OntoApp : une approche déclarative pour la simulation du fonctionnement d'un logiciel dès une étape précoce du cycle de vie de développement

Tuan Anh Pham

UNIVERSITÉ DE CÔTE D'AZUR

ÉCOLE DOCTORALE DES SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Présentée en vue de l'obtention du grade de

Docteur en Sciences

de l'Université de Côte d'Azur

Spécialité : Informatique

Présentée et soutenue par

Tuan Anh PHAM

OntoApp : une approche déclarative pour la simulation du fonctionnement d'un logiciel dès une étape précoce du cycle de vie de développement

Thèse dirigée par: **Nhan LE THANH**

équipe WIMMICS,I3S, CNRS UMR-7271, INRIA Sophia Antipolis

soutenue le 22 Septembre, 2017

Devant le jury composé de :

Rapporteurs :	Myriam LAMOLLE	-	Professeur, Université Paris 8
	Ladjel BELLATRECHE	-	Professeur, ISAE - ENSMA
	Parisa GHODOUS	-	Professeur, Université Lyon 1
Directeur :	Nhan LE THANH	-	Professeur, Université Côte d'Azur
Examinateurs :	Andrea TETTAMANZI	-	Professeur, Université Côte d'Azur
	Jean-Yves TIGLI	-	MdC, Université Côte d'Azur

Remerciements

Je voudrais tout d'abord remercier les membres du jury d'avoir jugé mon travail. Je remercie Mme. Myriam LAMOLLE, Mme. Parisa GHODOUS et M. Ladjel BELLATRECHE d'avoir accepté d'être les rapporteurs de ce manuscrit. Je remercie M. Andrea TETTAMANZI et M. TIGLI Jean-Yves d'avoir accepté d'être l'examinateur de ce manuscrit.

J'adresse ma grande reconnaissance à mon directeur de thèse M. Nhan LE THANH d'avoir accepté ma candidature de thèse et m'avoir laissé sur le domaine du web sémantique et l'ingénierie logiciel avec des connaissances intéressantes. Je le remercie profondément d'avoir été disponible pour me guider tout au long de ma thèse. Grâce à sa direction des cours dès le début aux suggestions et corrections pour finir ce manuscrit j'ai appris les méthodes et les approches de recherche.

J'adresse également mes remerciements à l'équipe WIMMICS de m'avoir donné une ambiance amicale. Je n'oublierai jamais les moments inoubliables de pause du café partagé avec tous les membres de l'équipe.

Je tiens à remercier le laboratoire I3S et INRIA Sophia Antipolis de m'avoir accueilli dans son établissement et aussi pour tout son soutien, sa sollicitude. Merci également aux personnels administratifs, techniques et de la bibliothèque : Christine Foggia,..

Merci à mes amis vietnamiens qui ont partagé des voyages intéressants, des moments de nouvel an, des événements spéciaux et m'ont aidé à surmonter les moments les plus durs pendant mon séjour de quatre ans en France.

J'adresse mes remerciements aux projets 322, maintenant 911, du gouvernement du Vietnam qui m'ont offert une occasion d'étudier en France.

Enfin, j'ai une grande pensée pour ma famille, plus particulièrement pour mes parents, ma femme et ma fille qui m'ont toujours soutenu et sont toujours le point d'appui de ma vie.

OntoApp : une approche déclarative pour la réutilisation et la simulation du fonctionnement d'un logiciel dès l'étape précoce du cycle de vie de développement

Résumé: dans cette thèse, nous étudions plusieurs modèles de collaboration entre l'ingénierie logiciel et le web sémantique. À partir de l'état de l'art, nous proposons une approche d'utilisation de l'ontologie dans la couche de métier d'une application. L'objectif principal de notre travail est de fournir au développeur des outils pour concevoir la matière déclarative une couche de métier "exécutable" d'une application afin de simuler son fonctionnement et de montrer ainsi la conformité de l'application par rapport aux exigences du client au début du cycle de vie du logiciel. Un autre avantage de cette approche est de permettre au développeur de partager et de réutiliser la description de la couche de métier d'une application dans un domaine en utilisant l'ontologie. Celle-ci est appelée "patron d'application". La réutilisation de la description de la couche de métier d'une application est un aspect intéressant à l'ingénier logiciel. C'est le point-clé que nous voulons considérer dans cette thèse. Dans la première partie de notre travail, nous traitons la modélisation de la couche de métier. Nous présentons d'abord une approche fondée sur l'ontologie pour représenter les processus de métiers et les règles de métiers et nous montrons comment vérifier la cohérence du processus et de l'ensemble des règles de métier. Puis, nous présentons le mécanisme de vérification automatique de la conformité d'un processus de métier avec un ensemble de règles de métier. La deuxième partie de cette thèse est consacrée à définir une méthodologie, dite de personnalisation, de création une application à partir d'un "patron d'application". Cette méthode permettra à l'utilisateur d'utiliser un patron d'application pour créer sa propre application en évitant les erreurs de structures et les erreurs sémantiques. Nous introduisons à la fin de cette partie, la description d'une plateforme expérimentale permettant d'illustrer la faisabilité des mécanismes proposés dans cette thèse. Cette plateforme est réalisée sur un SGBD relationnel.

Mots clés : Génie logiciel, web sémantique, ontologie, OWL, SWRL, RDF, conformité aux processus de métiers, couche de métier.

OntoApp : A declarative approach for software reuse and simulation in early stage of software development life cycle

Abstract : In this thesis, we study several models of collaboration between Software Engineering and Semantic Web. From the state of the art, we propose an approach to the use of ontology in the business application layer. The main objective of our work is to provide the developer with the tools to design, in the declarative manner, a business "executable" layer of an application in order to simulate its operation and thus show the compliance of the application with the customer requirements defined at the beginning of the software life cycle. On the other hand, another advantage of this approach is to allow the developer to share and reuse the business layer description of a typical application in a domain using ontology. This typical application description is called "Application Template". The reuse of the business layer description of an application is an interesting aspect of software engineering. That is the key point we want to consider in this thesis.

In the first part of this thesis, we deal with the modeling of the business layer. We first present an ontology-based approach to represent business process and the business rules and show how to verify the consistency of business process and the set of business rules. Then, we present an automatic check mechanism of compliance of business process with a set of business rules.

The second part of this thesis is devoted to define a methodology, called personalization, of creating of an application from an "Application Template". This methodology will allow the user to use an Application Template to create his own application by avoiding deadlock and semantic errors. We introduce at the end of this part the description of an experimental platform to illustrate the feasibility of the mechanisms proposed in the thesis. This platform s carried out on a relational DBMS.Finally, we present, in a final chapter, the conclusion, the perspective and other annexed works developed during this thesis.

Keywords: Software Engineering, Semantic Web, Ontology, OWL, SWRL, RDF, Business Process Compliance, Business Logic Layer.

Table des Matières

Liste des Figures

Liste des Tables

Acronyme

BPO Business Process Ontology

BRO Business Rule Ontology

DMO Data Matching Ontology

CPN Colored Petri Net

OWL Web Ontology Language

SWRL Semantic Web Rule Language

OntoApp Ontology Based Application Demonstration

SBVR Semantics of Business Vocabulary and Business Rules

ECA Event Condition Action

SDLC Software Development Life Cycle

LTL Linear Temporal Logic

CTL Computation Tree Logic

BPEL Business Process Execution Language

BPC Business Process Compliance

BPM Business ProcessManagement

BPMN Business ProcessModelling Notations

CMF ComplianceManagement Frameworks

ECAE Event Condition Action Event

EC Event-Calculus

FCL Formal Contract Language

WF-net Workflow-Nets

RDF Resource Description Framework

RDFS Resource Description Framework Schema

SPARQL SPARQL Protocol and RDF Query Language

ODA Ontology Driven Architecture

OMG Object Management Group

FSM Finite State Machine

Chapitre 1

Introduction

Contenu

1.1 Contexte, motivation, objectifs

1.1.1 Contexte

Le web sémantique donne une structure typique qui permet de partager et de réutiliser l'information sur application [SEM]. Il s'agit d'un " *Web of Data*" (Traitement des données) pouvant être traité à l'aide de la machine, [SEM]. Les méthodes du web sémantique sont convaincantes pour améliorer l'apprentissage sur le web.

Au cœur du web sémantique, l'ontologie est utilisée pour représenter explicitement nos conceptualisations. Les ontologies sont conçues pour représenter les concepts et les relations entre eux dans un domaine et en soutenant le raisonnement. L'ingénierie de l'ontologie dans le web sémantique est principalement représentée

par le langage de l'ontologie, par exemple *Resource Description Framework* (RDF) [W3Ca], *Resource Description Framework Schema* (RDFS) [W3Cb] et *Web Ontology Language* (OWL) [OWL].

L'ingénierie logicielle consiste à disséquer les besoins des clients et à planifier, développer et tester les applications des utilisateurs finaux qui répondent à ces exigences en utilisant le langage de programmation du logiciel. C'est l'utilisation de la conception de normes pour le développement de logiciels. Plutôt que la programmation de base, l'ingénierie logicielle est utilisée pour un système de logiciel plus vaste et plus complexe, qui est utilisé comme un système important pour les organisations.

La figure 1.1 représente le cycle de vie du développement logiciel général. Normalement, un projet commence toujours avec la phase d'analyse des besoins, cette phase aide les personnes du projet à comprendre l'exigence du client. L'étape suivante consiste à concevoir la fonctionnalité du logiciel. Après la phase de conception, le développeur élaborera le logiciel en fonction de la conception du système. Avant de déployer le logiciel, il faut vérifier qu'il n'y ait pas d'erreur lors de l'exécution du logiciel.

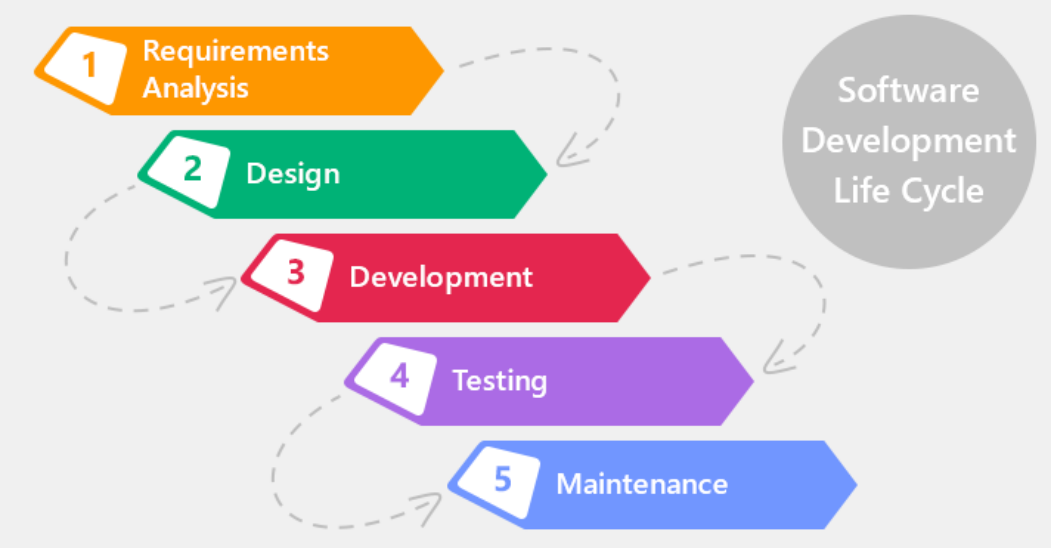

Figure 1.1: Cycle de vie de développement

Il existe de nombreuses recherches qui ont attiré l'attention de la collaboration du domaine entre l'ingénierie logicielle et le web sémantique. Elles ont essayé d'innover en appliquant les avantages du web sémantique au domaine de l'ingénierie

logicielle. Les études publiées de manière prolifique ont démontré les avantages des ontologies dans le domaine de l'ingénierie logicielle, ce qui nous motive à explorer d'autres possibilités disponibles dans ce domaine collaboratif.

Des études récentes ont fait connaître la collaboration entre le domaine de l'ingénierie logicielle (*software engineering*) et la technique de web sémantique (*semantic web*), qui montrent les avantages de l'intégration de techniques sémantiques avec l'ingénierie logicielle [PZ14]. Les ontologies ont joué un rôle important dans cette direction [GPZ+13], [BC13]. La direction de recherche en utilisant des ontologies pour échanger et interconnecter les connaissances en génie logiciel a été reconnue et acceptée par la communauté du génie logiciel.

Ce groupe de génie logiciel avec le web sémantique a tiré avantage des organismes de normalisation et a ouvert de nouvelles directions de recherche. *Ontology-Driven Architecture (ODA)*, qui est la création de *W3C's Software Engineering Best Practices Working Group*, tente de mettre en évidence les meilleures pratiques pour l'utilisation d'ontologies dans le génie logiciel [PJD+05]. *Ontology Definition Metamodel (ODM)* du *Object Management Group (OMG)* [OMG] permet d'intégrer les langages d'ontologie web (c'est-à-dire les ontologies) dans le processus de développement de logiciels basé sur des principes d'ingénierie axés sur le modèle. Ainsi, la motivation est fort pour mener la recherche sur l'étude de l'acceptation et de l'utilisation l'outil logiciel.

1.1.2 Motivation

Après avoir étudié les travaux précédents sur la collaboration de l'ingénierie logicielle et du web sémantique, nous nous rendons compte qu'il n'y a pas beaucoup de recherches antérieures qui envisagent d'appliquer l'avantage du technologie web sémantique sur la couche de métier d'une application (ce problème sera analysé dans la section l'état de l'art). De ce point de vue, nous nous sommes décidés à poursuivre notre recherche dans ce sens et à nous concentrer sur la création d'une méthodologie en utilisant une ontologie pour le partage de la description de la couche de métier exécutable d'une application typique dans un domaine . Étant donné que de nombreuses personnes rencontrent le même problème dans un domaine spécifique, il est préférable d'avoir une méthodologie qui permet aux gens de partager la logique de leur solution à d'autres personnes sous une forme exécutable.

Lorsqu'une personne réutilise une couche de métier exécutable partagée, il suffit de personnaliser le modèle pour qu'il corresponde à ses besoins et à ses données. L'avantage de réutiliser la conception d'une couche de métier est de réduire le coût et le temps de développement du logiciel. Cette approche peut changer la façon dont les gens partagent les expériences, au lieu de partager l'information de manière écrite, orale... Ils peuvent partager l'expérience sur une forme exécutable. Ce point vue nous a inspiré à faire ce travail.

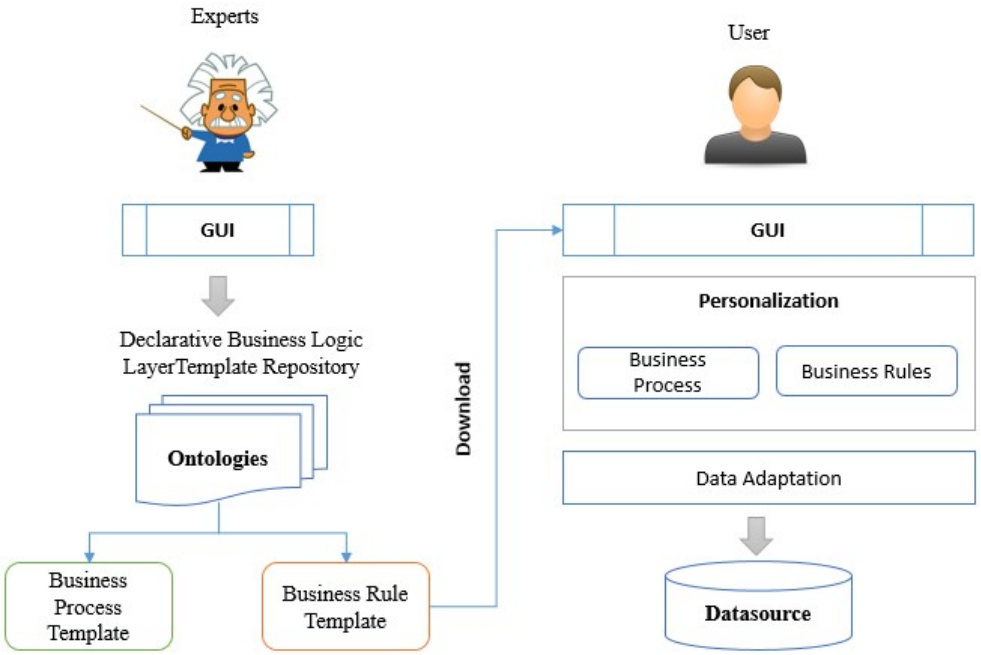

Figure 1.2: Objectif de notre travaux

1.1.3 Objectifs

La figure 1.2 est l'esquisse de notre objectif pour laquelle nous proposons deux parties principales dans nos recherche.

La première partie consiste à fournir un outil qui aide les experts et les personnes à partager leurs expériences sur le développement d'une application typique dans un domaine en décrivant un modèle de couches de métier (patron de couches de métier déclaratif). Le patron de couches de métier a deux parties principales: processus de

métier et règles de métier. Le processus de métier décrit la logique de l'exécution de l'application. Les règles de métier décrivent les contraintes qui sont appliquées sur une instance de processus de métier. Le logiciel OntoApp doit assurer qu'il n'y a pas des erreurs de structure dans le processus de métier, et assurer également qu'il n'y a pas de conflit entre les règles dans l'ensemble des règles de métier. Enfin, l'outil doit avoir une fonctionnalité pour vérifier la conformité d'un processus de métier avec le jeu de règles de métier.

La deuxième partie consiste à fournir un outil permettant aux personnes de télécharger un patron de processus de métier pour le personnaliser et le réutiliser dans leur système. Cet outil doit pouvoir personnaliser un patron de processus de métier et un patron de règles de métier pour qu'il soit correspondant avec ses propres exigences. Une autre caractéristique de cet outil est d'adapter la source de données de l'utilisateur au processus de métier personnalisé.

1.2 Questions de recherche

A partir de l'objectif de notre recherche, nous avons divisé notre travail en questions de recherche majeures suivantes :

- **Question de recherche 1 :**
  comment construire un modèle de couche de métier "exécutable" d'une application typique dans un domaine (y compris le modèle de processus de métier, le modèle de règles de métier) et assurer son exactitude ?

- **Question de recherche 2 :**
  comment vérifier la conformité d'un processus de métier avec un ensemble de règles de métiers prédéfinies automatiquement et s'assurer que les processus de métiers personnalisés sont conformes aux règles de métier personnalisées ?

- **Question de recherche 3 :**
  comment autoriser les utilisateurs à personnaliser un patron de couche de métier d'une application pour correspondre avec leur propre exigence sans faire d'erreur de structure ni d'erreur sémantique ?

- **Question de recherche 4 :**

comment intégrer la source de données d'un utilisateur à un patron de processus de métier et le rendre exécutable sur le système de l'utilisateur ?

1.3 Méthodologie de recherche

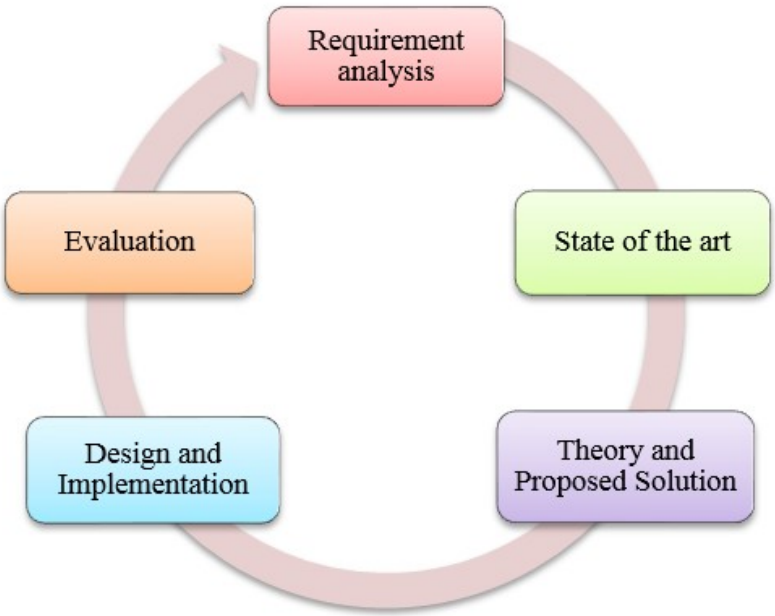

Figure 1.3: Méthodologie de recherche

La figue. 1.3 démontre la méthodologie de recherche. Au début, nous avons demandé un examen des exigences en vue d'enquêtes contextuelles. Cela nous a permis de planifier une avancée de l'approbation des processus de métier. Nous avons choisi une prémisse hypothétique, avons proposé une réponse et l'avons exécuté dans des modèles. Les arrangements ne traitent pas vraiment toutes les parties de l'examen préalable sur le prototype, mais peuvent plutôt se concentrer sur des points de vue spécifique. En utilisant les modèles que nous avons créés, nous avons examiné et évalué nos réponses en utilisant des informations provenant d'applications. Cela peut provoquer un cycle supplémentaire sur l'avancement et l'utilisation (par exemple, dans le cas où les idées créées ne couvrent pas encore tous les angles pertinents ou ne donnent pas suffisamment d'arrangements). L'évaluation des arrangements créés peut également provoquer un nouvel accord qui entraîne des ajustements ou

des raffinements de l'arrangement lorsque les études révèlent des nécessités supplémentaires.

1.4 Contributions

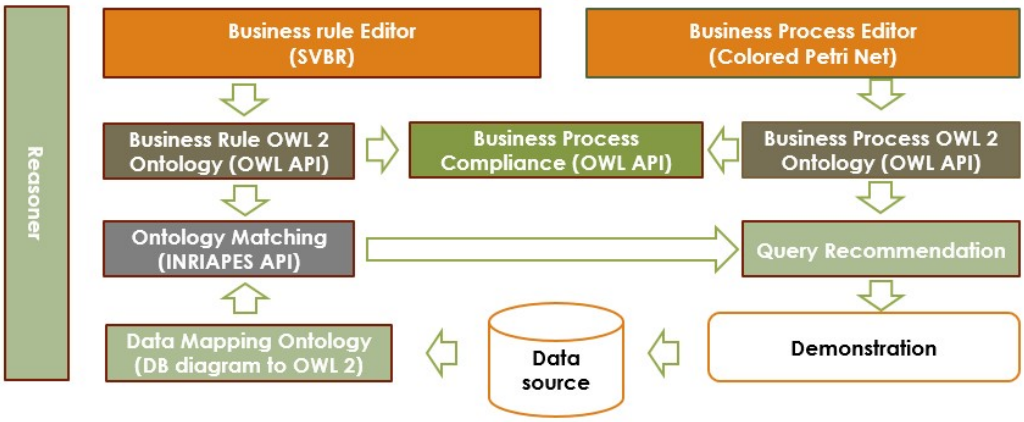

Figure 1.4: La structure principale

L'étude de ces questions concrètes de recherche a conduit à deux contributions principales dans les domaines de la collaboration entre l'ingénierie logicielle et le web sémantique.

La première contribution se concentre sur la construction d'une couche de métier exécutable d'une application typique dans un domaine utilisant l'ontologie. Elle apporte, tout d'abord, une approche basée sur l'ontologie pour représenter la couche de métier d'une application typique. En fonction des concepts et des règles, la couche de métier est vérifiée pour en assurer l'exactitude pendant la phase de conception du cycle de développement logiciel.

La deuxième contribution principale est de permettre au développeur de partager et de réutiliser une couche de métier d'une application en utilisant un patron d'application. Afin de faire cette contribution, nous proposons une méthode pour modifier un patron d'application correctement sans faire d'erreurs, et nous proposons également une méthode pour permettre à l'utilisateur d'intégrer une source de données dans une couche de métier pour aider chaque étape dans un processus de métier qui a besoin de se connecter et de travailler avec la source de données lors de l'exécution de la couche de métier. Nous allons entrer plus en détail dans

les deux sous-sections suivantes :

1.4.1 Couche de métier exécutable basée sur l'ontologie

Cette contribution répond à la question de recherche 1 et à la question de recherche 2. Nous divisons cette contribution en trois sous contributions comme suit :

- **Patron de processus de métier fondée sur l'ontologie**
  Nous utilisons réseaux de Pétri coloré (*Colored Petri Net*-CPN) pour modéliser un processus de métier. Ensuite, le processus de métier (un graphe de CPN) est représenté par une ontologie appelée "*Busines Process Ontology*"(BPO). Cette ontologie comporte un ensemble de concepts pour représenter un graphe de CPN (processus de métier), elle comporte aussi un ensemble d'instances des processus de métiers, et les contraints utilisées par un raisonneur pour vérifier automatiquement la cohérence des processus de métier. La contribution de cette partie est de fournir une méthode pour représenter et vérifier le modèle d'un processus de métier en utilisant l'ontologie et le raisonnement. Cette contribution sera analysée plus en détail au chapitre IV.

- **Définition de règle de métier fondée sur l'ontologie**
  La deuxième partie importante d'une couche de métier est la règle de métier. Nous proposons une méthode pour traduire l'ontologie *Semantics of Business Vocabulary and Business Rules* (SBVR) à OWL pour permettre à l'utilisateur de définir les règles de métier. Lorsqu'un ensemble de règles de métier est défini par SBVR, la terminologie, le verbe et la phrase sont traduites en une ontologie qui s'appelle "*Business Rule Ontology* (BRO)". Lorsqu'une nouvelle règle de métier est ajoutée, un raisonneur vérifiera sur la base de connaissances des règles de métier pour détecter s'il y a un conflit qui se produit ou non. Cette contribution sera analysée plus en détail au chapitre V.

- **Vérification de la conformité aux processus de métiers fondée sur l'ontologie**
  Cette contribution consiste à proposer une approche basée sur l'ontologie pour la vérification de la conformité des processus de métier. Lorsqu'un processus de métier est créé, il doit suivre les règles de métier prédéfinies. Notre solution est d'aligner le concept dans BPO et BRO ; après la cartographie, la règle

dans BRO sera appliquée à l'instance de processus de métier dans BPO. Cette contribution sera discutée plus en détail au chapitre V.

D'autre part, au niveau théorique, nous avons proposé une extension de CPN nommé CPN-BP pour modéliser un processus de métier dans l'ingénierie logiciel. CPN-BP est un graphe CPN avec deux nouvelles définitions, la définition de *Internal-Token* et *External-Token*. Ces définitions permettent une transition dans un graphe CPN qui peut travailler avec l'événement extérieur ou l'objet extérieur. Cette contribution sera introduite dans le chapitre IV.

1.4.2 Réutilisation d'une couche de métier

Dans la partie précédente, nous proposons une approche déclarative pour la modélisation d'une couche de métier en utilisant l'ontologie. Logiciel OntoApp peut être utilisé afin de définir un patron d'application pour une application spécifique dans un domaine. C'est un point fort de notre approche, car le patron d'application peut être réutilisé par la communauté. Dans d'autres cas, nous devons répondre à la question de recherche 3, et à la question de recherche 4.

Le deuxième apport de notre travail répond à ces questions de recherche. Nous proposons une approche permettant à l'utilisateur de personnaliser un modèle d'application pour que ce dernier soit correspondant avec le système d'information de l'utilisateur. Pour ce faire, nous proposons une méthode afin de personnaliser un patron de processus de métier sans créer d'erreurs de structures dans le modèle de processus de métier pendant la modification. Nous proposons également une approche pour la recommandation d'un ensemble de requêtes, selon la source de données utilisée dans le système d'information de l'utilisateur. Un processus de métier personnalisé est en fait un modèle de processus de métier qui est modifié pour s'adapter à un système d'information existant d'un utilisateur. Cette adaptation comprend l'adaptation des processus de métier, l'adaptation des données, l'adaptation des terminologies et l'adaptation des règles de métier. Chaque adaptation sera résolue par la contribution suivante :

- **Opérations de changement :**
  nous proposons un ensemble d'opérations de changement qui permettent aux utilisateurs de personnaliser un patron de processus de métier. Nous fournissons également une limitation de la modification des utilisateurs afin de ne

pas mettre en conflit le processus modifié avec un patron de règles de processus de métier prédéfini.

- **Intégration de sources de données :**
  nous proposons une approche pour aligner la source de données de l'utilisateur avec une ontologie. Avec les alignements, la requête a été définie dans le modèle de processus de métier qui sera traduit en une requête recommandée pour la source de données de l'utilisateur.

- **Recommandation de requête :**
  chaque utilisateur a son propre système d'information, y compris la source de données. Afin que les processus de métiers personnalisés soient prêts à être utilisés, nous proposons une méthode recommandant une requête qui correspond au système de source de données de l'utilisateur en utilisant la correspondance de l'ontologie.

1.5 Publications

1. Tuan Anh Pham and Nhan Le Thanh. **Ontology-based business process compliance**. International Journal of Research in Engineering and Science(IJRES), 2017 (Accepted).

2. Tuan Anh Pham and Nhan Le Thanh. **Checking the compliance of business process in business process life cycle**. In Proceedings of the RuleML 2016 Challenge, and Industry Track hosted by the 10th International Web Rule Symposium, RuleML 2016, New York, USA, July 6-9, 2016.

3. Tuan Anh Pham and Nhan Le Thanh. **An ontology-based approach for business process compliance checking**. In Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication, IMCOM 2016, Danang, Vietnam, January 4-6, 2016.

4. Tuan Anh Pham and Nhan Le Thanh. **A rule-based language for integrating business process and business rules**. In Proceedings of the RuleML 2015 Challenge, the Special Track on Rule-based Recommender Systems for the Web of Data, the Special Industry Track hosted by the 9th In-

ternational Web Rule Symposium (RuleML 2015), Berlin, Germany, August 2-5, 2015.

5. Tuan Anh Pham and Nhan Le Thanh. **Ontology-based business logic layer**. In Poster session of the 9th International Web Rule Symposium (RuleML 2015), Berlin, Germany,August 2-5,2015.

6. Tuan Anh Pham and Nhan Le Thanh. **Checking the compliance of business processes and business rules using OWL 2 ontology and SWRL**. In Proceedings of the Second International Afro-European Conference for Industrial Advancement, AECIA 2015, Villejuif (Paris-sud), France,pages 11-20, 2015.

7. Tuan Anh Pham, Thi-Hoa-Hue Nguyen, and Nhan Le Thanh. **Ontology-based workflow validation**. In The 2015 IEEE RIVF International Conference on Computing Communication Technologies Research, Innovation, and Vision for Future, RIVF 2015, Can Tho, Vietnam, January 25-28,pages 41-46, 2015.

8. Tuan Anh Pham. **Ontology-based business logic layer**. In Poster session of The 10th Reasoning Web Summer School, Athens, Greece, Sept 8-13, 2014.

1.6 Structure de thèse

Ce document est organisé en quatre parties et neuf chapitres. La partie I présente la couche de métier basée sur l'ontologie. Cette partie comprend le chapitre IV et le chapitre V. La partie 2 présente la réutilisation de la couche de métier. La partie 2 comprend le chapitre VI. La partie III présente la mise en œuvre et l'évaluation de notre recherche. La dernière partie est la conclusion de notre travail. Les chapitres sont organisés comme suit:

- **Chapitre 1 :** dans ce chapitre, nous présentons les questions de recherche, notre méthode de recherche. Nous clarifions également les principales contributions et notre motivation.

- **Chapitre 2 :** ce chapitre présente quelques antécédents théoriques qui ont été utilisés pour déployer notre approche: CPN comme théorie de vérification

de modèle, Ontologie, langage OWL.

- **Chapitre 3 :** l'état de l'art sur la collaboration entre ingénierie logicielle et web sémantique est présenté. Nous énumérons et analysons les travaux précédents sur cette direction.

- **Chapitre 4 :** dans ce chapitre, nous répondrons au RQ1, RQ2. Nous présentons notre approche basée sur l'ontologie pour représenter les processus de métier et la vérification des processus de métier. Nous classifions l'impasse du processus de métier et définissons les règles pour l'éviter.

- **Chapitre 5 :** nous proposons une approche basée sur l'ontologie pour la représentation des règles de métier. Cette ontologie est utilisée pour vérifier la cohérence du jeu de règles de métier avant de l'utiliser dans un système. Ce chapitre fournit également la solution pour la vérification de la conformité des processus de métier.

- **Chapitre 6 :** ce chapitre présente notre approche pour l'intégration de la source de données et la personnalisation de la couche de métier avant de créer une démonstration pour une application.

- **Chapitre 7 :** ce chapitre présente la mise en œuvre d'un prototype, les principales caractéristiques de ce prototype.

- **Chapitre 8 :** ce chapitre présente l'évaluation du prototype réalisé. Il est nécessaire de vérifier notre solution dans cette thèse.

- **Chapitre 9 :** ce chapitre fournit les conclusions de notre travail. Il fournit également des perspectives et des travaux futurs de le prolongement de cette thèse.

Chapitre 2

Contexte théorique

Contenu

2.1 Théorie des vérifications de modèles

2.1.1 Réseaux de Pétri

Un réseaux de Pétri [PN] est un type particulier de graphes dirigés peuplés par plusieurs types d'objets, qui sont des places, des transitions, des arcs dirigés, des jetons. Les arcs dirigés relient les places aux transitions ou les transitions vers les places. Une transition est activée lorsque chaque place dans la condition préalable de la transition est remplie.

Définition 1 : un graphe de réseaux de Pétri (ou structure de réseaux de Pétri) est un graphe bipartite (P, T, A, w) Où :

- P est l'ensemble fini de places (un type de nœud dans le graphe)
- T est l'ensemble fini de transitions (l'autre type de nœud dans le graphe)
- A $\subseteq$(P$\times$T)$\cup$(T$\times$P) est l'ensemble des arcs des places aux transitions et des transitions aux places du graphe.
- w:A$\rightarrow$1,2,3,... est la fonction de poids sur les arcs..

Exemple 2.1 : La figure 2.1 présente un graphe de réseaux de Pétri avec les ensembles suivants:

P=$\{P_1, P_2, P_3, P_4\}$
T=$\{T_1, T_2\}$
A=$\{P_1T_1, T_1P_2, T_1P_3, P_2T_2, P_2T_2, T_2P_4, T_2P_1\}$

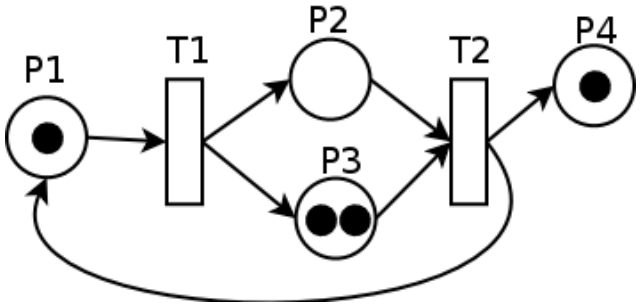

Figure 2.1: un graphe de réseaux de Pétri. Source: wikipedia

Donc, un réseaux de Pétri peut être représenté par une transition avec une place d'entrée et une place de sortie. Les jetons sont un concept spécial des réseaux de Pétri. La présence ou l'absence d'un jeton dans une place peut indiquer si une condition d'une place est vraie ou fausse. L'état est désigné par le mouvement des jetons et est causé par le déclenchement d'une transition. Le tir représente une occurrence de l'événement ou une action prise. Le tir est soumis aux conditions d'entrée, notées par la disponibilité des jetons.

2.1.1.1 Réseaux de Pétri coloré

Un réseaux de Pétri n'a pas de types ni de modules, un seul type de jetons et le réseau est plat. Avec les réseaux de Pétri coloré (CPN) [PN], il est possible d'utiliser des types de données et une manipulation de données complexes :

- Chaque jeton a attaché une valeur de données appelée couleur jeton.
- Les couleurs symboliques peuvent être étudiées et modifiées par les transitions apparentes.

Avec les CPN, il est possible de faire des descriptions hiérarchiques:

- Un grand modèle peut être obtenu en combinant un ensemble de sous-modèles.
- Interfaces bien définies entre les sous-modèles.
- Sémantique bien définie du modèle combiné.
- Les sous-modèles peuvent être réutilisés.

Les CPN sont une extension compatible avec l'arrière du concept de réseaux de Pétri. Le CPN conserve les propriétés utiles des réseaux de Pétri, en même temps, étend le formalisme initial pour permettre la distinction entre tokens [PN].

Définition 2 : Réseaux de Pétri coloré (Color Petri Net) est un ensemble N = (P, T, A, *Sigma*, C, N, E, G, I) où :

- P est un ensemble de places
- T est un ensemble de transitions
- A est un ensemble de arcs
- Σ est un ensemble de couleurs définies dans le modèle CPN. Cet ensemble contient toutes les couleurs possibles, les opérations et les fonctions utilisées dans CPN.
- C est une fonction de couleurs. Il aligne les places dans P en couleurs dans Σ.
- N est une fonction de nœuds. Il aligne A en $(P \times T) \bigcup (T \times P)$.
- E est une fonction d'expression d'arc. Il aligne chaque arc $a \in A$ dans l'expression e. Les types d'entrée et de sortie des expressions d'arc doivent correspondre aux types de nœuds auxquels l'arc est connecté. L'utilisation de la fonction de nœud et de la fonction d'expression d'arc permet à plusieurs arcs de connecter le même paire de nœuds avec différentes expressions d'arcs.

- G est une fonction de garde. Il correspond à chaque transition t ∈ T dans l'expression de garde g. La sortie de l'expression de garde doit évaluer la valeur booléenne vrai ou faux.

- I est une fonction d'initialisation. Il mappe chaque emplacement p dans une expression d'initialisation i. L'expression d'initialisation doit être évaluée à un ensemble de jetons avec une couleur correspondante à la couleur de l'endroit C(p)

2.2 Web sémantique

2.2.1 Logique de description

Les logiques de description (DL) sont une famille de langues formelles de représentation des connaissances. La logique de description consiste en les concepts, les rôles, les individus et leurs relations dans un domaine. Le concept de modélisation fondamentale d'une DL est l'axiome - une déclaration logique reliant les rôles et/ou les concepts.

