Pupa: Moteur de génération des contrats intelligents pour les processus d'affaires avec évènement de minuterie et branchement inclusif

par

Rodrigue TONGA NAHA

MÉMOIRE PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE
AVEC MÉMOIRE EN GÉNIE LOGICIEL
M. Sc. A.

MONTRÉAL, LE 30 NOVEMBRE 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

M. Kaiwen Zhang, directeur de mémoire
Département de génie logiciel et des technologies de l'information, École de technologie supérieure

M. François Coallier, président du jury
Département de génie logiciel et des technologies de l'information, École de technologie supérieure

M. Alain April, membre du jury
Département de génie logiciel et des technologies de l'information, École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE "DATE DE SOUTENANCE"

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je tiens à exprimer mes sincères remerciements à tous ceux et celles qui par leurs travaux, leurs idées, leurs présentations, leurs collaborations, ou leurs relectures ont participé de près ou de loin à la réalisation de ce mémoire.

Je tiens à remercier également les membres du jury d'avoir accepté d'évaluer mon mémoire et examiner le travail que j'ai réalisé durant ces deux dernières années.

Ma plus profonde gratitude va à mon superviseur Professeur Kaiwen Zhang pour sa disponibilité, son encadrement, ainsi que pour ses conseils éclairés qui ont grandement aidé à la complétude de ce mémoire.

Je tiens à exprimer ma très profonde gratitude à mon épouse pour sa présence à mes côtés, son amour, sa confiance et ses sacrifices. À mes enfants, merci d'être toujours une source de motivation pour moi. Je voudrais remercier également mes parents, mes frères et sœurs qui ont toujours cru en moi tout au long de ce parcours.

Enfin, je remercie tous mes amis et collègues du laboratoire de recherche FUSÉE LAB pour leur encouragement.

Pupa: Moteur de génération des contrats intelligents pour les processus d'affaires avec évènement de minuterie et branchement inclusif

Rodrigue TONGA NAHA

RÉSUMÉ

La chaîne de blocs et les autres registres distribués sont reconnus comme des technologies clés aptes à offrir des propriétés de confiance et de transparence. À cet effet, un besoin en matière d'applications décentralisées et capables de déployer les processus d'affaires sur les contrats intelligents se pose. Les solutions existantes de BPM basées sur la chaîne de blocs, prennent en charge diverses plateformes de chaîne de blocs (exemple Ethereum) et différents langages de modélisation de processus (exemple BPMN).

Cependant, à cause des limitations liées à la programmation des contrats intelligents notamment l'impossibilité de planifier l'exécution d'une transaction, la majorité de ces approches ne soutient pas les processus métiers avec les évènements de minuterie et décisions inclusives. Afin de renverser les défis mentionnés ci-dessus, nous proposons un moteur logiciel appelé Pupa. Pupa est une solution décentralisée basée sur Ethereum capable de convertir les processus d'affaires avec évènement de minuterie et branchement inclusif en contrat intelligent.

Pour réaliser ce service, notre solution ajoute les fonctionnalités de la tâche utilisateur à l'évènement de minuterie et augmente des procédures de contrôle sur la tâche succédant la minuterie. De plus, la solution Pupa implémente des variables d'écoute sur les flux de séquence des branchements inclusifs. Nous avons implémenté notre solution en allongeant la solution Caterpillar. C'est une solution BPMN existante qui utilise Solidity et Ethereum. Nous avons évalué les performances de Pupa et nous l'avons comparée à celle de Caterpillar. Nos résultats montrent que Pupa, en plus de supporter des éléments additionnels de BPMN, est assez performante en termes de coût de déploiement et de nombre de lignes du code généré.

Mots-clés: Chaîne de blocs, Business Process Management, Contrat intelligent, BPMN, Événement de minuterie, Branchement inclusif

Pupa : Smart Contracts for BPMN with Time-Dependent Events and Inclusive Gateways

Rodrigue TONGA NAHA

ABSTRACT

The digital transformation of business processes face a major hindrance due to the lack of trust and transparency. As blockchain and other distributed ledger (DLT) are considered key enabling technologies, there is a need for supporting tools which can deploy business models over smart contracts in order to leverage these decentralized platforms. Existing blockchain-based Business Process Management (BPM) solutions support various blockchain platforms and different types of modelling language, i.e., Ethereum and Business Process Model Notation (BPMN).

