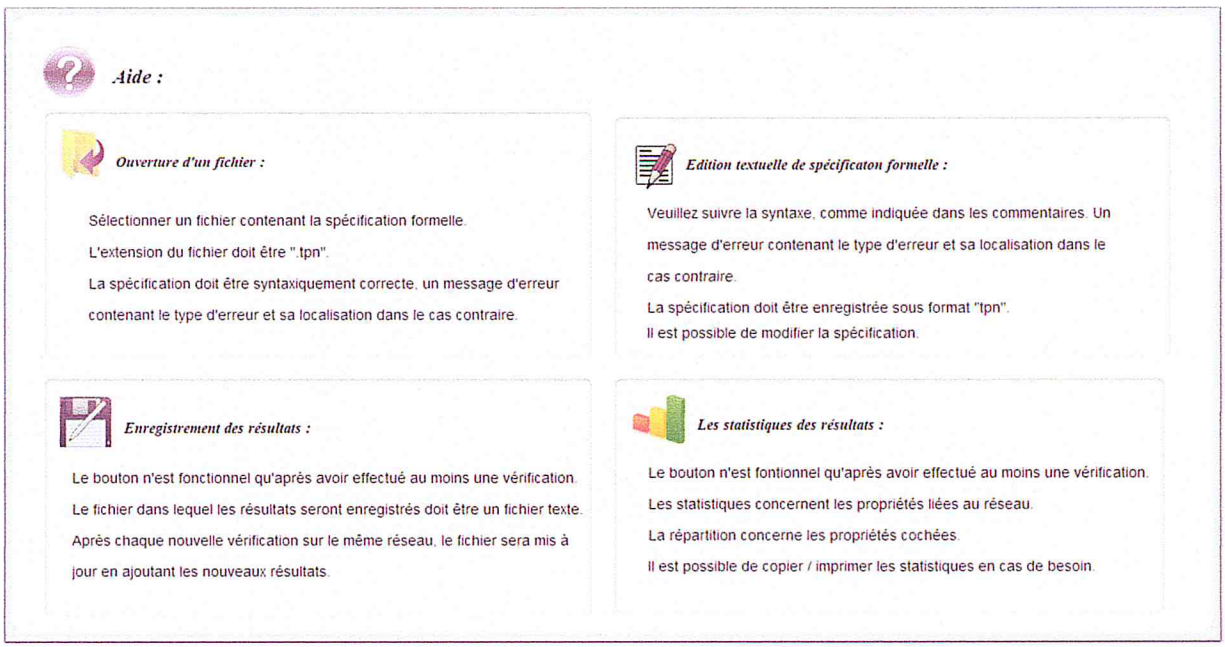


Figure 38 : Bouton aide.

A ce niveau, il est nécessaire de mentionner qu'à l'aide de nos travaux antérieurs, nous avons pu réaliser nos principaux objectifs ainsi que toute fonctionnalité exposée dans ce chapitre.

D'après l'étude faite au cours du deuxième chapitre (tableau comparatif). Nous avons pu constater que la correction d'erreur n'a pas été mise en œuvre, nous avons donc voulu dépasser les objectifs fixés au départ en optant pour la correction des propriétés violées. Or, nous nous sommes aperçues que cela n'est pas possible dans la plupart des cas, car en la corrigeant toute la sémantique du processus métier risque de changer. Ainsi, une propriété violée ne désigne pas toujours une erreur mais au contraire une confirmation de la propriété. En effet, si par exemple la propriété d'absence de blocage est violée alors que le système est supposé s'arrêter il n'y a aucune erreur, donc si on la corrige on ne fait que falsifier le processus métier.

IX - ***Conclusion :***

Au cours de ce chapitre, nous avons présenté l'implémentation de notre vérificateur de modèle de processus métier. Après avoir mis en œuvre les algorithmes et les techniques indispensables au bon fonctionnement de l'outil, nous avons abouti aux résultats estimés.

Conclusion générale

L'objectif de ce travail consiste en la vérification de modèle de processus métier dans le cadre d'un projet de recherche SCIO-Web Social (Service de Collaboration Inter-Organisationnelle basée web social) au sien du Centre de Développement des Technologies Avancées (CDTA).

Pour ce faire, plusieurs technologies ont été utilisées. Nous citons les études préalablement faites concernant la modélisation des processus métier, à savoir : BPMN et les réseaux de Petri, les techniques de vérification formelles, la spécification formalisant les réseaux de Petri ainsi que le langage de programmation Java et les interfaces graphiques.

De cette étude et recherche minutieuses effectuées, nous avons pu élaborer un vérificateur de modèle de processus métier, essentiellement nécessaire à la détection des comportements non souhaitables au sein de ces processus.

Enfin, de cette étude établie par nos soins, nous avons pu faire ressortir, à ce qui a été imaginé, une plate-forme d'appréciation qui résulte sous forme de réalité de notre travail préparatoire. Ainsi, en ce qui est des objectifs fixés au début de l'étude, nous pouvons humblement affirmer qu'ils ont été intégralement atteints avec satisfaction. Toutefois et afin d'offrir plus de fonctionnalités, il serait souhaitable de rendre plus bénéfique ce projet en l'enrichissant par :

- Un outil qui permet la modélisation du processus collaboratif inter-organisationnel (englobant tous les partenaires de la collaboration) prenant en entrée toutes les connaissances concernant la collaboration à établir (sur les partenaires, les organisations et le réseau de collaboration) et fournissant en sortie le modèle BPMN du processus collaboratif validé par tous les partenaires.
- Un outil qui permet de transformer automatiquement un modèle BPMN en un réseau de Petri sous format « tpn » qui servira comme entrée à notre outil de vérification.