Il existe de nombreuses variétés de logique de description et il existe une convention de dénomination informelle. L'expressivité est codée dans l'étiquette pour une logique commençant par l'une des logiques de base suivantes:

- $\mathcal{AL}$, c'est la langue de base qui permet:
  Négation atomique (négation des noms de concepts qui n'apparaissent pas sur le côté gauche des axiomes)
  Concept intersection
  Restrictions universelles
  Quantification existentielle limitée

- $\mathcal{FL}$ permet :
  Concept intersection
  Restrictions universelles
  Quantification existentielle limitée
  Restriction de rôle

- $\mathcal{EL}$ langage existentiel, permet:

Concept intersection

Restrictions existentielles (de quantification complète existentielle)

Suivi par l'une des extensions suivantes :

- $\mathcal{F}$ propriétés fonctionnelles, un cas particulier de quantification unique.
- $\mathcal{F}$ propriétés fonctionnelles, un cas particulier de quantification unique.
- $\mathcal{E}$ qualification complète existentielle (restrictions existentielles qui ont des charges autres que $\top$).
- $\mathcal{U}$ union de concept
- $\mathcal{C}$ négation de concept complexe.
- $\mathcal{H}$ hiérarchie des rôles (subproperties - rdfs:subPropertyOf)
- $\mathcal{R}$ axiomes complexes d'inclusion de rôle complexes; La réflexivité et l'irréflexivité ; disjonction de rôle.
- $\mathcal{O}$ Les nominales. (classes énumérées de restrictions de valeur d'objet - owl:oneOf, owl:hasValue).
- $\mathcal{I}$ propriétés inverse
- $\mathcal{N}$ restrictions de cardinalité (owl:cardinality, owl:maxCardinality)
- $\mathcal{Q}$ restrictions de cardinalité qualifiées (disponibles dans OWL 2, restrictions de cardinalité qui ont des charges autres que $\top$).
- ($\mathcal{D}$) utilisation de propriétés de type de données, valeurs de données ou types de données

Certains DL qui ne correspondent pas exactement à cette convention sont:

- $\mathcal{S}$ une abréviation pour $\mathcal{ALC}$ avec des rôles transitifs.
- $\mathcal{FL}^-$ une sous-langue de $\mathcal{FL}$, qui est obtenue en refusant la restriction de rôle. Cela équivaut à $\mathcal{AL}$ sans négation atomique.
- $\mathcal{FL}_o$ une sous-langue de $\mathcal{FL}^-$, qui est obtenue en refusant une quantification existentielle limitée.

- $\mathcal{EL}^{++}$ Alias pour $\mathcal{ELRO}$.

Exemple 2.2: OWL 2 fournit l'expressivité de $\mathcal{SROIQ}^{(\mathcal{D})}$, OWL-DL est basé sur $\mathcal{SHOIN}^{(\mathcal{D})}$ et pour OWL-Lite est $\mathcal{SHIF}^{(\mathcal{D})}$.

2.2.2 OWL

Abstract Syntax	DL Syntax	Semantics
Descriptions (C)		
A (URI Reference)	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
owl:Thing	$\top$	$\texttt{owl:Thing}^{\mathcal{I}} = \Delta^{\mathcal{I}}$
owl:Nothing	$\bot$	$\texttt{owl:Nothing}^{\mathcal{I}} = \emptyset$
intersectionOf(C_1 C_2 ...)	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
unionOf(C_1 C_2 ...)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
complementOf(C)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
oneOf(o_1 ...)	$\{o_1, \ldots\}$	$\{o_1^{\mathcal{I}}, \ldots\}$
restriction(R someValuesFrom(C))	$\exists R.C$	$\{x \mid \exists y \ (x,y) \in R^{\mathcal{I}} \cup y \in C^{\mathcal{I}}\}$
restriction(R allValuesFrom(C))	$\forall R.C$	$\{x \mid \forall y \ (x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
restriction(R hasValue(o))	$R : o$	$\{x \mid (x, o^{\mathcal{I}}) \in R^{\mathcal{I}}\}$
restriction(R minCardinality(n))	$\geqslant nR$	$\{a \in \Delta^{\mathcal{I}} \mid \ \mid \{b \mid (a,b) \in R^{\mathcal{I}}\} \mid \geq n\}$
restriction(R maxCardinality(n))	$\leqslant nR$	$\{a \in \Delta^{\mathcal{I}} \mid \ \mid \{b \mid (a,b) \in R^{\mathcal{I}}\} \mid \leq n\}$
restriction(U someValuesFrom(D))	$\exists U.D$	$\{x \mid \exists y \ (x,y) \in U^{\mathcal{I}} \cup y \in D^{\mathcal{D}}\}$
restriction(U allValuesFrom(D))	$\forall U.D$	$\{x \mid \forall y \ (x,y) \in U^{\mathcal{I}} \rightarrow y \in D^{\mathcal{D}}\}$
restriction(U hasValue(v))	$U : v$	$\{x \mid (x, v^{\mathcal{I}}) \in U^{\mathcal{I}}\}$
restriction(U minCardinality(n))	$\geqslant nU$	$\{a \in \Delta^{\mathcal{I}} \mid \ \mid \{b \mid (a,b) \in U^{\mathcal{I}}\} \mid \geq n\}$
restriction(U maxCardinality(n))	$\leqslant nU$	$\{a \in \Delta^{\mathcal{I}} \mid \ \mid \{b \mid (a,b) \in U^{\mathcal{I}}\} \mid \leq n\}$
Data Ranges (D)		
D (URI reference)	D	$D^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^{\mathcal{I}}$
oneOf(v_1 ...,)	$\{v_1 \ldots,\}$	$\{v_1^{\mathcal{I}} \ldots,\}$
Object Properties (R)		
R (URI reference)	R	$\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
	R^-	$(R^{\mathcal{I}})^-$
Datatype Properties (U)		
U (URI reference)	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$
Individuals (o)		
o (URI reference)	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
Data Values (v)		
v (RDF literal)	v	$v^{\mathcal{D}}$

Figure 2.2: OWL et logique de description. Source:[DL]

La figure 2.2 présente l'équivalent entre la syntaxe OWL et la syntaxe de logique de description (DL), elle donne aussi leur sémantiques aussi. Le langage OWL [OWL] est un langage proposé basé sur la logique de description. C'est un des

langages populaire du web sémantique. Il est utilisé pour la représentation de concepts et de propriétés, en tant qu'exemples de concepts et de propriétés (atomiques) de base. L'ensemble de ces concepts pour un domaine est la boîte terminologique (TBox) d'une ontologie. Les assertions impliquant des concepts et des propriétés des individus forment la boîte d'assertion (ABox) de l'ontologie. Le raisonnement est appliqué sur les deux, les définitions TBox et les assertions ABox.

2.2.3 SWRL

Semantic Web Rule Language (SWRL) [SWR] est un langage pour le web sémantique qui peut être utilisé pour représenter des règles ainsi que la logique, SWRL est un sous-ensemble du langage de *Rule Markup Language*. La règle est représentée sous la forme suivante:

$$antécédent \Rightarrow conséquent$$

Les antécédents et conséquences sont des conjonctions d'atomes écrit $a_1 \wedge ... \wedge a_n$. Les variables sont indiquées en utilisant la convention standard de préfixation avec un point d'interrogation (par exemple, ?x).

Exemple 2.3 : en utilisant cette syntaxe, une règle affirmant que la composition des propriétés *parent* et *frère* implique la propriété de l'*oncle* serait écrite:

$$parent(?x,?y) \wedge frère(?y,?z) \Rightarrow oncle(?x,?z)$$

Dans cette syntaxe, les relations intégrées fonctionnelles peuvent être écrites en notation fonctionnelle, c'est-à-dire, op: numeric-add (?X, 3,?z) peut être écrit plutôt que: x = op: numeric-add (3,?z)

2.2.4 Raisonneur

Dans cette section, nous fournissons un bref introduction des raisonneurs. Raisonneur est un outil qui peut exécuter des tâches de raisonnement, généralement basées sur RDFS, OWL ou un moteur de règles. Il peut être utilisé pour vérifier la cohérence d'une base de connaissance ou pour déduire la nouvelle connaissance depuis une base de connaissance. En gros, les raisonneurs peuvent être groupés selon les techniques de raisonnement en trois groupes suivantes:

- Dans la première classe de raisonneur DL traditionnels (par exemple, RacerPro [Gmb], Pellet [CL]), des algorithmes basés sur des tableaux sont utilisés pour implémenter le calcul d'inférence.

- La deuxième alternative repose sur la réutilisation des techniques des bases de données déductives, en fonction d'une transformation d'une ontologie OWL en une programme de données de disjonction et à l'utilisation d'un moteur de données disjonctif afin de le raisonnement implanté dans KAON2 [oM].

- La dernière classe de raisonneur - y compris Sesame [Adu] et OWLIM [Ont] - utilisez le moteur de règle standard pour raisonner avec OWL. Les conséquences sont matérialisées souvent lorsque l'ontologie est chargée. Cependant, ceci la procédure est en principe limitée à des fragments de langage moins expressifs.

2.2.4.1 Sesame

Sesame [Adu] est un dépôt open source pour stocker et interroger des informations RDF et RDFS. Les ontologies OWL sont simplement traitées au niveau des graphes RDF. Sesame permet la connexion au DBMS (actuellement MySQL, PostgreSQL et Oracle) via le module SAIL (Storage and Inference Layer). Sesame fournit l'inférence de RDFS et permet d'interroger via SeRQL, RQL, RDQL et SPARQL. Via la SAIL, il est également possible d'étendre les capacités d'inférence du système.

2.2.4.2 OWLIM

OWLIM [Ont] est un dépôt sémantique et un raisonneur, emballé comme SAIL pour la base de données RDF. OWLIM utilise le moteur TRREE pour effectuer le RDFS et le raisonnement OWL DLP. OWLIM offre un support et une performance de raisonnement configurables. Dans la version "standard" du raisonnement OWLIM (appelé SwiftOWLIM) et l'évaluation de la requête est effectuée en mémoire, tandis qu'une stratégie de persistance assure la préservation, la cohérence et l'intégrité des données.

2.2.4.3 KAON2

KAON2 [oM] est un raisonneur Java gratuit pour l'extension SHIQ avec le fragment DL-safe de SWRL. Contrairement à la plupart des facteurs de DL actuellement disponibles, le calcul du tableau n'est pas implémenté. Plutôt, le raisonnement dans KAON2 est mis en œuvre par de nouveaux algorithmes qui réduisent une connaissance SHIQ(D). Ces nouveaux algorithmes permettent d'appliquer des techniques de base de données déductives bien connues.

2.2.4.4 HermiT

HermiT [Gro] est un raisonneur gratuit pour les logiques de description. Le raisonneur gère actuellement le DL SHIQ. Le support de SHOIQ est en cours de traitement. L'inférence principale soutenue est le calcul de la hiérarchie de la subsomption. HermiT peut également calculer l'ordre partiel des classes qui se produisent dans une ontologie. HermiT met en œuvre un nouvel algorithme de raisonnement "hypertableau". L'aspect principal de cet algorithme est qu'il est beaucoup moins non déterministe que les algorithmes tableau existants.

2.2.4.5 RacerPro

Le système RacerPro [Gmb] est un raisonneur tableau optimisé pour SHIQ(D). Il peut gérer plusieurs TBoxes et plusieurs ABox et traite les individus sous l'unique l'hypothèse du nom. En plus des tâches de raisonnement de base, telles que la satisfaction et la subsomption, il offre des requêtes ABox basées sur les optimisations nRQL. Il est implémenté dans le langage de programmation *Common Lisp*.

2.2.4.6 Pellet

Pellet [CL] est un raisonneur pour SROIQ avec des types de données simples. Il met en œuvre une procédure de décision basée sur tableau pour les TBoxes générales (subsomption, satisfaction et classification) et ABoxes (récupération, réponse de requête conjoncturelle). Pellet emploie plusieurs des optimisations pour le raisonnement standard de DL comme d'autres raisonneur DL. Il est directement prend en charge les contrôles d'implication et des requêtes ABox optimisées via son interface.

2.3 Génie logiciel

L'ingénierie logicielle est une approche du développement de produits logiciels utilisant les meilleurs principes scientifiques, la méthodologie et les procédures. La sortie de l'ingénierie logicielle est un produit logiciel. Le cycle de vie de développement de logiciel est un ensemble d'étapes dans l'ingénierie logicielle pour construire le logiciel attendu.

2.3.1 Activités SDLC

Software Development Life Cycle (SDLC) fournit un ensemble d'étapes à suivre pour concevoir et développer un produit logiciel. La figure 2.3 présente un cadre SDLC qui comprend les étapes suivantes :

2.3.1.1 Communication

C'est la première étape où l'utilisateur effectue la demande d'un produit logiciel attendu. Il communique avec le client et essaie de négocier le contrat et le terme. Il envoie sa demande au client par écrit.

2.3.1.2 Rassemblement d'exigences

Cette étape est l'une des étapes les plus importantes du développement de logiciels. L'équipe doit conserver les discussions avec le client participant de manière différente à son problème et doit essayer de noter autant d'informations que possible sur ses besoins. Les exigences sont clarifiées et écrites dans le document sur les exigences des utilisateurs, les exigences du système et les exigences fonctionnelles.

2.3.1.3 Etude de faisabilité

Après la collecte des exigences, l'équipe fera un plan temporaire d'un processus logiciel. À cette étape, l'équipe analyse s'ils peuvent créer un logiciel pour répondre à toutes les exigences de l'utilisateur ou s'il existe des problèmes qui rendent le logiciel plus utile. Il existe de nombreuses méthodes existantes, qui aident les développeurs à examiner la faisabilité d'un produit logiciel.

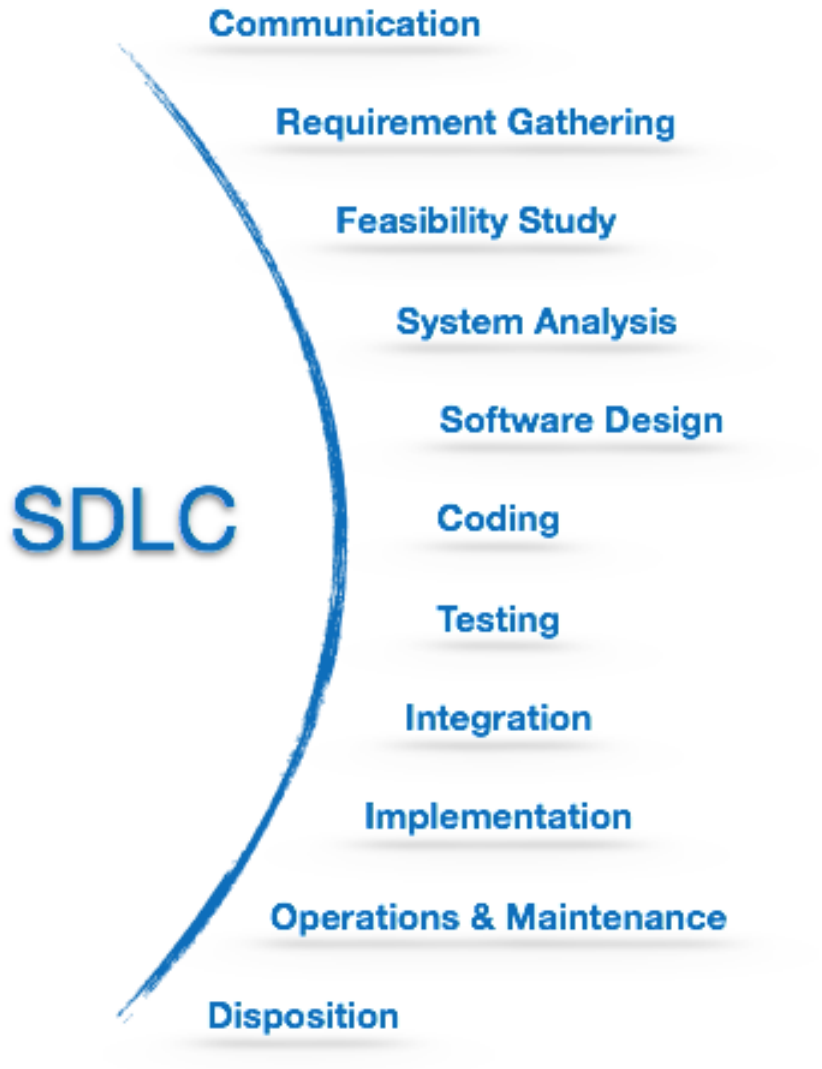

Figure 2.3: SDLC activités

2.3.1.4 Analyse des systèmes

Cette phase de SDLC est l'étape principale du plan du développeur qui va tenter de créer le modèle de logiciel. Après avoir analysé un système, l'équipe est capable de comprendre le point fort et la faiblesse du produit logiciel, identifier et détecter les problèmes importants d'un projet. L'équipe doit clarifier la zone appliquée du projet et planifier le plan et les ressources en conséquence.

2.3.1.5 Conception de logiciel

Cette phase consiste à concevoir des produits logiciels à partir des exigences et des analyses de l'utilisateur. L'exigence de l'utilisateur et les informations recueillies dans la phase de collecte des besoins sont utilisées dans cette étape qui fournit deux sorties possibles : la conception logique et la conception physique. Les ingénieurs établissent la base de données, les diagrammes logiciels, les diagrammes de flux de données et certains codes possibles.

2.3.1.6 Programmation

Cette étape est également connue comme phase de mise en œuvre. La mise en œuvre de la conception du logiciel est l'écriture en code du programme dans le langage de programmation spécifique et le développement efficace d'un programme.

2.3.1.7 Test

Avant de déployer un produit logiciel, tous les processus de développement logiciel doivent être testés. Les erreurs peuvent bloquer le logiciel ou fournir une mauvaise sortie. La détection précoce des erreurs et leur remède est la clé pour créer un bon logiciel.

2.3.1.8 Intégration

Le logiciel peut avoir besoin d'être intégré au programme externe ou à la source de données. Cette étape de SDLC est l'intégration de logiciels avec un objet externe.

2.3.1.9 Déploiement

Cette étape consiste à déployer le logiciel sur un système réel. Normalement, le logiciel doit être configuré à l'environnement de l'utilisateur. Le logiciel est vérifié pour l'adaptabilité et les problèmes liés à l'intégration sont résolus pendant le déploiement.

2.3.1.10 Fonctionnement et maintenance

Cette phase consiste à confirmer l'exécution du logiciel en termes d'efficacité et sans erreurs. Les utilisateurs peuvent être formés, ou peuvent s'entraider avec le guide

en utilisant le logiciel et comment maintenir l'exécution du logiciel. Le logiciel est maintenu en modifiant le code en fonction des changements qui se produisent dans l'environnement ou de la technologie de l'utilisateur. Dans cette phase, il existe des risques liés aux erreurs cachées et aux problèmes du monde réel.

2.3.2 Modèle à trois couches

Le modèle à trois couches pour le développement de logiciels est essentiellement une architecture décomposé à trois couches. A savoir :

- **Présentation :** la couche de présentation fournit l'interface utilisateur (UI) de l'application. En règle générale, il s'agit d'un formulaire d'application et d'une interface, ainsi que du document html créé à partir du serveur web vers le client.
- **Couche de Métier :** la couche de métier applique la logique de l'application en utilisant le processus de métier (*business process*). Et la règle métier de la demande.
- **Couche de données** : la couche de données permet d'accéder à des systèmes externes tels que des bases de données.

2.4 SBVR

Semantics of Business Vocabulary and Business Rules (SBVR) est un méta-modèle pour les développements de modèles sémantiques de vocabulaires de métier et de règles de métier. SBVR définit le vocabulaire et les règles pour documenter la sémantique des vocabulaires de métier, des faits de métiers et des règles de métiers.

L'approche orientée vers le fait de SBVR provient du *Business Rules Manifesto*, affirmant que les règles s'appuient sur les faits et que ceux-ci s'appuient sur les concepts exprimés par les termes. Par conséquent, les termes expriment les concepts de métier, les faits font des assertions sur ces concepts et les règles limitent et soutiennent ces faits. Le SBVR soutient cette approche en fournissant des concepts de noms et des concepts de verbes correspondants respectivement aux notions de termes et de faits. La figure 2.5 montre l'organisation structurelle de ces composants.

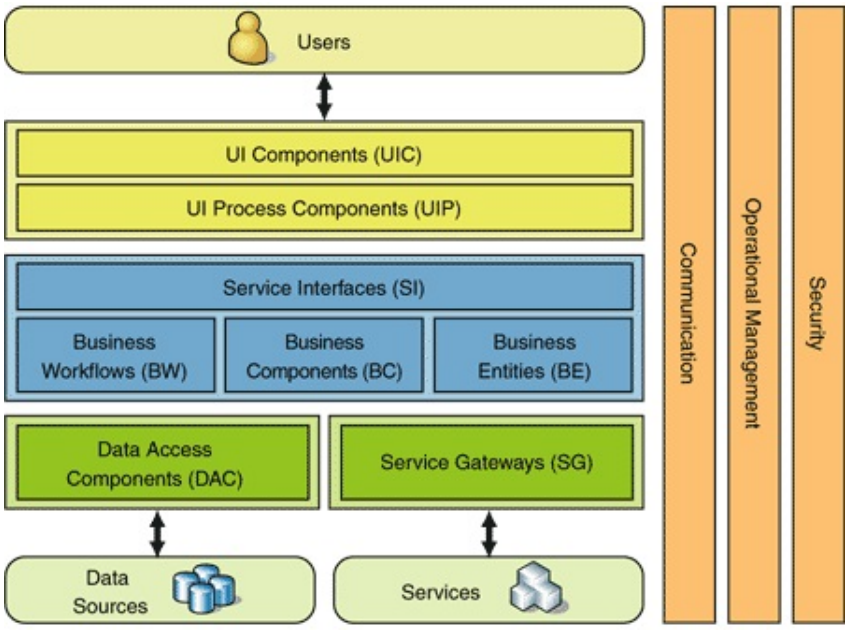

Figure 2.4: Modèle à trois couches

Un concept nominal est un concept signifiant un nom ou une expression nominale, spécialisée par :

1. Types d'objets, qui sont des concepts de noms classifiant les choses en fonction de leurs propriétés communes ;

2. Concepts individuels, qui sont un concept correspondant à un seule type d'objet ;

3. Rôles, qui sont des concepts de noms correspondant à des éléments basés sur leur rôle de modèle, en supposant une fonction ou en étant utilisés dans certaines situations.

   En outre, les rôles de type de fait sont définis comme les rôles qui caractérisent spécifiquement ces instances par leur implication dans une instance d'un type de fait donné. Un concept de verbe aussi appelé type de fait est un concept signifiant une phrase verbale et qui implique un ou plusieurs concepts de noms, représentant des relations un-aires, binaires ou n-aires.

Enfin, une règle SBVR est un élément d'orientation qui introduit une obligation ou une nécessité et distingue deux types généraux :

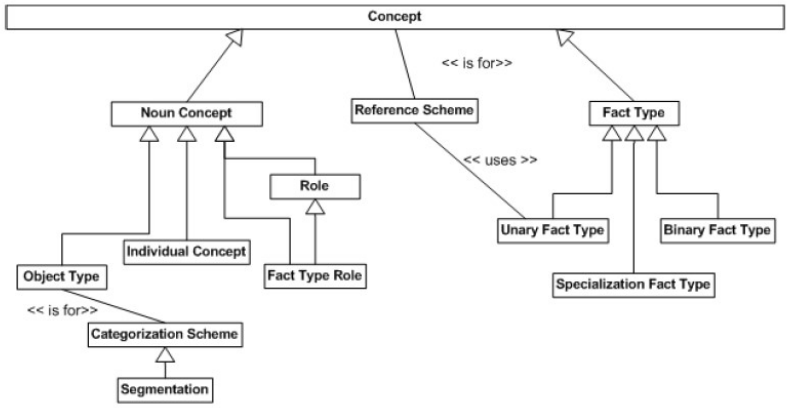

Figure 2.5: Semantic of Business Vocabulary and Business Rules

- **Règes de structures :** décrivant la façon dont l'entreprise choisit d'organiser les choses qu'elle traite;
- **Règles opératoires :** régissant la conduite de l'activité en décrivant les processus de métier. Les deux types de règles sont construits en imposant des restrictions sur les types de faits et en utilisant des quantificateurs, des opérateurs logiques, etc. La figure 2.4 montre l'organisation structurelle des quantifications et des opérations logiques.

Comme indiqué précédemment, SBVR adopte une approche linguistique qui permet de définir les vocabulaires et d'exprimer les règles opérationnelles. Selon ce point de vue, SBVR définit un *Control Natural Language* (CNL) - nommé *SBVR Structured English* - et décrit la façon de mapper mécaniquement ces expressions CNL aux concepts formels du SBVR.

2.5 Ontologie pour l'ingénierie logicielle

Pour aider le lecteur à comprendre ce chapitre plus facilement, nous présenterons quelques définitions acceptées dans ce domaine de collaboration d'ingénierie logicielle et de web sémantique avant d'expliquer plus en détail dans chaque sous-section.

- ***Upper Ontology* :** *upper ontology* contient des concepts très généraux qui sont utilisés dans tous les domaines. Ces domaines partagent des concepts

communs d'une ontologie. Cette ontologie fournit l'interopérabilité sémantique à toutes les autres ontologies qui se situent en dessous [UPP],[AVR15].

- ***Software Process Ontology*** **:** *software process ontology* contient les différents modèles de cycle de vie du logiciel et leurs étapes qui sont utilisées pour développer le logiciel. Chaque étape est structurée par la séquence des tâches. Par exemple: *Waterfall Model, Prototype Model, Incremental Process Model, Rapid Application Development (RAD) Model, Iterative Enhancement Model, V Model, Spiral Model, Agile Model etc.*, [HMS03],[LL14], [TR06].

- ***Domain Ontology*** **:** *Application domain ontology* fournit la connaissance d'un domaine typique (par exemple, l'ontologie du domaine médical représente la connaissance du domaine médical) et les informations requises pour développer des applications logicielles dans un domaine. Cette ontologie représente les différentes relations qui relient différentes définitions liées au domaine. Tout en représentant la relation entre différents concepts, ils sont les représentants de la vie réelle [TR06], [AdSdLdS04b],[PK15a].

- ***System Behavior Ontology*** **:** *System Behavior Ontology* représente le comportement de l'application, c'est-à-dire ce que demande une application dans un scénario typique. Il contient également les conditions qui doivent être satisfait avant qu'une tâche ne soit exécutée et indique l'état du système une fois la tâche effectuée. [CK07a],[CK07a].

- ***Architecture Ontology*** **:** *Architecture Ontology* contient la définition de l'architecture telle que les styles d'architecture (structure de l'application), le module principal de l'application, les propriétés du module et leur relation [PZ14]

- ***Pattern Ontology*** **:** cette ontologie modélise un ensemble de modèles utilisés pour créer une application. Cette ontologie pourrait contenir le motif de conception, le modèle de processus de métier, le modèle d'application web, le modèle de cas d'utilisation, etc. [HA05],[HAHA],[GPSV14].

- ***Software Artifact Ontology*** **:** *Software Artifact Ontology* fournit la définition de plusieurs artefacts générés lors du développement de l'application. Ces définitions nous permettent de les séparer en fonction de leur type ou de leurs

formats tels que le code source, le fichier image, les manuels de documentation, etc. [AdSdLdS04a], [AGS],[HKST06].

- ***Object Oriented Source Code Ontology*** **:** *OO source code ontology* représente les concepts liés à la programmation orientée objet (OOP). Les concepts principaux de l'OOP tels que l'abstraction, l'héritage, le polymorphisme et l'encapsulation seront représentées par cette ontologie. Il contiendra également des concepts tels que la classe, l'interface, la constante ou les données, la fonction, l'objet, etc.[KB07],[WZR07],[ZWRH06].

- ***Document Ontology*** **:** l'ontologie de la documentation fournit les documents liés à l'application. Il fournit de nombreux documents générés lors de plusieurs étapes telles que la description formelle, le diagramme d'état et le diagramme de flux de contrôle généré lors de l'analyse des besoins et de l'étape de spécification; le diagramme de flux et le diagramme de relation d'entité généré pendant la phase de conception. [KB07],[WZR07],[ZWRH06].

- ***Testing Ontology*** **:** l'ontologie de test fournit les terminologies qui peuvent être utilisées par le testeur, les tests d'environnement effectués, quelles sont les techniques de test disponibles, sur le type de test effectué, etc. Cette ontologie peut également contenir une méthodologie de test, par exemple, de nombreux tests techniques pouvant être utilisés tels que les tests de boîtes noires (tests fonctionnels) et les tests de boîtes blanches (tests structurels); De nombreux niveaux pour tester le système, comme les tests unitaires, les tests d'intégration et les tests système.

Chapitre 3

L'état de l'art

Contenu

Dans ce chapitre, nous présentons les travaux précédents sur l'application de l'ontologie en ingénierie logicielle. Bien que notre travail se concentre uniquement sur la phase de conception du cycle de développement du logiciel, nous présentons encore le travail en utilisant l'ontologie dans une autre phase du cycle de développement du logiciel pour aider le lecteur à avoir une vue général de la direction de recherche de la collaboration de l'ingénierie logicielle et du web sémantique. Il existe de nombreuses ontologies existantes qui ont été proposées pour s'appliquer à toutes les phases du cycle de vie du développement logiciel, elles ont été répertoriées dans le document d'enquête récent [BKB16]. Nous allons les examiner. Dans chaque sous-section suivante, nous présenterons les ontologies qui sont appliquées à chaque phase du cycle de vie du développement logiciel.

3.1 Réutilisation du logiciel

La réutilisation du logiciel est un attribut dans lequel le logiciel ou son module est réutilisé avec très peu ou pas de modification. Pour toute organisation, l'amélioration des performances de l'entreprise implique l'exécution de leur développement logiciel. La réutilisation du logiciel offre un plus grand potentiel de gains significatifs pour une organisation, en réduisant les coûts et l'effort, et en accélérant le temps de commercialisation des produits logiciels. Depuis plusieurs années, plusieurs travaux de recherche, y compris les rapports d'entreprises [End93], [Bau93], [Gri94], [GW95], [Joo94], la recherche informelle [FI94] et les études empiriques [Rin97], [MET02], [RDKN03] ont montré qu'un moyen efficace d'obtenir les avantages de réutilisation du logiciel est d'adopter un processus de réutilisation. Cependant, les processus de réutilisation existants présentent des problèmes cruciaux tels que des lacunes dans des activités importantes, comme le développement et la réutilisation, et mettent davantage l'accent sur certaines activités spécifiques (analyse, conception et mise en œuvre). Même aujourd'hui, avec les idées de lignes de produits logiciels, il n'y a toujours pas de consensus clair sur les activités (intrants, produits, artefacts) et les exigences qu'un processus de réutilisation efficace doit avoir.

3.1.1 Processus d'ingénierie de domaine

L'ingénierie de domaine est l'activité de collecte, d'organisation et de stockage d'expérience acquise lors de la construction de systèmes ou parties de systèmes dans un domaine particulier sous la forme d'actifs réutilisables, ainsi que l'activité fournissant des moyens adéquats pour réutiliser ces actifs lors de la construction de nouveaux systèmes [CE00] .

Parmi les travaux des années 80 et 90, tels que [Nei80], [fAS93], [SCK+96], [JGJ97], [GFd98], [KKL+98]], un accent particulier est mis sur les processus d'ingénierie de domaine pour développer des logiciels réutilisables.

Un exemple de ce travail peut être trouvé dans [Nei80]. Dans ce travail, *Neighbors* a proposé la première approche d'ingénierie de domaine, ainsi qu'un prototype - Draco - basé sur la technologie de transformation. Les idées principales introduites par Draco comprennent : l'analyse de domaine, les langues spécifiques au

domaine et les composants en tant que séries de transformations. Draco soutient l'organisation de la connaissance de la construction de logiciels dans un certain nombre de domaines connexes. Chaque domaine Draco intègre les besoins, les exigences et les différentes implémentations d'une collection de systèmes similaires.

Dans ce contexte, en 1992 [fAS93], *Software Technology for Adaptable, Reliable Systems (STARS)* a développé le *Conceptual Framework for Reuse Processes (CFRP)* en tant que véhicule pour comprendre et appliquer le paradigme de l'ingénierie logicielle réutilisable spécifique au domaine STARS. Le CFRP a mis en place un cadre pour examiner les processus d'ingénierie logicielle liés à la réutilisation, la manière dont ils s'interconnectent et comment ils peuvent être intégrés entre eux et avec des processus non liés à la réutilisation afin de former des modèles de processus de cycle de vie axés sur la réutilisation qui sont adaptés à l'organisation besoins.

Quatre ans après le début du projet STARS, Mark Simos [SCK+96] et son groupe ont développé la méthode *Organisation Domain Modeling (ODM)*. Leur motivation était d'avoir une lacune dans les méthodes d'ingénierie du domaine et les processus disponibles pour réutiliser les connaissance des experts.

Trois experts en développement de logiciels - Jacobson, Griss et Jonsson - ont créé *Reuse-driven Software Engineering Business(RSEB)* [JGJ97]. RSEB est un processus de réutilisation systématique basé sur l'utilisation et sur la notation UML. La méthode a été conçue pour faciliter à la fois le développement de logiciels réutilisables orientés objet et la réutilisation de logiciels. À l'instant du *Unified Process* [JBR99], le RSEB est également itératif et axé sur l'utilisation.

3.1.2 Lignes de produits d'application

Jusqu'en 1998, les processus de réutilisation du logiciel ne concernaient que les problèmes d'ingénierie de domaine. Cependant, durant cette même période, une nouvelle tendance a commencé à être explorée : la zone de lignes de produits d'application [20001]. Les lignes de produits d'applications ont commencé à être considérées comme l'une des avancées les plus prometteuses pour un développement logiciel efficace. Une ligne de produits d'application est définie comme un ensemble de systèmes à forte intensité de logiciel partageant un ensemble de fonctionnalités communes gérées qui répondent aux besoins spécifiques d'un segment ou d'une

mission de marché particulier et qui sont élaborés à partir d'un ensemble commun d'actifs de base de manière prescrite [20001].

La ligne de produits d'application s'est révélée compatible avec la réutilisation systématique dans l'ensemble des produits similaires que les sociétés de logiciels offrent. Les avantages principaux de l'adoption des lignes de produits d'application ont été discutés par plusieurs auteurs [20001].

Bayer *et al* [BFK+99] a proposé la méthodologie de *Product Line Software Engineering* (PuLSE). Celle-ci a été développée avec la proposition permettant la conception et le déploiement de lignes de produits d'application dans une grande variété de contextes d'entreprises. Une caractéristique importante de PuLSE résulte d'un effort ascendant : la méthodologie capte et exploite les résultats (les leçons apprises) des activités de transfert de technologie avec des clients industriels.

Selon Atkinson *et al.* [ABM00], PuLSE a été appliqué avec succès dans différents contextes à différentes fins. Parmi les avantages, il s'est avéré utile d'introduire une documentation solide et des techniques de développement dans les pratiques de développement existantes.

3.1.3 Ontologie pour la réutilisation du logiciel

Bien que le sujet de la réutilisation ait été largement discutée dans le domaine de l'ingénierie logicielle depuis plusieurs années, parmi plusieurs chercheurs et praticiens, aucun n'est encore satisfait avec l'état actuel de la pratique. [FK05].

Selon *Uschold* [Usc98], "une ontologie peut prendre une variété de formes, mais nécessairement cela inclura un vocabulaire des termes et certaines spécifications de leur sens. Cela comprend des définitions et une indication de manières pour expliquer les concepts qui imposent collectivement une structure sur le domaine et contraignent les interprétations de termes possibles qui sont interconnectés". Ainsi, une ontologie se compose des concepts et relations et leurs définitions, propriétés et les contraintes exprimées en axiomes [FdMdR98].

[UJ99] a classé les applications des ontologies dans quatre catégories principales, soulignant qu'une application peut intégrer plus d'une de ces catégories :

- **Autorisation neutre :** une ontologie est développée en une seule langue, est traduite en différents formats et est utilisée dans plusieurs applications cibles.

- **Ontologie comme spécification** : une ontologie d'un domaine donné est créée et fournit un vocabulaire pour spécifier les exigences pour une ou plusieurs applications cibles. En fait, l'ontologie est utilisée comme la base de la spécification et du développement de certains logiciels, ce qui permet une réutilisation des connaissances.

- **Accès commun à l'information :** une ontologie est utilisée pour permettre à plusieurs applications cibles (ou humaines) d'avoir accès à des sources d'informations hétérogènes exprimées à l'aide d'un vocabulaire diversifié ou d'un format inaccessible.

- **Recherche basée sur l'ontologie :** une ontologie est utilisée pour rechercher un référentiel d'informations sur les ressources souhaitées, améliorer la précision et réduire le temps total consacré à la recherche.

3.2 Ontologie en ingénierie logicielle

3.2.1 Phase d'analyse des besoins

Il s'agit de la première phase du cycle de développement du logiciel, cette phase consiste à collecter l'exigence du client, puis celle-ci sera analysée pour extraire l'information principale afin de définir et prendre une décision pour la prochaine étape du développement de logiciels. Il existe des travaux qui ont porté sur l'application de l'ontologie sur l'analyse des besoins logiciels. Il existe deux types d'exigences : fonctionnelles et non fonctionnelles. L'exigence fonctionnelle est de spécifier l'ensemble des actions qu'un système exécute. Les exigences non fonctionnelles décrivent les aspects de qualité du système logiciel.

[CK07b],[MJ15] présentent deux ontologies de comportement système. Ce type d'ontologie représente le comportement du système en fonction d'un scénario typique, par exemple : quelle activité fera un logiciel dans un scénario spécifique ? Le cas d'utilisation dans le diagramme UML représente les scénarios d'un logiciel. Cela montre l'interaction entre l'utilisateur et le logiciel. Que fera l'utilisateur ? Et quelle est la réponse du logiciel dans un cas d'utilisation ? Dans ce cas, le logiciel est considéré comme une "boîte noire", [CK07b] propose une approche pour fournir un référentiel de cas d'utilisation à l'aide du comportement du logiciel et de l'ontologie

du domaine d'application.

[RK15] propose une approche pour réutiliser les exigences du logiciel en utilisant l'ontologie. Pour comparer l'exigence actuelle avec l'exigence existante pour développer de nouveaux logiciels, il propose un mécanisme pour cartographier l'exigence en langage naturel aux axiomes d'une ontologie. [WGGM14] propose une approche pour considérer le logiciel comme un artefact abstrait, distinct du code. Cette phase doit utiliser l'ontologie supérieure pour récupérer les terminologies générales afin de définir le concept et les règles. Il existe plusieurs ontologies supérieures existantes. Par exemple, *IDEAS, WordNet, Bunge-Wand-Weber (BWW) ontology, Upper Mapping* et *Binding Exchange Layer (UMBEL) ontology*, *Unified Foundational Ontology (UFO)*, *CIDOC object-oriented Conceptual Reference Model*, *Basic Formal Ontology (BFO), Business Objects Reference Ontology (BORO), Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE),(CRM) ontology, COmmon Semantic MOdel (COSMO) ontology, General Formal Ontology (GFO), PROTo ONtology (PROTON), Sowa's ontology*, et *Suggested Upper Merged Ontology* (SUMO) [PK15b]. Une comparaison des sept dernières ontologies a été examinée par [MCR06]. L'exigence doit être cohérente, sans ambiguïté, vérifiable. Il existe de nombreuses approches pour vérifier l'exigence, par exemple, axées sur les points de vue, les approches basées sur des scénarios, réseaux petri [LG05],[JB04]. L'ontologie du domaine pourrait être utilisée dans cette phase du cycle de vie du développement logiciel. L'ontologie du domaine est en mesure de proposer une bonne question lors de la collecte des besoins. Il peut être utilisé pour vérifier que les exigences suivent la norme du domaine ou non. [ZJ05b],[ZJ05a]. [BdPL03] soutient que les ontologies devraient être des sous-produits de la phase d'ingénierie des exigences. Ainsi, leur processus d'ingénierie des exigences a un sous-processus particulier pour la construction de l'ontologie. Ce processus s'inspire de l'approche d'ingénierie de l'ontologie en couches [MS01], où la source principale de création d'ontologies est le lexique étendu du langage. Le lexique est construit en suscitant les termes importants à partir des documents sources pertinents et en mappant les termes aux constructions appropriées (par exemple, les classes) du langage ontologique utilisé dans la demande de l'étude. En ce qui concerne les autres technologies de développement de l'ontologie, nous pouvons également découvrir que l'ingénierie des exigences et l'ingénierie de

l'ontologie partagent même certaines méthodologies communes. Par exemple, le cadre d'ingénierie de l'ontologie DOGMA [STM] utilise une approche basée sur les scénarios pour créer des ontologies pour les domaines d'application.