However, the majority of these methods do not support processes with time events and inclusive gateways due to severe limitations imposed by smart contract programming languages. In other words, mainstream blockchain platforms do not offer a straightforward way to execute a transaction at a later time. To overcome these aforementioned issues, we propose an engine called Pupa, a blockchain-based decentralized protocol to translate business processes with time events and inclusive gateways to smart contracts. Pupa accomplishes this by adding task feature to time events, check function on top of activities succeeding time events and, listening variables to sequence flow forking or joining inclusive gateways.

We implemented Pupa by extending Caterpillar, an existing BPMN solution using Solidity and Ethereum, and evaluated the performance of our proposed engine and its generated smart contracts with a baseline solution. Our results show that Pupa is competitive with baseline solutions in terms of cost and performance, while offering additional advantages in terms of decentralization and supporting additional BPMN semantics.

Keywords: Blockchain, Business Process Management, Smart Contract, BPMN, Timer event, Inclusive gateway

TABLE DES MATIÈRES

Page

INTRODUCTION 1

CHAPITRE 1 NOTIONS DE BASE 5
1.1 Chaîne de blocs 5
1.1.1 Architecture 5
1.1.2 Fonctionnement des transactions 7
1.1.3 Types de système de chaîne de blocs 8
1.2 Ethereum 9
1.2.1 Comptes Ethereum 10
1.2.2 Machine virtuelle Ethereum 10
1.2.3 Contrats intelligents et Solidity 10
1.2.4 Coûts d'exécution 11
1.3 Gestion par processus d'affaires 12
1.3.1 BPM et BPMS 12
1.3.2 BPMN 14
1.4 La minuterie et la décision inclusive de BPMN 15
1.4.1 L'évènement de minuterie 16
1.4.2 Le branchement inclusif 17
1.5 Conclusion 18

CHAPITRE 2 REVUE DE LA LITTÉRATURE 19
2.1 Moteurs de génération des processus d'affaires basés sur les chaînes de blocs 19
2.2 Événement de minuterie dans la gestion par processus d'affaires basée sur la chaîne de bloc 23
2.3 Branchement inclusif dans la gestion par processus d'affaires basée sur la chaîne de bloc 24
2.4 Conclusion 25

CHAPITRE 3 ARCHITECTURE ET CONCEPTION DE LA SOLUTION PUPA 27
3.1 Architecture de la solution 27
3.1.1 Les composants on-chain 28
3.1.1.1 Le magasin des processus 28
3.1.1.2 Les composants contrats intelligents 29
3.1.1.3 Les logs 30
3.1.2 Les composants off-chain 30
3.1.3 Le portail web 31
3.2 Conception de Pupa 32
3.2.1 Spécifications fonctionnelles 32
3.2.1.1 Événement de minuterie 32
3.2.1.2 Branchements inclusifs 33

3.2.2 Séquences d'exécution 35
3.2.2.1 Événement de minuterie 35
3.2.2.2 Branchements inclusifs 37
3.2.3 Spécifications non fonctionnelles 38
3.2.3.1 Sécurité des contrats intelligents 38
3.2.3.2 Traçabilité des processus 39
3.3 Conclusion 39

CHAPITRE 4 IMPLÉMENTATION DE LA SOLUTION PUPA 41
4.1 Description étude de cas : gestion des commandes chez une entreprise fournisseur d'accès d'internet 41
4.1.1 Diagramme BPMN 41
4.1.2 Spécifications conceptuelles du modèle du processus d'affaires 42
4.1.2.1 Au niveau de la racine du modèle 42
4.1.2.2 Au niveau de l'évènement de minuterie 43
4.1.2.3 Au niveau de la tâche qui précède le branchement inclusif divergent 44
4.1.2.4 Au niveau des flux de séquences associés aux branchements inclusifs 44
4.2 Conversion des processus d'affaires en contrats intelligents 45
4.2.1 Les étapes de conversion des diagrammes BPMN en contrats intelligents 46
4.2.2 Les composants d'exécution d'un processus d'affaires 47
4.2.2.1 La fonction *step* 48
4.2.2.2 La variable *marking* 48
4.2.2.3 Les opérations de bit à bit 49
4.3 Génération des contrats intelligents de l'étude de cas 50
4.3.1 Intégration de l'évènement de minuterie 50
4.3.2 Intégration des branchements inclusifs divergents et convergents 51
4.4 Conclusion 54