Bibliographie :

[1] R.ElMansouri, Modélisation et Vérification des processus métiers dans les entreprises virtuelles: (Une approche basée sur la transformation de graphes), pp.42-85.

[2] G.Decalf, 2013. BPMN, un véritable standard ?. http://www.bpms.info/bpmn-un-veritable-standard/, (05/01/2015).

[3] SA.White, Introduction to BPMN, pp. 01-05.

[4] F.Bouvart, 2014. BPMN 2.0 Modélisation de processus Introduction à BPMN & Bizagi process modeler. http://fr.slideshare.net/fredericbouvart/bpmn-bizagi-modelisation-processus, (12/02/2015).

[5] D.Brookshier, BPMN 2.0 Tutorial, pp. 30-39.

[6] L.Bocher, 2010. Business Process Modeling Notation (BPMN) - Système de notation graphique pour la description des precessus métier. http://www.allaboutbpm.com/bonnes-pratiques-bpm/bpmn-business-process-modeling-notation-definition, (08/12/2014).

[7] B.Debauche, 2013. BPMN, Le premier standard du BPM ?. http://www.bpms.info/bpmn-un-veritable-standard/, (20/12/2014).

[8] JL.Camelio, 2014. Standards et bonnes pratiques, le duo gagnant du BPM. http://www.w4software.com/blog/article-sur-w4-bpmn-dans-magazine-qualite-reference-0614, (02/01/2015).

[9] Chemla , 2007.Notions de base RDP, http://www.tn.refer.org/hebergement/cours/sys_disc/notions_de_base_RdP.html , (13/12/2014).

[10] Y.Morére, Cours de réseau de Petri, 2002, pp.15-44.

[11] Valette Robert, 2007.Bonnes propriétés et invariants, http://www.laaas.fr/robert, (26/02/2015).

[12] A.Geniet, Modélisation des Structures de contrôle (Fiabilité et sûreté de fonctionnement): Réseau de Petri, pp.08-32.

[13] R.Elmansouri, Modélisation et Vérification des processus métiers dans les entreprises virtuelles : Une approche basée sur la transformation de graphes, 2010, pp.46-67.

[14] M.Bourcerie, La modélisation des systèmes de production par réseau de Petri, 2010-2011, pp.5-16.

[15] RM.Dijkman, Information and Software Technology, 2008, pp. 1282-1289.

[16] MO.Khebbouche, Contributions à la gestion de l'évolution des processus métier, 2013, pp. 71-97.

[17] C.Choppy, M.Mayero, L.Petrucci, Electronic Notes in Theoretical Computer Science, 214, pp. 231-254.

[18] Z.Sbai, Contribution à la Modélisation et à la Vérification de Processus Workflow, 2010, pp. 120.

[19] WMP. van der Aalst, Woflan : A Petri-net-based workflow analyzer, pp. 03-08.

[20] C.Ouyang, E.Verbeek, WMP.Van der Aalst, MP.Will, WS.Breutel, M.Dumas, AHM.ter Hofstede, WofBPEL: A Tool for Automated Analysis of BPEL Processes. Lecture Notes in Computer Science, 2005, pp.484-489.

[21] Suresh Raj Gopalan, 03/07/2006.BPEL: What are partnerLinkTypes roles and partnerLinks, https://blogs.oracle.com/gopalan/entry/bpel_what_are_partnerlinktypes_roles, (24/12/2014)

[22] Oracle, 2010. Understanding Correlation. Using the Correlation Wizard, http://docs.oracle.com/cd/E19182-01/821-0539/corr_wizard/index.html, (24/12/2014).

[23] G6G Consulting Group, 2015. Platform Independant Petri net Editor (PIPE), http://g6g-softwaredirectory.com/bio/cross-omics/agent-based/20610-Imperial-Coll-London-PIPE.php, (22/01/2015).

[24] M.Boukhebouze, Gestion de changement et vérification formelle de processus métier : approche orientée règle, 2010, pp. 68-71.

[25] B. Berthomieu, D. le Botlan, S. Dal Zilio, F.Vernadat, Vérification avec les outils TINA, 2010, pp. 02-06.

[26] B.Berthomieu, F.Vernadat, Time Petri Nets Analysis with TINA,2004, pp.2741–2756.

[27] LoLA : A Low Level Petri net Analyzer, http://service-technology.org/lola/, (26/01/2015).

[28] Universität Rostock , 2010. LoLA - a low-level Petri net analyzer, http://fr.slideshare.net/correctsystems/lola-a-lowlevel-petri-net-analyzer-3837556, (16/01/2015).

[29] Laboratoire de Paris 6, 2007. Modélisation et vérification : Model Checking, http://move.lip6.fr/software/CPNAMI/MANUAL_SERV/behav-prop.html, (27/11 /2014).

[30] C.Baier , J.P. Katoen, Principles of Model Checking, 2008, pp. 07-16.

[31] A.Mebsout, Inférence d'Invariants pour le Model Checking de Systèmes Paramétrés, 2014, pp. 10-11.

[32] N.Sznajder, Vérification formelle de systèmes par Model-Checking, 2012, pp.11-140.

[33] S.Bardin, Introduction au model Checking, 2008, pp. 06-40.

[34] C.Pajault, Model checking parallèle et reparti de réseaux de Petri colores de haut niveau: application à la vérification automatique de programmes Ada concurrents, 2013, pp.10-22.