Si les ontologies sont encore utilisées dans la phase de conception (par exemple les modèles), l'ingénierie des exigences implique également l'utilisation de différentes méthodologies telles que les approches axées sur les objectifs, les orientations orientées vers les points de vue et les approches basées sur des scénarios, ou leurs combinaisons [DBL05]. La vérification des exigences est l'une des choses les plus importantes de cette phase. Il existe des travaux existants qui envisagent d'utiliser l'approche basée sur l'ontologie pour la vérification des exigences [MFCC06], [Bür07], et [HHK+07].

3.2.2 Phase de conception

[dGTLvV12] décrit une approche basée sur l'ontologie pour la construction de documents d'architecture. Il a utilisé une ontologie logicielle dans un wiki sémantique [dGTLvV12] optimisée pour la documentation d'architecture. Ils ont une évaluation sur cette approche et démontrent que cette dernière est meilleure que celle basée sur les fichiers. [AT06] propose une approche du développement de logiciels axée sur les décisions d'architecture et implique l'utilisation de l'ontologie. Dans cette approche, l'architecture est captée par une instance d'une ontologie. L'ontologie comporte quatre composantes majeures : les atouts de l'architecture, les décisions d'architecture, les préoccupations des parties prenantes et une feuille de route d'architecture. [PGH07] propose une approche ontologique pour la modélisation de style architectural basée sur la logique de description en tant qu'instrument de modélisation abstraite et méta-niveau. Les styles d'architecture sont souvent négligés dans les architectures logicielles. Nous introduisons un cadre pour la définition et la combinaison de style. [dG15] est une thèse sur la représentation du document d'architecture en utilisant l'ontologie, il existe une comparaison avec l'approche basée sur les fichiers. [dGLT+14] décrit également une approche afin de construire une ontologie pour la documentation d'architecture logicielle. [AOAN12] propose une approche visant la réutilisation des processus logiciels basée sur des architectures logicielles. Il étudie et se concentre sur les approches de réutilisation basées sur les architectures logicielles et l'ontologie de domaine. [FRR] propose

un cadre qui permet la recherche d'informations sur l'architecture du logiciel dans les artefacts générés dans les environnements des communautés virtuelles. Un mécanisme de recherche sémantique basé sur l'ontologie est utilisé dans le cadre pour récupérer non seulement les propriétés de l'architecture, mais aussi la raison d'être des décisions prises lors de la construction du logiciel. [TRPR07] décrit une méthode pour identifier les concepts majeurs dans l'architecture de logiciel qui peuvent être utilisés dans une telle méta-connaissance. Les termes du concept sont identifiés à l'aide de deux techniques différentes, grâce à l'index du *back-of-the-book* de certains des textes majeurs de l'architecture logicielle par une technique semi-automatique en analysant les pages *Wikipedia*. La connaissance de l'architecture générique est considérée dans cette approche seulement. [WLS+07] présente une approche de la modélisation et de la vérification des diagrammes de fonctionnalités à l'aide d'ontologies OWL. Ils utilisent des ontologies OWL DL pour capturer précisément les interrelations entre les fonctionnalités d'un diagramme de fonctionnalités. Les moteurs de raisonnement OWL tels que FaCT++ [dGTLvV12] sont déployés pour vérifier l'incohérence des configurations de fonctionnalités de manière entièrement automatique [ZDP09]. [ZKDT09] a fourni un cadre qui représente, intègre les modèles de fonctionnalités distribuées et les valide en utilisant OWL et SWRL.

3.2.3 Phase de développement

[Sak06] propose un modèle d'échange d'informations entre les utilisateurs potentiels du système, tels que les employeurs et les employés potentiels. L'objectif du travail est un prototype de service web public qui offrirait un moyen facile et compréhensible d'échange d'informations entre les utilisateurs potentiels du système, tels que les employeurs et les employés.

Cette sous-section montre des artefacts logiciels, des ontologies de code source orientées objet, qui peuvent être mise en œuvre. *Software Artifact Ontology* caractérise le concept des différents artefacts créés lors de la phase de développement du logiciel. Ces concepts nous permettent de les regrouper en fonction de leur type ou format, par exemple, fichier audio, document, etc.[DBL04], [AGS],[HKST06].

Cette ontologie peut également définir le concept, qui contient des données de métadonnées, par exemple, à quel stade un artefact spécifique est livré, par qui

(individuel) il est créé et dans lequel le projet est réalisé. Selon [ZDMP], une fois que nous avons toutes les données de méta-données sur l'artefact livré, cette ontologie ainsi que l'ontologie du domaine d'application peuvent être utilisées dans le cadre de la génération de la documentation.

3.2.4 Phase de test

Cette sous-section présente les ontologies de qualité, les ontologies de tests, qui peuvent être utilisées pour examiner ses caractéristiques de qualité améliorant le développement du logiciel. Quelques modèles de qualité et benchmarks ont été créés, par exemple, *Capability Maturity Model* (CMM), ISO 9001, et ISO/IEC 9126, qui sont utilisés pour améliorer la programmation [HSGPML14].

3.2.5 Phase de maintenance

L'ontologie de maintenance logicielle définit les concepts liés à la maintenance et leur relation [ADD06a]. Elle peut définir les concepts décrivant le type de maintenance tels que la maintenance parfaite, la maintenance adaptative, la maintenance corrective et tout autre type d'entretien ; quelles procédures peuvent être utilisées pour maintenir le logiciel ? c'est-à-dire les activités de maintenance ; qui effectue la maintenance ? Il comprend les concepts comme les activités telles que l'activité de gestion et l'activité de modification; une personne comme l'ingénieur de maintenance et le gestionnaire de maintenance; procédure telle que la méthode, la technique et le paradigme ; ressources telles que la maintenance des ressources humaines, les ressources humaines du client, etc. L'ontologie de maintenance peut également être utilisée pour modéliser divers modèles de maintenance tels que *Iterative Enhancement Model*, *Reuse Oriented Model*, *Boehm's Model*, *Taute Maintenance Model*, etc. [ADD06b]. L'ingénierie inverse est également effectuée à des fins de maintenance qui comprend des activités telles que la découverte d'informations inconnues et cachées d'un système logiciel. L'ontologie de modèle ainsi que l'architecture logicielle et les ontologies de conception orientées objet peuvent être utilisées pour faciliter la tâche de découverte des informations si inconnues et cachées, car les modèles tendent à révéler le style utilisé pour créer le système logiciel [DE05]. Les approches web sémantiques peuvent également être facilitées pour estimer les coûts de maintenance. Nous pouvons utiliser des modèles tels que 'Belady et Lehman Model',

Boehm Model, etc. pour estimer le coût de maintenance. [ADD06a] a montré qu'un outil de maintenance de logiciel web sémantique peut être créé. Cet outil utilisera l'ontologie de maintenance logicielle. [HWCK06] a également proposé une approche pour le maintien du système logiciel en utilisant des techniques web sémantiques. Il a construit une ontologie qui capture l'information sur les méta-données des composants logiciels tels que la documentation des exigences (à la fois fonctionnelle et non fonctionnelle), les métriques, les tests et les moyens par lesquels différents composants interagissent les uns avec les autres. Si une modification est apportée à n'importe quel composant logiciel, elle déclenchera toutes les informations de méta-données liées à ce composant logiciel modifié afin de comprendre et maintenir ce composant. À cette fin, ils ont encodé les informations de méta-données dans le graphe RDF et les requêtes SPARQL ont été utilisées pour faciliter la compréhension et la maintenance du composant logiciel. Les requêtes SPARQL ont été construites pour vérifier si une nouvelle validation du composant logiciel modifié est requise afin de savoir quel cas de test a échoué et à quelle exigence le cas de test qui a échoué appartient [Kha15].

3.2.6 Conclusion

Après avoir étudié les travaux précédents, nous nous rendons compte que le domaine de la collaboration entre ingénierie logicielle et web sémantique intéressait le milieu de la recherche. Ils ont augmenté la performance en appliquant les avantages du web sémantique au le domaine de l'ingénierie logicielle. Une ontologie a été appliquée dans toutes les phases du cycle de vie du développement logiciel. Ils se sont concentrés sur l'utilisation de l'ontologie pour partager les connaissances du logiciel et réutiliser le logiciel autant que possible dans une gamme de produits logiciels. Un autre aspect de l'ontologie qui a été utilisé pour améliorer les techniques d'ingénierie logicielle, c'est la représentation de connaissances. Ils ont essayé de représenter le document, la structure et toutes les phases dans le cycle de vie du développement logiciel sous la forme sémantique. De là, pour vérifier l'exigence ; ... Ils peuvent interroger ces connaissances pour détecter le point d'incohérence et assurer l'exactitude de l'exigence.

Partie I

Couche de métier fondée sur l'ontologie

Chapitre 4

Modélisation de couche de métier fondée sur l'ontologie

Contenu

4.1 Introduction

Dans ce chapitre, nous présentons notre approche pour construire une couche de métier d'une application basée sur l'ontologie. Le but principal de cette idée est de permettre à l'utilisateur de concevoir une couche de métier exécutable d'une application à partir de ses besoins. Un processus de métier doit utiliser les terminologies définies dans l'ontologie de la règle de métier (cette ontologie sera introduite dans le chapitre suivant). La couche de métier assure l'exactitude pendant le temps de

conception et elle pourrait également être intégrée à une source de données pour simuler l'exécution de l'application. Pour implémenter cette idée, la couche de métier d'une application est représentée dans une forme compréhensible par machine (dans ces travaux, nous proposons d'utiliser l'ontologie OWL). Selon la base de connaissances du processus de métier et le raisonnement sur ontologie, la structure principale d'une application est vérifiée au moment de conception et pendant l'exécution. L'avantage de cette approche est de réduire le temps de développement d'une application. Nous choisissons CPN d'utiliser comme modèle de la théorie des contrôles dans notre travail. L'ontologie proposée s'appelle "Ontologie des processus de métier (BPO)" et elle comporte deux couches principales comme suit :

- Couche méta : cette couche définit tous les concepts de CPN et la relation entre eux pour créer un graphe de CPN à l'intérieur de BPO. Cette couche contient deux parties principales, la première partie est un ensemble de concepts et de propriétés qui permettent à l'utilisateur de définir un processus de métier. La deuxième partie est un ensemble de règles de structure utilisé pour s'assurer qu'un processus de métier est créé sans erreur de structure.

- Couche individuelle : cette couche contient les instances de processus de métier. Chaque instance de processus de métier est un ensemble d'individus de concepts CPN dans la couche méta. Cette couche est contrôlée par la couche méta. S'il existe une instance de processus de métier qui ne suit pas le concept et la règle dans la couche méta, l'ontologie des processus de métier sera incompatible et l'instance de processus de métier ne sera pas validée.

Ce chapitre est structuré comme suit :

- Dans la première partie, nous présentons le processus de métier basé sur CPN.

- Dans la deuxième partie, nous présentons notre algorithme pour détecter l'impasse d'un processus de métier afin d'assurer l'exactitude des processus de métier.

- Dans la dernière partie, nous présentons l'ontologie du processus de métier utilisée pour construire le processus de métier d'une application.

4.2 Travaux connexes

4.2.1 Vérification des processus de métier

Il existe de nombreuses méthodes de vérification qui sont utilisées pour vérifier un processus de métier. Les réseaux de Pétri gére la théorie de contrôle du modèle utilisée pour disséquer les cadres distribués. Le réseaux de Pétri comporte à une notation graphique intuitive, avec une définition mathématique de sa sémantique d'exécution. Il existe de nombreuses sous-classes de réseaux de Pétri qui ont été caractérisées, parmi les plus importantes sous-classes de *Workflow nets* (WF) [vdA98] qui est utilisée comme formalisme dans la vérification d'un processus de métier. Une autre méthodologie est *Finite State Machine*(FSM) - un diagramme coordonné de nœuds et de bords, avec des nœuds représentant un état du système et des bords représentant un réglage dans un état. Ensuite, les modèles génériques, par exemple, SPIN [Hol04] et nuSMV2 [CCGR99], actualisent les algorithmes de recherche pour confirmer tout cadre démontré dans les dialectes d'information du vérificateur modèle. *Temporal logics* incorporate *Linear Temporal Logic (LTL)* et *Computation Tree Logic* (CTL).

[GOSC11] fournit d'abord un logiciel de vérification de processus de métier en tenant compte de la logique FCL, puis un procédé en améliorant ce logiciel. Il présente des poches caractéristiques d'adaptabilité à l'intérieur du processus déterminé de manière critique pour présenter la variabilité du temps de conception, en utilisant des impératifs qui donnent encore des données de demande et d'incorporation qui ne sont pas indiquées en utilisant une logique formelle. Les cadres formels construisent leurs détails décisifs sur la logique temporelle LTL et CTL. [DALV10] utilisent LTL pour déterminer le processus complètement déclaratif, [DGG+11] indiquent un processus déclaratif utilisant la programmation temporelle du paramètre de réponse (ASP) et le LTL dynamique, [PvdA06] utilisent LTL pour déterminer le processus d'exécution adaptable.

Enfin, différents systèmes obligent les clients à appliquer des logiques de spécification récemment proposées ou étirées afin de spécifier les propriétés correctes. L'exemple inclut [PFS10], qui étend les CTL avec la capacité de se séparer entre les événements et les fonctions, [GMS06], qui propose une logique déontique totalement nouvelle, et [BLA11], qui propose une autre logique de processus temporel

pour permettre des fusions de processus.

4.3 Processus de métier basé sur CPN

Comme mentionné ci-dessus, nous utilisons le CPN pour vérifier le modèle d'un processus de métier. Dans cette section, nous allons présenter quelques définitions et constructions de routage de base d'un processus de métier qui sont représentées par CPN.

4.3.1 Construction de routage de base du processus de métier

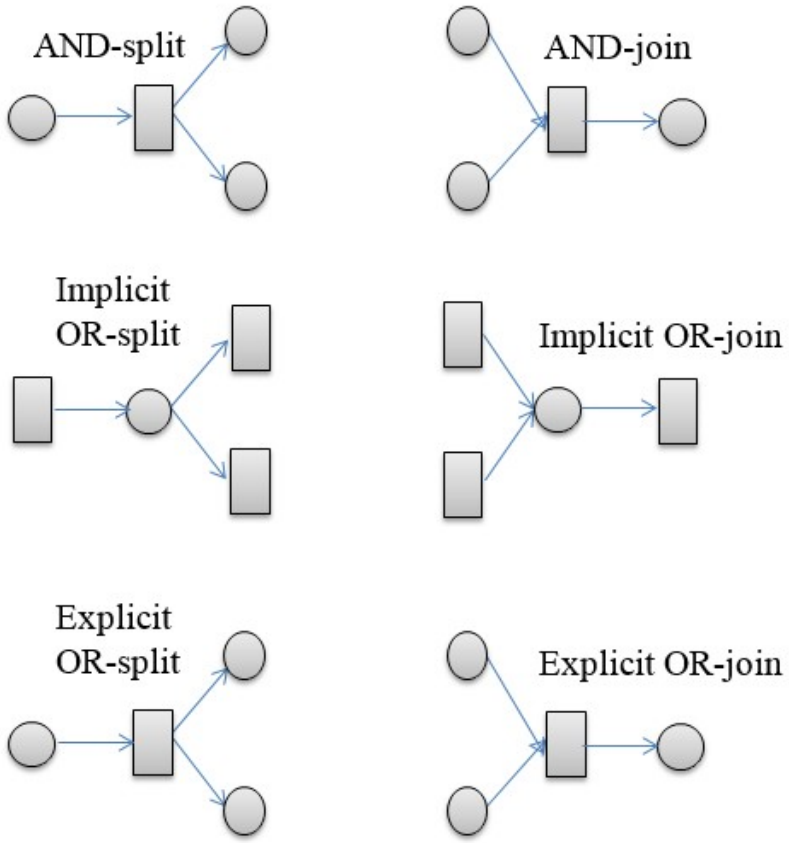

Figure 4.1: Bloc de base de la structure de routage

Dans la mesure du processus, les pièces de construction, par exemple, la

AND−Split, AND−join, OR−Split et OR−Join sont utilisées pour développer des flux séquentiels, conditionnels, parallèles et itératifs (WfMC [Coa96]). Un processus de métier peut être utilisé pour indiquer la direction des cas. Dans le prochain sous-segment, quatre sortes de direction dont parle le CPN seront présentées : séquentiels, conditionnels, parallèles et itératifs.

4.3.1.1 Routage séquentiel

Ce type de direction est une direction essentielle, une direction séquentielle utilisée pour gérer les connexions causales entre les tâches. Considérons deux transitions A et B. Si la transition B est exécutée après la fin de la transition A, alors A et B sont exécutés séquentiellement. La figure 4.2 démontre que la direction séquentielle peut être affichée en incluant des places. Un processus de métier doit avoir deux points exceptionnels: place i et place o, nous les considérons comme un état de début et une condition d'achèvement d'un processus de métier. Entre chaque transition voisine A et B, une place modélise la connexion causale entre la place A et la place B, c'est-à-dire met i à une condition intérieure pour la place A.

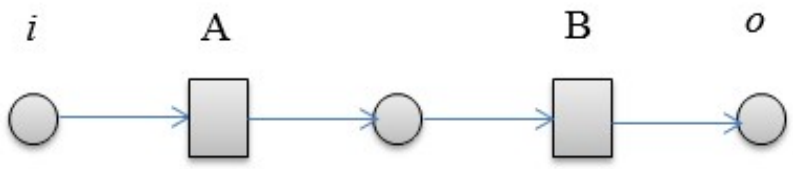

Figure 4.2: Structure séquentielle

4.3.1.2 Routage parallèle

La direction parallèle est utilisée pour caractériser les transitions qui doivent être exécutées en parallèle. Pour modéliser une telle direction parallèle, deux opérateurs logiques de construction sont utilisés: (1) la AND−Split et (2) la AND−Join. La figure 4.3 montre que les deux logiques opérateurs de construction peuvent être utilisés par des structures standards.

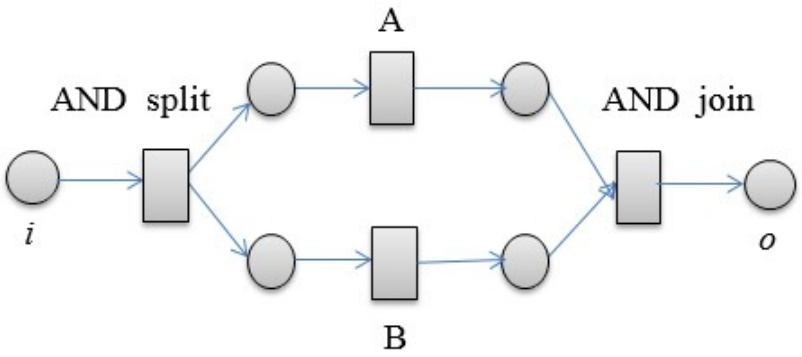

Figure 4.3: Structure parallèle

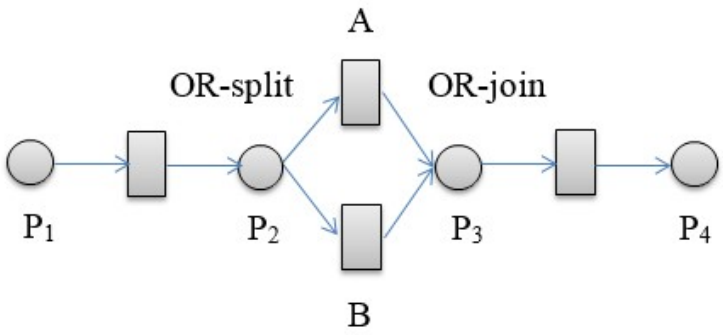

Figure 4.4: Structure OR implicite

4.3.1.3 Routage conditionnel

La direction conditionnelle est utilisée pour tenir compte d'une orientation qui peut fluctuer entre les cas. Pour montrer une décision entre deux autres choix, deux opérateurs logiques de construction sont utilisés : (1) la OR−Split et (2) la OR−Join (dans les deux cas, un OR sélectionné). Une OR−Split peut être affichée par une place avec divers segments circulaires actifs, une OR−Join est démontrée par une place avec de nombreuses courbes continues.

4.3.1.4 Déclencheur

Un *trigger* est une action externe qui mène au déclenchement d'une transition activée. Le déclenchement d'une transition pour un cas spécifique commence au moment où la transition est déclenchée. Une transition ne peut être déclenchée que si le cas correspondant est dans un état qui permet le déclenchement de la transition.

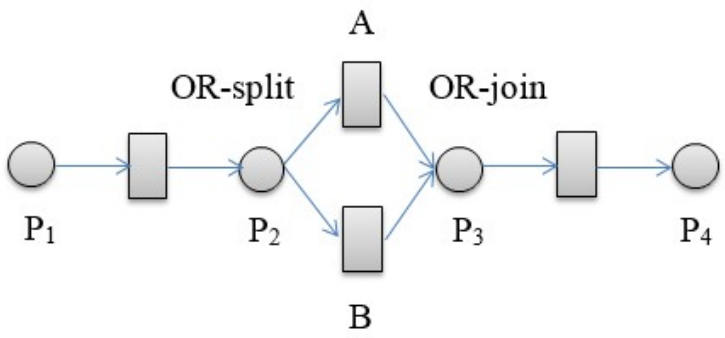

Figure 4.5: Structure OR explicite

Dans cette section, nous distinguons quatre types de transitions.

- *Automatique* : une transition se déclenche au moment où elle est activée. Ce type de déclencheur est utilisé pour une transition qui est déclenchée par une application qui ne nécessite pas d'interaction humaine.
- *Client* : une transition est déclenchée par un participant humain, c'est-à-dire qu'un utilisateur sélectionne une transition activée à déclencher.
- *Message* : un événement externe (c'est-à-dire un message) déclenche une transition activée. Des exemples de messages sont les appels − mobiles, les messages, les e-mails ou les messages EDI.
- *Temps* : une transition activée est déclenchée par une horloge, c'est-à-dire que la transition est déclenchée à un temps prédéfini. Par exemple, la transition "supprimer le document" est déclenchée si une affaire est piégée dans un état spécifique pendant plus de 15 heures.

Ce n'est que pour la transition automatique que l'activation et le début réel de l'exécution coïncident.

4.3.2 Processus de métier basé sur un CPN

La définition de processus de métier basé sur CPN a été introduite par W.M.P Val dee Aalst [vdA98]. Elle a été utilisée par de nombreuses recherches dans le domaine de la gestion des processus de métier. Dans notre solution, nous étendons la définition de W.M.P Val dee Aalst pour le processus de métier en ingénierie logicielle.

Les définitions suivantes sont définies pour un cas particulier dans l'ingénierie logicielle, car il existe une différence entre le processus de métier dans la gestion des processus de métier et le processus de métier en ingénierie logicielle. Afin d'adapter le diagramme de CPN pour le domaine de l'ingénierie logicielle, nous définissons une extension du CPN avec les propriétés qui permettent à un graphe CPN d'interagir avec une source de données, et le jeton peut être déclenché par un langage de requête.

Définition 1 : *Un processus de métier basé sur CPN* $CPN{-}BP{=}\{P,T,F,\sum,\Delta\}$ *est un CPN−BP si et seulement si :*

- P *a deux place spéciales : i et o. Place i est un place source:* $\bullet\, i{=}\emptyset$. *Place o est un place de sortie:* $o\bullet{=}\emptyset$;
- *T est un ensemble de transitions ;*
- $\sum$ *est un ensemble de couleurs définies dans le modèle CPN. Cet ensemble contient toutes les couleurs possibles, les opérations et les fonctions utilisées dans le CPN*
- *Si nous ajoutons une transition* t^* *à CPN-BP qui relie l'emplacement o avec i, le réseaux de Pétri résultant est fortement connecté ;*
- *T a trois sous-ensembles:* $T{-}Get$, $T{-}Post$ *et* $T{-}Control$

  Avec $T{=}T{-}Get \cup T{-}Post \cup T{-}Control$

  - $T{-}Get$ *est un ensemble de transitions qui reçoit le jeton pour obtenir les données d'une source spécifique.*
  - $T{-}Post$ *est un ensemble de transitions qui publie les données du jeton à une cible spécifique.*
  - $T{-}Control$ *est un ensemble de transitions qui contrôle le chemin d'exécution de l'application.*
- Δ T représente une source de données qui sera utilisée pour interagir avec un graphe de CPN.

Lors de l'utilisation de CPN pour définir un processus de métier, les places correspondent à des conditions, les transitions correspondent à des tâches.

Définition 2 : *Nous définissons deux types de jeton à savoir : external token Ex−Token et internal token In−Token:*

- *Externe−Token est un ensemble de jetons dont la valeur est remplie par des activités extérieures. Nous classons ce type de jeton en deux ensembles :*
  - *Typed Data token : ce type de jeton est la donnée saisie par l'utilisateur.*
  - *Event Token : ce type de jeton est un événement, par exemple : email envoyé,...*
- *Internal−Token est un ensemble de jetons qui a été transféré d'une transition précédemment lancée. La valeur de ce jeton est remplie par une activité dans le graphe CPN*

Définition 3 : ***(Firing Query)*** *Chaque type de transition correspond à une Firing Query, le résultat d'une Firing Query est rempli dans un jeton prédéfini. La Firing Query est classée comme suit:*

- *Si une transition est un T−Get, nous appelons la requête de tir Get Firing Query Get(FQ-G):*
  - *GET ALL* ***Color*** */ ORDERBY* ***Property*** */ GROUP BY* ***Property****/ HAVING* ***Property***
  - *GET* ***Color*** *WHERE* ***Property****='Token Value' / ORDERBY* ***Property*** */ GROUP BY* ***Property****/ HAVING* ***Property***
- Si une transition est un *T−Post*, nous appelons la firing query Firing Query Post (FQ-P):
  - *DELETE* ***Color*** */ WHERE* ***Property****='Token Value'*
  - *UPDATE* ***Color*** *SET* ***Property****='External Token Value' WHERE* ***Property****='Token Value'*
  - *SELECT* ***Color*** *WHERE* ***Property****='Token Value'*

La définition *Firing query* ci-dessus est utilisée pour l'interaction entre un graphe de CPN et une base de données relationnelle. La requête de tir, dans ce cas, est représentée pour la requête SQL. Cette requête peut être remplacée par une requête

SPARQL [W3Cc] pour récupérer les informations à partir d'une source de données sémantiques RDF.

Dans la figure 4.6, nous faisons un exemple d'un processus de métier basé sur le CPN. Ce processus est utilisé pour construire un comportement d'un site web d'émission de télévision. Il ne couvre pas toutes les fonctions de ce type de site web, mais nous l'utilisons pour présenter notre modèle.

Exemple 1 : Processus de métier du site web de l'émission de télévision.
$CPN-BP=\{ P,T,F,\sum\}$
$P=\{ i,C_1,C_2,C_3,C_4,C_5,C_6,C_7,C_8,C_9,C_{10},C_{11},C_{12},C_{13},C_{14},C_{15} \}$
T={Load Home Page, New TV Show, Ongoing TV Show, Latest Update, Login,AND−Join,Bookmark List, Profile, Request TV Show, Select TV Show, Episode, Logout, Change Password, OR−Join, Request Movies Confirmation }
T−Get={New TV Show,Ongoing TV Show,Latest Update,Bookmark List,Profile,Select TV Show,Episode }
T−Post={Login,Request TV Show,Logout,Change Password }
T−Control={Load Home Page,Login,AND−Join,OR−Join }
$\sum$*={TV Show,User,Episode,Bookmark}*
Set of FQ-G :
FQ-G-1= GET ALL TVShow ORDERBY AddedDate
FQ-G-2= GET ALL Ongoing Movie
FQ-G-3= GET ALL TVShow ORDERBY UpdatedDate
FQ-G-4= GET ALL Bookmark OF User
FQ-G-5= GET ALL Profile OF User
FQ-G-6= GET TVShow WHERE id='Token Value'
FQ-G-7= GET ALL Episode
Set of FQ-P :
FQ-P-1= SELECT User Where UserName='Token value' AND Password='Token Value'
FQ-P-2= INSERT Movie
FQ-P-1= UPDATE User SET Password='Token value' Where UserName='Token Value'

- Place i est l'état de début du processus.

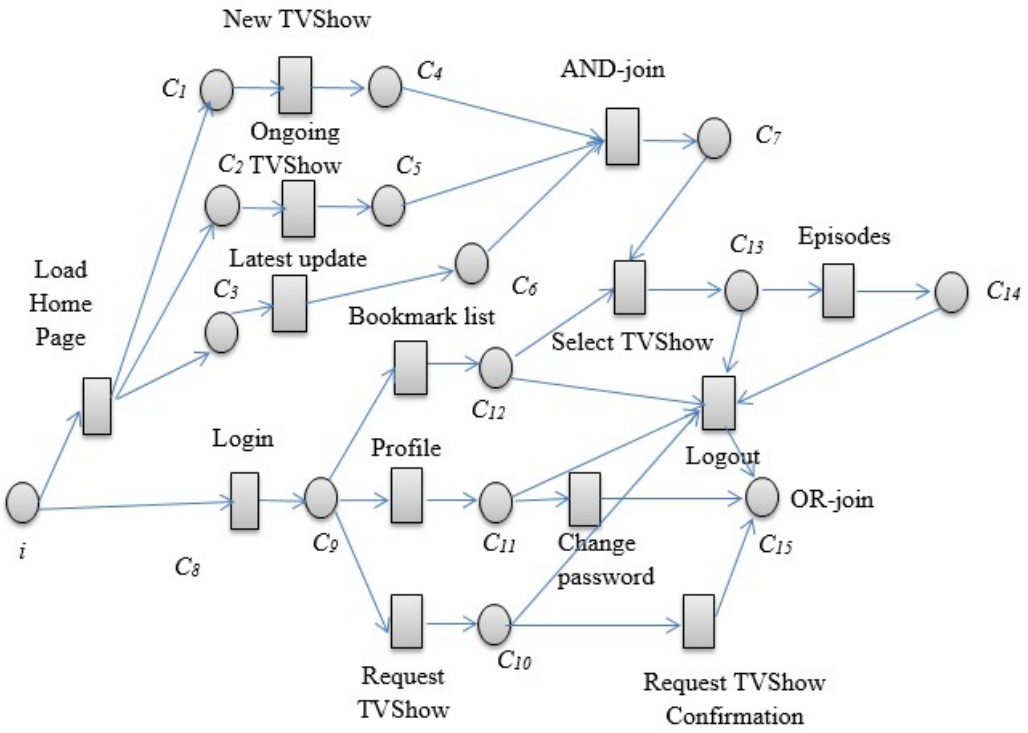

Figure 4.6: Processus de métier pour le site web d'émission de télévision

- Transition *Load Home Page* est une transition AND−Split utilisée pour définir ces trois transitions *New TV Show*, *Ongoing TV Show*, *Lastest Update* doit être exécuté en parallèle.

- Transition *Login* présente la tâche de connexion du site, une fois que cette transition a été déclenchée, l'utilisateur peut exécuter trois transition *Bookmark List*, *Profile* et *Request TV Shows*.

- Transition *Bookmark List* permet à l'utilisateur de montrer sa liste d'émission de télévision favoris .

- Transition *Profile* permet à l'utilisateur de montrer son profil.

- Transition *Request TV Show* permet à l'utilisateur de demander ses films préférés.

- Transition *New TV Show* est utilisé pour obtenir le nouvelle émission de télévision dans la base de données.

- Transition *Ongoing TV Show* est utilisé pour sélectionner la série en cours dans la base de données.
- Transition *Latest update* est utilisé pour sélectionner l'émission de télévision mis à jour.

4.4 Vérification des processus de métier

Afin d'éviter une mauvaise exécution d'un processus de métier, il est nécessaire de vérifier l'exactitude des processus opérationnels. Dans cette section, nous présentons certaines propriétés qui doivent être vérifiées pour s'assurer qu'un CPN − BP est construit sans blocage.

Définition 4 : (Possible Path) *Lorsqu'il existe au moins un chemin entre la transition A et la transition B, il existe un Possible Path entre A et B. Ceci est exprimé comme suit :*

$$PATH(A,B)$$

Possible Path est transitif. Soit A, B, C (A, B, C *en* T) est une série de transitions. S'il y a un *PATH (A, B)* et *PATH (B, C)*, il existe un *PATH (A, C)*...
Définition 5 : (Only Transferable Path) S'il existe un seul chemin d'une transition A vers la transition B, il n'y a que le Only Transferable Path entre A et B. Ceci est exprimé comme suit :

$$OTP(A,B)$$

4.4.1 Exemple d'impasse

Une halte dans un modèle de procédure est donnée si un cas spécifique du modèle (mais pas vraiment de tous) ne peut pas fonctionner, alors qu'il n'a pas encore atteint sa fin. La figure 4.7 démontre la situation où un jeton évite quelques concentrateurs. À l'heure actuelle, dans ce tableau, le jeton qui a passé AND−Split est séparé. Les jetons sont échangés à deux points de rendement de AND−Split. Après qu'un jeton passe l'un des deux places de rendement, il résiste au mouvement de AND−Join pour se joindre au jeton d'un autre lieu de rendement de AND−split. Ensuite, le jeton sera échangé à un mouvement OR−Split. Deux jetons ne peuvent pas se joindre au mouvement AND−Join. De cette manière, le jeton s'arrête au

mouvement de AND−JOin. L'arrêt implique que le jeton ne peut déplacer qu'à l'endroit donné.

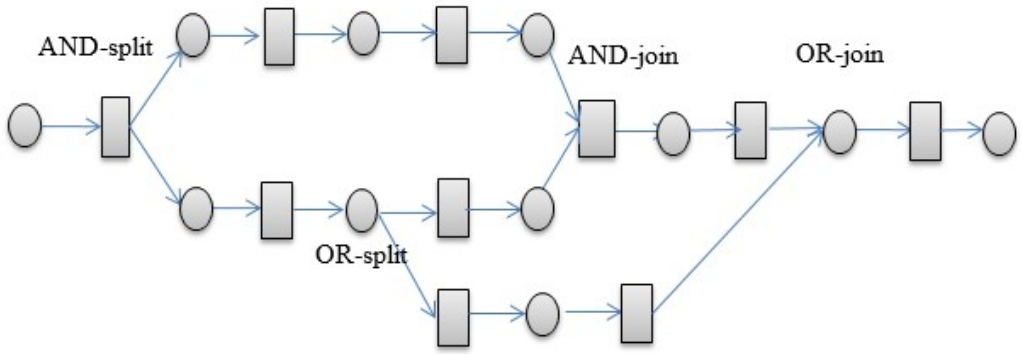

Figure 4.7: Exemple de blocage

4.4.2 Classification des impasses

Lorsque le jeton ne peut pas accéder aux deux places d'entrée d'une transition AND − join, un blocage se produit. A partir de définitions 3 et 4 qui ont été définies ci-dessus et avec le résultat de l'analyse des causes d'une impasse dans l'exemple de blocage, nous classons cinq modèles comme suit :

- *Possible Path* avec *Only Transferable Path.* Le chemin d'exécution d'une transition A à l'arc d'entrée d'une AND−Join est libre de XOR et OR-Split. Selon la définition du seul chemin transférable, aucun blocage ne se produit.
- *Possible Path* sans *Only Transferable Path.* Ici, le chemin d'exécution d'une transition A à l'arc de saisie d'une transition AND−Join comprend des divisions XOR(OR). Ici, des blocages peuvent se produire
- *No Possible Path.* Cela signifie qu'il n'y a pas de chemin d'une certaine transition vers les arcs d'entrée d'une transition AND−Join, il n'y a pas de blocage.
- Boucle infinie. Cela signifie qu'il existe une possibilité pour *Possible Path* sans *Only Transferable Path* à partir de la sortie d'une transition AND−Join son entrée. Ce type de blocage se produit lorsqu'il existe un chemin exécutable

Type de base de réseaux de Pétri coloré	Type de base de XML Schema
INTEGER	integer positiveInteger negativeInteger nonPositiveInteger nonNegativeInteger int long
Reals	decimal
Strings	string
Booleans	boolean

Table 4.1: Équivalence des types de données entre réseaux de Pétri coloré et schème XML

à partir du lieu de sortie d'une transition AND−Join vers un sous-ensemble de sa transition précédente. Si ce chemin contient une XOR−Split, l'impasse se produit uniquement lorsque la branche qui mène à la boucle est choisie. Au cas où il existe un chemin qui ne contient pas XOR−Split l'occurrence du blocage est certaine.

4.5 Ontologie des processus de métiers

Cette section présente une méthode pour représenter un CPN-BP par une ontologie OWL 2.

4.5.1 Modèle CPN-BP à l'ontologie OWL 2

4.5.1.1 Règles de alignement

1. **Valeur et types :** Cette section présente l'équivalence entre le type de type CPN et le type de données de schéma XML. Ce mappage est utilisé pour définir l'ensemble de couleurs de CPN Le tableau 4.1 montre ces types d'équivalences.