CHAPITRE 5 ÉVALUATION DE LA SOLUTION ET DISCUSSION DES RÉSULTATS 57
5.1 Environnement de travail 57
5.1.1 Composants off-chain 57
5.1.2 Composants on-chain 58
5.2 Métriques de performance 59
5.2.1 Consommation du gaz 59
5.2.2 Taille de la transaction 60
5.2.3 Le nombre de lignes de code générées 60
5.3 Comparaison avec la solution Caterpillar 60
5.3.1 Consommation de gaz 61
5.3.2 La limite de gaz autorisée par une transaction 62
5.3.3 Lignes de code générées 62

5.4 Évaluation de la solution Pupa 63
5.4.1 Impact de la taille du processus sur la consommation de gaz 63
5.4.2 Impact de la taille du processus sur la limite de gaz de la transaction 64
5.4.3 Impact de l'évènement de minuterie et des tâches utilisateurs sur le nombre de lignes du contrat 64
5.5 Conclusion 65

CHAPITRE 6 EXPÉRIENCE PERSONNELLE 67

CHAPITRE 7 CONCLUSION ET RECOMMANDATIONS 71

ANNEXE I PUPA : SMART CONTRACTS FOR BPMN WITH TIME-DEPENDENT EVENTS AND INCLUSIVE GATEWAYS 73

ANNEXE II CONTRAT INTELLIGENT GÉNÉRÉ POUR LE PROCESSUS D'AFFAIRE DU CAS D'UTILISATION 93

ANNEXE III LOG D'EXÉCUTION DU CAS D'UTILISATION 101

LISTE DE RÉFÉRENCES 107

LISTE DES TABLEAUX

Page

Tableau 1.1 Comparaison des types de chaînes de blocs 9

Tableau 4.1 Exemple d'expressions qui réalisent des opérations de bit à bit 50

Tableau 5.1 Comparaison du gaz utilisé pour déployer les solutions Caterpillar et Pupa 61

LISTE DES FIGURES

Page

Figure 1.1 Architecture des chaînes de blocs 8

Figure 1.2 Cycle de vie de BPM Adapté de Sturm et al.(2019, p.2) 14

Figure 1.3 Types d'évènements de minuterie de BPMN 17

Figure 1.4 Exemple de branchement inclusif 18

Figure 3.1 Architecture Pupa Adaptée de Lòpez-Pintado et al.(2019, p.8) 28

Figure 3.2 Séquences d'exécution de l'évènement de minuterie 36

Figure 3.3 Séquences d'exécution du branchement inclusif 38

Figure 4.1 Diagramme BPMN du cas d'utilisation 42

Figure 4.2 Spécifications conceptuelles des séquences des branchements 45

Figure 4.3 Étapes de compilation de Pupa Adaptées de Lòpez-Pintado et al.(2019, p.12) 47

Figure 4.4 Index des éléments BPMN du processus d'affaires 49

Figure 5.1 Interface de traçabilité des processus 59

Figure 5.2 Consommation de gaz - Caterpillar vs Pupa 62

Figure 5.3 Taille processus/nombre de lignes de code générées - Caterpillar vs Pupa 63

Figure 5.4 Impact de la taille du processus sur la consommation du gaz 64

Figure 5.5 Impact de la taille du processus sur le nombre de lignes de code générées 65

LISTE DES ALGORITHMES

Page

Algorithme 4.1 Spécifications conceptuelles variables globales de marquage des branchements inclusifs 43

Algorithme 4.2 Spécifications conceptuelles de l'évènement de minuterie 44

Algorithme 4.3 Spécifications tâche précédant le branchement inclusif divergent 44