[35] N.Abed, Exploration randomisée de larges espaces d'états pour la vérification, 2011, pp. 11-39.

[36] A.Duret-Lutz, Algorithmes pour la vérification de formules LTL par l'approche automate, 2004, pp. 11-13.

[37] A.Duret-Lutz, Introduction au Model Checking, 2013, pp.9-32.

[38] P.Lescanne, Les modèles de Kripke, 2004, pp. 02-04.

[40] C.Attiogbé, Méthodes de spécification et développement formel : l'approche orientée modèle, 2001, pp. 05-08.

[41] L.Audibert, Traitement automatique du langage naturel (TALN) Outils d'analyse de données textuelles, 2010, pp.09-54.

[42]https://sites.google.com/site/coursdetal/introduction-au-tal/analyse-syntaxique-automatique

[43] S.Mustière, N.Abadie, N.Aussenac-Gilles, M.N.Bessagnet, M.Kamel, E.Kergosier, C.Reynaud, B.Safar, GéOnto : Enrichissement d'une taxonomie de concepts topographiques, 2009, pp.04-13.

[44] E.Tiphène, Vérification formelle des systèmes matériels, 2010, pp.2-19.

[45] I.Mounier, Spécification des systèmes et la vérification des formules de CTL, pp.1-74.

[46] Razclos, Réseaux de Petri - cours de programmation des systèmes II, 1998.

[47] M.C.Boukala, Towards Distributed Verification of Petri Nets Properties, 2007, pp. 01-11.

[48] A.Geniet, Modélisation des Structures de contrôle Fiabilité et sûreté de fonctionnement Réseau de Petri, pp.16-21.

[49] P.Molinaro, Un algorithme efficace pour calculer le graphe des marquages accessibles, 2013, pp.02-06.

Annexes

Sommaire

Annexe 1 : Business Process Modeling Notation (BPMN) :

Cette notation est utilisée pour modéliser un processus métier. Pour cela elle dispose d'un BPD (Business Process Diagram) constitué des 4 éléments de base suivants :

1 - Les objets de flux :

1.1 - ***Les évènements :*** Ils événements affectent le flux du processus et ont généralement une cause (déclencheur) ou un impact (résultat). Il existe trois types d'événements, la différence entre eux réside au moment où ils affectent le flux.

Figure 1 : Types d'évènement.

- **Les événements de début :** décrivent la circonstance de déclenchement du processus.
- **Les événements intermédiaires :** décrivent un état intermédiaire.
- **Les événements de fin :** identifient la fin d'un processus. La forme de l'événement de fin permet de distinguer le résultat du processus.

Cependant, lors de la modélisation des flux de processus, on a, parfois, besoin de modéliser des événements métier plus complexes, tels que des messages, des minuteries, des règles de gestion et les conditions d'erreur. BPMN permet de spécifier le type de l'événement déclencheur, et le désigne avec une icône représentative.

1.2 - ***Les activités :*** Une activité est un terme générique pour le travail que l'entreprise réalise. Elle peut être atomique (dite tâche) ou non atomique (composée dite sous-processus).

1.2.1 - ***Tâches :***

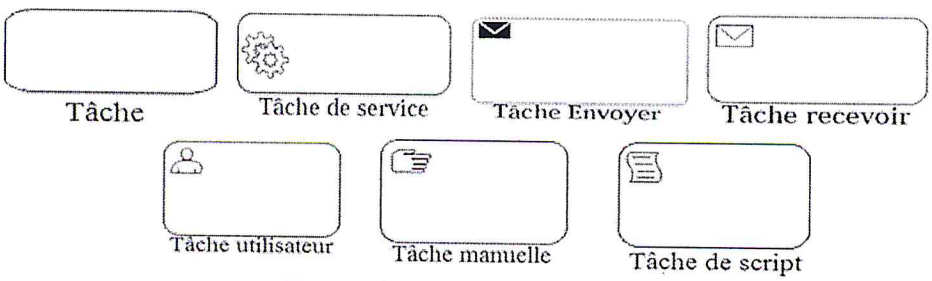


Figure 2 : Types de tâches.

1.2.2 - ***Sous-processus :*** Il est toujours dans le contexte du processus et peut accéder aux données contextuelles. Il peut également être développé ou réduit pour afficher les détails du sous-processus ou pour cacher les détails. Un Sous processus doit définir un processus interne avec un évènement de début et de fin. Il est seulement réutilisable au sein du processus parent.

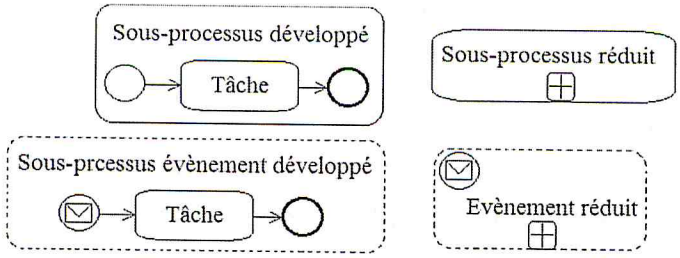


Figure 3 : Activité de type sous-processus.

1.3 - ***Les passerelles :*** Une passerelle est utilisée pour contrôler la divergence et la convergence du flux de séquence. Ainsi, elle déterminera les décisions traditionnelles et la manière dont les activités vont s'enchaîner. Elles sont au nombre de 4 :

- Les passerelles exclusives : un seul chemin est possible sur tous ceux représentés.
- Les passerelles inclusives : plusieurs chemins possibles. Cette passerelle est généralement suivie par une passerelle de même type.
- Les passerelles complexes : utilisées quand les comportements sont complexes.
- Les passerelles parallèles : plusieurs chemins empruntés en même temps.