2. **Opérateur de base et opérateur logique**

   (a) **Négation logique :** Il existe deux syntaxes d'équivalence pour la négation logique. Si l'opération est de type d'objet, elle sera représentée par cette syntaxe :

   ObjectComplementOf(op:operation)

   Si l'opération est de type de données littérales, elle sera représentée comme suit :

   DataComplementOf(op:ope)

   (b) **Conjonction :** Cette opération est représentée par des axiomes dans une ontologie en fonction du type d'opération. Si les opérations sont des types d'objet, les axiomes sont équivalents à

   ObjectIntersectionOf(op:ope1 op:ope2)

   Si les opérations sont des types de données littérales, l'axiome est représenté comme suit:

   DataIntersectionOf(op:ope1 op:ope2)

   (c) **Disjonction :** Cette opération est représentée par des axiomes dans une ontologie en fonction du type d'opération. Si les opérations sont des types d'objet, les axiomes sont équivalents à

   ObjectUnionOf(op:ope1 op:ope2)

   Si l'opération est de type de données littérales, l'axiome est représenté comme suit:

   DataUnionOf(op:ope1 op:ope2)

   (d) **Equivalence :** Si les opérations sont des types données littérales, l'axiome est représenté comme suit :

   EquivalentClasses(op:ope1 op:ope2)

   Si les opérations sont des concepts individuels, l'axiome est représenté comme suit :

   SameIndividual(op:ope1 op:ope2)

Si les opérations sont de type unaire, l'axiome est représenté comme suit :

EquivalentDataProperties(op:ope1 op:ope2)

Si les opérations sont de type binaire, l'axiome est représenté comme suit :

EquivalentObjectProperties(op:ope op:ope)

(e) **If Else statement** Nous utilisons SWRL pour représenter l'affirmation "If Else" : par exemple,

```
Implies(Antecedent(hasToken(x1 x2))
        hasBrother(x2 x3))
  Consequent(hasUncle(x1 x3)))
```

3. **Déclarations de l'ensemble de couleur** Les déclarations d'ensemble de couleurs d'un graphe de CPN seront représentées par un ensemble de concepts dans l'ontologie OWL.

   Par exemple, nous avons un ensemble de déclarations d'ensemble de couleurs comme suit :

```
color Student=record d:StudentID * r:Name;
color Professor=record d:ProfID * r:Name;
color Subject=record d:SubID * r:Name;
```

   Ces déclarations d'ensemble de couleurs peuvent être représentées par la syntaxe fonctionnelle OWL comme suit :

```
Declaration(Class(color:Student))
Declaration(Class(color:Professor))
Declaration(Class(color:Subject))

Declaration(DataProperty(color:StudentID))
DataPropertyDomain(color:StudentID color:Student)
DataPropertyRange(color:StudentID xsd:int)

Declaration(DataProperty(color:StuName))
DataPropertyDomain(color:StuName color:Student)
DataPropertyRange(color:StuName xsd:string)

```

```
Declaration(DataProperty(color:ProfID))
DataPropertyDomain(color:ProfID color:Professor)
DataPropertyRange(color:ProfID xsd:int)

Declaration(DataProperty(color:SubID))
DataPropertyDomain(color:SubID color:Student)
DataPropertyRange(color:SubID xsd:int)

Declaration(DataProperty(color:ProfName))
DataPropertyDomain(color:ProfName color:Professor)
DataPropertyRange(color:ProfName xsd:string)

Declaration(DataProperty(color:SubName))
DataPropertyDomain(color:SubName color:Subject)
DataPropertyRange(color:SubName xsd:string)
```

4. **Multi-Sets** Pour initialiser les places avec des jetons multiples, mais aussi à diverses fins, nous avons besoin de *multi-Set* également appelés sacs. Un *multi-set* peut être représenté par une syntaxe fonctionnelle OWL 2 comme exemple suivant.

```
1'"NumberOfStudent" ++ 2'"NumberOfProfessor" ++ 2'"NumberOfSubject
    "
```

Le *multi-set* ci-dessus peut être représenté comme suit :

```
Declaration(Class(initial:InitialPlace))
EquivalentClasses(qa:InitialPlace ObjectIntersectionOf(
    DataExactCardinality(1 init:NumberOfStudent)
    DataExactCardinality(2 init:NumberOfProffessor)
    DataExactCardinality(2 init:NumberOf)))
```

5. **Firing Query** : une *Firing Query* est utilisée pour interroger les données de la source de données afin de déclencher une transition. Elle doit être représentée dans la syntaxe fonctionnelle OWL 2. Selon le type de transition, nous classons la requête de mise en route en deux types :

   - Si une transition est un $T-Get$, nous appelons la firing query est Firing Query Get(FQ-G) :

- *GET ALL **Color** | ORDERBY **Property** | GROUP BY **Property**| HAVING **Property***

  Cette *Firing Query* peut être représentée par un SWRL (First Order Logic) comme suit:

  Fired(token,x)← Color(x) ∧ | ORDERBY(object) ∧ hasObject-Property(x, object)|ORDERBY(object) ∧ hasDataProperty(x, token)|GROUPBY(object) ∧ | hasObjectProperty(x, object)

- *GET **Color** WHERE **Property**='Token Value' | ORDERBY **Property** | GROUP BY **Property**| HAVING **Property***

  Cette *Firing Query* peut être représentée par un SWRL (First Order Logic) comme suit:

  Fired(token,x)← Color(x) ∧ | ORDERBY(object) ∧ hasObjectProperty(x, object)|GROUPBY(object) ∧ | hasObjectProperty(x, object)

- Si une transition est un *T−Post*, nous appelons la firing quey *Firing Query Post*(FQ-P) :

  - *DELETE **Color** | WHERE **Property**='Token Value'*

    Cette *Firing Query* peut être représentée par un SWRL (*First Order Logic*) comme suit :

    Fired(token,x)← Color(x) ∧ | DELETE(x) ∧ hasObjectProperty(x, object)|GROUPBY(object) ∧ | hasObjectProperty(x, object)

  - *UPDATE **Color** SET **Property**='External Token Value' WHERE **Property**='Token Value'*

    Cette *Firing query* peut être représentée par un SWRL (*First Order Logic*) comme suit:

    Fired(token,x)← Color(x) ∧ | UPDATE(x) ∧ hasObjectProperty(x, object)|GROUPBY(object) ∧ | hasObjectProperty(x, object)

  - *SELECT **Color** WHERE **Property**='Token Value'*

    Cette *Firing query* peut être représentée par un SWRL (*First Order Logic*) comme suit :

    Fired(token,x)← Color(x) ∧ hasObjectProperty(x, object)

6. **Structure principale** Les règles d'équivalence de la structure principale sont importantes pour représenter un graphe CPN. Il s'agit d'un ensemble

d'axiomes dans le ABOX. Une instance graphe de CPN est un ensemble d'individus des concepts définissant dans cette partie. Un graphe de CPN connectif s'il satisfait aux axiomes suivants :

- Concept *CPN* représente un graphe du CPN, cette contrainte est de s'assurer qu'un graphe du CPN doit avoir au moins un *InputArc* et *OutputArc*, deux *Places* et un *Transition*, parce que le plus petit CPN est construit par une seule transition avec l'état de début et l'état de fermeture.

```
Declaration(Class(qa:CPN))
SubClassOf(qa:CPN ObjectIntersectionOf(ObjectMinCardinality(1
    qa:hasInputArc qa:InputArc) ObjectMinCardinality(1 qa:
    hasOutputArc qa:OutputArc) ObjectMinCardinality(2 qa:
    hasPlace qa:Place) ObjectMinCardinality(1 qa:hasTransition
    qa:Transition)))
```

- Concept *AND-Join* est un sous-concept de *Transition*, ce concept représente un nœud de contrôle pour lier le flux qui a été divisé par un nœud *AND-Split*. Donc, un *AND-Join* doit avoir au moins deux points d'entrée. Car pour rejoindre le flux, il doit joindre au moins deux parties entrantes.

```
Declaration(Class(qa:AND-Join))
SubClassOf(qa:AND-Join ObjectIntersectionOf(qa:Transition
    ObjectMinCardinality(2 qa:hasInputPlace qa:Place)))
```

- Concept *AND-Split* est un sous-concept de *Transition*, ce concept représente un nœud de contrôle pour diviser le flux en plus de deux parties et permettre à ces parties fractionnées d'exécuter en parallèle. A *AND-Split* doit avoir au moins deux points de sortie. Car pour diviser le flux afin d'effectuer la tâche parallèle, il doit diviser le flux dans au moins deux parties à venir.

```
Declaration(Class(qa:AND-Split))
SubClassOf(qa:AND-Split ObjectIntersectionOf(qa:Transition
    ObjectMinCardinality(2 qa:hasOutputPlace qa:Place)))
```

- Concept *Arc* représente l'arc dans un graphe du CPN

```
Declaration(Class(qa:Arc))
```

- *InputArc* est un concept pour représenter l'arc qui relie un lieu à une transition. Ainsi, les axiomes suivants représentent qu'un *InputArc* est un sous-concept d'*Arc* et il doit se connecter à un endroit exact sur le côté gauche et une transition sur le côté droit. Un arc d'entrée possède une expression maximale.

```
Declaration(Class(qa:InputArc))
SubClassOf(qa:InputArc qa:Arc)
SubClassOf(qa:InputArc ObjectIntersectionOf(
    ObjectExactCardinality(1 qa:hasSourcePlace qa:Place)
    ObjectExactCardinality(1 qa:hasTargetTransition qa:
    Transition) ObjectMaxCardinality(1 qa:hasExpression qa:
    Expression)))
```

- Concept *OR-Join* est un sous-concept de *Place*, ce concept représente un nœud de contrôle pour rejoindre le flux qui a été divisé par un nœud *OR-Split*. A *OR-Join* doit avoir au moins deux *InputTransitions*. Car pour rejoindre le flux, il doit joindre au moins deux parties entrantes.

```
Declaration(Class(qa:OR-Join))
SubClassOf(qa:OR-Join ObjectIntersectionOf(qa:Place
    ObjectMinCardinality(2 qa:hasInputTransition qa:Transition
    )))
```

- Concept *OR-Split* est un sous-concept de *Place*, ce concept représente un nœud de contrôle pour diviser le flux en plus de deux parties et permettre à ces parties séparées de s'exécuter dans le choix. A *OR-Split* doit avoir au moins deux transitions de sortie. Car pour diviser le flux afin d'accomplir les tâches qui s'exécutent, il faut diviser le flux dans au moins deux parties à venir.

```
Declaration(Class(qa:OR-Split))
SubClassOf(qa:OR-Split ObjectIntersectionOf(qa:Place
    ObjectMinCardinality(2 qa:hasOutputTransition qa:
    Transition)))
```

- *OutputArc* est un concept pour représenter l'arc qui relie une transition à une place. Ainsi, les axiomes suivants représentent qu'un *InputArc*• est un sous-concept d'*Arc* et il doit se connecter à une transition exacte sur

le côté gauche et à une place sur le côté droite. Un arc d'entrée possède une expression maximale.

```
Declaration(Class(qa:OutputArc))
SubClassOf(qa:OutputArc qa:Arc)
SubClassOf(qa:OutputArc ObjectIntersectionOf(
    ObjectExactCardinality(1 qa:hasSourceTransition qa:
    Transition) ObjectExactCardinality(1 qa:hasTargetPlace qa:
    Place) ObjectMaxCardinality(1 qa:hasExpression qa:
    Expression)))
```

- *InputArc* est un concept pour représenter l'arc qui relie une place à une transition. Ainsi, les axiomes suivants représentent qu'un *InputArc* est un sous-concept d'*Arc* et il doit se connecter à un place exact sur le côté gauche et une transition sur le côté droite. Un arc d'entrée possède une expression maximale.

```
Declaration(Class(qa:Place))
SubClassOf(qa:Place ObjectIntersectionOf(ObjectMinCardinality
    (1 qa:hasArc ObjectUnionOf(qa:InputArc qa:OutputArc))
    ObjectMinCardinality(0 qa:hasToken qa:Token)))
```

- ```
  Declaration(Class(qa:Token))
  ```

- ```
  Declaration(Class(qa:Transition))
  SubClassOf(qa:Transition ObjectIntersectionOf(
      ObjectMinCardinality(1 qa:hasGuardFunction qa:Expression)
      ObjectMinCardinality(1 qa:hasInputArc qa:InputArc)
      ObjectMinCardinality(1 qa:hasOutputArc qa:OutputArc)))
  ```

Afin d'aider le lecteur familier avec la logique de description, nous réécrivons les axiomes ci-dessus dans la logique de description comme suit :

(a) $CPN \sqsubseteq \geq 2 hasPlace.Place \sqcap \geq 1 hasTransition.Transition \sqcap \geq 1 hasInputArc.InputArc \sqcap \geq 1 hasOutputArc.OutputArc$

(b) $Place \sqsubseteq \geq 0 hasToken.Token \sqcap \geq 1 hasArc.(InputArc \sqcap OutputArc)$

(c) $Transition \sqsubseteq \geq 1 hasInputArc.InputArc \sqcap \geq 1 hasOutputArc.OutputArc \sqcap \leq 1 hasGuardFunction.Expression$

(d) $Arc \sqsubseteq \leq 1 hasExpresssion.Expression$

(e) $InputArc \sqsubseteq = 1 hasSourcePlace.Place \sqcap = 1 hasTargetTransition.Transition$

(f) $OutputArc \sqsubseteq = 1 hasTargetPlace.Place \sqcap = 1 hasSourceTransition.Transition$

(g) $AND\text{-}Split \sqsubseteq Transition \sqcap \geq 2 hasOutputPlace.Place$

(h) $AND\text{-}Join \sqsubseteq Transition \sqcap \geq 2 hasInputPlace.Place$

(i) $OR\text{-}Split \sqsubseteq Place \sqcap \geq 2 hasOuputTransition.Transition$

(j) $OR\text{-}Join \sqsubseteq Place \sqcap \geq 2 hasInputTransition.Transition$

Le CPN est le concept de représentation de tous les graphe CPN. Un graphe CPN est bien défini si et seulement s'il possède au moins une place, une transition, un arc d'entrée et un arc de sortie. Nous définissons les classes suivantes de BPO : *CPN*, *Place*, *Transition*, *OutputArc*, *InputArc* et certaines propriétés qui définissent les relations entre eux, *hasPlace*, *hasTransition*, *hasInputArc*, *hasOutputArc*.

Place représente les propriétés de la place, nous définissons un concept Place. Une place peut avoir un jeton ou non, elle a également au moins un *InputArc* ou un *OutputArc*. Le concept transition est défini pour toutes les transitions. Une transition doit avoir au moins un *InputArc* et un *OutputArc*. C'est l'une des conditions minimales pour avoir un bon graphe CPN. Une transition peut avoir une seule fonction de garde ou non. Le concept *InputArc* est défini pour tous les arcs d'entrée. Un arc d'entrée a une seule place source et une transition cible. Il peut être marqué par une seule expression ou non. Un *OutputArc* n'a qu'une seule transition source et une seule place cible. Il ne peut y avoir qu'une seule expression ou non. Nous définissons également un nœud de contrôle en suivant les concepts suivants: $AND-Split$ est un sous-concept de *Transition*, $AND-Join$ est un sous-concept de *Transition*. $OR-Split$ est un sous-concept de *Place*, $OR-Join$ est un sous-concept de *Place*.

4.5.2 Règles de vérification de la structure

À partir de la classification des règles de l'impasse dans la section 4.4.2, nous définirons certaines règles correspondantes à l'intérieur de l'ontologie pour éviter

l'impasse dans un processus de métier par raisonnement ontologique pendant la période de conception.

- *Possible Path* sans *Only Transferable Path.* Ici, le chemin d'exécution d'une transition A à l'arc de saisie d'une transition AND−Join comprend XOR (OR) se divise. Afin d'éviter cette erreur, nous définissons une règle SWRL dans l'ontologie BPO. Lorsqu'une instance de processus de métier est créée, le raisonneur détectera les erreurs potentielle qui peut se produire dans l'ontologie.

```
TransitiveObjectProperty(hasAfterTransition)

DLSafeRule(Body(ClassAtom(:Transition Variable(var:x)) ClassAtom(:
    Transition Variable(var:y)) ObjectPropertyAtom(:
    hasAfterTransition Variable(var:y) Variable(var:x))
    ObjectPropertyAtom(:hasBeforePlace Variable(var:x) Variable(
    var:y))Head(ClassAtom(owl:Nothing Variable(var:z))))
```

- Boucle infinie.

  Cela signifie qu'il existe une possibilité pour un *Possible Path* sans *Only Transferable Path* à partir de la sortie d'une transition AND−Join de rejoindre son entrée. Ce type de blocage se produit lorsqu'il existe un chemin exécutable à partir du Place de sortie d'une transition AND−Join vers un sous-ensemble de sa transition précédente. Si ce chemin contient une XOR−Split, une impasse se produit uniquement lorsque la branche qui mène à la boucle est choisie. Dans le cas où il existe un chemin qui ne contient pas XOR−Split l'occurrence du blocage des blocages est certaine.

```
TransitiveObjectProperty(hasAfterTransition)

TransitiveObjectProperty(hasBeforePlace)

DLSafeRule(Body(ClassAtom(:Place Variable(var:x)) ClassAtom(:
    Transition Variable(var:y)) ObjectPropertyAtom(:
    hasAfterTransition Variable(var:y) Variable(var:x))
    ObjectPropertyAtom(:hasBeforePlace Variable(var:x) Variable(
    var:y))Head(ClassAtom(owl:Nothing Variable(var:z))))
```

4.6 Conclusion

Dans ce chapitre, nous présentons notre approche pour représenter et vérifier un processus de métier dans la couche de métier d'une application. Le contenu principal consiste à introduire un ensemble de motifs de blocage qui permet de détecter l'impasse possible dans un processus de métier. Afin de détecter l'impasse, les processus de métier sont représentés comme un ensemble d'individus dans une ontologie OWL. Chaque processus de métier est une ontologie de processus de métier. Nous utilisons un raisonneur pour raisonner et vérifier la cohérence de chaque processus de métier. Un processus de métier est cohérent seulement si l'instance de processus de métier (un ensemble d'individus dans BPO) satisfait les règles prédéfinies et la couche TBOX dans le BPO. L'avantage de cette approche est de permettre au système de vérifier automatiquement un processus de métier pendant le temps de conception et de temps d'execution également. Lors de l'application de cette approche, l'utilisateur peut réduire le temps de développement d'une application. Le développeur peut éviter l'erreur structurée. Le travail de ce chapitre a été publié sur trois articles [PNT15], [PT16a] et [PT15b].

Chapitre 5

Vérification de conformité des processus de métier

Contenu

5.1 Introduction

Dans ce chapitre, nous présentons deux méthodes de création de règles de métiers et la vérification de la conformité d'un processus de métier avec un ensemble de règles de métier. L'idée principale de notre solution est de représenter un processus de métier et un ensemble de règles de métier par deux ontologies appelées "Ontologie des processus de métiers (BPO)" et "Ontologie des règles de métiers (BRO)", BPO a été introduit dans le chapitre précédent. Dans ce chapitre, nous proposons une solution qui permet à l'utilisateur de définir la règle métier de la couche de métier de son application. L'ensemble de la règle de métier peut être utilisé pour vérifier que les processus de métier sont conçus et exécutés correctement. L'avantage de cette

approche est de permettre à l'utilisateur de vérifier la cohérence de la règle de métier et des processus de métier ainsi que de vérifier automatiquement par le raisonnement la conformité du processus de métier avec les règles de métier. Ce chapitre est construit comme suit : Après l'introduction, dans le travaux connexes, nous allons énumérer et analyser les travaux sur la vérification de la conformité des processus de métier. Dans la section suivante, nous présenterons notre approche basée sur l'ontologie pour représenter l'ontologie des règles de métiers. Avant la conclusion, nous présenterons notre solution en fonction de l'intégration des processus de métier et règles de métier.

5.2 Travaux connexes

Dans cette section, nous présentons les travaux connexes sur la vérification de la conformité des processus de métier. Il existe de nombreuses approches existantes pour le faire, mais nous les classons en trois approches principales comme suit :

5.2.1 Vérification de conformité des processus de métier

5.2.1.1 Approches du cycle de vie des objets

[KRG07] présente la notion de cycle de vie et de couverture de l'objet pour vérifier si un modèle de processus est conforme au cycle de vie de l'objet référencé au moment du design. La technique proposée génère d'abord un modèle de processus à partir d'un ou plusieurs cycles de vie d'objets référencés. Dans la première étape, le cycle de vie de l'objet est utilisé pour générer un ensemble d'actions pour le modèle de processus afin d'identifier les transitions dans les cycles de vie des objets donnés. Cela garantit que les états composites non valides ne peuvent pas être atteints dans le cycle de vie composite des objets. L'ordre du modèle de processus est ensuite déterminé, et les actions sont combinées avec des fragments de processus respectifs à la deuxième et à la troisième étape. Les fragments de processus sont connectés dans la dernière étape. Cette approche fournit un support aux concepteurs de processus. Cependant, il n'est pas entièrement automatisé car les points de synchronisation entre les cycles de vie des processus doivent être définis manuellement dans le cas de plusieurs cycles de vie des objets. En outre, dans certains cas, le nombre de cycles de vie des objets peut être très important, ce qui peut augmenter la taille du modèle

de processus. La taille accrue des modèles de processus peut les rendre difficiles à manipuler. Dans cette approche, aucun mécanisme n'est fourni pour déterminer comment la vérification de la conformité sera affectée si la taille des modèles de processus devient relativement grande ; comment un grand nombre de cycles de vie d'objet référencés sont pris comme entrée ; ou comment la conformité sera conservée si un processus généré est personnalisé. Cela pose également la question de savoir si les dépendances entre les règles de conformité dynamique et les alternatives sont préoccupantes (c'est-à-dire en réalisant les règles de conformité correctes) lorsqu'un modèle de processus est personnalisé.

[SALM09] étend le travail de [KRG07], pour résoudre le problème d'un point de synchronisation (variabilité) et pour préserver la conformité dans les modèles de processus personnalisés. Les auteurs présentent une approche basée sur le concept d'un modèle de processus de métier qui implique implicitement des contraintes de conformité et des points de variabilité pour éviter que les conceptions de processus ne violent pas les contraintes de conformité au moment du design. L'algorithme garantit que les contraintes de conformité ne sont pas violées lorsqu'un modèle de processus est personnalisé. Le problème avec l'algorithme est qu'il ne fournit aucun mécanisme pour gérer les dépendances entre les alternatives et les règles de conformité dynamique, comme mentionné ci-dessus.

5.2.1.2 Approches des modèles/Approches basées sur des graphes

La vérification automatisée de la conformité des exigences légales des processus de métier est hautement souhaitable. Ces exigences sont souvent écrites en langage naturel et doivent être traduites en un format lisible par machine pour une vérification automatisée. Généralement, les langages formels (tels que *Event-Calculus*, *Logique Temporelle*, L*ogique Deontique*), qui fournissent le support de raisonnement, sont utilisés pour traduire les exigences légales. Cependant, en raison de leur complexité, la compréhension et la facilité d'utilisation de ces langages sont difficiles, en particulier pour les utilisateurs non techniques tels que l'analyse de processus et les experts en conformité. Ainsi, la facilité d'utilisation des langages officiels est l'une des principales préoccupations pour les utilisateurs non techniques qui possèdent moins de connaissance de ces langages [Elg12]. Pour répondre à la préoccupation d'utilisation des langages formels, les chercheurs ont proposé d'intégrer les formules

dans un langage formel qui traduit les exigences de conformité en motifs visuels faciles à comprendre ou en graphes. Cela a conduit à l'émergence d'approches de vérification de conformité fondées sur des graphes / modèles dans le domaine de conformité des processus de métier.

[HJL+07] propose un tel langage de spécification de propriété basé sur un modèle, PROPOLS pour spécifier les règles de métier temporelles. Le PROPOLS définit une collection de propriétés pour une composition de service, chaque propriété étant une règle ou une composition logique de règles qui régissent la commande des services primaires dans une composition de service. Chaque règle consiste en un élément de motif et un élément de portée.

Étant donné que chaque modèle spécifie le comportement d'existence d'une activité unique ou d'une relation temporelle entre les activités, PROPOLS permet aux concepteurs de processus d'insérer, de supprimer ou de réorganiser les processus en fonction des règles de métier temporelles. Les écarts par rapport aux règles de métier sont identifiés à l'aide d'automates d'états finis (FSA) pour informer les concepteurs des procédés non conformes. Les automates sont dérivés d'un ensemble de règles de métier et de schéma de processus existant. [YHH+08] étend le travail de Han *et al.* Et propose un cadre de synthèse pour générer des modèles de processus à partir d'un ensemble de règles de métier temporelles. L'approche proposée génère un modèle de processus et un modèle d'exigences (règles temporelles) pour obtenir des spécifications intuitives et une correction par conception. Cela aide les concepteurs de processus à corriger les erreurs de temps de conception. En outre, il permet également la vérification automatisée des modèles de processus générés semi-automatiquement.

[SBO07] discute d'une approche basée sur l'ontologie pour représenter les processus de service et leurs exigences de conformité, afin de vérifier si les processus de service conçus sont conformes. L'approche proposée utilise deux ontologies distinctes: une ontologie de processus définissant les concepts nécessaires pour représenter les processus de service et une ontologie des exigences de conformité constituée des concepts qui représentent les objectifs et les exigences des règles de conformité. Les auteurs présentent trois catégories distinctes d'exigences de conformité dans leur modèle: syntaxique, sémantique et pragmatique. Pour vérifier les éléments de processus conformes aux exigences sémantiques des processus de service, un raison-

neur est appliqué. Le problème avec l'approche proposée est qu'il isole uniquement les processus dont les exigences sont instanciées en tant que processus conformes ; D'autres processus non confirmés ne sont pas inclus. De plus, il n'y a aucune indication sur la façon dont l'approche proposée traite des processus non conformes car aucune action corrective ne peut être prise dans l'approche proposée.

[YMH+06] introduit une approche de vérification de la conformité au schéma BPEL, qui utilise un langage d'ontologie pour les spécifications de propriété. Le processus de vérification commence par une description de haut niveau d'un schéma BPEL à mettre en œuvre dans le processus. Ensuite, l'équivalence sémantique entre les opérations est défini dans le langage ontologie, et un modèle de *Labelled Transition Sytem* (LTS) fini et déterministe est généré. À partir de ce modèle LTS, un automate fini total et déterministe (TDFA) est construit. Cela inclut l'ensemble des états finaux et des états d'erreur pour collecter une liste de tous les événements indésirables de chaque état. Dans la dernière étape, la vérification du schéma BPEL de conformité détermine si toutes les séquences d'événements acceptables du schéma BPEL sont présentes dans la liste des séquences acceptables générées dans la TDMA.

[FES05] présente une approche de processus axée sur les modèles pour exprimer visuellement les contraintes de conformité sur le comportement du processus. Les auteurs utilisent PPSL, une extension des diagrammes d'activité UML (OMG, 2011). Les diagrammes d'activité sont utilisés pour spécifier les modèles possibles qui doivent être appliqués dans les modèles de processus de métier. Cela permet aux concepteurs de processus d'avoir une vue abstraite d'un comportement possible d'un processus de métier.

Bien que l'approche proposée offre de manière souple aux concepteurs de processus de vérifier la qualité de la conformité, l'approche n'est pas exempte de problèmes. La définition du comportement des processus en tant que modèles visuels au moment du design est une telle question, car ces modèles peuvent dépendre l'un de l'autre, voire reflètent des comportements contradictoires. À l'heure actuelle, l'approche ne fournit aucun mécanisme pour acquérir (potentiellement antérieur) la connaissance des interdépendances entre les différents modèles.

En outre, ces modèles ne sont pas en mesure de modéliser expressément une négation ; c'est-à-dire, une règle peut stipuler des conditions qui empêchent certaines

activités de se produire, tandis que d'autres ont déjà été exécutées. Essentiellement, du point de vue de la conformité des processus de métier, la négation est un aspect important des interdictions de modélisation, cependant, aucun soutien explicite n'est fourni dans ce cadre pour modéliser les négations. De plus, cette approche se concentre uniquement sur l'aspect du flux de contrôle des processus de métier et ne fournit aucun soutien pour la modélisation et la vérification de leur conformité aux données, aux aspects de ressources d'un processus de métier.

[NS07] utilise une approche basée sur les modèles pour modéliser les contrôles internes d'une entreprise. Ils construisent leur modèle sur la norme de contrôle interne de fait (autrement connu sous le nom COSO9). Dans la phase d'exécution du processus, une interaction bidirectionnelle entre le BPM et la gestion du contrôle interne est établie. Plus tard, toutes les informations sur l'instance actuelle du processus de métier sont adoptées. En cas de violation, une action de récupération (définie dans les contrôles) est exécutée. Le principal avantage de leur approche est sa capacité à définir différents contrôles au-delà des flux de travail, et dans différents environnements, à la réutilisation des modèles de processus. Cependant, le modèle proposé n'est pas entièrement automatisé car il nécessite une sélection manuelle d'un modèle de contrôle et sa conception sur un processus de métier correspondant aux exigences de conformité spécifiques au domaine. En outre, il n'existe aucun support pour gérer les dépendances inter-contrôle ; Par exemple, différents contrôles peuvent contredire, sous-ensembles ou même bloquer l'exécution d'autres contrôles dans une interaction de processus de métier. Cela implique la nécessité d'établir une corrélation plus forte entre les processus et les contrôles. En outre, cette approche ne prend pas en charge la vérification de la conformité au-delà du temps d'exécution, et elle n'appuie pas non plus les aspects liés à la ressource et au temps du processus de métier.

[AKM08] présente REO Tool-kit, un langage de coordination basé sur les canaux pour la vérification du temps de conception des modèles de processus de métier. Le langage utilise des techniques de vérification de la modélisation et de bisimulation pour analyser formellement l'exactitude des processus de métier par rapport aux contraintes imposées. Pour la vérification des comportements conformes, dans cette approche, les processus de métier sont d'abord modélisés par les diagrammes d'activité BPMN (OMG, 2010) ou UML (OMG, 2011), qui peuvent être mappés

dans les contraintes. Les exigences de conformité sont représentées à l'aide de la Logique temporelle linéaire (LTL), puis les techniques de vérification du modèle intégrées au kit d'outils REO servent à vérifier la conformité des processus de métier. Le travail rapporté dans [STK+10] est ancré dans le kit d'outils REO, où le REO est utilisé pour la vérification automatisée de la conformité des fragments de processus de métier par rapport aux contraintes de métier alignées dans LTL.

5.2.1.3 Approches basées sur la requête

[ADW08] discute d'un BPMN-Q, une approche basée sur la requête pour la vérification de la conformité. L'approche est capable de répondre aux questions Oui/Non pour vérifier si une procédure est conforme. Les auteurs utilisent une technique de réduction de graphe pour obtenir la réponse Oui/Non. En tant qu'exécution d'un graphe de requête, l'approche de réduction de graphe divise un graphe de processus en un ensemble de chemins d'exécution du premier au dernier nœud du graphe. Ensuite, l'ordre d'exécution est déterminé par rapport à un chemin d'exécution en trouvant la priorité entre les occurrences de noeuds. Dans la dernière étape, un graphe de processus correspond à un graphe de requête. Si elle satisfait tous les débits de séquence et de chemin, le BPMN-Q renvoie un OUI à une règle représentant un processus de plainte. Dans le cas où BPMN-Q ne trouve pas de correspondance, un NO est renvoyé pour transmettre une violation d'une règle.

L'approche [ADW08] fournit une réponse à la requête des règles de manière efficace et permet de vérifier les commandes entre les activités impliquées dans un processus. Cependant, un problème avec l'approche de réduction du graphe est qu'il pourrait supprimer certaines activités qui, à première vue, pourraient ne pas être pertinentes pour une requête. Cela pourrait inclure les activités qui doivent jouer un rôle important dans l'achèvement d'un processus. Le travail est ensuite étendu avec les auteurs présentant les moyens de visualiser les violations de la commande de flux de contrôle dans les règles de conformité [AW10].

Encore une fois, ils utilisent des requêtes structurelles BPMN-Q pour exprimer les règles de conformité, appelées 'patterns'. Ces requêtes sont utilisées pour trouver l'ensemble des modèles de processus qui sont soumis à la vérification de la conformité dans un référentiel de processus. Les formules temporelles sont ensuite dérivées des requêtes pour vérifier le modèle de processus. Dans la dernière étape, les anti-

modèles sont dérivés automatiquement des requêtes BPMN-Q pour signaler toute violation de règle dans les modèles de processus. [AW10] utilise une réduction de graphe et un vérificateur de modèle dans cette approche ultérieure. Sa solution proposée pour dériver des requêtes ansti-modèles a certaines limites. Les anti-modèles générés dépendent du système de transition d'état d'entrée du modèle de processus. Si le système de transition est généré à partir d'un modèle de processus réduit, l'anti-modèle résultant ne serait pas utilisable sur le modèle de processus original. De même, comme les anti-modèles générés sont donnés en tant que non-respect d'une violation de règle, il est possible que certaines infractions ne soient pas signalées par le vérificateur de modèle. De plus, une ré-implémentation d'un logiciel de traduction sera requise dans le cas où des modifications sont apportées au logiciel de vérification de modèle.

5.3 SBVR à ontologie de la règle de métier

5.3.1 Classification de la règle de métier

Les règles de métier sont des listes d'énoncés qui vous indiquent si vous pouvez ou non faire quelque chose ou vous donner les critères et les conditions nécessaires pour prendre une décision. Un facteur d'une exigence métier est ce que vous devez faire pour permettre la mise en œuvre et la conformité à une règle métier. Les différentes catégories structurelles des règles de métier sont :

- **Intégrité** (ou contraintes) ; Par exemple, une entreprise a un seul directeur.
- **Dérivation** (conditions résultant de conclusions) ; Par exemple: les clients qui habite à Nice reçoivent une réduction de 5
- **Réaction** (événement, condition, action, alternative d'action, post-condition) ; Par exemple, une facture est reçue. Si le montant de la facture est supérieur à 2 000 USD, un superviseur doit l'approuver.
- **Production**(condition, action); Par exemple: s'il n'y a pas de défauts dans le dernier lot de voitures, le lot est approuvé.
- **Transformation**(changement d'état) ; Par exemple: l'âge d'un homme peut passer de 28 à 29, mais pas de 29 à 28 ans.

Example 2 : EU-Rent a 1000 succursales dans des villes de plusieurs pays. Dans chaque succursale, les voitures (classées par groupe automobile) sont disponibles à la location. Chaque succursale dispose d'un gestionnaire et d'un nombre de commis de réservation qui gèrent les locations.

Règles de métier :

un permis de conduire doit être considéré comme valide si tous les éléments suivants sont vrais:

- **Locations**

  La plupart des locations sont effectuées par anticipation. La période de location et le groupe de voitures sont précisés au moment de la réservation. EU-Rent accepte également des locations immédiates («walk-in»), si les voitures sont disponibles.

  À la fin de chaque journée, les voitures sont affectées aux réservations pour le lendemain. Si plus de voitures ont été demandées que celles disponibles dans un groupe de voitures dans une succursale, le directeur de succursale peut demander à d'autres branches si elles ont des voitures à transférer

- **Retours**

  Les voitures louées à partir d'une branche de l'UE-Rent peuvent être retournées dans une succursale différente. La succursale de location doit s'assurer que la voiture a été retournée dans une succursale à la fin de la période de location. Si une voiture est retournée dans une branche autre que celle qui l'a louée, la propriété de la voiture est attribuée à la nouvelle succursale.

- **Service client**

  EU-Rent dispose également de dépôts de service, chacun desservant plusieurs succursales. Les voitures peuvent être réservées pour maintenance à tout moment, à condition que le dépôt de service ait la capacité le jour en question.

  Pour simplifier, une seule réservation par voiture par jour est autorisée. Une location ou un service peut couvrir plusieurs jours.

- **Clients**

Un client peut avoir plusieurs réservations, mais une seule voiture louée à la fois. EU-Rent conserve les enregistrements des clients, leur location et les mauvaises expériences (comme le retour tardif, les problèmes de paiement et les dommages aux voitures). Cette information est utilisée pour décider d'approuver une location.

5.3.2 Terminologie du domaine

Semantics of Business Vocabulary and Business Rules (SBVR) est un méta modèle développé par OMG pour définir les concepts, les verbes et les règles de métier d'un domaine sous forme de langage naturel basé sur des logiques formelles. Afin d'aider l'utilisateur à définir la règle de métier et celle de la logique métier d'une application, nous proposons une approche pour définir la règle de métier en utilisant SVBR. Cette section présente une méthode pour traduire un ensemble de règles SBVR en ontologie OWL 2. Cette solution permet au système de générer une ontologie OWL automatiquement formant un ensemble de spécifications SBVR dans un domaine. Cette ontologie est utilisée pour vérifier la conformité du processus de métier qui a été mentionné au chapitre 4. Comme présenté au chapitre 2, SBVR contient les éléments suivants: *types*, **type de fait**, **individus** et **rôles de types de faits**. Afin de traduire un ensemble de définitions SBVR, nous proposons des règles générales comme suit :

1. **Type** est représenté par un concept dans l'ontologie OWL.
   Par exemple: une déclaration dans la syntaxe fonctionnelle comme suit:

   Declaration(Class(prefix:Car))

2. **Type de fait** est représenté par des individus du concept de type.
   Par exemple: une déclaration dans la syntaxe de la fonction:
   Declaration(NamedIndividual(prefix:BMW)

   ClassAssertion(prefix:Car url:Mercedes)

3. **Type de fait unaire** est représenté par les propriétés des données. Par exemple, une déclaration d'un fait unaire "seriesOfCar" en syntaxe fonctionnelle.

Declaration(DataProperty(prefix:seriesOfCar)
DataPropertyDomain(prefix:seriesOfCar prefix:Car)
DataPropertyRange(prefix:seriesOfCar prefix:int)

4. **Type de fait binaire** est représenté par les propriétés de l'objet. Par exemple:

Declaration(ObjectProperty(prefix:isBelongTo)
ObjectPropertyDomain(prefix:isBelongTo prefix:Car)
ObjectPropertyRange(prefix:isBelongTo prefix:Person)

5. **Quantification universelle**

   Si la formulation logique est de type de données littérales, elle est représentée par l'axiome suivant:

DataAllValuesFrom(prefix:DataProperty prefiex:DataRange)

   Si la formulation logique est de type de type binaire, elle est représentée par les axiomes suivants:

ObjectAllValuesFrom(prefix:ObjectProperty prefix:Class)

6. **Quantification existentielle**

   Si la formulation est de type de fait unaire, elle est représentée comme l'axiome suivant :
   DataSomeValuesFrom(prefix:DataProperty prefix:DataRange)

   Si la formulation est de type de fait binaire, elle est mise en correspondance avec

ObjectSomeValuesFrom(prefix:ObjectProperty prefix:Class)

7. **Au plus-n Quantification**, n est un nombre entier positif.

   Si la formulation est de type de fait unaire, elle est représentée comme l'axiome suivant :

   DataMaxCardinality(n prefix:DataProperty prefix:DataRange)

   Si la formulation est de type de fait binaire, elle est représentée comme l'axiome suivant :

   ObjectMaxCardinality(n prefix:ObjectProperty a:Class)

8. **Au moins-n Quantification**, n est un nombre entier positif.

   Si la formulation est de type de fait unaire, elle est représentée comme l'axiome suivant :

   DataMinCardinality(n prefix:DataProperty prefix:DataRange)

   Si la formulation est de type de fait binaire, elle est représentée comme l'axiome suivant :

   ObjectMinCardinality(n prefix:ObjectProperty prefix:Class)

9. **Exactement-n Quantification**, n est un nombre entier positif.