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ABI	Application Binary Interface
BPM	Business Process Management
BPMN	Business Process Model and Notation
BPMS	Business Process Management System
EJS	Embedded JavaScript Templating
HTML	HyperText Markup Language
IDE	Integrated Development Environment
IDM	Ingénierie Dirigée par les Modèles
IPFS	InterPlanetary File System
MVE	Machine Virtuelle d'Ethereum
OMG	Object Management Group
XML	Extensible Markup Language

LISTE DES SYMBOLES ET UNITÉS DE MESURE

CAD	Dollar canadien
gaz	Unité de mesure utilisée pour exprimer le coût de calcul de l'exécution des opérations sur Ethereum
Ether	Cryptomonnaie utilisée par Ethereum

INTRODUCTION

L'une des révolutions technologiques les plus marquantes de ces dernières années est la technologie de chaîne de blocs. Grâce à ses propriétés innovantes telles que la décentralisation, la suppression des parties tiers, le stockage sécurisé des données, l'immutabilité, la technologie de chaîne de blocs a la capacité de transformer les applications d'entreprise (Nakamoto, 2008; Norman, 2017). Dans le domaine de la gestion collaborative des processus d'affaires, la technologie de chaîne de blocs permet à plusieurs parties sans confiance de coopérer sans avoir besoin d'une autorité centrale. En effet, la centralisation des informations présente de nombreux risques en matière de sécurisation des données. De même, les interactions et les accords entre plusieurs organisations dans un environnement centralisé exposent des risques en matière de violabilité des données. Ainsi, il apparaît clairement que les applications standards centralisées de gestion interorganisationnelle des processus métiers présentent de nombreuses limites notamment en matière de transparence et de traçabilité. Afin de remédier à ces limites, des systèmes combinant la technologie de chaîne de bloc aux systèmes de gestion des processus métiers ont vu le jour.

La majorité de ces systèmes fonctionne sur la chaîne de blocs Ethereum et par conséquent bénéficie des avantages du concept de contrat intelligent. Les contrats intelligents sont des programmes qui lorsqu'ils sont déployés, sont capables de gérer et de valider les informations issues des processus métiers. Une solution capable de réaliser cet objectif est la production manuelle des contrats intelligents à l'aide de programmes spécialisés. Toutefois, afin de faire abstraction de la technicité liée à l'écriture des contrats intelligents et par ailleurs réduire les barrières d'adoption de la technologie, il est préférable d'utiliser des outils de conversion. Les outils de conversion se chargeront de traduire de modèles de processus d'affaires (notamment BPMN) en contrats intelligents (par exemple les contrats écrits en langage Solidity).

En effet, plusieurs travaux ont été effectués parmi lesquels la solution Caterpillar (López-Pintado, García-Bañuelos, Dumas & Weber, 2017; Lòpez-Pintado, García-Bañuelos, Dumas, Weber & Ponomarev, 2019), la solution Lorikeet (Tran, Lu & Weber, 2018). Nous retrouvons également les solutions suivantes :(Corradini *et al.*, 2020; Sturm, Szalanczi, Schönig & Jablonski, 2018; Nakamura, Miyamoto & Kudo, 2018). Ces outils ont la capacité de convertir des modèles BPMN de processus d'affaires en contrats intelligents prêts à être déployés sur la chaîne de blocs. Cependant, ces outils ne prennent pas en charge dans le processus de génération des contrats intelligents deux éléments fondamentaux du langage BPMN à savoir : les évènements de minuterie et les branchements inclusifs. Les évènements de minuterie sont utilisés pour illustrer des évènements déclenchés après un certain montant de temps prédéfini. Les branchements inclusifs quant à eux, sont utilisés pour intégrer au sein d'un processus d'affaires des décisions pouvant être à la fois alternatives et parallèles. L'absence de prise en charge de ces deux éléments participe à produire des contrats intelligents qui n'exécutent pas le comportement exprimé par le processus métier. De plus, la portée des processus d'affaires pouvant bénéficier des avantages de la technologie de chaîne de blocs se trouve réduite. La difficulté à implémenter les évènements de minuterie provient du fait que les contrats intelligents sous Ethereum sont sujets à des restrictions attachées aux contraintes temporelles. En effet, il n'est pas possible de planifier l'exécution d'une tâche sur la chaîne de blocs Ethereum. En ce qui concerne les branchements inclusifs, la combinaison des décisions parallèles et exclusives contribue déjà à complexifier son utilisation. De plus, l'intégration de cet élément requiert une grande flexibilité durant la conception.