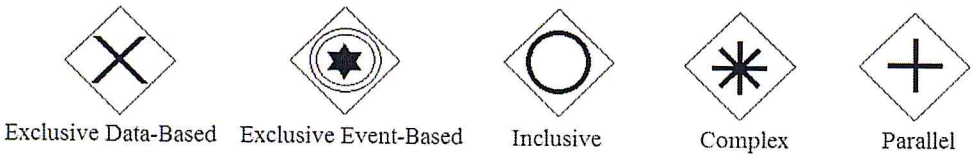


Figure 4 : Types de passerelle.

2 - ***Les objets de liaison :***

Les connexions sont reliées entre elles dans un diagramme pour créer la structure squelettique de base d'un processus métier. Il y a trois connexions qui offrent cette fonction :

2.1 - ***Flux de séquence (Sequence flow) :*** Il est utilisé pour montrer l'ordre (la séquence) dont lequel les activités seront effectuées dans un processus.

2.2 - ***Flux de message (Message flow) :*** Il est utilisé pour montrer le flux de messages entre deux participants séparés d'un processus (entités métier ou acteurs métier) qui les envoient et les reçoivent. Ils ne peuvent pas être utilisés à l'intérieur d'un processus pour lier des tâches.

2.3 - ***Association :*** les associations associent des données, du texte et d'autres objets avec des objets de flux. Elles sont utilisées pour montrer les entrées et les sorties des activités.

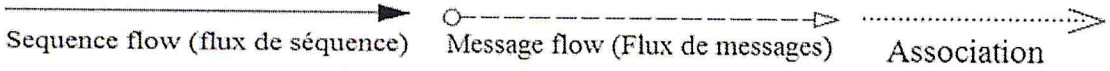


Figure 5 : Les objets de liaison.

3 - ***Les couloirs :***

Comme beaucoup de méthodes de modélisation de processus, BPMN utilise le concept de Swimlanes comme un mécanisme pour organiser les activités dans des catégories visuelles distinctes afin d'illustrer les différentes capacités fonctionnelles ou les responsabilités.

- Pool (souvent appelé « liste de ressouces »).
- Lane (souvent appelé « ligne » ou encore « couloir »).

Les objets d'un modèle sont organisés sur des lignes (lanes), chaque ligne est associée à un objet logique. Les lignes sont regroupées en Pool. Il est nécessaire que les flux entre les lignes sont connectées via des liaisons de type « Messages ».

3.1 - ***Pool :*** Un pool représente un participant dans un Processus. Il agit aussi en tant que conteneur graphique pour le partitionnement d'un ensemble d'activités d'autres pools.

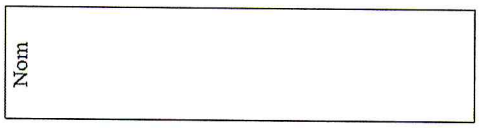


Figure 6 : Forme d'un pool

3.2 - ***Lane :*** Un Lane est une sous-partition dans un Pool et va étendre sur toute sa longueur verticalement ou horizontalement. Ils sont utilisés pour organiser et classer les activités.

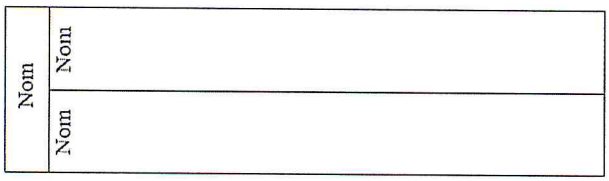


Figure 7 : Deux lanes regroupés en un pool.

4 - ***Les artefacts :***

Les modélisateurs peuvent créer leurs propres types d'artefacts, qui ajoutent plus de détails sur la façon dont le processus est effectué, souvent pour montrer les entrées et les sorties des activités dans le processus. Cependant, la structure de base du processus, telle que déterminée par les activités, les passerelles et les flux de séquence, n'est pas modifiée par l'ajout d'objets dans le diagramme.

La version actuelle de la spécification BPMN prédéfinit trois types d'artefacts BPD, qui sont les suivants :

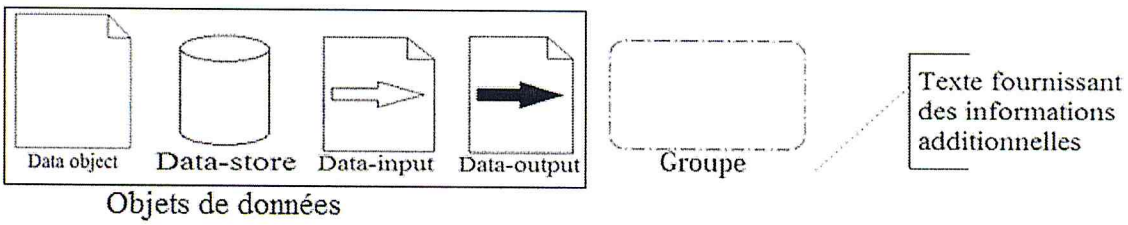


Figure 8 : Types d'artefacts.