   Si la formulation est de type de fait unaire, elle est représentée comme l'axiome suivant :

   DataExactCardinality(n prefix:DataProperty prefix:DataRange)

   Si la formulation est de type de type binaire, elle est représentée comme l'axiome suivant :

ObjectExactCardinality(n prefix:ObjectProperty prefix:Class)

10. **Opérations logiques**

- **Négation logique** Il existe deux syntaxes de mappage pour la négation logique, car une opération est de type d'objet est représentée avec cette syntaxe:

  ObjectComplementOf(op:operation)

  Si l'opération est de type de données littérales, elle est représentée comme suit :

  DataComplementOf(op:ope)

- **Conjonction** Cette opération est représentée par des axiomes dans une ontologie en fonction du type d'opération. Si les opérations sont des types d'objet, les axiomes sont mappés à

  ObjectIntersectionOf(op:ope1 op:ope2)

  Si les opérations sont de type de données littérales, l'axiome est représenté comme suit :

  DataIntersectionOf(op:ope1 op:ope2)

- **Disjunction** Cette opération est représentée par des axiomes dans une ontologie en fonction du type d'opération. Si les opérations sont des types d'objet, les axiomes sont mappés à

  ObjectUnionOf(op:ope1 op:ope2)

  Si l'opération est de type de données littérales, l'axiome est représenté comme suit :

  DataUnionOf(op:ope1 op:ope2)

- **Équivalence** Si les opérations sont de type de données littérales, l'axiome est représenté comme suit :

  EquivalentClasses(op:ope1 op:ope2)

  Si les opérations sont de types de concepts individuels, l'axiome est représenté comme suit :

SameIndividual(op:ope1 op:ope2)

Si les opérations sont de type unaire, l'axiome est représenté comme suit :

EquivalentDataProperties(op:ope1 op:ope2)

Si les opérations sont de type binaire, l'axiome est représenté comme suit :

EquivalentObjectProperties(op:ope op:ope)

Exemple 3 : nous continuons l'exemple 2, car la voiture de location de l'UE est un exemple populaire d'OMG. Dans cet exemple, nous traduirons la spécification de la voiture UR-Rental en OWL 2 comme suit : **Concept**: CarJourney, Agency, RentalAgency,ReturnAgency,Manager,CarGroup
Ces concepts seront représentés dans la syntaxe fonctionnelle comme suit:

```
1 Declaration(Class(eu:CarJourney))
2 Declaration(Class(eu:Agency))
3 Declaration(Class(eu:RentalAgency))
4 Declaration(Class(eu:ReturnAgencys))
5 Declaration(Class(eu:Manager))
6 Declaration(Class(eu:Cargroup))
```

Propriétés des données : les propriétés qui ne concernent pas d'autre objet sont représentées par l'objet de propriété Data. Par exemple: CarName, CarID,...

```
1 Declaration(DataProperty(eu:has CarName))
2 DataPropertyDomain(eu:CarName eurent:CarJourney)
3 DataPropertyRange(eu:CarName xsd:string)
```

Propriété d'objet : les propriétés qui concernent un autre objet sont représentées par l'objet de propriété Data. Par exemple:

```
1 Declaration(ObjectProperty(eu:hasReturnAgency))
2 ObjectPropertyDomain(eu:hasReturnAgency eu:RentalAgency)
3 ObjectPropertyRange(eu:hasReturnAgency eu:ReturnAgency)
```

5.3.3 Règle d'ordre d'exécution

Dans cette section, nous présentons la méthode de construction de la règle à l'intérieur de l'ontologie de la règle de métier. Il existe cinq types de règles. Pour

chaque type de règle, nous créons un ensemble d'axiomes dans le BRO. Nous introduisons également quelques règles d'additionel pour permettre au raisonneur de raisonner sur BRO et BPO afin de détecter automatiquement les erreurs sémantiques potentielle.

5.3.3.1 Règle d'intégrité

La règle d'intégrité a la même signification avec une contrainte dans la base de données relationnelle. Dans le tableau 5.1, nous définissons les règles de cardinalité. Il sera traduit en un ensemble d'axiomes de cardinalité à l'intérieur de BRO.

Exemple de règle	OWL **and** $SWRL$
Something that owns at least 2 cars	ObjectMinCardinality(n R C)
Something that owns at most 2 cars	ObjectMaxCardinality(n R C)
Something that owns exactly 2 cars	ObjectExactCardinality(n R C)

Table 5.1: Règle d'intégrité

5.3.3.2 Règle de dérivation

Ce type de règle permet au système de déduire une nouvelle connaissance. Si un ensemble de faits satisfont la règle de dérivation, le raisonneur déduit un fait nouveau des faits existants. Nous utilisons la règle SWRL pour représenter ce type de règles. Ceci est l'avantage de l'ontologie. L'information est représentée dans une forme compréhensible pour la machine, de sorte que le système peut raisonner sur notre information dans le dut de faire une proposition à l'utilisateur.

Rule Example	OWL **and** $SWRL$
Platinum customers receive a 5 USD discount	PaltiumCustomer(w)→ hasDiscount(x,5)

Table 5.2: Règle de dérivation

5.3.3.3 Règle de réaction

L'une des règles importantes est la règle de réaction qui permet à l'utilisateur de définir la relation entre un ensemble d'actions dans un domaine spécifique. Nous proposons six types de relations : la dépendance, l'exécution parallèle, l'exécution des choix, l'exclusion séquentielle, l'exclusion parallèle et l'exclusion des choix.

Rule Example	*OWL* **and** *SWRL*
Task A is depended on task B	hasDependencyTask(B,A)TransitiveObjectProperty
Task A exclude task B in sequential	hasExSequentialTask(B,A)SymmetricObjectProperty
Task A exclude task B in parallel	hasExParallelTask(B,A)SymmetricObjectProperty
Task A exclude task B in choice	hasExChoiceTask(B,A) SymmetricObjectProperty
Task A executed in parallel with task B	hasParallelTask(B,A) SymmetricObjectProperty
Task A must be executed in choice with task B	hasChoiceTask(B,A) SymmetricObjectProperty

Table 5.3: Règle de réaction

5.3.3.4 Règle de production et règle de transformation

Ce type de règles de métier est représenté sous la forme "si quelque chose fait quelque chose". Nous représentons ce formulaire en utilisant la langage OWL et SWRL.

Rule Example	*OWL* **and** *SWRL*
If something do something	PaltiumCustomer(w)→ hasDiscount(x,5)

Table 5.4: Règle de production

5.4 Vérification de la conformité des processus de métier à l'aide de raisonnements

Dans la figure 5.1, nous présentons l'esquisse de notre solution. Les processus de métier (graphe CPN) sont représentés par un ensemble de concepts et individuels dans BPO. Les règles de métier sont créées et modifiées par un éditeur. Chaque règle est représentée par un ensemble d'axiomes et une règle SWRL à l'intérieur de BRO. Afin de vérifier la conformité des processus de métier avec les règles de

métier, nous fusionnons BRO et BPO en une ontologie ; deux concepts : Transition dans BPO et Task en BRO sont définie comme deux concepts d'équivalence. Le terme de métier peut être utilisé comme une couleur et un jeton dans le graphe CPN (processus de métier). Au cours de l'exécution d'un processus de métier, la valeur d'un individu peut être modifiée, mais elle doit respecter les contraintes de BRO (TBox et Properties). Au moment de la conception, lorsqu'un utilisateur définit un processus de métier, le terme de métier sera utilisé pour nommer un élément. Chaque individu de transition dans BPO est équivalent à un individu d'action dans BRO. En fonction des commandes de l'utilisateur, l'éditeur BPO générera un ensemble d'axiomes à l'intérieur de BPO. Par exemple, il existe deux tâches à l'intérieur de BRO, qui est défini que b dépend de a comme suit :

ObjectPropertyAssertion(:hasDependencyTask :b :a)

Cela signifie qu'il faut que a soit exécuté avant b, mais au moment de la conception, un utilisateur définit que a s'effectue après b et que la règle est générée comme suit :

ObjectPropertyAssertion(:hasDependencyTask :a :b)

Deux règles ci-dessus sont opposées, de sorte que l'ontologie fusionnée de BRO et BPO sera incohérente. Il peut être vérifié par un raisonneur. Étant donné que la propriété *hasDependencyTask* est définie comme *TransitiveObjecProperty* dans la règle de réaction 1 du tableau 5.3, alors au moment de l'exécution, nous utilisons la même approche pour vérifier la cohérence de l'ontologie fusionnée. Si un utilisateur modifie un processus de métier, la modification sera générée et insérée dans BPO ; pour chaque modification, le raisonnement vérifie la cohérence de l'ontologie fusionnée et notifie le résultat à l'utilisateur automatiquement. Dans le tableau 8.1, la liste des erreurs sémantiques potentielles pouvant être détectées par notre approche est présentée. Il permet au concepteur d'éviter l'erreur sémantique au moment du design. Cela permet de réduire le temps de construction d'un système et d'assurer l'exactitude de ce système au début du développement.

5.5 Conclusion

Dans ce chapitre, une approche ontologique pour détecter l'erreur sémantique potentielle du processus de métier et des règles de métier est proposée. Il faut des

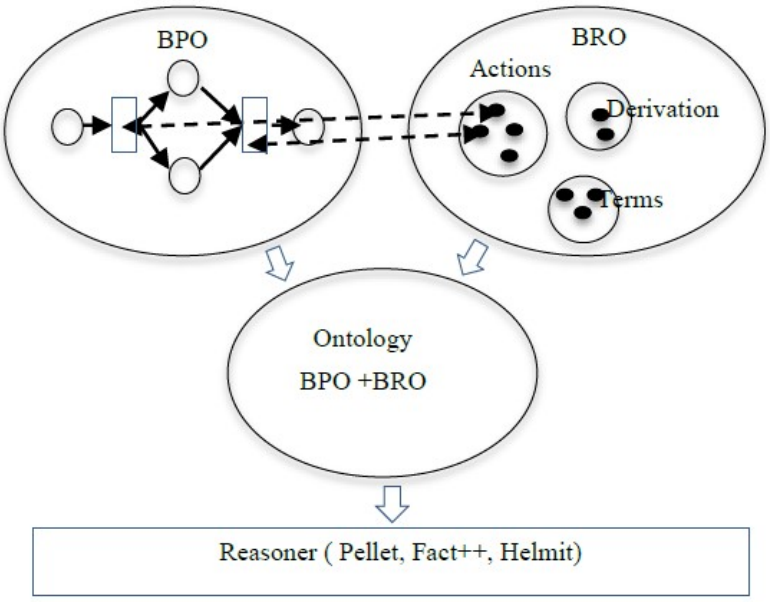

Figure 5.1: Intération de BRO et BPO

fonctionnalités importantes de l'ontologie qui sont les capacités de raisonnement, la possibilité d'exprimer des actions complexes et sa sémantique déclarative pour valider non seulement la cohérence des règles de métier et des processus de métier, mais aussi la conformité des processus de métier avec un ensemble de règles de métier. L'avantage de cette approche est de permettre au système de détecter les défauts du processus de métier automatiquement au moment du design et du temps d'exécution en utilisant les capacités de raisonnement de l'ontologie. Néanmoins, en utilisant cette approche, si BRO a de nombreux concepts et propriétés, le raisonnement peut prendre beaucoup de temps pour vérifier la cohérence de l'ontologie BPO et BRO. Pour cela, les travaux théoriques futurs comportent trois problèmes principaux. La première consiste à se concentrer sur le raisonnement distribué. La seconde est réalisée en sélectionnant la règle relative à une action pour la validation. Et le dernier objectif est de considérer l'exécution des processus de métier et de travailler avec une source de données. Le travail de ce chapitre a été publié dans quatre articles [PT16b], [PT15c], [PT15a] et [PT17].

Partie II

Réutilisation d'une couche de métier

Chapitre 6

Personnalisation de la couche de métier

Contenu

6.1 Introduction

Dans cette section, nous allons présenter notre approche pour permettre à l'utilisateur de personnaliser un patron de processus de métier afin que ce dernier corresponde avec son système d'information et ses exigences sans faire l'erreur de structure dans le processus de métier personnalisé. De nos jours, il existe un grand nombre de systèmes d'information utilisant une technologie à différence, qui a une base de données différente. Ce qui nous amène à nous poser les questions suivantes :

- comment modifier un modèle de processus de métier tout en gardant sa structure correcte ?
- comment personnaliser un processus de métier pour qu'il puisse être exécuté dans le système d'information de l'utilisateur ?

Pour répondre aux deux questions ci-dessus, cette section est divisée en sous-section comme suit :

Nous commençons cette section avec une introduction, puis nous présentons notre approche pour personnaliser la structure d'un processus de métier sans créer d'erreur de structure. Dans la sous-section suivante, nous présentons notre solution pour adapter le processus de métier à la source de données de l'utilisateur.

6.2 Processus de métier personnalisé

6.2.1 Personnalisation de la structure des processus de métier

6.2.1.1 Réseaux de Pétri d'état

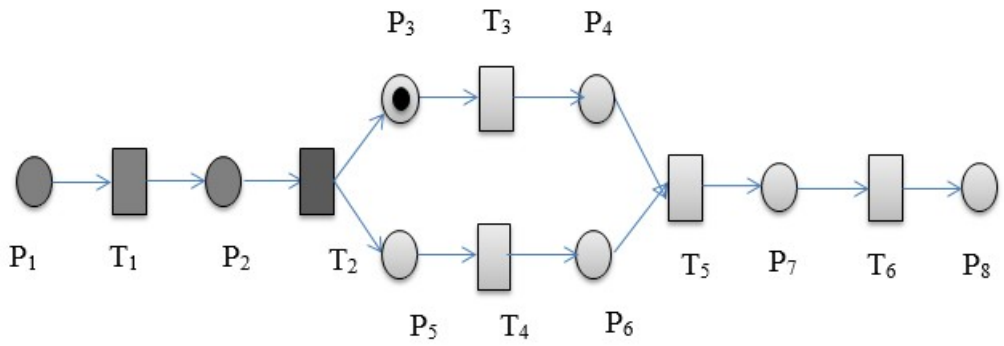

Figure 6.1: Réseaux de Pétri d'état

Dans cette partie, une extension de réseaux de Pétri pour décrire l'aspect de gestion de changement d'un système dans la syntaxe mathématique est introduite, cette méthode formelle est étendue à partir de réseaux de Pétri, appelée réseaux de Pétri d'état.

Definition 4 : un graphe de réseaux de Pétri d'état (State Petri net) est un graphe de six ensembles (P, T, A, S, C, s) où (P, T, A, w) est un graphe du réseau

de Pétri.

- C est l'ensemble fini de points modifiés dans un graphe de réseaux de Pétri.
- S est l'ensemble fini d'état de transition.
- S: T -> $t_i(j)$ est l'ensemble fini d'états d'une transition avec $ti \in T, j \in S, j \in N$

Il existe deux ensembles finis qui sont ajoutés dans la définition du graphe de réseaux de Pétri d'état. S est l'ensemble fini d'états de transition qui permet à l'utilisateur de représenter plusieurs états d'une transition. C est un ensemble fini de points modifiés dans un graphe de réseaux de Pétri, un point modifié est une opération XOR qui a toujours une valeur vraie. Ce type de nœud a été ajouté dans le graphe réseaux de Pétri pour modifier le flux de contrôle du graphe réseaux de Pétri sans affecter les données au moment de l'exécution. L'intention principale de l'ensemble du point C changé est d'aider les utilisateurs à contourner un nœud dans un système au moment de l'exécution.

Exemple 3 : il y a deux états d'une transition et l'ensemble des nœuds factices est vide. Considérons le graphe du réseaux de Pétri illustré à la figure. 6.1. Le réseaux de Pétri qu'il représente est spécifié par :

- P=$\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$
- T=$\{t_1, t_2, t_3, t_4, t_5, t_6\}$
- A=$\{(p_1, t_1), (p_2, t_2), (t_1, p_2), (t_2, p_3), (t_2, p_5), (p_3, t_3),$
  $(p_5, t_4), (t_3, p_4), (t_4, p_6), (p_4, t_5), (p_6, t_5), (t_5, p_7), (p_7, t_6), (t_6, p_8)\}$
- S=$\{0, 1\}$
- s:T=$\{t_1(1), t_2(1), t_3(0), t_4(0), t_5(0), t_6(0)\}$
- C =$\{\emptyset\}$

6.2.1.2 Opérations modifiées

Dans cette section, un ensemble d'opérations modifiées est proposé. Cet ensemble de modifications doit respecter les règles de métier prédéfinies dans l'ontologie des

règles de métier. Nous l'expliquerons plus en détail dans la prochaine section, afin de respecter l'ensemble des règles sémantiques prédéfinies et des règles syntaxiques dans l'étape de conception que nous avons mentionnées ci-dessus. Dans l'étape de personnalisation, nous fournissons certaines limitations générales suivantes :

- Une transition contrôlée ne peut pas être supprimée, les utilisateurs ne peuvent modifier que les conditions pour déclencher le jeton dans le nœud de contrôle.
- Une transition d'activité peut être supprimée et modifiée, mais la modification doit revoir les règles sémantiques prédéfinies et les règles syntaxiques.
- Pour ajouter un nouveau nœud, l'utilisateur peut seulement ajouter un nœud à chaque fois.

Pour les limitations ci-dessus, dans la figure 6.2, l'utilisateur ne peut pas supprimer la transition T_2 et T_5, car ce sont des nœuds de contrôle. Ils ne peuvent modifier que les conditions pour déclencher la transition contrôlée. On peut voir aussi dans la figure 6.3, quand nous insérons un nœud XOR toujours vrai, la transition B sera ignorée de manière logique. D'autre part, notre idée principale de construire l'ensemble des opérations modifiées permet de respecter la structure d'un modèle de processus de métier, lorsque nous conservons toujours la structure du flux de production, nous vérifions uniquement l'efficacité du flux de travail lors de l'opération d'ajout. Pour que la transition soit supprimée, divisée ou fusionnée, nous n'avons pas vraiment supprimé du flux de travail, il est uniquement bloqué (ou désactivé) et ne peut pas être déclenché tant que l'utilisateur n'a pas annulé sa modification.

Ajouter une nouvelle activité

Un nœud d'activité dans un processus de métier est une transition, un lieu et un arc entre eux dans le graphe du réseaux de Pétri. Il existe des cas suivants à considérer lorsque nous ajoutons un nouveau nœud à un processus de métier :

- **Ajout séquentiel** Cette opération permet aux utilisateurs d'insérer une nouvelle transition perméable dans un processus de métier. Cette transition sera exécutée avant ou après une transition existante dans un processus

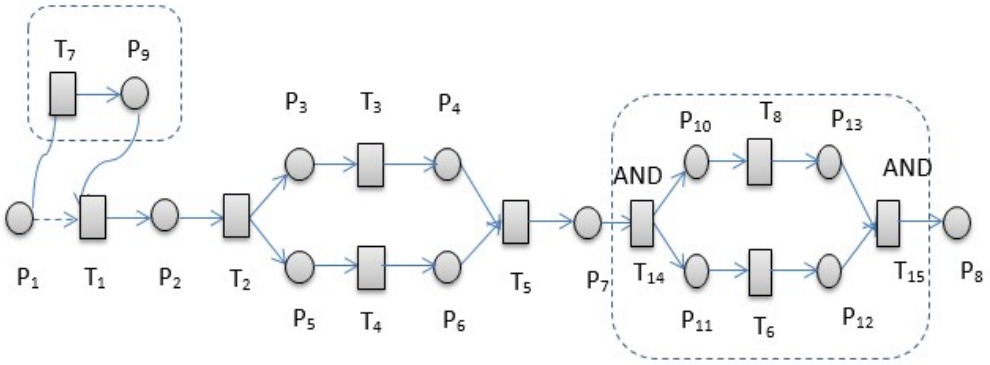

Figure 6.2: Ajout séquentiel et ajout parallèle

de métier. Selon le théorème 1, lorsque nous ajoutons une nouvelle transition t' dans un processus métier, le PN de réseaux de pétri de couleur PN, l'extension de PN est définie comme suit: $PN' = (P', T', F')$ avec $P' = P \cup \{p'\}, T' = T \cup \{t'\}, F = F' \cup \{(p', t')$ or $(t', p')\}$. Nous pouvons également insérer un ensemble de transitions séquentielles dans un processus opérationnel correct sans provoquer d'erreur syntaxique. Dans la figure 6.2, les transitions T_7 et P_9 sont insérées dans le flux de travail avant la transition T_1.

- **Ajout parallèle** Les utilisateurs peuvent utiliser cette opération pour insérer une nouvelle transition perméable dans un processus de métier. Il est exécuté en parallèle avec une transition existante dans le processus de métier. Dans la figure 6.2, nous insérons la transition T_8 pour exécuter en parallèle avec T_6. Cette opération insère dans le processus de métier une paire de blocs de construction AND-Split, AND-join et une transition amorçable. Pour la définition 4, PN' a encore deux places spéciales i et o, et il est fortement connecté. Nous ajoutons AND-split pour permettre à T_6 et T_8 et au AND-join de synchroniser les deux flux parallèles.

  Dans la figure 6.2, la transition T_8 est insérée dans le processus de métier avec deux blocs de construction XOR-Split et XOR-Join. Comme XOR-Split permet T_8 ou T_6, seul un d'entre eux sera exécuté et dépend du résultat de la vérification de la condition de XOR-Split. Le flux de travail étendu 'a une "bonne" structure.

Supprime un nœud

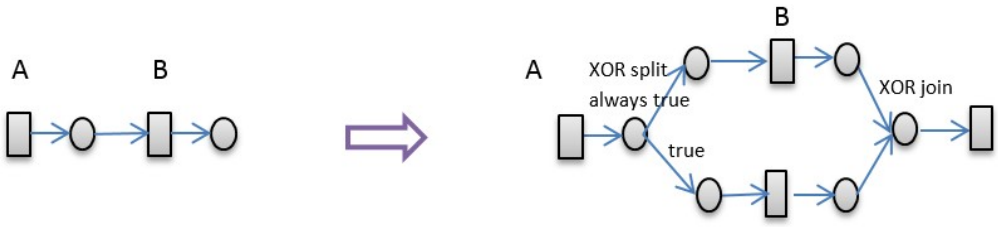

Figure 6.3: Supprimer temporairement un nœud

L'idée principale de cette opération est que nous ne supprimons pas vraiment une transition dans le processus de métier. Une paire de blocs de construction spéciaux XOR-Split et XOR-Join est utilisée. La condition de XOR-Split est toujours vraie. Par conséquent, le XOR-Split permet toujours une transition factice. En fait, cette opération nous aide à passer une transition. Au lieu d'exécuter une transition qui doit être supprimée, la transition fictive (la transition ne fait rien) sera exécutée. Le jeton sera transféré directement à la prochaine transition. D'autre part, selon le théorème 1, le flux de travail modifié est bien structuré et n'a pas d'erreur syntaxique, sauf en cas d'un ensemble de transition qui contient le bloc de construction AND-Split/Join, XOR-Split/Join, ou une structure itérative. Afin d'éviter l'erreur syntaxique, la partie sans être supprimé doit commencer avant le point de départ d'un bloc de construction ou de la boucle, et s'arrêter après le point final de celles-ci. Dans la figure 6.3, une paire de blocs de construction spéciaux XOR-Split et XOR-Join et la transition factice T_8 sont ajoutés au processus métier. La transition T_6 est temporairement supprimée. Cela ne se fera pas parce que la division XOR est toujours vraie. Le jeton sera transféré de P_7 à P_8 directement. Cette opération permet également aux utilisateurs de revenir à l'étape précédente et nous l'expliquerons plus en détail dans la section 4. Un avantage de cette solution est de ne pas influer sur la structure du processus métier d'origine. Cela nous permet d'éviter l'erreur syntaxique autant que possible pendant la modification.

Diviser un nœud

Cette opération permet aux utilisateurs de diviser une tâche en plusieurs tâches. En fait, l'opération de fractionnement est la somme de l'opération de suppression et de l'opération d'ajout. En général, lorsqu'un utilisateur veut diviser un nœud, ce dernier sera ignoré automatiquement par une opération de suppression. Après cela, selon le type de fractionnement, le système ajoutera de nouveaux nœuds. Le nœud divisé n'est pas vraiment supprimé du processus métier. Il est seulement sauté, il peut être réactivé si l'utilisateur souhaite revenir à l'étape précédente. Il existe les types suivants d'opérations divisées :

- **Split séquentiel**

  La formule de cette opération :

  Split séquentiel ≡ Supprimer le nœud divisé + Ajout séquentiel

  La figure 6.4 est un exemple de split séquentielle d'un nœud. Dans cet exemple, la transition B est divisée en deux transitions C et D. La division B n'est pas vraiment supprimée B du processus métier. Il est uniquement bloqué par un nœud de contrôle OR qui est toujours vrai.

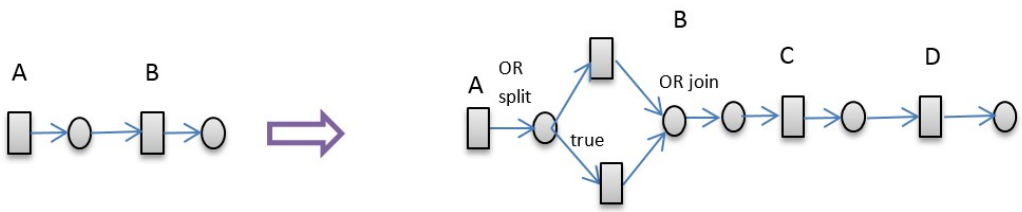

Figure 6.4: Split séquentiel

- **Split parallèle**

  AND split ≡ Supprimer le nœud divisé + Ajout séquentielle

  Pour cette opération, nous ignorons également la transition fractionnée par un opérateur OR et nous utilisons l'opérateur AND pour que les nouvelles transitions soient exécutées en parallèle.

- **OR split**

  La formule de cette opération :

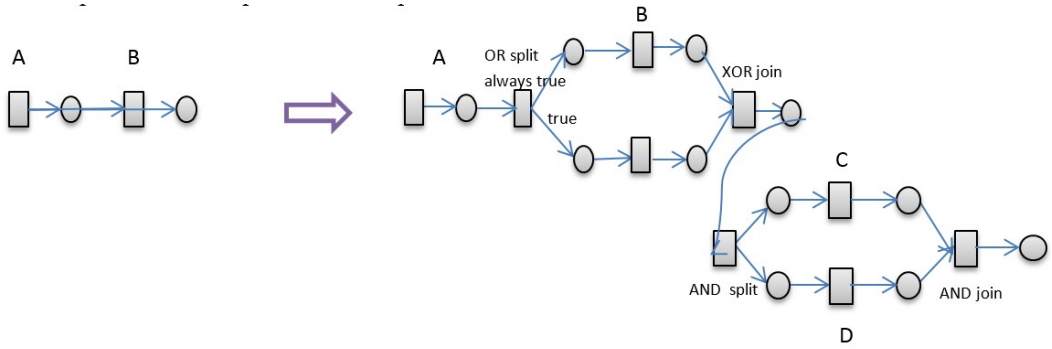

Figure 6.5: Split parallèle une transition

OR split ≡ *Remove splitted node* + *OR insert*

Fusionner un ensemble de nœuds

Cette opération est une opération opposée à l'opération fractionnée. Nous allons ignorer un ensemble de transitions qui doivent être fusionnées et insérer une nouvelle transition vers le processus de métier. Afin d'éviter l'erreur syntaxique en cas de saut d'un ensemble de transitions qui contient le bloc de construction AND-Split / Join, XOR-Split / Join, ou la structure itérative, la partie sans être supprimé doit être démarrée avant le point de départ d'un bloc de construction ou la boucle, et arrêter après le point final de celui-ci.

La formule de cette opération :

Fusion séquentielle ≡ *Supprimer les noeuds fusionnés* + *Ajout séquentiel*

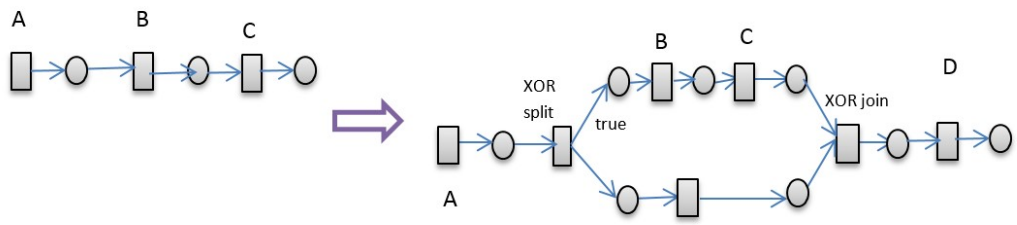

Figure 6.6: Fusion séquentielle

Dans l'exemple de la figure 6.6, nous fusionnons la transition B et C dans la transition D.

Changement de contenu

Dans cette section, nous considérons l'opération de changement de contenu. Cela signifie qu'il n'y a pas de partie ajoutée ou de partie qui saute dans le processus de métier.

Modifications de contenu

Nous pouvons classer les types de transition. Il existe deux types de transition : transition d'activité et transition de contrôle.

- **Transition d'activité**

  Ce type de transition exécute une tâche spécifique. Ainsi, les utilisateurs peuvent modifier leur contenu afin d'avoir la tâche qu'ils souhaitent.

- **Transition de contrôle**

  Ce type de transition contient la condition pour prendre la décision de contrôle de flux. Cette condition peut être modifiée par cette opération.

Ajustement de position L'utilisateur peut ajuster la position d'une transition d'activité, mais il n'a pas la permission d'ajuster la position de la transition de contrôle, car l'erreur syntaxique se produirait. Par exemple, XOR-join ne peut pas être exécuté avant XOR-split.

Parallélisation

Cette opération permet à l'utilisateur de définir deux transitions à exécuter en parallèle. Cette opération ne provoque pas l'erreur syntaxique.

De-Parallélisation

C'est l'opération inverse de l'opération de parallélisation. Nous modifions l'ordre d'exécution en deux transitions de la parallélisation à la séquence.

Itération d'un nœud ou d'un ensemble de nœuds

Cette opération peut être effectuée si elle fournit une condition pour quitter la boucle. Il existe deux types de conditions :

- Un nombre d'itérations prédéfini : la boucle sera exécutée n fois (n est le nombre d'itérations).

- Un déclencheur et un délai d'attente.

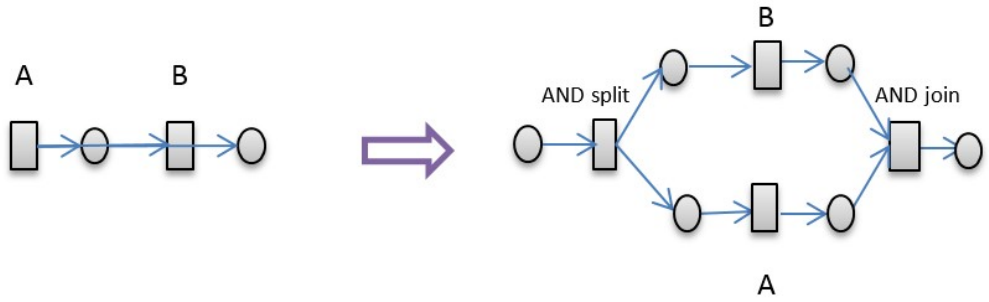

Figure 6.7: Parallélisation

6.3 Adaptation du processus de métier à la source de données

6.3.1 Génération de l'ontologie correspondante pour une base de données relationnelle

Cette section présente notre approche pour mapper les constructions d'une base de données relationnelle à une ontologie, le nom des tables, des colonnes, des lignes de la base de données relationnelle sont utilisés comme noms de concept, propriété,... de l'ontologie. Afin de mapper la construction d'une base de données relationnelle à une ontologie, nous définissons les règles suivantes pour garantir que les sémantiques de la base de données relationnelles sont complètement traduites dans une ontologie.

- Règle d'alignement de table
- Règle d'alignement des colonnes
- Règle d'alignement de type de données

Nous continuons l'exemple 1 sur le site web d'émission de télévision. La figure 6.8 montre le diagramme de base de données de site web de film définie par nous. Nous utiliserons cet exemple pour montrer notre algorithme d'alignement.

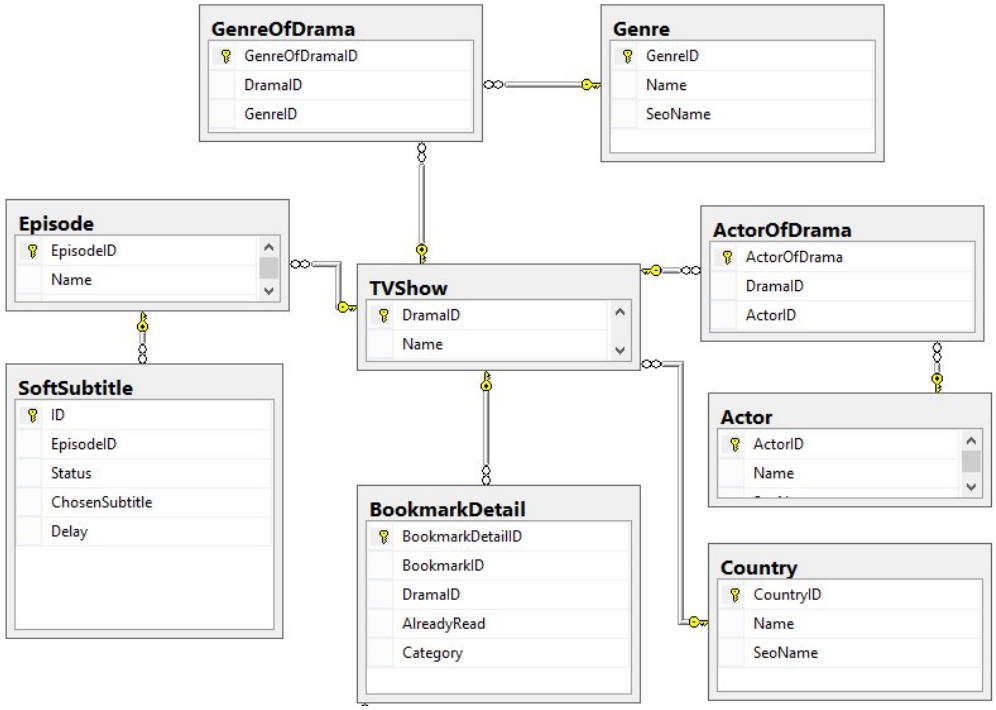

Figure 6.8: Base de données du site web de l'émission de télévision

6.3.1.1 Règles d'alignement de tableaux

Une table est représentée par une classe dans l'ontologie. Les clés étrangères de la table sont mappées aux propriétés de l'objet dans l'ontologie.

Le tableau *TVShow* de la figure 6.8 comporte une clé étrangère À la table Country, indiquant une relation binaire de plusieurs à plusieurs. Cette clé étrangère est mappée aux propriétés de l'objet : *FilmOfCountry* (Qui utilise les classes *Country* et *TVShow* comme domaine et gamme, respectivement). Ce dernier est un inverse du premier, ce qui signifie que la relation est bidirectionnelle (c'est-à-dire un *TVShow* a été réalisé par une équipe dans plusieurs pays). Les règles seront illustrées par l'exemple du site de film.

```
1 CREATE TABLE TVShow(CountryID INTEGER REFERENCES Country)
```

Listing 6.1: Règles de cartographie de tableaux

OWL functional syntax

```
1 Declaration(Class(db:TVShow))
2 Declaration(ObjectProperty(db:CountryID))
3 ObjectPropertyDomain(db:CountryID db:TVShow)
```

```
4 ObjectPropertyRange(db:CountryID db:Country)
```

6.3.1.2 Règles de cartographie de colonnes

Les colonnes d'une table sont représentées par un ensemble de propriétés de type de données du concept de table. Dans la Figure 6.8, la colonne *Name* dans une table *TVShow* est une colonne normale. Ainsi, cette colonne est représentée par une propriété de type de données *TVShowName* qui a *TVShow* en tant que domaine et la valeur de cette propriété est mappée en utilisant la règle de mappage de données.

```
1 Declaration(DataProperty(db:TVShowName))
2 DataPropertyDomain(db:TVShowName db:TVShow)
3 DataPropertyRange(db:TVShowName xsd:string)
```

Comme chaque colonne dans la rangée d'une table n'a qu'une seule valeur, ou pas de valeur, alors dans l'ontologie, nous devons définir la propriété de données avec une valeur maximale, ce problème sera représenté par l'axiome suivant.

```
1 SubClassOf(db:TVShow DataExactCardinality(1 db:TVShowName))
```

6.3.1.3 Règles de cartographie de type de donnés

Comme mentionné ci-dessus, lors de la représentation d'une colonne de table, nous devons mapper le type de données de cette colonne au type de données XSD. Dans le tableau 6.1, nous présentons plus en détail le mappage entre deux types de données.

Colonne *Name* dans la figure 6.8, qui a le type de données CHARACTER. Par conséquent, un type de données propriéte *TVShowName* utilise la chaîne comme étant sa gamme. Le type de données CHARACTER sera traduit en XML type de données xsd:string comme suit :

```
1
2 Declaration(DataProperty(db:TVShowName))
3 DataPropertyDomain(db:TVShowName db:TVShow)
4 DataPropertyRange(db:TVShowName xsd:string)
```

6.3.1.4 Règles de cartographie de PRIMARY KEY

Il existe deux types de clés primaires, la clé avec juste une colonne et la clé avec plusieurs colonnes. De la même façon que le mappage d'une colonne normale,

type de donnés de SQL	Type de donnés de XML Schema
SMALLINT	short
INTEGER	integer positiveInteger negativeInteger nonPositiveInteger nonNegativeInteger int long
DECIMAL	decimal
NUMERIC	decimal
FLOAT	float
REAL	float
DOUBLE PRECISION	double
CHARACTER	string
CHARACTER VARYING	decimal
TIME	time
TIME WITH TIME ZONE	time
DATE	date
TIMESTAMP	datetime
TIMESTAMP WITH TIME	datetime
CHARACTER VARYING	decimal
INTERVAL	duration
BIT	boolean
BIT VARYING	byte

Table 6.1: Cartographie du type de données

la colonne de clé sera représentée par une propriété de données. Mais ce type de colonne doit avoir une valeur exacte, et c'est aussi une propriété fonctionnelle inverse.