Afin de combler l'écart existant dans les solutions actuelles de gestion de processus d'affaires basée sur la chaîne de blocs, nous présentons une nouvelle solution appelée Pupa. Pupa est un moteur logiciel qui prend en entrée un diagramme BPMN et génère des contrats intelligents écrits en code Solidity qui peuvent être déployés sur Ethereum. Au-delà de la prise en charge des évènements de minuterie et des branchements inclusifs lors de la génération des contrats intelligents, la solution Pupa supporte une version récente et plus sécurisée de contrats intelligents.

De plus, la solution Pupa fournit une fonctionnalité qui permet l'auditabilité des processus exécutés sur la chaîne de blocs.

Dans le cadre de notre mémoire, nos contributions principales sont les suivantes :

1. Nous prenons en charge les évènements de minuterie lors de la génération des contrats intelligents. Notre solution supporte les contraintes temporelles de type chronomètre, intermédiaires entre deux activités. La durée du chronomètre est spécifiée par l'utilisateur, la vérification de la fin du chronomètre est effectuée durant l'exécution des contrats intelligents,
2. Nous prenons en charge les branchements inclusifs à travers l'utilisation de variables de marquage tant dans la conception du modèle BPMN que dans les contrats intelligents. Les contrats intelligents déployés sont capables de supporter convenablement les comportements de divergence et les comportements de convergence du branchement inclusif tels que définis dans le standard BPMN 2.0,
3. Notre solution est implémentée par extension de l'outil Caterpillar. Nous avons évalué le code généré par Pupa et nous l'avons comparé par rapport à Caterpillar en termes de coût et de performance.

La solution Pupa a fait l'objet d'un article intitulé « Pupa : Smart Contracts for BPMN with Time-Dependent Events and Inclusive Gateways » qui a été publié à la conférence dénommée « International Conference on Business Process Management (BPM 2022, Münster, Germany) ».

À la suite de ce qui précède, nous allons énoncer le plan de notre mémoire. Tout d'abord dans le chapitre 1, nous présentons les notions de base qui vont permettre de mieux appréhender les concepts clés qui sont les plus utilisés dans le cadre de notre travail. Dans le chapitre 2, nous proposons une revue de littérature à travers laquelle nous parcourons les travaux de recherche qui se sont focalisés sur les outils de gestion de processus d'affaires basés sur la chaîne de bloc. Par la suite, nous présentons un état des lieux des solutions existantes de gestion de processus d'affaires basées sur la chaîne de blocs avec, prise en charge des évènements de minuterie et des

branchements inclusifs. Dans le chapitre 3, nous fournissons les détails de l'architecture de Pupa ainsi que les différentes étapes de conception qui ont été prises en compte. Dans le chapitre 4, nous présentons l'implémentation de la solution à travers un cas d'utilisation. Ici, nous détaillons les spécifications conceptuelles à prendre en compte lors de la réalisation du modèle BPMN du processus. Par la suite, nous examinons le processus de conversion du modèle BPMN en contrats intelligents. Dans la dernière partie du chapitre, nous présentons l'implémentation du suivi d'exécution du processus d'affaires sur la chaîne de bloc au niveau de l'évènement de minuterie et des différents branchements inclusifs. Le chapitre 5 est consacré à l'évaluation et à la comparaison de la solution Pupa avec l'outil Caterpillar. Par la suite, une discussion des résultats est fournie. Dans le chapitre 6, nous faisons une synthèse du parcours de deux années de ce mémoire ainsi que les principaux challenges rencontrés durant la réalisation de ce travail. En dernier lieu, dans le chapitre 7 nous allons conclure notre travail et proposer des recommandations qui pourront servir de base pour des recherches futures.