Annexe 2 : Les Réseaux de Petri (RdP) :

La nécessité de pouvoir disposer de méthodes formelles permettant de vérifier un certain nombre de propriétés d'intérêt du système modélisé. Les réseaux de Petri, parmi l'ensemble des formalismes existants, semblent répondre à l'ensemble de ces attentes. En effet, les réseaux de Petri sont largement utilisés pour la modélisation et l'analyse de systèmes à événements discrets ainsi qu'à des validations et de vérifications dont ils font preuve. La théorie des réseaux (General Net Theory) a été créée par Carl Adam Petri, un mathématicien Allemand contemporain (d'où l'absence d'accent dans Petri).

Ce succès est dû à de nombreux facteurs. Parmi ceux-ci nous pouvons relever leur simplicité de compréhension, leur nature graphique se prêtant sans grande difficulté à la modélisation de phénomènes complexes, et la possibilité de disposer d'un arsenal de résultats mathématiques analytiques.

Alors qu'un processus peut être vu comme une collection d'événements qui sont déclenchés suite à des conditions satisfaites et rendent des conditions satisfaites après leur terminaison. Un réseau de Pétri décrit cette intuition de manière idéale et sépare de manière explicite les conditions et les événements introduits dans un processus et modélise le changement d'état du système à l'aide de la simulation du mouvement des jetons.

1 - Concept général :

C'est un outil de modélisation utilisé généralement en phase préliminaire de conception de système pour leur spécification fonctionnelle, modélisation et évaluation. Il permet notamment :

- La modélisation des systèmes informatiques,
- L'évaluation des performances des systèmes discrets, des interfaces homme-machine,
- La commande des ateliers de fabrication,
- La conception de systèmes temps réel
- La modélisation des protocoles de communication,
- La modélisation des chaines de production (de fabrication),
- En fait, tout système dans lequel circule objets et information.

2 - Notions de base d'un réseau de Petri ordinaire :

2.1 - Représentation informelle :

A - ***Condition :*** c'est un prédicat ou une description logique d'un état du système. Elle est vraie ou fausse. Donc, un état du système peut être décrit comme un ensemble de conditions.

B - ***Evénement :*** Les événements sont des actions se déroulant dans le système. Le déclenchement d'un événement dépend de l'état du système.

C - ***Déclenchement, pré-condition, post-condition :*** Les conditions nécessaires au déclenchement d'un événement sont les *préconditions* de l'événement. Lorsqu'un événement

se produit, certaines de ses pré-conditions peuvent cesser d'être vraies alors que d'autres conditions, appelées *postconditions* de l'événement, deviennent vraies.

Donc les notions graphiques sont:

- Les places permettent la description des états possibles du système (qui sont discrets), elles correspondent aux activités des processus modélisées,
- Les transitions permettent la description des événements ou les actions qui causent le changement de l'état c.-à-d. Elles représentent les évènements ou les conditions à vérifier pour avancer dans le processus,
- Les arcs sont associés aux évolutions du processus et des flux d'informations.

Un réseau de Petri est composé donc de places, transitions et arcs :

- Une place (des prédicats) est représentée par un cercle → ◯
- Une transition par un trait ou rectangle → ▮ ou ▮
- Une marque (jeton) par un point noir → •
- Prédicat est vrai → ◉
- Prédicat est faux → ◯
- Un arc relie soit une place à une transition (arc ne relie jamais deux sommets de la même famille), soit une transition à une place → ◯—▮
- Une place P_i est en amont de T_j s'il existe un arc menant de P_i à T_j. On définit de même P_i en aval de T_j. Cette définition est également valable pour les transitions :

Pi est en amont de Tj Pi est en aval de Tj

Figure 9 : Différence entre place en amont et place en aval.

PS : l'état du système (réseau) est souvent appelé marquage d'un réseau de Petri.

2.2 - ***Représentation formelle (algèbre linéaire) :***

Un réseau de Petri non marqué est un quadruplet Q=<P, T, Pré, Post> tel que :

- P = {P1, P2, …, Pn} est un ensemble fini et non vide de places ;
- T = {T1, T2, …, Tm} est un ensemble fini et non vide de transitions ;
- Pré : PxT →{0, 1} est l'application d'incidence avant ;
- Post : PxT →{0, 1} est l'application d'incidence arrière.

Tel que :

« pré (P_i, T_j) » est le poids « k » de l'arc reliant une place à une transition :

$$\text{pré}(P_i, T_j) = \begin{cases} k \text{ si l'arc } (P_i, T_j) \text{ existe} \\ 0 \text{ sinon} \end{cases}$$

« post (P_i , T_j) » est le poids « k » de l'arc reliant une transition à une place :

$$\text{poste}(T_j, P_i) = \begin{cases} k \text{ si l'arc } (T_j, P_i) \text{ existe} \\ 0 \text{ sinon} \end{cases}$$

2.3 - ***Notion de flot :***

Un réseau de Petri est un modèle mathématique servant à représenter divers systèmes (informatiques, industriels…) dans lesquels il existe des flots contrôlés. Ces flots peuvent concerner des objets concrets (par exemple : pièces détachées, périphériques,..) ou abstraits (messages, entiers,...). Dans le cas le plus général nous parlerons de flots d'information.

2.4 - ***Marquage :***

A chaque place est associée un entier positif ou nul de marques ou de jetons qui est appelé marquage unitaire. Le marquage M définit l'état du système par le réseau à un instant donné. C'est un vecteur des marquages de préférence en colonne : M= [m1, m2, ..]. Le iéme élément du vecteur correspond au nombre de jetons contenus dans la place Pi.