Dans la figure 6.8, dans le tableau *TVShow*, la colonne *DramaID* a le type de données int, et c'est la clé principale de table *TVShow*, de sorte que le mappage de cette colonne est représenté comme suit :

```
1 Declaration(DataProperty(db:DramaID))
2 DataPropertyDomain(db:DramaID db:TVShow)
3 DataPropertyRange(db:DramaID xsd:nonNegativeInteger)
4 InverseFunctionalObjectProperty(db:DramaID}
5 SubClassOf(db:TVShow DataMinCardinality( 1 db:hasDramaID))
```

Listing 6.2: Cartographie de constraintes PRIMARY KEY

6.3.1.5 Règles de cartographie de FOREIGN KEY

Une colonne qui est une clé étrangère peut être représentée par une propriété d'objet. Dans la figure 6.8, on peut voir que le colonne *CountryID* dans table *TVShow* est une clé étrangère qui sert à mapper deux tables *TVShow* et *Country*. Cette relation est représentée par une propriété d'objet comme suit :

```
1
2 Declaration(ObjectProperty(db:CountryID))
3 ObjectPropertyDomain(db:CountryID db:TVShow)
4 ObjectPropertyRange(db:CountryID db:Country)
```

Listing 6.3: Cartographie de constraintes FOREIGN KEY

6.3.1.6 Algorithme de génération d'ontologie de base de données relationnelle

L'algorithme 1 présente l'idée principale de notre approche pour générer une ontologie de base de données relationnelle qui est représentée pour une base de données relationnelle. Chaque table dans la base de données est représentée par un concept dans l'ontologie. La colonne est représentée par une propriété d'objet avec la colonne de clé étrangère et la propriété de données pour une colonne normale. Avec la colonne de la clé primaire, elle sera représentée par une propriété de données.

Nous générons également un autre ensemble d'axiomes pour assurer les contraintes de la base de données.

6.3.2 Réécriture de requêtes

6.3.2.1 Faire des alignements de RDO et BRO

Afin d'interroger les données à partir de la base de données relationnelle, nous devons créer l'alignement entre l'ontologie des règles de métier, l'ontologie des processus métier avec l'ontologie de la base de données relationnelle. Il existe plusieurs systèmes et API existants. Avec notre solution, nous choisissons l'API de l'équipe INRIA Grenoble pour développer la fonction d'appariement. Cette fonction est utilisée dans l'algorithme de recommandation de requête. La sortie de la fonction ontologie correspondante est un ensemble d'alignements entre deux ontologies. Chaque alignement a un indice de similarité qui a une valeur entre 0 et 1, si un alignement a l'indice de similarité égal à 1, cela signifie que deux concepts dans l'alignement ont la même signification.

6.3.2.2 Recommandation de requête

À partir de l'alignement de l'ontologie, nous proposons une méthode pour recommander une requête possible pour interagir avec la base de données de l'utilisateur à partir de la requête de démarrage dans le processus métier. À chaque transition dans un processus de métier (graphe CPN), il a sa propre requête de tir qui est utilisée pour décrire l'activité principale de la transition. La requête de démarrage a été définie au chapitre 4. Pour que le modèle de processus métier soit exécutable sur le système de l'utilisateur, la requête de démarrage doit être traduite dans la requête correspondante avec la source de données de l'utilisateur.

En fonction de l'indice de similarité de chaque alignement, nous calculerons l'indice d'exécution possible de chaque *Firing Query*. Si toutes les *Firing Query* à l'intérieur du processus de métier ont l'indice d'exécution possible égal à 1, cela signifie que le modèle de processus de métier peut être exécuté immédiatement sur le système d'information de l'utilisateur.

Dans l'algorithme 2, nous allons prendre toutes les terminologies à partir de la *Firing Query* de chaque transition. Ensuite, nous le rechercherons dans un ensemble

Algorithm 1: Cartographie d'une base de données relationnelle à une ontologie OWL 2

input : A relational database with a database diagram

output: An corresponded OWL 2 ontology

```
1  Read the database diagram from the relational database;
2  for i ⟶ numberOfTable do
3      DeclareConcepts (table[i].name);
4      for j ⟶ numberOfColums do
5          if colum is not a primary key then
6              if colum is a foreign key then
7                  Declaration(ObjectProperty(db:ColumName))
                    ObjectPropertyDomain(db:ColumName db:TableName)
8                  if Not Exist RangeTable then
9                      DeclareConcepts (RangeTable);
10                 end
11                 ObjectPropertyRange(db:ColumName db:RangeTable)
12             end
13             else
14                 Declaration(DataProperty(db:ColumName))
                    DataPropertyDomain(db:ColumName db:TableName)
                    DataPropertyRange(db:ColumName xsd:datatype)
                    SubClassOf(db:TableName DataExactCardinality(1
                    db:ColumName))
15             end
16         end
17         else
18             Declaration(DataProperty(db:ColumName))
                DataPropertyDomain(db:ColumName db:TableName)
                DataPropertyRange(db:ColumName xsd:nonNegativeInteger)
                InverseFunctionalObjectProperty(db:ColumName)
                SubClassOf(db:TableName DataMinCardinality( 1
                db:ColumName))
19         end
20     end
21 end
```

d'alignements entre l'ontologie des règles de métier et l'ontologie de la base de données relationnelle. S'il existe dans les alignements, nous prendrons l'indice de similarité de l'alignement qui contient la terminologie du modèle. Dans le cas où l'indice de similarité est égal à 1, l'algorithme choisira la terminologie de l'utilisateur immédiatement et générera la requête recommandée pour informer l'utilisateur. Dans le cas où l'indice de similarité est supérieur à 0,5. Nous choisirons les terminologies de l'utilisateur pour générer la requête recommandée.

Si de nombreux alignements contiennent la terminologie de la *Firing Query*. Nous choisirons l'alignement dont l'indice de similarité est supérieur ou égal à 0,5 pour générer la requête recommandée à l'utilisateur. L'utilisateur doit choisir la bonne requête qui correspond à son système. Si tous les alignements ont un indice de similarité égal à 1, le modèle de processus métier peut fonctionner immédiatement sur le système de l'utilisateur sans aucune modification.

6.4 Simulation d'application basée sur ECA

Dans cette section, nous proposons une approche basée sur *Event Condition Action* (ECA) pour construire un modèle d'exécution d'application dont l'idée principale est de traduire les processus de métier et les règles de métier en une base de connaissances. Il s'agit d'un ensemble de règles ECA. Comme les deux utilisent un format de règle, le processus de métier doit donc suivre la règle dans la base de connaissances de la règle de métier.

6.4.1 Modèle de processus métier basé sur la ECA

Example 1: Dans la figure 6.9, nous concevons un graphe CPN pour représenter les opérations du compte bancaire.

```
1 color Account = int with 1..1000;
2 color Balance = int;
3 color Amount = int with 1..5000;
4 color AB = product Account * Balance;
5 color AA = product Account * Amount;
6 var a:Account; var x:Amount;
7 var y:Balance;
```

Algorithm 2: Algorithme de recommandation des requêtes

input : A set of alignments: α

A business process template: CPN

output: A set of recommended query:ϱ

```
1  a=(template − terminology, user − terminology,similarityindex), ∀ a ∈
   α;
2  foreach transition t of CPN do
3      Q ← GetTerminologyInQuery(t.firingquery)
4      foreach q in Q do
5          foreach a in α do
6              if q == a.template − terminology then
7                  if a.similarityindex==1 then
8                      t.firingquery.replaceTerminology(a.template −
                       terminology,a.user − terminology);
9                  end
10                 else
11                     if a.similarityindex>=0.5 then
12                         SaveToPropose(a.user − terminology);
                           t.firingquery.replaceTerminology(a.template −
                           terminology,a.user − terminology);
13                     end
14                 end
15             end
16         end
17     end
18 end
```

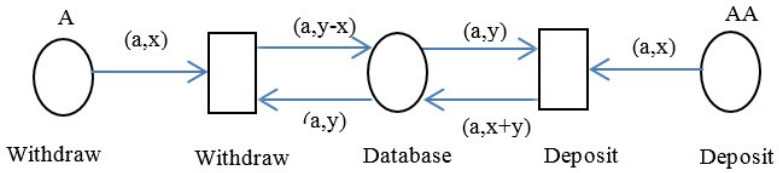

Figure 6.9: Opérations du compte bancaire

6.4.1.1 Langage intermédiaire de processus métier basé sur ECA

Nous commençons cette section en introduisant de manière informelle les différentes constructions du langage. Notre solution s'inspire des travaux de [ABB06]. Dans cette solution, nous visons à définir un langage présentant à la fois les avantages des langages ECA et de la programmation logique[ET06]. En tant que telles, les expressions dans ECAE sont divisées en deux parties :

- **Règles :** règle réactive, règle d'inférence et règles d'inhibition.
- **Définitions :** objet, événement et action

Les règles réactives sont les même dans les langages ECA et ont la forme suivante : l'événement est un événement de base ou complexe exprimé en algèbre ; la condition est une conjonction de littéraux (positive ou négative) et l'action est une action de base ou complexe. Les règles d'inférence sont des règles de programmation logique avec une négation par défaut, où les têtes négatives par défaut sont autorisées. Enfin, ECAE comprend également les règles d'inhibition de la forme :

If Condition ***Do Not*** *Action*
If Condition ***Do*** *Action*

Lorsque la condition est une conjonction de littératures et d'événements, une telle expression signifie intuitivement : si la *Condition* est vraie, n'exécutez pas action. Les règles d'inhibition sont utiles pour mettre à jour le comportement des règles réactives. Si la règle d'inhibition ci-dessus est affirmée, toutes les règles avec action dans la tête sont mises à jour avec la condition supplémentaire qui ne doit pas être satisfaite pour exécuter l'action.

L'ECAE permet de combiner les événements de base pour obtenir des complexes en utilisant l'algèbre d'événements. Les opérateurs que nous utilisons sont : $\wedge | \vee | S |$

NOT. Intuitivement, $e_1 \wedge e_2$ se produit à un instant i si e_1 et e_2 se produisent à i; $e_1 \vee e_2$ se produit à l'instant i si e_1 ou e_2 se produisent à l'instant i; *NOT e* se produit à l'instant i si e ne se produit pas i. *S(e_1, e_2, e_3)* se produit au même instant de e_3, dans le cas où e_1 a eu lieu avant, et e_2 au milieu. L'opérateur S est très important car il permet de combiner et de raisonner avec des événements se produisant à différents moments.

Les actions peuvent être basiques ou complexes. Les actions externes de base sont liées à l'application spécifique du langage. Les actions internes de base sont destinées à ajouter ou à rétracter des faits et des règles (règles d'inférence, réactives ou d'inhibition), de la forme *assert(tau)* et *retract(τ)* respectivement, pour augmenter les événements de base, E). Il existe également une action interne définie (d) pour ajouter de nouvelles définitions d'actions et d'événements (voir plus sur ces définitions ci-dessous).

Des actions complexes sont obtenues en appliquant des opérateurs algébriques sur des actions de base. Ces opérateurs sont : $\Rightarrow$ / $\Updownarrow$ / *IF*, le premier pour l'exécution séquentielle des actions et la seconde pour les exécuter simultanément. L'exécution de *IF (C, a_1, a_2)* équivaut à l'exécution a_1 dans le cas où C est vrai, sinon ou en exécutant a_2.

Pour permettre une définition modulaire des actions et des événements complexes, ECAE autorise des expressions de définition d'événements et d'actions. Ceux-ci sont de la forme e_{def} est un événement et a_{def} est une action où e_{def} (resp. $a_{def}s$) est un atome représentant un nouveau événement et e (resp. a) est un événement (resp. action) obtenu par l'événement (resp. action) algèbre ci-dessus. Il est également possible d'utiliser des événements définis (resp. actions) dans la définition d'autres événements (resp. actions).

Comme mentionné, un processus de métier est représenté par un graphe CPN; L'idée de notre solution est de traduire un graphe CPN sur un ensemble de règles ECAE. Nous proposons un algorithme pour la traduction CPN-ECAE : Les règles de l'ECAE sont traduites à partir d'un modèle de processus métier basé sur un Réseau Petri Coloré utilisé pour réaliser l'exécution des processus métier. L'algorithme de traduction comporte 4 étapes comme suit :

Example 4 : Le graphe CPN de l'exemple 1 sera traduit en un ensemble de règles ECAE.

Algorithm 3: ECAE Translation Algorithm

input : A CPN graph

output: A set of ECAE rules

1 The condition part of ECAE reactive rule is a collection of color sets, guard function related to a transition.

2 Translate each transition to ECAE rule.

3 Add starting condition and ending condition.

4 Connect all ECAE rule transition as their triggered sequence.

```
1  BPR1: If Withdraw and AA(a,x)Do Withdraw  and AB(a,y-x)
2
3  BPR2:On Withdraw and AB(a,y-x) If Done Do AB(a,y)
4
5  BPR3: If Deposit and AA(a,x) Do Deposit and AB(a,y+x)
6
7  BPR4:On Deposit and AB(a,y+x) If Done Do AB(a,y)
8
9  BPR5:On AB(a,y) If Done Do EndWorkflow
10
11 BPR6: If Account>5000 and Account<0 Do EndWorkflow
12
13 BPR7: If Amount>1000 and Amount<0 Do EndWorkflow
```

Dans l'exemple 4, R_1 et R_3 sont les règles pour commencer le processus de métier pour deux cas, retirer et déposer, respectivement. R5 est la règle pour abandonner le processus de métier.

6.4.1.2 Langage intermédiaire de règle métier basé sur ECA

L'un des principaux objectifs de ECAE est de créer un ensemble de règles de métiers. Lorsqu'un processus métier est exécuté, il doit respecter un ensemble de règles de métier qui se compose de deux parties :

- **Définitions :** cette partie contient toutes les définitions d'actions, d'événements et de couleurs définies dans un domaine spécifique.
- **Règles d'inhibition :** cette partie consiste en un ensemble de règles d'inhibition qui sont utiles pour mettre à jour le comportement des règles

réactives.

Exemple 5 : nous étendons l'exemple 4 en ajoutant certaines actions et des règles d'inhibition simples.

```
1
2 BRR1: Withdraw(a,x) is Login(user,pass) Amount(a,y−x)
3
4 BRR2: Deposit(a,x) is Login(user,pass) Amount(a,y+x)
5
6 BRR3: If y−x<0 Do Not Withdraw(a,x)
7
8 BRR4: If Not Login(user,pass) Do EndWorkflow
```

Lorsque ces règles d'inhibition sont intégrées à l'ensemble des règles ECAE dans l'exemple 5, le solde du compte bancaire ne sera jamais négatif. Nous pouvons utiliser ECAE pour définir cette règle de métier plus complexe.

6.4.2 Intégration du processus métier et des règles de métiers

Cette section présente notre méthode d'intégration et de vérification d'un processus métier et de règles de métier. Comme présenté ci-dessus, l'ensemble des règles de métier et des processus métier sont représentés par la langage ECAE. Par conséquent, afin de vérifier la conformité entre eux, nous fusionnons deux ensembles de règles ECAE dans une base de connaissances unique. Continuons notre exemple 5. Nous avons une base de connaissances comme suit :

```
1
2 BPR1: If Withdraw and AA(a,x)Do Withdraw and AB(a,y−x)
3
4 BPR2:On Withdraw and AB(a,y−x) If Done Do AB(a,y)
5
6 BPR3: If Deposit and AA(a,x) Do Deposit and AB(a,y+x)
7
8 BPR4:On Deposit and AB(a,y+x) If Done Do AB(a,y)
9
10 BPR5:On AB(a,y) If Done Do EndWorkflow
11
12 BPR6: If Account>5000 and Account<0 Do EndWorkflow
13
14 BPR7: If Amount>1000 and Amount<0 Do EndWorkflow
```

```
15
16 BRR1:Withdraw(a,x) is Login(user,pass) Amount(a,y-x)
17
18 BRR2:Deposit(a,x) is Login(user,pass) Amount(a,y+x)
19
20 BRR3:If y-x<0 Do Not Withdraw(a,x)
21
22 BRR4:If Not Login(user,pass) Do EndWorkflow
```

Nous pouvons voir que le processus métier et les règles de métier sont représentés dans la syntaxe ECAE (c'est un ensemble de règles). Par conséquent, nous pouvons facilement vérifier la conformité des processus métier avec un ensemble de règles de métier en détectant le conflit entre les règles dans une base de connaissances en utilisant le raisonnement.

6.5 Conclusion

Dans ce chapitre, nous proposons une approche basée sur les réseaux de Pétri colorés afin d'éviter les erreurs syntaxiques lorsque l'utilisateur modifie un processus métier. Un ensemble d'opérations modifiées est proposé. L'idée principale se base sur l'ajout de l'opération et l'opération de saut. Dans le cas où nous devons supprimer une transition ou un ensemble de transitions, nous ne l'enlevons pas vraiment, elle sera bloqué par une paire de blocs spéciaux XOR-Split/XOR-Join. La condition de XOR-Split est toujours vraie. Par conséquent, le XOR-Split permet toujours une transition factice au lieu d'exécuter la transition qui doit être supprimée. Lorsque l'utilisateur exécute des processus métier modifiés, la transition bloquée ne sera pas exécutée. Toutes les opérations sont basées sur cette idée, par exemple l'opération de fractionnement, l'opération de fusion. Une transition ou un ensemble de transitions qui doivent être divisées ou fusionnées sera ignoré et nous utilisons également l'opération d'ajout pour insérer de nouvelles transitions dans le processus métier. L'un des avantages de cette approche est que la modification n'affecte pas la structure du flux de travail original, nous pouvons donc prévenir l'erreur syntaxique dans certains cas. D'autre part, en raison de l'utilisation d'un ensemble d'opérations prédéfinies, il est plus facile de fournir la limite pour justifier l'exactitude syntaxique d'un processus métier modifié. Un autre avantage de cette solution est d'aider les utilisateurs à gérer la modification d'un processus métier

par un ensemble de booléens. Pour modifier l'état des processus métier, il suffit de définir la valeur d'un élément dans le tableau de vrai à faux, ou de faux à vrai. Cela peut réduire la complexité de notre algorithme et permettre au système de gérer de nombreux changements sur un processus métier. Dans ce chapitre, nous présentons également notre nouvelle approche pour permettre à l'utilisateur d'adapter une couche de métier à ses besoins et à son système d'information, y compris la base de données et la structure de processus métier.

Partie III

Implémentation et évaluation

Chapitre 7

Implementation

Contenu

7.1 Introduction

Dans ce chapitre, nous présentons la mise en œuvre de notre recherche. Cette implémentation est une combinaison des aspects théoriques. Chaque question de recherche sera solutionné par chaque fonction du logiciel OntoApp. Logiciel OntoApp fournit un éditeur de processus de métier (un graphe de CPN) d'une application. Cette fonction peut vérifier la consistance d'un processus de métier et vérifier aussi la conformité d'un processus de métier avec un ensemble de règles de métier. Chaque transition dans un processus de métier peut fonctionner avec la source de données grâce à l'accès aux données et à la fonction de requête recommandée. Nous allons entrer plus en détailé dans la sous-section suivante.

7.2 Ontology-Based Application Simulation : OntoApp

7.2.1 Résumé des fonctionnalités

OntoApp possède les fonctionnalités suivantes :

- **OntoApp : Business Process Editor** : Cette fonction est un éditeur graphique qui permet à l'utilisateur de concevoir un processus de métier

en utilisant l'icône réseaux de Pétri coloré. Pour développer l'interface de cette fonctionnalité, nous utilisons une source ouverte appelée *PNEditor* pour sauvegarder le temps de développement. Un processus de métier sera validé avant de sauvegarder dans une ontologie que nous appelons l'ontologie des processus de métier (BPO) comme mentionné dans le chapitre précédent. La figure 7.2 est l'interface de l'éditeur de processus de métier.

Cette fonction est le premièr interface d'utilisateur. Dans la figure 7.2, on peut choisir l'autre fonction par le menu : *Query Management, Colors Management, Ontology Domain, Business Rule Editor, Configuration.*

- ***OntoApp : Business Rule Editor*** **:** Cette fonction est un éditeur de texte qui sert à déclarer l'ensemble des règles de métier d'une application. Selon le domaine d'application, la règle sera construite à partir d'un ensemble de terminologie définie dans une ontologie de domaine. La figure 7.3 est l'interface de cette fonction. L'utilisateur peut ajouter chaque règle de métier. Avec chaque nouvelle règle, l'utilisateur peut valider et vérifier la cohérence d'ensemble de règle de métier.

- ***OntoApp : Configuration*** **:** Cette fonction consiste à se connecter à une source de données relationnelle et à générer une ontologie à partir de la base de données. Il s'agit de la mise en œuvre du chapitre 6. La figure 7.1 est l'interface de la configuration. L'utilisateur peut saisir l'adresse du server, et le compte pour connecter à la base de données.

- ***Query Validator*** **:** Cette fonction génère l'alignement de l'ontologie entre l'ontologie de la base de données relationnelle et la terminologie dans l'ontologie du domaine. Si l'indice de similarité de toute terminologie est égal à 1, la requête sera générée automatiquement, sinon le mappage doit être validé par l'utilisateur. La figure 7.6 est l'interface de fonction de validation des requêtes, et adaptation des requêtes avec la base de données. Après l'adaptation de requête, cette fonction peut exécuter la requête personnalisée. les résultats sont sauvegardés dans un fichier JSON. Nous n'avons pas créé les individuels dans BPO. Parce que quand un processus de métier travaille avec une grand base de données, la performance du sémantique système est l'un des problèmes le plus important.

- ***Application Simulation Generator* :** Après avoir terminé la conception et la configuration ci-dessus, l'utilisateur doit utiliser cette fonction pour générer la simulation de leur application. La simulation est une petite application web qui contient toutes les tâches dans le processus de métier. La couche de présentation est générée automatiquement et dépend du processus de métier. Cette fonction est intégrée à un petit HttpServer pour lancer la simulation d'application par un petit site web.

7.2.2 Architecture

Cette section présente l'architecture de notre implémentation. La figure 7.2 montre que l'architecture de l'implémentation comporte les modules suivants :

- **Editeur de processus métier**
  Comme mentionné au chapitre 4, l'éditeur de processus de métier est implémenté pour concevoir le processus de métier d'une application typique dans un domaine. Cet éditeur prend en charge la boîte à outils qui permet à l'utilisateur de concevoir un graphe de CPN, mais aussi de définir la requête formelle dans chaque transition qui sera utilisée dans la recommandation de requête du module. Cet éditeur sauvera le processus métier dans BPO. L'éditeur de processus métier fonctionne également afin de vérifier un processus métier en utilisant un raisonneur pour raisonner sur l'ontologie des processus de métier. Cette fonction est représentée par la flèche entre le raisonnement du module, l'ontologie des processus métier et l'éditeur de processus métier. Pour implémenter ce module, nous utilisons OWL API [API] afin de travailler avec l'ontologie et le parrainé.

- **Editeur de règles de métier**
  Ce module est un éditeur de texte. L'utilisateur se sert de cet outil pour définir un ensemble de règles de métier par SBVR. SBVR est composé de terminologies, de verbes et de phrases. Les terminologies et les verbes sont prédéfinis dans l'ontologie. SBVR fournit un modèle de phrase pour aider l'utilisateur à définir la règle métier en utilisant la suggestion de terminologies et de verbes. L'utilisateur peut définir de nouvelles terminologies et de nouveaux verbes dans le cas où ils n'existent pas dans l'ontologie des règles

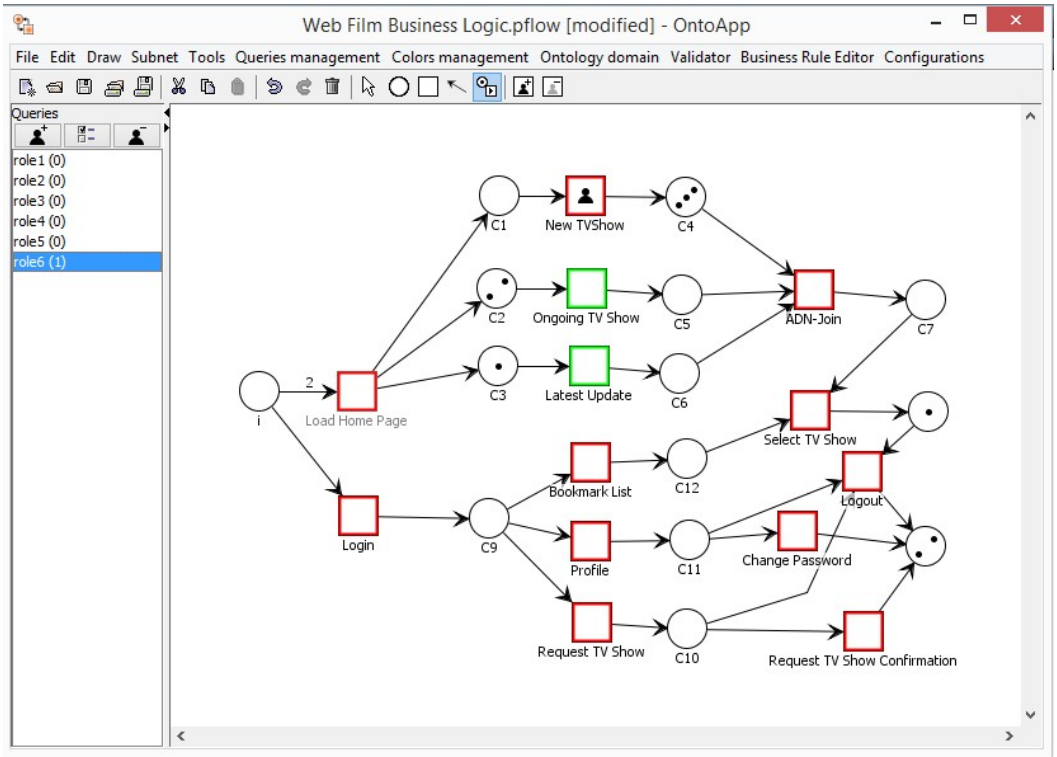

Figure 7.1: Editeur de processus de métier

de métier. L'ensemble des règles de métier est vérifié par un raisonneur, cette fonction est nécessaire car si nous appliquons des règles de métier inconsidérées, le processus métier ne peut pas être exécuté correctement. Ce module a été développé en langage Java et utilisé OWL API pour fonctionner avec l'ontologie et le parrainé.

- **BPO**

  BPO a été introduit au chapitre 4. Il s'agit d'une ontologie utilisée pour représenter un processus métier. Selon le type d'applications, nous pouvons disposer d'un dépôt BPO contenant beaucoup de types d'application. L'utilisateur peut télécharger à partir de ce dépôt et modifier le modèle pour qu'il corresponde à ses besoins. Nous utilisons l'API OWL pour traiter les connaissances du processus métier.

- **BRO**

  BRO a été présenté plus en détail au chapitre 5. Pour définir les règles, nous fournissons un langage naturel de règle, OWL API pour générer les axiomes

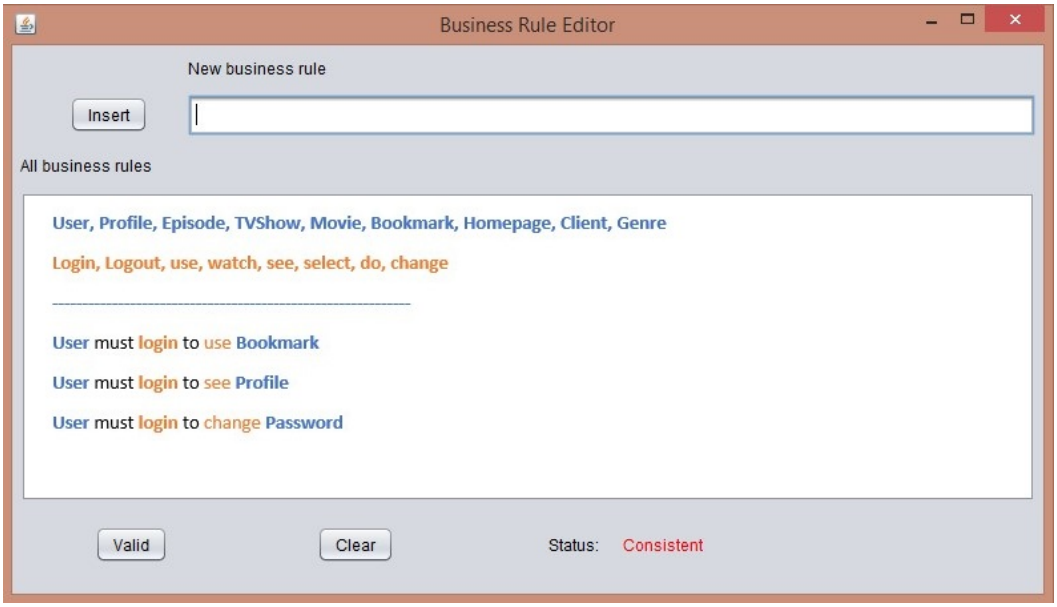

Figure 7.2: Editeur de règles de métier

dans BRO.

- **Conformité des processus de métier**
  Ce module a été présenté au chapitre 5. Cette fonction est nécessaire pour éviter l'erreur de démarrage du processus métier pendant la phase de conception ou la modification. Nous utilisons également l'API OWL comme une technologie de base pour développer ce module.

- **DMO**
  Cette ontologie est une ontologie créée à partir du diagramme de base de

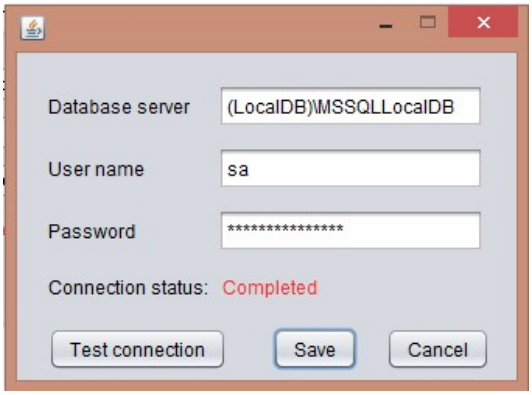

Figure 7.3: L'interface de la configuration

données d'une base de données relationnelles. Ce travail a été présenté au chapitre 6. Selon la table, les colonnes, la clé et les contraintes dans le diagramme de la base de données, l'ontologie de la cartographie des données est créée pour l'utilisation dans le module d'ontologie correspondant.

- **Ontology Alignment**

  Ce module consiste à calculer la similitude entre le concept dans deux on-

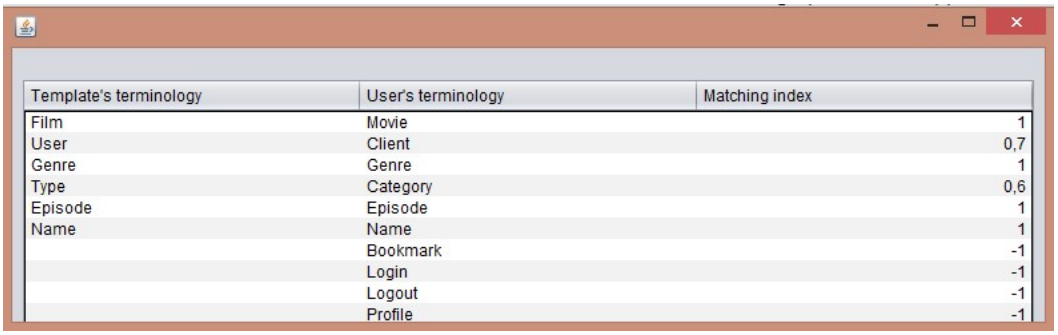

Figure 7.4: Gestion des alignements

tologies, l'ontologie des règles de métier et l'ontologie de la correspondance des données. La sortie de ce module est une table de mappage qui contient une liste d'index de similarité entre chaque couple de concepts dans deux ontologies. La sortie fournit le module de recommandation de requête. Dans ce module, nous utilisons une ontologie open source correspondant à api d'INRIA ALPES pour réaliser les alignements de l'ontologie. La figure 7.5 est l'interface de la fonction *Ontology Alignment*. Il y a trois colonnes pour présenter les alignements entre BRO et DMO.

- **Query Recommendation**

  Ce module utilise le tableau d'alignement pour exécuter une requête dans une transition si l'indice de similarité est égal à 1 ou recommande une requête si l'indice de similarité est inférieur à 1. Ce module a été exprimé plus en détailé dans le chapitre 6. La figure 7.6 est l'interface d'affichage la requête recommandée et la requête validée. Chaque transition a une requête recommandée. Si la requête recommandée n'adapte pas avec la base de données de l'utilisateur, ce dernier peut la modifier pour avoir une requête adaptée.

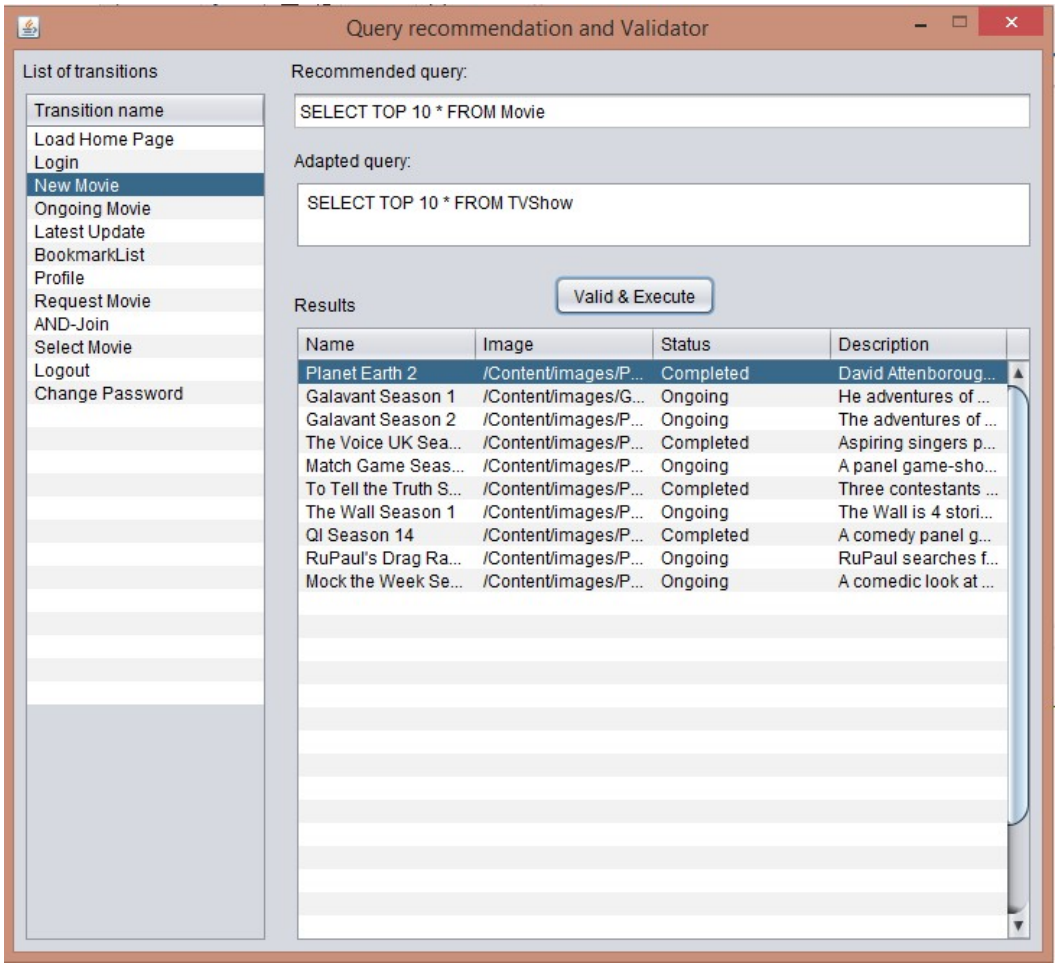

Figure 7.5: Recommandation de requêtes

7.3 Conclusion

Ce chapitre présente la caractéristique principale et l'architecture de la mise en œuvre. Cette implémentation n'est qu'un prototype qui démontre notre approche. Ce n'est pas un produit complet. Mais il montre qu'un développement de logiciel déclaratif est une approche réalisable, cette approche, qui peut modifier certaines routines dans le cycle de vie du développement de logiciels. La mise en œuvre ne peut pas fonctionner automatiquement, mais elle peut aider le programme de démarrage et de projet à avoir une vue détaillée sur leur produit attendu. OWL API est une API que nous avons utilisé avec l'ontologie et un raisonneur, pour faire correspondre le concept dans l'étape de personnalisation, nous utilisons une apologie de l'ontologie correspondante de l'équipe INRIA ALPES.

Chapitre 8

Évaluation

Contenu

8.1 Introduction

Ce chapitre fournit l'évaluation de notre approche. Nous le commençons avec une simulation de fonctionnement d'un site web d'émission de télévision. Cette évaluation consiste en l'évaluation de processus de métier, l'évaluation de règle de métier, l'évaluation du conformité du processus de métier, et l'intégration de source de données. D'autre part, nous voulons évaluer la qualité d'ontologie générée par notre implémentation. La qualité d'ontologie est importante avec la performance de la simulation. Il y a trois ontologies principales : Ontologie des processus de métier, Ontologie des règles de métier, et ontologie des bases de données relationnelles. Dans la deuxième partie de ce chapitre, nous donnons une évaluation sur la vérification de la qualité de trois ontologies principales mentionnées ci-dessus.

8.2 La simulation d'un site web d'émission de télévision

Dans cette section, nous évaluons l'implémentation de notre travail en créant une simulation d'un site web d'émission de télévision. Cette simulation a été analysée dans quelques petits exemples dans quelques sections ci-dessus. Nous allons expliquer cette évaluation de manière plus détaillée dans les sous-sections suivantes :

8.2.1 L'exigence principale

Dans cette simulation, nous voulons simuler seulement le "front-end" du site web d'émission de télévision. Avec "front-end" partie du site web d'émission de télévision, il doit avoir au minimum les fonctions suivantes :

- Exigence 1 : Il peut fournir la liste de nouvelle émission de télévision, la liste d'émission de télévision en cours.
- Exigence 2 : les utilisateurs peuvent s'identifier sur le site web, et déconnecter aussi.
- Exigence 3 : Chaque utilisateur doit avoir son profil affiché s'il est en ligne.
- Exigence 4 : L'utilisateur peut ajouter une émission de télévision préféré à sa liste des films souhaités.
- Exigence 5 : L'utilisateur peut demander une émission de télévision sur le site web.
- Exigence 6 : La liste de souhaits peut afficher si l'utilisateur s'est déjà identifié.