CHAPITRE 1

NOTIONS DE BASE

La révolution financière apportée par les monnaies cryptographiques a grandement contribué à faire connaître la technologie de chaîne de blocs. Au-delà de soutenir les cryptomonnaies telles que le Bitcoin, la technologie des chaînes de bloc permet également de réaliser des applications décentralisées par l'implémentation du concept de contrat intelligent. À cet effet, dans le domaine de la gestion par processus d'affaires, la technologie de chaîne de blocs peut être utilisée pour garantir le respect des accords entre différentes parties collaborants dans un environnement sans confiance. Dans ce chapitre, nous allons présenter les concepts de base majoritairement utilisés dans le cadre de ce travail. Après avoir défini la technologie des chaînes de blocs, nous présenterons les types de chaînes de blocs. Un accent sera mis sur la chaîne de bloc Ethereum. Ensuite, nous allons introduire les notions concernant la gestion par processus d'affaires. Enfin, une grande attention sera portée sur deux éléments utilisés dans la modélisation des processus d'affaires à savoir : les évènements de minuterie et les branchements inclusifs.

1.1 Chaîne de blocs

La chaîne de blocs est une technologie émergente qui dispose de plusieurs propriétés qui la rendent très attractive dans plusieurs secteurs d'activité. Parmi ces secteurs, on peut citer notamment le domaine de la gestion des procédés d'affaires. Dans cette section, nous allons présenter la technologie des chaînes de blocs, son architecture, son fonctionnement, ainsi que les types de chaînes de blocs.

1.1.1 Architecture

La technologie de chaîne de blocs est une technologie de registre distribué qui produit une liste immuable de blocs liés les uns après les autres, dans un réseau décentralisé. Cette innovation

technologique est rendue possible grâce à la combinaison des réseaux pair à pair, des mécanismes de consensus, de la cryptographie et des mécanismes du marché (Mendling *et al.*, 2018).

La notion de bloc renvoie à un ensemble de données et d'états stockés de manière successive. La notion de chaîne se traduit par le fait que chaque bloc est lié de manière cryptographique à son parent. Ainsi, la chaîne de blocs est une liste enchaînée de blocs contenant un ensemble de transactions et de métadonnées de ces transactions. Ces blocs sont distribués à travers un réseau pair à pair où chaque nœud du réseau maintient la version la plus récente de la liste. L'ajout d'un nouveau bloc à la chaîne est validé par des techniques cryptographiques. Chaque bloc est associé à une fonction de hachage qui prend en entrée le contenu du bloc ainsi que la signature du bloc précédent afin de produire la signature unique du bloc. L'utilisation d'une fonction de hachage cryptographique permet de prévenir tout changement malicieux sur une transaction. En effet, toute tentative de modification d'une transaction incluse dans un bloc nécessitera le calcul de la fonction de hachage de tous les blocs créés après le bloc modifié. Cependant, la réalisation de ce calcul exige des coûts énormes de traitement ce qui contribue, à décourager tout éventuel adversaire. Le mécanisme de création de blocs encore appelé minage est fait par un consensus. Le consensus entre les nœuds du réseau concernant la nouvelle version de la chaîne de blocs incluant le nouveau bloc doit être fait. Le mécanisme de consensus permet de garantir l'inviolabilité de la chaîne de blocs entre plusieurs nœuds méfiants les uns envers les autres. Les algorithmes de consensus utilisés par la technologie de chaîne de blocs sont basés sur différentes techniques. Parmi ces techniques, on peut citer la preuve de travail, la preuve de participation. La preuve de travail est un concept où le minage du prochain bloc valide exige la résolution d'un puzzle cryptographique avant les autres mineurs. Quant à la preuve de participation, elle consiste à choisir les valideurs de nouveaux blocs sur la base de leur participation (niveau de possession d'une certaine quantité) dans la cryptomonnaie associée à la plateforme de chaîne de blocs.