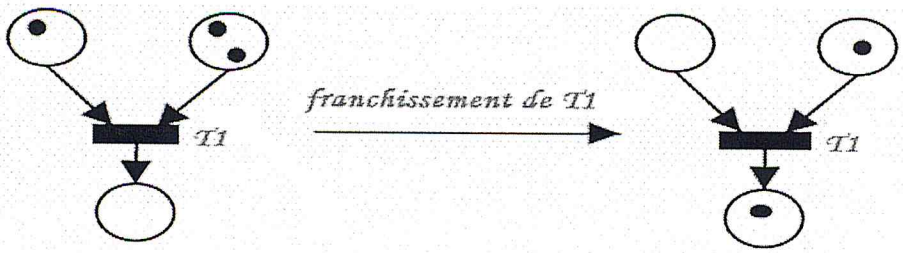
2.5 - ***Franchissement d'une transition:***

Le franchissement d'une transition consiste à retirer un jeton de chacune des places en amont (ou toutes les places d'entrée de la transition) et à rajouter un jeton à chacune des places en aval (toute les places de sortie de la même transition). Donc Une transition est franchissable s'il existe au moins un jeton dans toutes ses places qui lui sont en amont.

Quelques remarques :

- Une transition franchissable n'est pas forcément franchie (par exemple, deux transitions disposant en amont d'une seule place munie d'un seul jeton).
- Une transition sans place d'entrée est toujours franchissable (transition source).
- Le franchissement d'une transition source consiste à rajouter un jeton à chacune de ces places de sortie et une transition sans place de sortie est une transition puits.
- Le franchissement d'une transition puits consiste à retirer un jeton de chacune de ses places d'entrée.
- Le franchissement d'une transition n'est pas divisible.
- On peut remarquer qu'il n'y a pas de conservation du nombre de jetons.
- Lorsqu'une transition est validée, cela n'implique pas qu'elle soit immédiatement franchie, ce n'est qu'une possibilité.

- Il n'y a qu'un seul franchissement à la fois et un franchissement est de durée nulle.



RdP marqué avant franchissement de T1 RdP après franchissement de T1.

Figure 10 : Franchissement d'une transition.

2.6 - ***Séquence de franchissement :***

Une séquence de franchissement S est une succession de franchissement de transitions Ti Tj…Tk (suite de franchissement) à partir d'un marquage initial Mi pour aboutir au marquage final Mj. On note : $M_i [S \rightarrow M_j$ ou $M_i [S >$.

2.7 - ***Marquages accessibles (ou atteignable):***

L'ensemble des marquages accessibles (ou atteignables) M à partir d'un marquage initial M0 par une séquence de franchissement S c.-à-d. l'ensemble des marquages Mi qui peuvent être atteint par le franchissement d'une séquence S à partir du marquage initial M0.

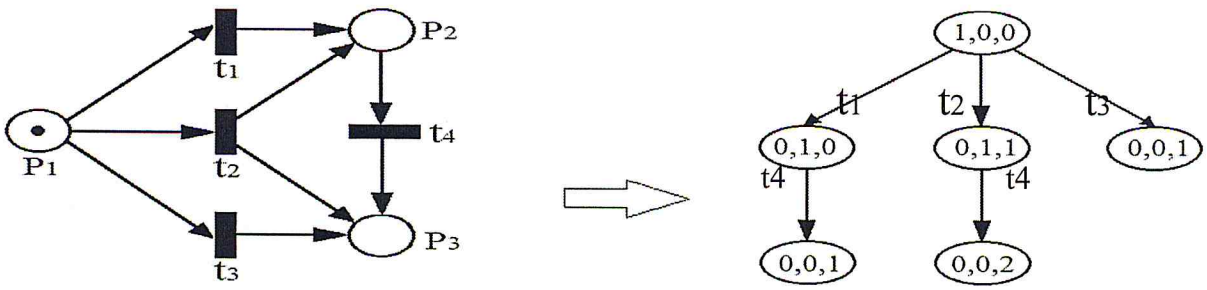
On le note $*M_0$ comme suit : $M_0 = \{ M_i$ tel que $M_i [S \rightarrow M_j \}$.

2.8 - ***Graphe de marquage :***

Il sert comme première idée à prospecter les propriétés d'un RdP, ses sommets correspondent à des marquages accessibles et ses arcs aux franchissements d'une transition permettant de passer d'un marquage à l'autre. Deux situations peuvent alors se présenter :

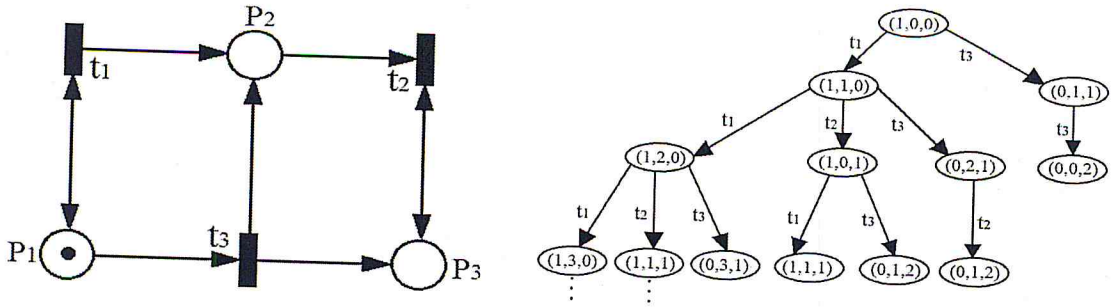
1. Le graphe est fini : C'est le cas le plus avantageux car dans ce cas toutes les propriétés peuvent être déduites simplement par vérification de celui-ci.