8.2.2 Processus de métier

Pour évaluer la fonction *OntoApp : Business Process Editor*, à partir de l'exigence principale d'un site web d'émission de télévision, nous allons utiliser l'éditeur de processus de métier pour créer un processus de métier d'un site web d'émission de télévision. Ce processus de métier vérifiera la consistance d'eux en utilisant le menu *Validator*. La figure 8.1 présente le processus de métier d'un site web d'émission de télévision. La transition *Load Home Page* permet trois transitions *New TV Show*, *Ongoing TV Show*, et *Latest Update* exécutés en parallèle. Quand la transition

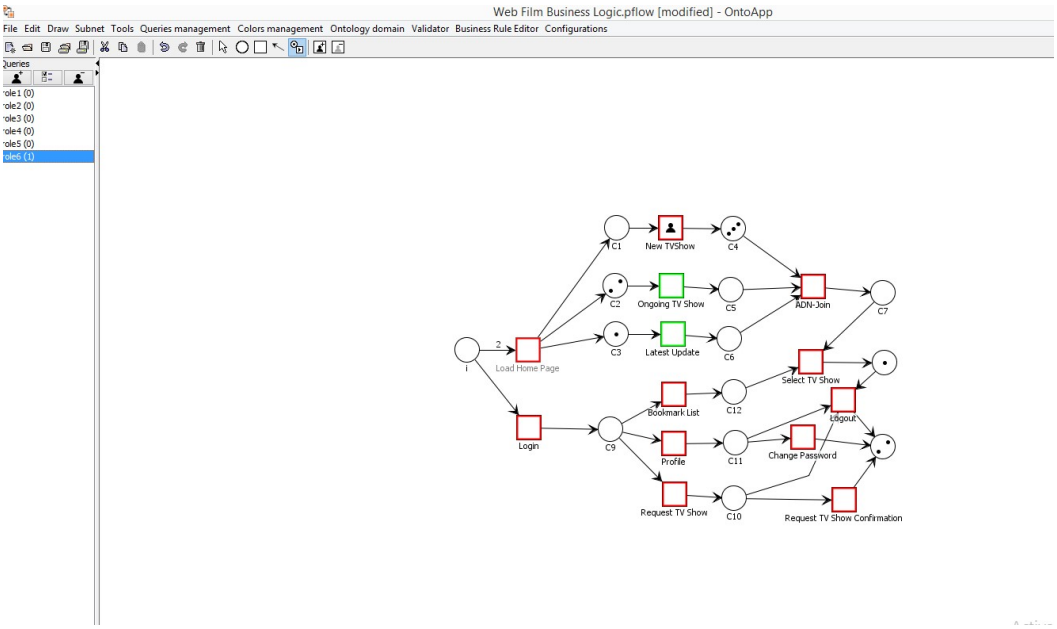

Figure 8.1: Processus de métier d'un site web de l'émission de télévision

AND-Join est déclenchée, la page d'accueil du site web est téléchargé complètement. L'autre direction dans le processus de métier, les transitions *Bookmark List*, *Profile*, *Request TV Show*, *Change Password*, *Logout* peuvent être exécutées si la transition *Login* a été exécutée.

Le processus de métier du site web d'émission de télévision est traduit en OWL 2 ontologie. Vous pouvez trouver l'ontologie BPO dans l'annexe A.

8.2.3 Evaluation de règles de métier

Dans la figure 8.2, nous fournissons un petit ensemble de règles de métier du site web d'émission de télévision. Les règles sont simples mais elles sont appliquées sur le processus de métier défini dans cette section. *OntoApp : Business Rule Editor peut valider un ensemble de règles de métier en utilisan*t le raisonneur Peller. L'ensemble de règles de métier dans la figure 8.2 est cohérent.

8.2.4 Evaluation de la conformité des processus de métier

Comme mentionné au chapitre 5, il existe trois types d'ordre de tâches, les tâches parallèles, les tâches exécutées dans le choix et les tâches séquentielles. Le tableau 8.1 montre la liste des erreurs sémantiques potentielles pouvant être détectées par

Table 8.1: Le cas du test de l'évaluation

Test de case	BPO	BRO
1	T_0 depend T_i with $1 \leq i \leq n$	T_0 seqExc T_1 and T_i depends T_1 with $2 \leq i \leq n$
2	T_0 depend T_i with $1 \leq i \leq n$	T_0 seqExc T_1 and T_i parallel T_1 with $2 \leq i \leq n$
3	T_0 parallel T_i with $1 \leq i \leq n$	T_0 paraExc T_1 and T_i parallel T_1 with $2 \leq i \leq n$
4	T_0 parallel T_i with $1 \leq i \leq n$	T_0 choiceExc T_1 and T_i parallel T_1 with $2 \leq i \leq n$
5	T_1 paralExc T_i with $1 \leq i \leq n$	T_1 parallel T_i with $2 \leq i \leq n$
6	T_1 sequenExc T_i with $1 \leq i \leq n$	T_1 depends T_i with $2 \leq i \leq n$
7	T_1 choiceExc T_i with $1 \leq i \leq n$	T_1 choice T_i with $2 \leq i \leq n$
8	T_1 sequenExc T_2	T_1 choiceExc T_2 T_1 choiceExc T_2
9	T_0 depend T_i or T_0 choice T_i or with $1 \leq i \leq n$	T_0 parallel T_1 and T_i depends T_1 with $2 \leq i \leq n$ Or T_0 parallel T_1 and T_i parallel T_1 with $2 \leq i \leq n$
10	T_0 depend T_i or T_0 parallel T_i with $1 \leq i \leq n$	T_0 choice T_1 and T_i parallel T_1 with $2 \leq i \leq n$
11	T_0 choice T_i with $1 \leq i \leq n$ T_0 depend T_i with $1 \leq i \leq n$	T_0 depends T_1 and T_i parallel T_1 with $2 \leq i \leq n$

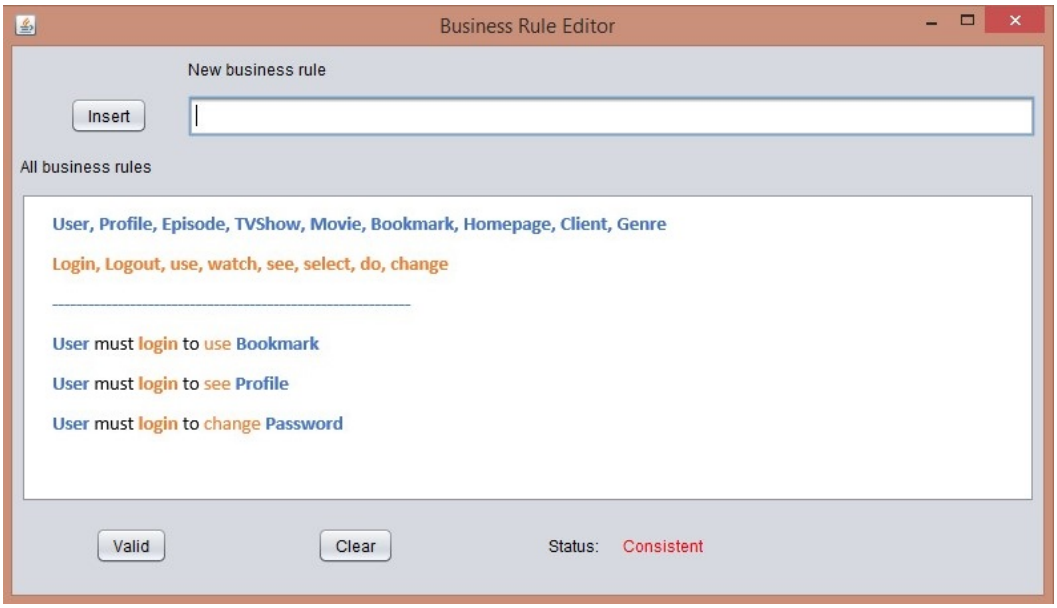

Figure 8.2: Exemple de règles de métier

notre approche lors du contrôle de la conformité des processus de métier avec un ensemble de règles de métier. Ce cas de test s'applique à l'ordre de la transition.

- **Cas de test 1 :** si la transition T_i doit exécutée avant la transition T_0 dans BPO, la transition T_0 doit être exécutée avant la transition T_1 et T_i doit être exécutée après la transition T_1. Le BPO ne suit donc pas la règle prédéfinie dans BRO. C'est un conflit potentiel qui ne peut pas être détecté à la main, car T_0 et T_i ne sont pas les voisins les uns des autres. Avec notre approche, nous utilisons la propriété transitive pour raisonner sur une propriété de séquence. Cela pourrait aider à détecter le conflit potentiel qui ne peut pas être vu manuellement.

- **Cas de test 2 :** si la transition T_i doit être exécutée avant la transition T_0 dans BPO, la transition T_0 doit être exécutée avant la transition T_1 et T_i doivent être exécutés en parallèle avec la transition T_1 dans BRO. Donc, il y a un conflit potentiel entre les règles dans BPO et les règle prédéfinie dans BRO. C'est un conflit potentiel qui ne peut pas être détecté à la main, car si T_0 et T_i ne sont pas les voisins les uns des autres, nous ne pouvons pas voir ce type d'erreur.

- **Cas de test 3 :** si la transition T_i est exécutée en parallèle avec la transition

T_0 dans BPO, la transition T_0 est déclarée exécuter en parallèle avec la transition T_1 et T_i doivent être exécutés en parallèle avec la transition T_1 dans BRO. Le BPO ne suit donc pas la règle prédéfinie dans BRO. C'est un conflit potentiel qui ne peut pas être détecté à la main, car si T_0 et T_i ne sont pas les voisins les uns des autres, nous ne pouvons pas voir ce type d'erreur.

- **Cas de test 4 :** si la transition T_i est exécutée en parallèle avec la transition T_0 dans BPO, la transition T_0 est déclarée à exécuter au choix avec la transition T_1 et T_i doivent être exécutés en parallèle avec la transition T_1. Le BPO ne suit donc pas la règle prédéfinie dans BRO. C'est un conflit potentiel qui ne peut pas être détecté à la main, car si T_0 et T_i ne sont pas les voisins les uns des autres, nous ne pouvons pas voir ce type d'erreur.

Le reste du cas de test a la même approche avec le cas de test 1, nous définissons la règle en fonction de ceux-ci pour éviter les conflits potentiels. Lorsque l'utilisateur crée ou personnalise un processus de métier, le raisonneur raisonnera sur BPO et BRO pour vérifier la conformité des processus de métier avec des règles de métier. Si elle détecte une erreur potentielle au moment de conception, le système informera l'utilisateur pour les aider à effectuer un changement nécessaire.

Le tableau 8.2 présente notre expérience sur la performance de conformité des processus de métier. Nous ajoutons de nouveaux concepts, règles et des individus à l'ontologie du processus de métier et à l'ontologie de la règle de métier pour évaluer la performance de cette fonction. L'heure dans la colonne "Heure d'exécution" est le temps de vérification qu'il n'y a pas de conflit entre le processus de transaction et les règles de métier.

8.2.5 Évaluation de l'intégration des données

Dans cette section, nous présentons la base de données de films. La figure 8.3 est le diagramme de base de données qui contient les informations suivantes : émission de télévision, épisode, genre, acteur, signet, etc. Selon le processus de métier que nous concevons, chaque transition contient chaque requête de tir dans un formulaire de règle. La recommandation de la requête le transformera en la requête correspondante en fonction de la base de données. Dans cette thèse, nous fournissons uniquement la transformation pour la base de données relationnelle.

Table 8.2: Performance de conformité des processus de métier

Processus de métier	**règles de métier**	**Temps d'exécution**
CPN concepts,	20 concepts,	
15 properties,	15 properties,	*24s*
30 tasks,	30 reaction, integrity, transformation	
50 individuals	30 tasks	
CPN concepts,	20 concepts,	
22 properties,	15 properties,	2578s
35 tasks,	40 reaction rules ,	
65 individuals	30 tasks	
CPN concepts,	44 concepts,	
22 properties,	39 properties,	3488s
35 tasks	40 rules,	
65 individuals	30 tasks	
CPN concepts,	44 concepts,	
22 properties,	39 properties,	5600s
40 tasks,	40 rules,	
70 individuals	35 tasks	
CPN concepts,	50 concepts,	
22 properties,	15 properties,	6050s
40 tasks,	45 reaction rules,	
70 individuals	40 tasks	
CPN concepts,	50 concepts,	
22 properties,	15 properties,	7690s
50 tasks,	45 reaction rules,	
80 individuals	40 tasks	

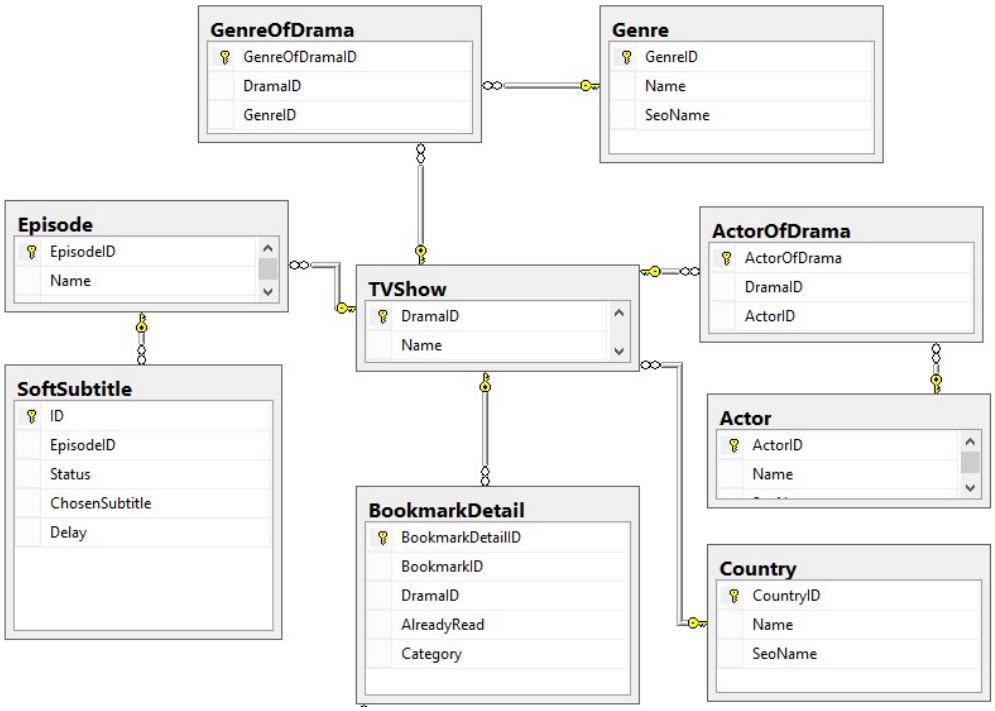

Figure 8.3: Base de données du site web de l'émission de télévision

Dans la figure 8.4, chaque transition a une *Firing Query* utilisée pour décrire le contenu principal de la transition. La requête de démarrage sera transformée en requête SQL pour être exécutée sur une base de données relationnelle. Le tableau 8.3 montre la transformation de chaque requête de démarrage en requête SQL. Par exemple : La transition *New TV Show* est pour obtenir le film le plus récent dans la base de données, cette transition a une *Firing Query* représentée par une règle SWRL dans l'ontologie des processus de métier comme suit :

$$TVShow(?x) \wedge hasAddedDate(?x,?date)$$
$$\rightarrow sqwrl{:}select(?x) \wedge sqwrl{:}orderBy(?date) \wedge Fired(?token,?x)$$

Cette règle SWRL sera transformée en une requête SQL :

$$SELECT\ *\ FROM\ TVShow\ ORDERBY\ AddedDate$$

Cette requête SQL sera exécutée lorsque l'utilisateur exécutera le processus de métier. Lorsqu'une transition est déclenchée, la requête de tir sera transformée en requête SQL pour l'exécuter.

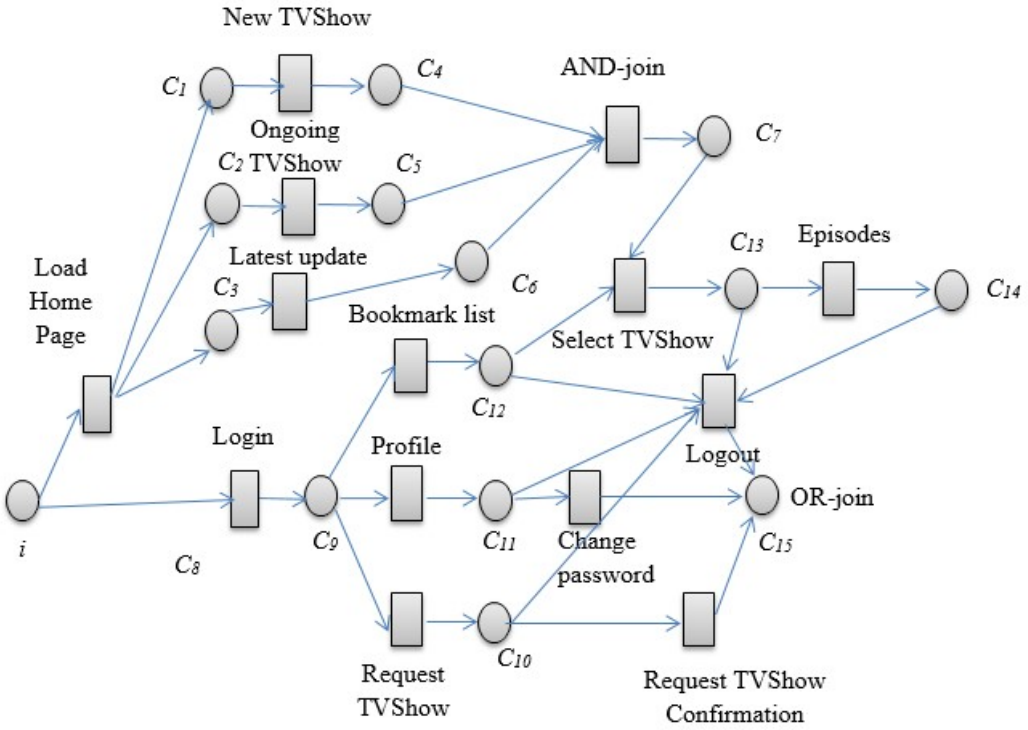

Figure 8.4: Structure du processus de métier du site web de l'émission de télévision

Comme mentionné au chapitre 4, il existe deux types de jetons : *internal-token* et *external-token*. Le *internal-token* est utilisé dans une transition qui n'a pas besoin de l'action extérieure à déclencher. La valeur du jeton interne est une modification dans le processus de métier pendant l'exécution. Le jeton externe est utilisé dans une transition qui nécessite une action provenant de l'extérieur du processus de métier. La valeur de *external-token* est remplie à la main ou par un événement d'un autre système.

Par exemple, transition *Request TVShow* et transition *Change Password* ont besoin de la valeur provenant de l'utilisateur. Pour la transition *Change Password*, cette transition doit être exécutée après la transition *Login*. L'utilisateur doit taper le nouveau mot de passe pour remplacer l'ancien mot de passe. Le nouveau mot de passe est un jeton externe utilisé dans la requête SQL pour mettre à jour les données dans la base de données.

Table 8.3: Transformation des requêtes

Trans	**Patron de requête**	**SQL Transformation**
New TVShow	*TVShow(?x) ∧ hasAddedDate(?x,?date) → sqwrl:select(?x) ∧ sqwrl:orderBy(?date) ∧ Fired(?token,?x)*	SELECT * FROM TVShow ORDERBY AddedDate
Ongoing TVShow	*TVShow(?x) ∧ hasStatus(?x,?status) ∧ stringEqualIgnoreCase(?statis,"ongoing") → sqwrl:select(?x) ∧ sqwrl:orderBy(?date) ∧ Fired(?token,?x)*	SELECT * FROM TVShow WHERE Status="Ongoing" ORDERBY AddedDate
Latest update	*TVShow(?x) ∧ hasUpdate(?x,?update) → sqwrl:select(?x) ∧ sqwrl:orderBy(?update) ∧ Fired(?token,?x)*	SELECT * FROM TVShow ORDERBY UpdateDate
Login	*Users(?x) ∧ hasUserName(?x,?username) ∧ hasPass(?x,?password) ∧ stringEqual(?username,ex-token) ∧ stringEqual(?password,token) → sqwrl:select(?x)∧ Fired(?token,?x)*	SELECT * FROM Users WHERE Username='token' AND Password='Token'
Bookmark List	*Bookmark(?x) ∧ hasUserName(?x,?username) ∧ stringEqual(?username,in-token) → sqwrl:select(?x)∧ Fired(?token,?x)*	SELECT * FROM Bookmarks WHERE Username='token'
Profile	*Users(?x) ∧ hasUserName(?x,?username) ∧ stringEqual(?username,in-token) → sqwrl:select(?x)∧ Fired(?token,?x)*	SELECT * FROM Users WHERE Username='token'
Request TVShow	*Request(?x) ∧ hasUserID(?x,ex-tokenUID) ∧ hasTVShowID(?x,?ex-tokenMID) → DoInsert(?x)∧ Fired(?token,?x)*	INSERT INTO Requests(UID,MID) VALUES(tokUID,tokMID)
Change Password	*Users(?x) ∧ hasUserID(?x,?uid) ∧swrlb:equal(?uid,token)∧hasPass(?x,?pass) → DoUpdate(?pass,token)*	UPDATE Users SET Password=token WHERE UserName='tokenUID'

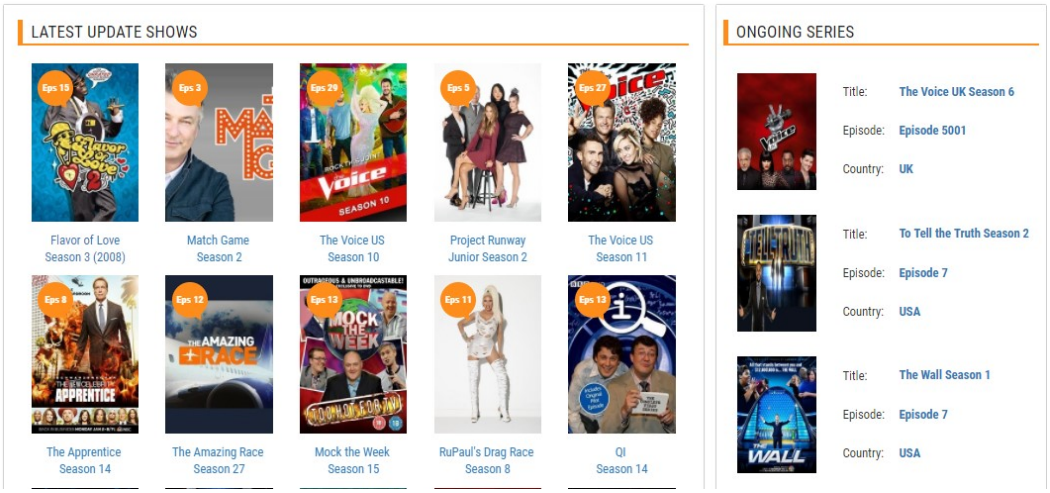

Figure 8.5: L'interface d'accueil du site web

8.2.6 La simulation du site web d'émission de télévision

La figure 8.5 est l'interface d'accueil du site web d'émission de télévision que nous avons fait avec simulateur l'OntoApp. Dans le processus de métier du site web du film que on a déjà présenté plus haut. Cet interface est généré après l'exécution de la transition AND-Join, parce que la page d'accueil a besoin des donnés produites par trois transitions *New TVShow*, *Ongoing TVShow*, et *Latest Update* pour afficher.

8.3 Évaluation de la qualité de l'ontologie

OQuaRE [DFSA11] est un cadre existant qui fournit une bonne fonction pour évaluer la qualité de l'ontologie. Ce cadre basé sur la norme SQuaRE utilisée dans l'évaluation de la qualité du logiciel [SQu]. OQuaRe utilise un modèle de qualité et des indicateurs de qualité pour évaluer l'ontologie. Le modèle de qualité a la dimension suivante pour évaluer l'ontologie: adéquation structurelle, fonctionnelle, maintenabilité, compatibilité, transférabilité, opérabilité, fiabilité. Nous expliquons chaque dimension comme suit :

- La dimension structurelle est utilisée pour évaluer les concepts et les propriétés des facteurs de qualité tels que la cohérence, la redondance ou l'enchevêtrement. Cette dimension est importante car au moins l'ontologie générée doit être cohérente.

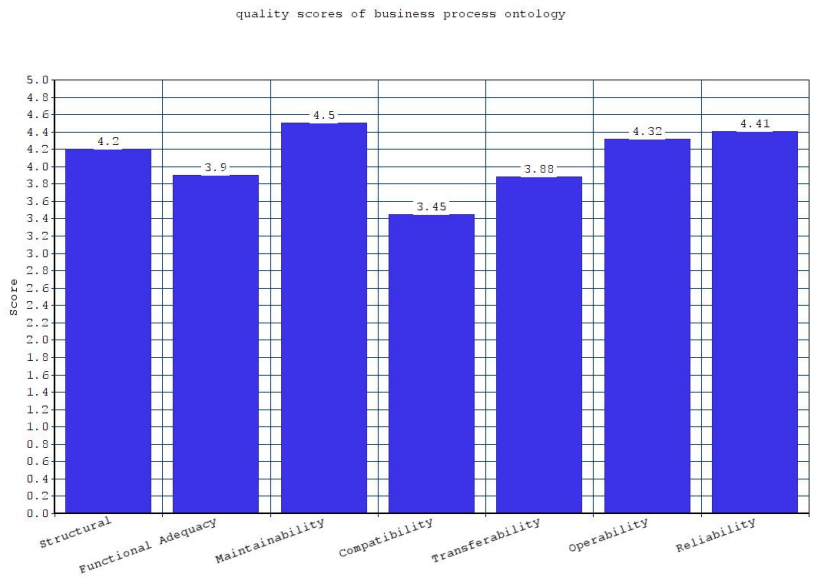

Figure 8.6: Niveau de qualité de l'ontologie des processus de métier

- La dimension de l'adéquation fonctionnelle est liée à la propriété de l'ontologie pour la raison proposée.

- La dimension de maintenance est utilisée pour vérifier l'ontologie de capacité personnalisable lorsque les exigences ont un changement.

- La dimension de compatibilité est liée à la capacité d'au moins deux ontologies à envoyer et à recevoir les données ainsi qu'à jouer leurs capacités requises tout en ayant un équipement ou des environnements de programmation similaires. La dimension de compatibilité peut être évaluée sur une seule ontologie même si elle inclut des propriétés sur plus d'une ontologie puisqu'elle est évaluée quantitativement par des méthodes pour un arrangement de mesures liées à chaque ontologie indépendamment.

- La dimension de transférabilité est la détection pour décider quelle ontologie peut être échangée d'un environnement à un autre environnement.

- La dimension de l'opération est liée à l'effort nécessaire pour utiliser l'ontologie et, dans l'évaluation individuelle d'une telle utilisation, par un ensemble d'utilisateurs déclaré ou implicite.

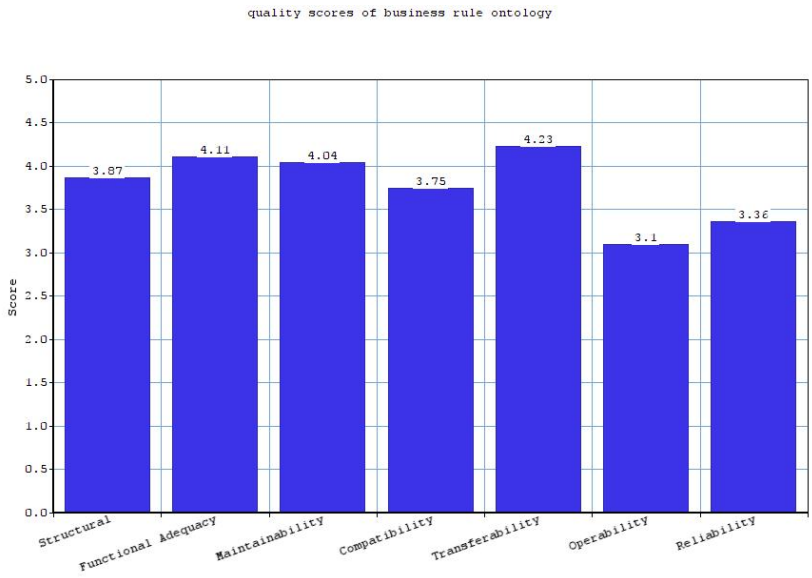

Figure 8.7: Niveau de qualité de l'ontologie des règles de métier

- La dimension de fiabilité est le score utilisé pour évaluer la capacité de l'ontologie à maintenir son niveau de performance dans les conditions indiquées pour une période donnée.

Chaque dimension a un score entre 1 et 5. Si le score est inférieur ou égal à 1, cela signifie qu'il n'est pas acceptable. S'il est égal à 3, cela signifie qu'il est possible de l'accepter et si le score est égal à 5, cela signifie que l'ontologie dépasse les exigences.

La figure 8.6 présente l'évaluation de l'ontologie des processus de métier qui a été introduite au chapitre 4 en utilisant le cadre OQuaRE. Le score de structure de BPO est égal à 3,12, cela signifie que BPO est une ontologie acceptable. Le score d'adéquation fonctionnelle, le score de maintenabilité, le score de compatibilité, le score de transférabilité, le score d'opérabilité et le score de réalalité sont supérieurs à 3. La qualité de BPO est acceptable selon la règle du cadre OQuaRE.

La figure 8.7 présente l'évaluation de l'ontologie des règles de métier. Cette évaluation a un bon résultat. Tous les scores de BRO sont supérieurs à 3. Donc, le BRO est généré de bonne qualité.

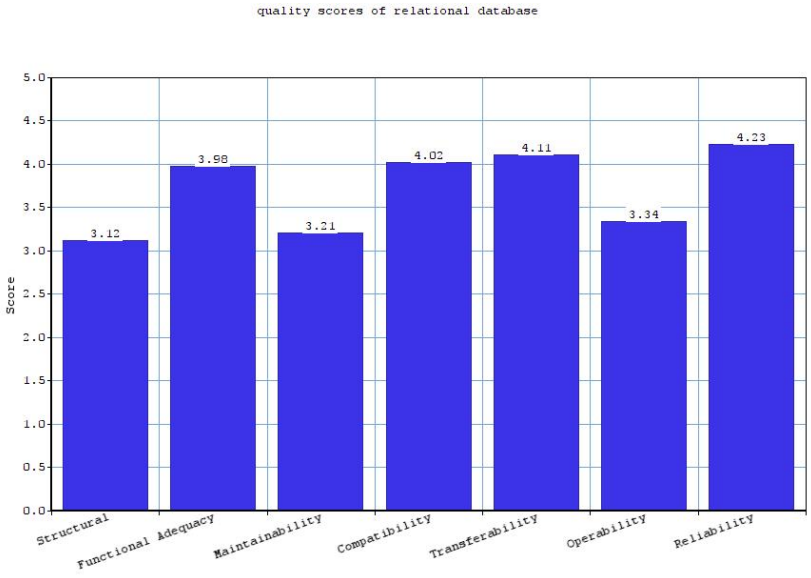

Figure 8.8: Niveau de qualité d'ontologie de base de données relationnelles

Partie IV

Conclusion

Chapitre 9

Conclusions et perspectives

Contenu

9.1 Introduction

Dans cette thèse, nous nous sommes concentrés sur l'application de l'ontologie sur la couche de métier à la phase de conception du cycle de vie du développement logiciel. Nous avons proposé une approche à base d'ontologie pour représenter le processus de métier, la règle de métier, vérifier la conformité d'un processus de métier, l'accès aux données en utilisant l'ontologie OWL. Nous décrivons brièvement les résultats obtenus, la contribution récapitulative et de nombreuses questions ouvertes pour des recherches complémentaires.

Au chapitre IV, nous utilisons le réseau de Pétri coloré pour modéliser un processus de métier. Ensuite, le processus de métier (un graphe de réseaux de Pétri coloré) est représenté par une ontologie appelée "*Business Process Ontology*". Cette ontologie comporte trois parties principales : TBOX, ABOX et Règles. TBOX est un ensemble de concepts utilisé pour représenter un graphe de réseaux petri et la relation entre eux. ABOX est un ensemble d'instances de processus de métier, et les règles sont définies pour être utilisées par un raisonneur afin de vérifier automatiquement la cohérence des processus de métiers. La contribution de cette partie

est de fournir une méthode pour représenter et vérifier le modèle d'un processus de métier en utilisant l'ontologie et le raisonnement.

Dans le chapitre V, nous définissons une méthode pour traduire un ensemble de règles SBVR en ontologie OWL 2. Lorsqu'un ensemble de règles de métier est défini par SBVR, la terminologie, le verbe et la phrase sont traduits en une ontologie qui s'appelle *Business Rule Ontology* (BRO). Lorsqu'une nouvelle règle est ajoutée, un raisonneur vérifiera sur la base de connaissances des règles de métier pour détecter s'il y a un conflit qui se produit ou non.

Dans le chapitre VI, nous présentons une approche permettant à l'utilisateur de personnaliser un modèle d'application pour correspondre avec le système d'information de l'utilisateur. Pour ce faire, nous proposons une méthode pour personnaliser un modèle de processus métier sans créer l'erreur de structure dans le modèle de processus de métier pendant la modification. Nous proposons également une approche pour la recommandation de la requête selon la source de données utilisée dans le système d'information de l'utilisateur. Dans ce chapitre, nous proposons également une approche pour aligner la base de données relationnelle de l'utilisateur avec une ontologie. Avec le cartographe, la requête a été définie dans le modèle de processus métier qui sera traduit en une requête recommandée pour la base de données relationnelle de l'utilisateur.

9.2 Résumé des contributions

Dans ce travail, nous nous concentrons sur la collaboration de l'ingénierie logicielle et web sémantique. Plus en détail, nous envisageons d'appliquer l'ontologie sur la couche de métier à la phase de conception d'une application. Grâce à cette motivation, nous divisons notre contribution en deux contributions principales dans les domaines de la collaboration entre ingénierie logicielle et web sémantique.

9.2.1 Couche de métier exécutable basé sur l'ontologie

Cette contribution répond à la question de recherche 1 et à la question de recherche 2. Nous divisons cette contribution en trois sous-contributions comme suit :

- **Modèle de processus métier basé sur l'ontologie**
  Nous utilisons réseau de Pétri coloré pour modéliser un processus de métier.

Ensuite, le processus de métier (un graphe de réseaux de Pétri coloré) est représenté par une ontologie appelée "Ontologie des processus de métier". Cette ontologie comporte trois parties principales : TBOX, ABOX et règles. TBOX est un ensemble de concepts utilisé pour représenter un graphe de réseaux Pétri et la relation entre eux. ABOX est un ensemble d'instances de processus de métier, et les règles sont définies pour être utilisées par un raisonneur pour vérifier automatiquement la cohérence des processus de métier. La contribution de cette partie est de fournir une méthode pour représenter et vérifier le modèle d'un processus de métier en utilisant l'ontologie et le raisonnement.

- **Définition de règle métier basée sur l'ontologie**
  La deuxième partie importante d'une couche logique de métier est la règle de métier. Nous définissons un mappage entre les règles SVBR et la syntaxe fonctionnelle OWL 2 pour permettre à l'utilisateur de définir les règles de métier. Lorsqu'un ensemble de règles de métier est défini par SVBR, la terminologie, le verbe et la phrase sont traduits en une ontologie qui s'appelle *Business Rule Ontology* (BRO). Lorsqu'une nouvelle règle est ajoutée, un raisonneur raisonnera sur la base de connaissances des règles de métier pour détecter s'il y a un conflit qui se produit ou non.

- **Vérification de la conformité aux processus métiers basée sur l'ontologie**
  Cette contribution propose une approche basée sur l'ontologie pour la vérification de la conformité des processus de métier. Lorsqu'un processus de métier est créé, il doit suivre les règles de métier prédéfinies. Notre solution est de mapper le concept dans BPO et BRO, après la cartographie, la règle dans BRO sera appliquée à l'instance de processus de métier dans BPO.

9.2.2 Réutilisation d'une couche de métier

Dans la contribution précédente, nous proposons une approche déclarative pour la modélisation d'une couche de métier en utilisant l'ontologie. Il peut être utilisé pour définir un modèle d'application pour une application spécifique dans un domaine. C'est un point fort de notre approche, car le modèle d'application peut

être réutilisé par la communauté. Dans d'autres cas, nous devons répondre à la question de recherche 3 et à la question de recherche 4. La deuxième contribution de ma thèse répond à ces questions de recherche. Nous proposons une approche permettant à l'utilisateur de personnaliser un modèle d'application pour correspondre avec le système d'information de l'utilisateur. Pour ce faire, nous proposons une méthode afin de personnaliser un modèle de processus de métier sans créer l'erreur de structure dans le modèle de processus métier pendant la modification. Nous proposons également une approche pour la recommandation de la requête selon la source de données utilisée dans le système d'information de l'utilisateur. Un processus de métier personnalisé est en fait un modèle de processus de métier modifié pour s'adapter à un système d'information existant d'un utilisateur. Cette adaptation comprend l'adaptation des processus de métier, l'adaptation des données, l'adaptation des terminologies et l'adaptation des règles de métier. Chaque adaptation sera résolue par la contribution suivante :

- **Opérations de changement**

  Nous proposons un ensemble d'opérations de changement qui permet aux utilisateurs de personnaliser un modèle de processus métier. Nous fournissons également une limitation de la modification des utilisateurs afin de ne pas mettre de conflit avec un ensemble de règles de métiers prédéfinis.

- **Intégration de source de données**

  Nous proposons une approche pour aligner la base de données relationnelle de l'utilisateur avec une ontologie. Avec l'alignement, la requête a été définie dans le modèle de processus de métier qui sera traduit en une requête recommandée pour la base de données relationnelle de l'utilisateur.

- **Recommandation de requête**

  Chaque utilisateur a son propre système d'information, y compris la source de données. Afin que les processus de métiers personnalisés soient prêts à être utilisés, nous proposons une méthode pour recommander une requête qui correspond au système de source de données de l'utilisateur en utilisant la correspondance de l'ontologie.

9.3 Directions futures

Dans cette thèse, nous proposons seulement une approche pour la démonstration de logiciels dans la phase de conception du cycle de vie du développement logiciel. Cette démonstration peut fonctionner avec la source de données et avoir une fonction complète comme une application commune complétée. Cette approche ne couvre pas tous les cas de développement de logiciels, mais pour l'application populaire, cela peut être démontré par notre outil. À l'avenir, nous souhaitons continuer la recherche dans ce sens et faire de la démonstration actuelle un produit complété, et vérifier s'il peut se déployer en tant que logiciel développé dans une procédure traditionnelle. Pour ce faire, nous devons améliorer les performances du prototype actuel. Un autre aspect, la couche logique métier doit être compilée et encapsulée dans un fichier d'exécution. Pour ce faire, une autre recherche doit poursuivre notre travail et proposer une méthode de compilation et le modèle d'exécution. Si cette cible est atteinte, pendant le cycle de développement du développement logiciel, la phase de mise en œuvre devient plus simple. La phase de test peut être effectuée en parallèle avec la phase de développement dans certains cas. Le développeur peut voir le comportement réel du produit au début du développement de logiciels. Bien que cette approche puisse remplacer l'approche traditionnelle d'ingénierie logicielle, nous croyons que l'application de la technique sémantique sur le domaine de l'ingénierie logicielle peut constituer un tournant intéressant dans le domaine de l'ingénierie logicielle.

Nous continuerons à améliorer l'OntoApp afin de permettre l'utilisateur peut intégrer plusieurs de source de données et et l'autre type de données, pas seulement base de données relationnelle.