2. Le graphe est infini : Dans ce cas, on élabore un autre graphe appelé "graphe de couverture" qui autorise la déduction de certaines propriétés.



Réseau de Petri Graphe des marquages fini

Figure 11 : Exemple d'un RdP ayant un graphe de marquage fini.



Réseau de Petri

Graphe des marquages infini

Figure 12 : Exemple d'un RdP ayant un graphe de marquage infini.

PS : On utilise le graphe de marquages quand le nombre de marquages accessibles est fini.

2.9 - ***Réseaux de Petri autonomes :***

Les réseaux de Petri autonomes est un RdP dont les instants de franchissement ne sont pas connus qui permettent la description du fonctionnement, de ce qui arrive, donc d'étudier les propriétés logiques des systèmes, c'est-à-dire les propriétés qui dépendent de l'ordre des événements, mais sans faire référence aux instants où ils arrivent.

A - ***Modélisation par réseaux de Petri autonome :***

Principe 1 : Les éléments actifs (dynamiques) et passifs (inactifs) du système doivent être bien distingués, un élément passif peut contenir/stocker des objets ou de l'information. Un élément actif peut transporter des objets ou de l'information.

Principe 2 : Les arcs entre éléments passifs et actifs ne doivent pas représenter un élément réel du système mais une relation abstraite entre éléments.

Principe 3 : Un arc ne doit jamais relier deux éléments du même type. Si ces principes ne sont pas suivis, on aboutit à terme à des problèmes de modélisation.

Principe 4 : La même structure de réseaux représentant l'architecture du système modélisé devra pouvoir être modélisée pour d'écrire sa dynamique.

Principe 5 : On obtient une description plus détaillée d'un système

- Soit en remplaçant un élément du système par un sous réseau,
- Soit en ajoutant un nouvel élément au système.

B - ***Réseaux de Petri généralisés :*** Des poids (nombres entiers strictement positifs) sont associés aux arcs. Lorsque qu'un arc possède un poids n et qui part d'une place Pi vers une transition Tj, sa implique que en amont de cette transition, ces poids représentent le nombre de jetons nécessaires dans les places amonts Pi pour rendre cette transition franchissable (ces jetons sont alors retirés des places amonts). En aval, ces poids représentent le nombre de jetons à rajouter dans les places Pi en aval. Lorsque le poids n'est pas signalé, il est égal à un par défaut.

Remarque : Tout RdP généralisé peut être transformé en RdP ordinaire. Malheureusement ce type de transformation bien que possible engendre souvent une grande complexité.

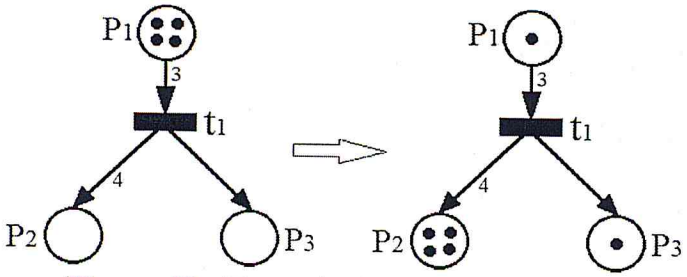


Figure 13 : Exemple d'un RdP généralisé.

C - ***Arbres de couverture :*** c'est un graphe particulier dans lequel il n'y a pas de boucle ni de circuit. Donc voici l'algorithme qui permet la construction de l'arbre de couverture :

2.10 - ***Réseaux de Petri temporisés :***

Un RdP non autonome temporisés est utiles pour l'évaluation des performances d'un système dont l'évolution est conditionnée par des événements externes ou par le temps. Soit les temporisations sont associées aux places (RdP P-temporisé) soit aux transitions (RdP T-temporisé). De plus l'exploitation de la théorie associée aux RdP permet, par la recherche des P et T invariants de répondre à de nombreux problèmes.

A - ***Réseaux de Petri P - temporisés :*** Pour chaque place Pi on associe une temporisation (valeur rationnelle positive évidemment). On notera di la temporisation de la place Pi.

A.1 - ***Principe de fonctionnement :*** Lorsqu'une marque apparu dans une place temporisée, on dit qu'elle est indisponible pendant un temps di. Quand le temps est écoulé, la marque devient disponible. On parle de fonctionnement à vitesse maximale quand on franchit chaque transition dès qu'elle devient tirable (franchissable). Dans ce cas, on peut compléter le graphe des marquages en ajoutant à chaque arc du graphe associant un marquage Mi à un marquage Mj par la transition Tk le temps séparant l'obtention du marquage Mi du franchissement de la transition Tk. Un RdP P-temporisé fonctionne en vitesse propre si toute marque ne reste dans une place que pendant sa durée d'indisponibilité. La fréquence de franchissement Fj d'une transition Tj est le nombre moyen de franchissements d'une transition par unité de temps.

B - ***Réseaux de Petri T - temporisés :*** On associe la temporisation aux transitions, le T-franchissement d'une transition T_j temporisé d'une durée d_j franchissable à une date D_a consiste soit :

- A réserver un jeton dans chacune des places en amont de T_j à la date D_a (on le représente évidé) pour franchir T_j, (cas avec réservation),
- A ne pas réserver les jetons qui pourront franchir une autre transition en cas de conflit.
- Et à ajouter un jeton dans chacune des places en aval de T_j à la date $D_a + d_j$.

B.1 - ***Principe de fonctionnement :*** Les événements (disparition/apparition de jeton) aux instants D_a et $D_a + d_j$ sont instantanés. D_a est appelé la « date de début du franchissement »

et $D_a + d_j$, la « date de fin du franchissement ». Une marque peut avoir deux états :

- disponible (ou non réservée)
- ou bien réservée pour le franchissement d'une transition.

PS : on peut toujours passer d'un RdP T-temporisé à un RdP P-temporisé.

2.11 - ***Extension des réseaux de Petri :***

La modélisation d'un système réel peut mener à des réseaux de Petri de taille trop importante rendant leur manipulation et/ou leur analyse difficile. La question est alors de modifier (étendre) la modélisation par RdP de façon à obtenir des modèles RdP de plus petite taille. Cette question a motivé la nécessité de faire des extensions des réseaux de Petri. Ainsi :

- Certaines propriétés ne peuvent pas être exprimées à l'aide des réseaux usuels.
- Besoin d'avoir une information plus précise sur les jetons transitant dans le réseau.

Ceci a donné naissance à plusieurs extensions de réseaux de Petri tels que les réseaux de Petri colorés, les réseaux de Petri à capacité, les réseaux de Petri Predicate/Transition, etc. Nous donnerons dans la section suivante un aperçu général sur ces derniers :

A - ***Graphe d'état :*** Un graphe d'état est un réseau de Petri non marqué mais chaque transition possède une seule place d'entrée et une seule place de sortie exactement.

B - ***Graphe d'événement :*** Un RdP est dit un graphe d'événement si chaque place possède exactement une seule transition d'entrée et une seule transition de sortie.

C - ***Réseau de Petri sans conflit :*** Un Rdp sans conflit est un réseau dans lequel chaque place a au plus une transition de sortie. Donc est un réseau qui possède une place avec au moins deux transitions de sorties.

Un conflit est noté: [Pi , {T1,T2,…,Tn}] ; avec T1,T2,…,Tn étant les transitions de sorties de la place Pi .

D - ***Réseau de Petri à choix libre :*** Un RdP est dit un réseau à choix libre si pour chaque conflit [Pi, {T1, T2,…,Tn}] aucune des transitions T1,T2,…,Tn ne possède aucune autre place d'entrée que Pi.

E - ***Réseau de Petri simple :*** Un Réseau de Pétri simple est un RdP dans lequel chaque transition ne peut être concernée que par un conflit au plus.

F - ***Réseau de Petri pur :*** Un RdP pur est un réseau dans lequel il n'existe pas de transition ayant une place d'entrée qui soit à la fois place de sortie de cette transition.

G - ***Réseau de Petri à capacités :*** c'est un RdP dans lequel des capacités (nombres entiers strictement positifs) sont associées aux places. si une place P_i dispose d'une capacité cap(P_i), alors le franchissement d'une transition amont de P_i n'est possible que si le franchissement de cette transition ne conduit pas à un nombre de marques de P_i supérieur à cap(P_i).

H - ***Réseau de Petri à priorités :*** Dans un tel réseau si on atteint un marquage tel que plusieurs transitions sont franchissables, on doit franchir la transition qui a la plus grande priorité. Il est utilisé lorsqu'on veut imposer un choix entre plusieurs transitions validées. Il est composé d'un réseau de Petri et d'une relation d'ordre partiel sur les transitions du réseau.

Remarque :

- Un RdP à priorités ne peut pas être transformé en un RdP ordinaire.
- Pour les RdP généralisés on pourra placer des contraintes au niveau des transitions.

I - ***Réseaux de Petri FIFO*** **(First In, First Out, premier entré, premier sorti)** : les marques y sont différenciées de telle manière que l'on puisse modéliser différents messages et les règles de franchissement sont modifiées pour modéliser le mécanisme FIFO.

Chaque place du réseau représente une file FIFO et le marquage de la place représente le contenu de cette file. Formellement ce marquage est une lettre faisant partie d'un alphabet choisi. Par exemple le marquage prendra ses valeurs dans $A = f®; \bar{}; °; \pm g$.

Ce type de modèle convient particulièrement à la modélisation et à l'analyse des processus séquentiels communiquant par des canaux FIFO.

Remarque : En général, un réseau FIFO ne peut pas être transformé en RdP ordinaire.

J - ***Réseaux de Petri colorés :*** Dans cette extension, une information est associée aux jetons. Cette information est appelée couleur. Cette information supplémentaire permet de modéliser des comportements plus complexes ou de simplifier fortement la structure de RdP non colorés en « mettant en facteur» des parties de graphe similaires.

Un réseau coloré est constitué : de places, transitions et arcs, comme les réseaux de places et transitions, de marques individuelles différentiables les unes des autres, par leur couleur, d'ou le nom de réseau coloré.

K - ***Réseaux de Petri colorés à prédicats :***

Un réseau à prédicats est constitué de places, transitions, et arcs tout comme les réseaux colorés cités plus haut, de variables (notées le plus souvent par des lettres minuscules) étiquetant les arcs du réseau.

Dans un tel réseau, une opération de substitution est effectuée au niveau d'une transition t si et seulement si toutes les variables étiquetant les arcs en entrée et en sortie de t sont remplacées par des constantes.

Remarque :

1. Il existe également de nouvelles extensions : RdP continu, flous, objets, stochastiques, à arcs inhibiteurs, à priorité …etc. qui ne font pas l'objet de notre travail.
1. Les RdP généralisé, à capacité et coloré (dont le nombre de couleur est fini) peuvent tous être traduits en RdP ordinaires.