Au niveau de l'interface de l'utilisateur, nous essayerons d'intégrer le langage BPMN pour permettre l'utilisateur à modéliser un processus de métier plus facilement. Afin de faire ça, nous devons transformer le modèle d'un graphe BPMN à le modèle d'un graphe réseaux de Pétri. Parce que BPMN n'est pas un langage formel. Nous avons besoin de cette transformation pour valider le modèle d'un processus de métier d'une application.

L'autre part, nous allons essayer de créer un service web et un restful API pour publier notre prototype sur l'Internet afin de partager le résulta de notre recherche.

Appendix A

Business Process Ontology

```
1  ClassAssertion(qa:AND$-$Join qa:AND$-$Join$-$Homepage)
2  ClassAssertion(qa:Transition qa:Bookmark$-$list)
3  ClassAssertion(qa:Place qa:C1)
4  ClassAssertion(qa:Place qa:C10)
5  ClassAssertion(qa:Place qa:C11)
6  ClassAssertion(qa:Place qa:C12)
7  ClassAssertion(qa:Place qa:C13)
8  ClassAssertion(qa:Place qa:C14)
9  ClassAssertion(qa:OR$-$Join qa:C15)
10 ClassAssertion(qa:Place qa:C2)
11 ClassAssertion(qa:Place qa:C3)
12 ClassAssertion(qa:Place qa:C4)
13 ClassAssertion(qa:Place qa:C5)
14 ClassAssertion(qa:Place qa:C6)
15 ClassAssertion(qa:Place qa:C7)
16 ClassAssertion(qa:Place qa:C8)
17 ClassAssertion(qa:OR$-$Split qa:C9)
18 ClassAssertion(qa:Transition qa:Change$-$password)
19 ClassAssertion(qa:Transition qa:Episodes)
20 ClassAssertion(qa:CPN qa:FrontEndWebFilm)
21 ObjectPropertyAssertion(qa:hasPlace qa:FrontEndWebFilm qa:C1)
22 ObjectPropertyAssertion(qa:hasTransition qa:FrontEndWebFilm qa:Login)
23 ClassAssertion(qa:Transition qa:Latest$-$update)
24 ClassAssertion(qa:AND$-$Split qa:Load$-$Home$-$Page)
25 ClassAssertion(qa:Transition qa:Login)
26 ClassAssertion(qa:Transition qa:Logout)
27 ClassAssertion(qa:Transition qa:New$-$Movie)
28 ClassAssertion(qa:Transition qa:Ongoing$-$Movie)
29 ClassAssertion(qa:Transition qa:Profile)
30 ClassAssertion(qa:Transition qa:Request$-$Movies$-$Confirmation)
31 ClassAssertion(qa:Transition qa:Request$-$movies)
```

```
32 ClassAssertion(qa:Transition qa:Select$-$Movie)
33 ClassAssertion(qa:Place qa:i)
34 ClassAssertion(qa:InputArc qa:i$-$arc$-$1)
35 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$1 qa:i)
36 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$1 qa:Load$-
    $Home$-$Page)
37 ClassAssertion(qa:InputArc qa:i$-$arc$-$10)
38 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$10 qa:C9)
39 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$10 qa:
    Bookmark$-$list)
40 ClassAssertion(qa:InputArc qa:i$-$arc$-$11)
41 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$11 qa:C9)
42 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$11 qa:
    Profile)
43 ClassAssertion(qa:InputArc qa:i$-$arc$-$12)
44 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$12 qa:C9)
45 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$12 qa:
    Request$-$movies)
46 ClassAssertion(qa:InputArc qa:i$-$arc$-$13)
47 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$13 qa:C12)
48 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$13 qa:
    Select$-$Movie)
49 ClassAssertion(qa:InputArc qa:i$-$arc$-$14)
50 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$14 qa:C12)
51 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$14 qa:
    Logout)
52 ClassAssertion(qa:InputArc qa:i$-$arc$-$15)
53 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$15 qa:C11)
54 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$15 qa:
    Logout)
55 ClassAssertion(qa:InputArc qa:i$-$arc$-$16)
56 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$16 qa:C11)
57 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$16 qa:
    Change$-$password)
58 ClassAssertion(qa:InputArc qa:i$-$arc$-$17)
59 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$17 qa:C10)
60 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$17 qa:
    Logout)
61 ClassAssertion(qa:InputArc qa:i$-$arc$-$18)
62 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$18 qa:C10)
63 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$18 qa:
```

```
    Request$-$Movies$-$Confirmation)
64 ClassAssertion(qa:InputArc qa:i$-$arc$-$19)
65 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$19 qa:C13)
66 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$19 qa:
    Logout)
67 ClassAssertion(qa:InputArc qa:i$-$arc$-$2)
68 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$2 qa:i)
69 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$2 qa:Login)
70 ClassAssertion(qa:InputArc qa:i$-$arc$-$20)
71 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$20 qa:C14)
72 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$20 qa:
    Logout)
73 ClassAssertion(qa:InputArc qa:i$-$arc$-$21)
74 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$21 qa:C13)
75 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$21 qa:
    Episodes)
76 ClassAssertion(qa:InputArc qa:i$-$arc$-$3)
77 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$3 qa:C1)
78 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$3 qa:New$-
    $Movie)
79 ClassAssertion(qa:InputArc qa:i$-$arc$-$4)
80 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$4 qa:C2)
81 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$4 qa:
    Ongoing$-$Movie)
82 ClassAssertion(qa:InputArc qa:i$-$arc$-$5)
83 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$5 qa:C3)
84 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$5 qa:
    Latest$-$update)
85 ClassAssertion(qa:InputArc qa:i$-$arc$-$6)
86 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$6 qa:C4)
87 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$6 qa:AND$-
    $Join$-$Homepage)
88 ClassAssertion(qa:InputArc qa:i$-$arc$-$7)
89 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$7 qa:C5)
90 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$7 qa:AND$-
    $Join$-$Homepage)
91 ClassAssertion(qa:InputArc qa:i$-$arc$-$8)
92 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$8 qa:C6)
93 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$8 qa:AND$-
    $Join$-$Homepage)
94 ClassAssertion(qa:InputArc qa:i$-$arc$-$9)
```

```
95 ObjectPropertyAssertion(qa:hasSourcePlace qa:i$-$arc$-$9 qa:C7)
96 ObjectPropertyAssertion(qa:hasTargetTransition qa:i$-$arc$-$9 qa:
       Select$-$Movie)
97 ClassAssertion(qa:OutputArc qa:o$-$arc$-$1)
98 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$1 qa:Load$-
       $Home$-$Page)
99 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$1 qa:C1)
100 ClassAssertion(qa:OutputArc qa:o$-$arc$-$10)
101 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$10 qa:
       Profile)
102 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$10 qa:C11)
103 ClassAssertion(qa:OutputArc qa:o$-$arc$-$11)
104 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$11 qa:
       Request$-$movies)
105 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$11 qa:C10)
106 ClassAssertion(qa:OutputArc qa:o$-$arc$-$12)
107 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$12 qa:
       Change$-$password)
108 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$12 qa:C15)
109 ClassAssertion(qa:OutputArc qa:o$-$arc$-$13)
110 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$13 qa:
       Logout)
111 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$13 qa:C15)
112 ClassAssertion(qa:OutputArc qa:o$-$arc$-$14)
113 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$14 qa:
       Request$-$Movies$-$Confirmation)
114 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$14 qa:C15)
115 ClassAssertion(qa:OutputArc qa:o$-$arc$-$15)
116 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$15 qa:
       Select$-$Movie)
117 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$15 qa:C13)
118 ClassAssertion(qa:OutputArc qa:o$-$arc$-$16)
119 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$16 qa:
       Episodes)
120 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$16 qa:C14)
121 ClassAssertion(qa:OutputArc qa:o$-$arc$-$2)
122 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$2 qa:Load$-
       $Home$-$Page)
123 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$2 qa:C2)
124 ClassAssertion(qa:OutputArc qa:o$-$arc$-$3)
125 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$3 qa:Load$-
```

```
    $Home$-$Page)
126 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$3 qa:C3)
127 ClassAssertion(qa:OutputArc qa:o$-$arc$-$4)
128 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$4 qa:New$-
    $Movie)
129 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$4 qa:C4)
130 ClassAssertion(qa:OutputArc qa:o$-$arc$-$5)
131 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$5 qa:
    Ongoing$-$Movie)
132 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$5 qa:C5)
133 ClassAssertion(qa:OutputArc qa:o$-$arc$-$6)
134 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$6 qa:
    Latest$-$update)
135 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$6 qa:C6)
136 ClassAssertion(qa:OutputArc qa:o$-$arc$-$7)
137 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$7 qa:AND$-
    $Join$-$Homepage)
138 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$7 qa:C7)
139 ClassAssertion(qa:OutputArc qa:o$-$arc$-$8)
140 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$8 qa:Login)
141 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$8 qa:C9)
142 ClassAssertion(qa:OutputArc qa:o$-$arc$-$9)
143 ObjectPropertyAssertion(qa:hasSourceTransition qa:o$-$arc$-$9 qa:
    Bookmark$-$list)
144 ObjectPropertyAssertion(qa:hasTargetPlace qa:o$-$arc$-$9 qa:C12)
```

Listing A.1: BPO in Functional Syntax

Bibliography

[20001] *Software Product Lines: Practices and Patterns*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001. (Cited in pages 33 and 34.)

[ABB06] J. J. Alferes, F. Banti, and A. Brogi. *An Event-Condition-Action Logic Programming Language*, pages 29–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. (Cited in page 105.)

[ABM00] Colin Atkinson, Joachim Bayer, and Dirk Muthig. *Component-Based Product Line Development: The KobrA Approach*, pages 289–309. Springer US, Boston, MA, 2000. (Cited in page 34.)

[ADD06a] Alain April, Jean-Marc Desharnais, and Reiner R. Dumke. A formalism of ontology to support a software maintenance knowledge-based system. In *Proceedings of the Eighteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2006), San Francisco, CA, USA, July 5-7, 2006*, pages 331–336, 2006. (Cited in pages 39 and 40.)

[ADD06b] Alain April, Jean-Marc Desharnais, and Reiner R. Dumke. A formalism of ontology to support a software maintenance knowledge-based system. In *Proceedings of the Eighteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2006), San Francisco, CA, USA, July 5-7, 2006*, pages 331–336, 2006. (Cited in page 39.)

[AdSdLdS04a] A. P. Ambrosio, D. C. de Santos, F. N. de Lucena, and J. C. da Silva. Software engineering documentation: an ontology-based approach. In *WebMedia and LA-Web, 2004. Proceedings*, pages 38–40, Oct 2004. (Cited in page 29.)

[AdSdLdS04b] Ana Paula Ambrosio, Dirson C. de Santos, Fabio N. de Lucena, and Joao Carlos da Silva. Software engineering documentation: An ontology-based approach. In *Joint Conference 10th Brazilian*

Symposium on Multimedia and the Web & 2nd Latin American Web Congress, (WebMedia & LA-Web 2004), 12-15 October 2004, Ribeirao Preto-SP, Brazil, pages 38–40, 2004. (Cited in page 28.)

[Adu] Aduna. https://www.w3.org/2001/sw/wiki/sesame. (Cited in page 20.)

[ADW08] Ahmed Awad, Gero Decker, and Mathias Weske. Efficient compliance checking using BPMN-Q and temporal logic. In *Business Process Management, 6th International Conference, BPM 2008, Milan, Italy, September 2-4, 2008. Proceedings*, pages 326–341, 2008. (Cited in page 73.)

[AGS] Bruno Antunes, Paulo Gomes, and Nuno Seco. Srs: A software reuse system based on the semantic web. (Cited in pages 29 and 38.)

[AKM08] Farhad Arbab, Natallia Kokash, and Sun Meng. Towards using reo for compliance-aware business process modeling. In *Leveraging Applications of Formal Methods, Verification and Validation, Third International Symposium, ISoLA 2008, Porto Sani, Greece, October 13-15, 2008. Proceedings*, pages 108–123, 2008. (Cited in page 72.)

[AOAN12] Fadila Aoussat, Mourad Chabane Oussalah, and Mohamed Ahmed-Nacer. A Domain Ontology for Software Process Architecture Description. In *7th International Conference on Evaluation of Novel Software Approaches to Software Engineering*, wroclaw, Poland, June 2012. (Cited in page 37.)

[API] OWL API. http://owlapi.sourceforge.net/. (Cited in page 115.)

[AT06] A. Akerman and J. Tyree. Using ontology to support development of software architectures. *IBM Systems Journal*, 45(4):813–825, 2006. (Cited in page 37.)

[AVR15] Chauhan A, Vijayakumar V, and Ragala R, editors. *Towards a Multi-level Upper Ontology foundation Ontology Framework as Background Knowledge for Ontology Matching Problem.* Procedia Computer Science, 2015. (Cited in page 28.)

[AW10] Ahmed Awad and Mathias Weske. *Visualization of Compliance Violation in Business Process Models*, pages 182–193. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. (Cited in pages 73 and 74.)

[Bau93] D. Bauer. A reusable parts center [technical forum]. *IBM Systems Journal*, 32(4):620–624, 1993. (Cited in page 32.)

[BC13] Kevin P. Brown and Miriam A. M. Capretz. ODEP-DPS: ontology-driven engineering process for the collaborative development of semantic data providing services. *Information & Software Technology*, 55(9):1563–1579, 2013. (Cited in page 3.)

[BdPL03] K. K. Breitman and J. C. S. do Prado Leite. Ontology as a requirements engineering product. In *Proceedings. 11th IEEE International Requirements Engineering Conference, 2003.*, pages 309–319, Sept 2003. (Cited in page 36.)

[BFK+99] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. Pulse: A methodology to develop software product lines. In *Proceedings of the 1999 Symposium on Software Reusability*, SSR '99, pages 122–131, New York, NY, USA, 1999. ACM. (Cited in page 34.)

[BKB16] M. P. S. Bhatia, Akshi Kumar, and Rohit Beniwal. Ontologies for software engineering: Past, present and future. *Indian Journal of Science and Technology*, 9(9), 2016. (Cited in page 31.)

[BLA11] Pavel Bulanov, Alexander Lazovik, and Marco Aiello. Business process customization using process merging techniques. In *2011 IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2011, Irvine, CA, USA, December 12-14, 2011*, pages 1–4, 2011. (Cited in page 45.)

[Bür07] Tobias Bürger. Putting business intelligence into documents. In *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management SBPM 2007, held in conjunction with the 3rd European Semantic Web Conference (ESWC 2007), Innsbruck, Austria, June 7, 2007*, 2007. (Cited in page 37.)

[CCGR99] Alessandro Cimatti, Edmund M. Clarke, Fausto Giunchiglia, and Marco Roveri. NUSMV: A new symbolic model verifier. In *Computer Aided Verification, 11th International Conference, CAV '99, Trento, Italy, July 6-10, 1999, Proceedings*, pages 495–499, 1999. (Cited in page 45.)

[CE00] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000. (Cited in page 32.)

[CK07a] J. C. Caralt and J. W. Kim. Ontology driven requirements query. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 197c–197c, Jan 2007. (Cited in page 28.)

[CK07b] Jordi Conesa Caralt and Jong Woo Kim. Ontology driven requirements query. In *40th Hawaii International International Conference on Systems Science (HICSS-40 2007), CD-ROM / Abstracts Proceedings, 3-6 January 2007, Waikoloa, Big Island, HI, USA*, page 197, 2007. (Cited in page 35.)

[CL] Clark and Parsia LLC. https://www.w3.org/2001/sw/wiki/pellet. (Cited in pages 20 and 21.)

[Coa96] Workflow Management Coalition. Workflow management coalition terminology and glossary (wfmc-tc-1011), 1996. (Cited in page 47.)

[DALV10] Romain Demeyer, Maxime Van Assche, Ludovic Langevine, and Wim Vanhoof. Declarative workflows to efficiently manage flexible and advanced business processes. In *Proceedings of the 12th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 26-28, 2010, Hagenberg, Austria*, pages 209–218, 2010. (Cited in page 45.)

[DBL04] *Joint Conference 10th Brazilian Symposium on Multimedia and the Web & 2nd Latin American Web Congress, (WebMedia & LA-Web*

2004), 12-15 October 2004, Ribeirao Preto-SP, Brazil. IEEE Computer Society, 2004. (Cited in page 38.)

[DBL05] *12th Asia-Pacific Software Engineering Conference (APSEC 2005), 15-17 December 2005, Taipei, Taiwan*. IEEE Computer Society, 2005. (Cited in page 37.)

[DE05] J. Dietrich and C. Elgar. A formal description of design patterns using owl. In *2005 Australian Software Engineering Conference*, pages 243–250, March 2005. (Cited in page 39.)

[DFSA11] Astrid Duque-Ramos, Jesualdo Tomás Fernández-Breis, Robert Stevens, and Nathalie Aussenac-Gilles. Oquare: A square-based approach for evaluating the quality of ontologies. *Journal of Research and Practice in Information Technology*, 43(2):159–176, 2011. (Cited in page 131.)

[dG15] Klaas Andries de Graaf. Phd thesis:ontology-based software architecture documentation, 2015. (Cited in page 37.)

[DGG+11] Davide D'Aprile, Laura Giordano, Valentina Gliozzi, Alberto Martelli, Gian Luca Pozzato, and Daniele Theseider Dupré. Verifying compliance of business processes with temporal answer sets. In *Proceedings of the 26th Italian Conference on Computational Logic, Pescara, Italy, August 31 - September 2, 2011*, pages 147–161, 2011. (Cited in page 45.)

[dGLT+14] K.A. de Graaf, P. Liang, A. Tang, W.R. van Hage, and H. van Vliet. An exploratory study on ontology engineering for software architecture documentation. *Computers in Industry*, 65(7):1053 – 1064, 2014. (Cited in page 37.)

[dGTLvV12] K. A. de Graaf, A. Tang, P. Liang, and H. van Vliet. Ontology-based software architecture documentation. In *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pages 121–130, Aug 2012. (Cited in pages 37 and 38.)

[DL] OWL DL. https://www.obitko.com/tutorials/ontologies-semantic-web/owl-dl-semantics.html. (Cited in pages v and 18.)

[Elg12] Elgammal. Towards a comprehensive framewcompliacbusiness process compliance, phd thesis, 2012. (Cited in page 69.)

[End93] A. Endres. Lessons learned in an industrial software lab (software development). *IEEE Software*, 10(5):58–61, Sept 1993. (Cited in page 32.)

[ET06] Sandro Etalle and Miroslaw Truszczynski, editors. *Logic Programming, 22nd International Conference, ICLP 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4079 of *Lecture Notes in Computer Science*. Springer, 2006. (Cited in page 105.)

[fAS93] Software Technology for Adaptable and Reliable Systems (STARS). *The Reuse-Oriented Software Evolution (ROSE) Process Model*. PhD thesis, 1993. (Cited in pages 32 and 33.)

[FdMdR98] Ricardo de Almeida Falbo, Crediné Silva de Menezes, and Ana Regina C. da Rocha. *A Systematic Approach for Building Ontologies*, pages 349–360. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. (Cited in page 34.)

[FES05] Alexander Förster, Gregor Engels, and Tim Schattkowsky. Activity diagram patterns for modeling quality constraints in business processes. In *Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005, Proceedings*, pages 2–16, 2005. (Cited in page 71.)

[FI94] W. B. Frakes and S. Isoda. Success factors of systematic reuse. *IEEE Software*, 11(5):14–19, Sept 1994. (Cited in page 32.)

[FK05] W. B. Frakes and Kyo Kang. Software reuse research: status and future. *IEEE Transactions on Software Engineering*, 31(7):529–536, July 2005. (Cited in page 34.)

[FRR] Adriana Maria Figueiredo, Julio C. Dos Reis, and Marcos A. Rodrigues. Improving access to software architecture knowledge an ontology-based search approach. (Cited in page 37.)

[GFd98] M. L. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the rseb. In *Proceedings. Fifth International Conference on Software Reuse (Cat. No.98TB100203)*, pages 76–85, Jun 1998. (Cited in page 32.)

[Gmb] Racer Systems GmbH. https://www.w3.org/2001/sw/wiki/racerpro. (Cited in pages 20 and 21.)

[GMS06] Guido Governatori, Zoran Milosevic, and Shazia Wasim Sadiq. Compliance checking between business processes and business contracts. In *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), 16-20 October 2006, Hong Kong, China*, pages 221–232, 2006. (Cited in page 45.)

[GOSC11] Guido Governatori, Francesco Olivieri, Simone Scannapieco, and Matteo Cristani. Designing for compliance: Norms and goals. In *Rule-Based Modeling and Computing on the Semantic Web, 5th International Symposium, RuleML 2011- America, Ft. Lauderdale, FL, Florida, USA, November 3-5, 2011. Proceedings*, pages 282–297, 2011. (Cited in page 45.)

[GPSV14] Aldo Gangemi, Silvio Peroni, David Shotton, and Fabio Vitali. A pattern-based ontology for describing publishing workflows. In *Proceedings of the 5th International Conference on Ontology and Semantic Web Patterns - Volume 1302*, WOP'14, pages 2–13, Aachen, Germany, Germany, 2014. CEUR-WS.org. (Cited in page 28.)

[GPZ+13] Gerd Gröner, Jeff Z. Pan, Yuting Zhao, Elisa F. Kendall, and Ljiljana Stojanovic. Introduction to the proceedings of the 9th international workshop on semantic web enabled software engineering (SWESE) 2013. In *Service-Oriented Computing - ICSOC 2013 Workshops - CCSA, CSB, PASCEB, SWESE, WESOA, and PhD*

Symposium, Berlin, Germany, December 2-5, 2013. Revised Selected Papers, pages 223–224, 2013. (Cited in page 3.)

[Gri94] M. L. Griss. Software reuse experience at hewlett-packard. In *Proceedings of 16th International Conference on Software Engineering*, pages 270–, May 1994. (Cited in page 32.)

[Gro] Information System Group. http://www.hermit-reasoner.com/. (Cited in page 21.)

[GW95] M. L. Griss and M. Wosser. Making reuse work at hewlett-packard. *IEEE Software*, 12(1):105–107, Jan 1995. (Cited in page 32.)

[HA05] Scott Henninger and Padmapriya Ashokkumar. An ontology-based infrastructure for usability design patterns. In *Proc. Semantic Web Enabled Software Engineering (SWESE*, pages 41–55, 2005. (Cited in page 28.)

[HAHA] Scott Henninger, Padmapriya Ashokkumar, Scott Henninger, and Padmapriya Ashokkumar. An ontology-based metamodel for software patterns. In *Proceeding of the 18 th International Conference on Software Engineering and Knowledge Engineering (SEKE2006*, pages 5–7. (Cited in page 28.)

[HHK+07] Martin Hepp, Knut Hinkelmann, Dimitris Karagiannis, Rüdiger Klein, and Nenad Stojanovic, editors. *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management SBPM 2007, held in conjunction with the 3rd European Semantic Web Conference (ESWC 2007), Innsbruck, Austria, June 7, 2007*, volume 251 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007. (Cited in page 37.)

[HJL+07] J. Han, Y. Jin, Z. Li, T. Phan, and J. Yu. Guiding the service composition process with temporal business rules. In *IEEE International Conference on Web Services (ICWS 2007)*, pages 735–742, July 2007. (Cited in page 70.)

[HKST06] Hans-Jörg Happel, Axel Korthaus, Stefan Seedorf, and Peter Tomczyk. Kontor: An ontology-enabled approach to software reuse. In *IN: PROC. OF THE 18TH INT. CONF. ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING*, 2006. (Cited in pages 29 and 38.)

[HMS03] J. Scott Hawker, H. M. Hong Ma, and R. K. Smith. A web-based process and process models to find and deliver information to improve the quality of flight software. In *Digital Avionics Systems Conference, 2003. DASC '03. The 22nd*, volume 1, pages 3.B.3–31–11 vol.1, Oct 2003. (Cited in page 28.)

[Hol04] Gerard J. Holzmann. *The SPIN Model Checker - primer and reference manual*. Addison-Wesley, 2004. (Cited in page 45.)

[HSGPML14] Brian Henderson-Sellers, Cesar Gonzalez-Perez, Tom Mcbride, and Graham Low. An ontology for iso software engineering standards: 1) creating the infrastructure. *Comput. Stand. Interfaces*, 36(3):563–576, March 2014. (Cited in page 39.)

[HWCK06] D. Hyland-Wood, D. Carrington, and S. Kaplan. Toward a software maintenance methodology using semantic web techniques. In *2006 Second International IEEE Workshop on Software Evolvability (SE'06)*, pages 23–30, Sept 2006. (Cited in page 40.)

[JB04] J. B. Jorgensen and C. Bossen. Executable use cases: requirements for a pervasive health care system. *IEEE Software*, 21(2):34–41, March 2004. (Cited in page 36.)

[JBR99] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. (Cited in page 33.)

[JGJ97] I. Jacobson, M.L. Griss, and P. Jonsson. *Reuse-driven Software Engineering Business (RSEB), Addison-Wesley*. PhD thesis, 1997. (Cited in pages 32 and 33.)

[Joo94] Rebecca Joos. Software reuse at motorola. *IEEE Softw.*, 11(5):42–47, September 1994. (Cited in page 32.)

[KB07] Christoph Kiefer and Abraham Bernstein. Analyzing software with isparql. In *In Proc. of the 3rd Int. Ws. on Semantic Web Enabled Software Engineering*, 2007. (Cited in page 29.)

[Kha15] Aditya Khamparia. Performance analysis of sparql and dl-query on electromyography ontology. *Indian Journal of Science and Technology*, 8(17), 2015. (Cited in page 40.)

[KKL+98] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. Form: A feature-;oriented reuse method with domain-;specific reference architectures. *Annals of Software Engineering*, 5(1):143, 1998. (Cited in page 32.)

[KRG07] Jochen M. Küster, Ksenia Ryndina, and Harald Gall. *Generation of Business Process Models for Object Life Cycle Compliance*, pages 165–181. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. (Cited in pages 68 and 69.)

[LG05] S. W. Lee and R. A. Gandhi. Ontology-based active requirements engineering framework. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 8 pp.–, Dec 2005. (Cited in page 36.)

[LL14] Li Liao and Hareton K. N. Leung. A software process ontology and its application. In *Semantic Web Enabled Software Engineering*, pages 207–217. 2014. (Cited in page 28.)

[MCR06] Viviana Mascardi, Valentina Cordì, and Paolo Rosso. A comparison of upper ontologies. Technical report, 2006. (Cited in page 36.)

[MET02] M. Morisio, M. Ezran, and C. Tully. Success and failure factors in software reuse. *IEEE Transactions on Software Engineering*, 28(4):340–357, Apr 2002. (Cited in page 32.)

[MFCC06] Yannis Manolopoulos, Joaquim Filipe, Panos Constantopoulos, and José Cordeiro, editors. *ICEIS 2006 - Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases*

and Information Systems Integration, Paphos, Cyprus, May 23-27, 2006, 2006. (Cited in page 37.)

[MJ15] S. Murugesh and A. Jaya. Construction of ontology for software requirements elicitation. *Indian Journal of Science and Technology*, 8(29), 2015. (Cited in page 35.)

[MS01] Er Maedche and Steffen Staab. Ontology learning for the semantic web. *IEEE Intelligent Systems*, 16:72–79, 2001. (Cited in page 36.)

[Nei80] James Milne Neighbors. *Software Construction Using Components*. PhD thesis, 1980. AAI8106784. (Cited in page 32.)

[NS07] Kioumars Namiri and Nenad Stojanovic. Pattern-based design and validation of business process compliance. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I*, pages 59–76, 2007. (Cited in page 72.)

[oM] University of Manchester. https://www.w3.org/2001/sw/wiki/kaon2. (Cited in pages 20 and 21.)

[OMG] Omg odm [internet]. http://www.omg.org/cgi-bin/doc?ad/06-05-01.pdf Available from: . 2016 Jan 30. (Cited in page 3.)

[Ont] Ontotext. https://www.w3.org/2001/sw/wiki/graphdb. (Cited in page 20.)

[OWL] https://www.w3.org/OWL/. (Cited in pages 2 and 18.)

[PFS10] Elke Pulvermüller, Sven Feja, and Andreas Speck. Developer-friendly verification of process-based systems. *Knowl.-Based Syst.*, 23(7):667–676, 2010. (Cited in page 45.)

[PGH07] Claus Pahl, Simon Giesecke, and Wilhelm Hasselbring. *An Ontology-Based Approach for Modelling Architectural Styles*, pages

60–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. (Cited in page 37.)

[PJD+05] Tetlow P, Pan JZ, Oberle D, Wallace E, Uschold M, and Kendall E, editors. *Ontology driven architectures and potential uses of the semantic web in systems and software engineering*. W3C Working Draf, 2005. (Cited in page 3.)

[PK15a] Samad Paydar and Mohsen Kahani. A semantic web enabled approach to reuse functional requirements models in web engineering. *Automated Software Engineering*, 22(2):241–288, 2015. (Cited in page 28.)

[PK15b] M. Priya and Ch. Aswani Kumar. A survey of state of the art of ontology construction and merging using formal concept analysis. *Indian Journal of Science and Technology*, 8(24), 2015. (Cited in page 36.)

[PN] Petri net definition. https://en.wikipedia.org/wiki/Petrinet. (Cited in pages 13, 14 and 15.)

[PNT15] Tuan Anh Pham, Thi-Hoa-Hue Nguyen, and Nhan Le Thanh. Ontology-based workflow validation. In *The 2015 IEEE RIVF International Conference on Computing Communication Techno logies - Research, Innovation, and Vision for Future, RIVF 2015, Can Tho, Vietnam, January 25-28, 2015*, pages 41–46, 2015. (Cited in page 66.)

[PT15a] Tuan Anh Pham and Nhan Le Thanh. Checking the compliance of business processes and business rules using OWL 2 ontology and SWRL. In *Proceedings of the Second International Afro-European Conference for Industrial Advancement, AECIA 2015, Villejuif (Paris-sud), France, 9-11 September 2015*, pages 11–20, 2015. (Cited in page 84.)

[PT15b] Tuan Anh Pham and Nhan Le Thanh. Ontology-based business logic layer. In *Poster session of the 9th International Web Rule*

Symposium (RuleML 2015), Berlin, Germany, August 2-5, 2015., 2015. (Cited in page 66.)

[PT15c] Tuan Anh Pham and Nhan Le Thanh. A rule-based language for integrating business process and business rules. In *Proceedings of the RuleML 2015 Challenge, the Special Track on Rule-based Recommender Systems for the Web of Data, the Special Industry Track hosted by the 9th International Web Rule Symposium (RuleML 2015), Berlin, Germany, August 2-5, 2015.*, 2015. (Cited in page 84.)

[PT16a] Tuan Anh Pham and Nhan Le Thanh. Checking the compliance of business process in business process life cycle. In *Proceedings of the RuleML 2016 Challenge, and Industry Track hosted by the 10th International Web Rule Symposium, RuleML 2016, New York, USA, July 6-9, 2016.*, 2016. (Cited in page 66.)

[PT16b] Tuan Anh Pham and Nhan Le Thanh. An ontology-based approach for business process compliance checking. In *Proceedings of the 10th International Conference on Ubiquitous Information Management and Communication, IMCOM 2016, Danang, Vietnam, January 4-6, 2016*, pages 56:1–56:6, 2016. (Cited in page 84.)

[PT17] Tuan Anh Pham and Nhan Le Thanh. Ontology-based business process compliance. *International Journal of Research in Engineering and Science (IJRES)*, 2017. (Cited in page 84.)

[PvdA06] Maja Pesic and Wil M. P. van der Aalst. A declarative approach for flexible business processes management. In *Business Process Management Workshops, BPM 2006 International Workshops, BPD, BPI, ENEI, GPWW, DPM, semantics4ws, Vienna, Austria, September 4-7, 2006, Proceedings*, pages 169–180, 2006. (Cited in page 45.)

[PZ14] Jeff Z. Pan and Yuting Zhao, editors. *Semantic Web Enabled Software Engineering*, volume 17 of *Studies on the Semantic Web*. IOS Press, 2014. (Cited in pages 3 and 28.)

[RDKN03] M. A. Rothenberger, K. J. Dooley, U. R. Kulkarni, and N. Nada. Strategies for software reuse: a principal component analysis of reuse practices. *IEEE Transactions on Software Engineering*, 29(9):825–837, Sept 2003. (Cited in page 32.)

[Rin97] David C. Rine. Success factors for software reuse that are applicable across domains and businesses. In *Proceedings of the 1997 ACM Symposium on Applied Computing*, SAC '97, pages 182–186, New York, NY, USA, 1997. ACM. (Cited in page 32.)

[RK15] Michael Roth and Ewan Klein. *Parsing Software Requirements with an Ontology-based Semantic Role Labeler*, pages 15–21. Association for Computational Linguistics, 4 2015. (Cited in page 36.)

[Sak06] T. Sakalo. Ontology modeling and supporting software system implementation for the informnation sharing processes within labour market. In *2006 International Conference - Modern Problems of Radio Engineering, Telecommunications, and Computer Science*, pages 103–104, Feb 2006. (Cited in page 38.)

[SALM09] Daniel Schleicher, Tobias Anstett, Frank Leymann, and Ralph Mietzner. *Maintaining Compliance in Customizable Process Models*, pages 60–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. (Cited in page 69.)

[SBO07] Rainer Schmidt, Christian Bartsch, and Roy Oberhauser. Ontology-based representation of compliance requirements for service processes. In *Proceedings of the Workshop on Semantic Business Process and Product Lifecycle Management SBPM 2007, held in conjunction with the 3rd European Semantic Web Conference (ESWC 2007), Innsbruck, Austria, June 7, 2007*, 2007. (Cited in page 70.)

[SCK+96] M Simos, D Creps, C Klingler, L Levine, and D Allemang. *Organization Domain Modeling (ODM) Guidebook Version 2.0.* PhD thesis, 1996. (Cited in pages 32 and 33.)

[SEM] https://www.w3.org/standards/semanticweb/. (Cited in page 1.)

[SQu] International organization for standardization (iso) iso/iec 25000 2005, software engineering - soft- ware product quality requirements and evaluation (square). (Cited in page 131.)

[STK+10] David Schumm, Oktay Türetken, Natallia Kokash, Amal Elgammal, Frank Leymann, and Willem-Jan van den Heuvel. Business process compliance through reusable units of compliant processes. In *Current Trends in Web Engineering - 10th International Conference on Web Engineering, ICWE 2010 Workshops, Vienna, Austria, July 2010, Revised Selected Papers*, pages 325–337, 2010. (Cited in page 73.)

[STM] P. Spyns, Y. Tang, and R. Meersman. A model theory inspired collaborative ontology engineering methodology. *Journal of Applied Ontology*. (Cited in page 37.)

[SWR] Semantic web rule language. https://www.w3.org/Submission/SWRL/. (Cited in page 19.)

[TR06] S. Thaddeus and S. V. Kasmir Raja. Ontology-driven model for knowledge-based software engineering. In *Proceedings of the Eighteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2006), San Francisco, CA, USA, July 5-7, 2006*, pages 337–342, 2006. (Cited in page 28.)

[TRPR07] L. Babu T., M. Seetha Ramaiah, T. V. Prabhakar, and D. Rambabu. Archvoc–towards an ontology for software architecture. In *Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent, 2007. SHARK/ADI '07: ICSE Workshops 2007. Second Workshop on*, pages 5–5, May 2007. (Cited in page 38.)

[UJ99] Mike Uschold and Robert Jasper. A framework for understanding and classifying ontology applications. 1999. (Cited in page 34.)

[UPP] Upper ontology [internet]. 2016 jan 30. Available from: http://en.wikipedia.org/wiki/Upper ontology. (Cited in page 28.)

[Usc98] Mike Uschold. Knowledge level modelling: Concepts and terminology. *Knowl. Eng. Rev.*, 13(1):5–29, March 1998. (Cited in page 34.)

[vdA98] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998. (Cited in pages 45 and 49.)

[W3Ca] W3C. https://www.w3.org/rdf/. (Cited in page 2.)

[W3Cb] W3C. https://www.w3.org/rdfs/. (Cited in page 2.)

[W3Cc] W3C. https://www.w3.org/tr/rdf-sparql-query/. (Cited in page 52.)

[WGGM14] Xiaowei Wang, Nicola Guarino, Giancarlo Guizzardi, and John Mylopoulos. Towards an ontology of software: a requirements engineering perspective. In *Formal Ontology in Information Systems - Proceedings of the Eighth International Conference, FOIS 2014, September, 22-25, 2014, Rio de Janeiro, Brazil*, pages 317–329, 2014. (Cited in page 36.)

[WLS+07] Hai H. Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, and Jeff Pan. Verifying feature models using {OWL}. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):117 – 129, 2007. Software Engineering and the Semantic Web. (Cited in page 38.)

[WZR07] René Witte, Yonggang Zhang, and Juergen Rilling. Empowering software maintainers with semantic web technologies. In *In European Semantic Web Conf*, pages 37–52, 2007. (Cited in page 29.)

[YHH+08] Jian Yu, Yan-Bo Han, Jun Han, Yan Jin, Paolo Falcarin, and Maurizio Morisio. Synthesizing service composition models on the basis of temporal business rules. *Journal of Computer Science and Technology*, 23(6):885–894, 2008. (Cited in page 70.)

[YMH+06] Jian Yu, Tan Phan Manh, Jun Han, Yan Jin, Yanbo Han, and Jianwu Wang. Pattern based property specification and verification for service composition. In *Web Information Systems - WISE 2006,*

7th International Conference on Web Information Systems Engineering, Wuhan, China, October 23-26, 2006, Proceedings, pages 156–168, 2006. (Cited in page 71.)

[ZDMP] Yajing Zhao, Jing Dong, Senior Member, and Tu Peng. Ontology classification for semantic-web-based software engineering. (Cited in page 39.)

[ZDP09] Y. Zhao, J. Dong, and T. Peng. Ontology classification for semantic-web-based software engineering. *IEEE Transactions on Services Computing*, 2(4):303–317, Oct 2009. (Cited in page 38.)

[ZJ05a] Xuefeng Zhu and Zhi Jin. Inconsistency measurement of software requirements specifications: an ontology-based approach. In *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, pages 402–410, June 2005. (Cited in page 36.)

[ZJ05b] Xuefeng Zhu and Zhi Jin. Ontology-based inconsistency management of software requirements specifications. In *Proceedings of the 31st International Conference on Theory and Practice of Computer Science*, SOFSEM'05, pages 340–349, Berlin, Heidelberg, 2005. Springer-Verlag. (Cited in page 36.)

[ZKDT09] Lamia Abo Zaid, Frederic Kleinermann, and Olga De Troyer. Applying semantic web technology to feature modeling. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1252–1256, New York, NY, USA, 2009. ACM. (Cited in page 38.)

[ZWRH06] Yonggang Zhang, René Witte, Juergen Rilling, and Volker Haarslev. An ontology-based approach for traceability recovery. In *3rd International Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ATEM 2006*, pages 36–43, 2006. (Cited in page 29.)