1.1.2 Fonctionnement des transactions

La chaîne de bloc est utilisée dans un environnement distribué où les données sont partagées entre plusieurs utilisateurs ayant peu ou pas de confiance entre eux. Ainsi, les utilisateurs peuvent émettre des transactions qui vont modifier le registre. À cet effet, les transactions apparaissent comme étant des composants critiques du système de chaîne de bloc en raison leur capacité à modifier l'état de la chaîne. À une transaction sont associées plusieurs informations parmi lesquelles, l'adresse de l'émetteur, l'adresse du récepteur et le solde du compte de l'émetteur en monnaie cryptographique. Lorsqu'une nouvelle transaction est émise, elle est signée numériquement avec la clé privée de l'émetteur. La transaction signée sera ensuite diffusée sur le réseau pair à pair de la chaîne de blocs afin d'être vérifiée et ajoutée à la chaîne. Les nœuds spéciaux du réseau notamment les mineurs / valideurs vont collecter les transactions, les vérifier et les inclure dans leur groupe de transactions non confirmées. L'une des vérifications effectuées ici concerne le contrôle du solde de l'émetteur afin de savoir s'il dispose du montant nécessaire pour supporter l'exécution de la transaction. Les transactions valides seront regroupées afin de construire un bloc en fonction de l'algorithme de consensus utilisé. Lorsque le processus de création de blocs est réussi, le nouveau bloc est diffusé sur le réseau pour vérification et acceptation. Les autres nœuds vont vérifier le nouveau bloc et l'ajouter dans leur propre copie de la chaîne s'il est valide. En cas de division (fork en anglais) c'est-à-dire l'apparition plusieurs chaînes différentes dans le réseau, la chaîne la plus longue sera choisie comme la chaîne principale.

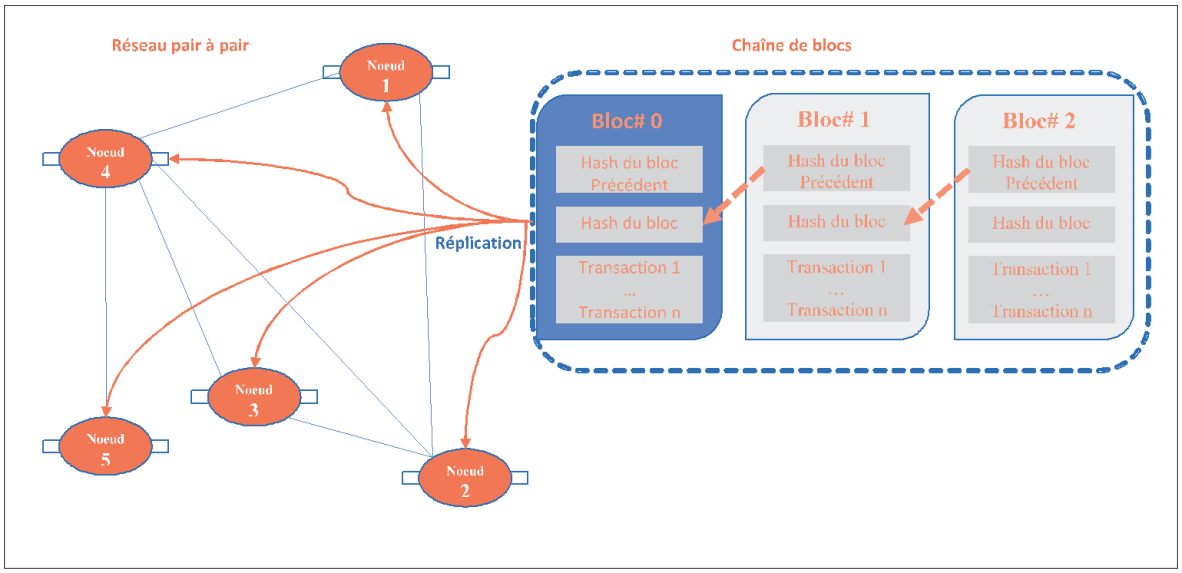


Figure 1.1 Architecture des chaînes de blocs

1.1.3 Types de système de chaîne de blocs

La divergence de motivation des applications basées sur la chaîne de bloc permet de classifier les chaînes de blocs en trois groupes : Les chaînes de blocs publiques, les chaînes de blocs privées et les chaînes de blocs consortium (Buterin, 2015).

- **La chaîne de blocs publique :** la chaîne de blocs publique est une chaîne de blocs décentralisée où n'importe qui peut joindre le réseau, être capable de lire, écrire ou participer au consensus de validation des blocs sans permission. Toutes les données de transactions sur ce type de chaîne de blocs sont publiques. Ce type de chaîne de bloc est vulnérable aux risques liés à la confidentialité des données, ainsi qu'aux attaques à 51%. Toutefois, la décentralisation offerte par la chaîne de blocs publique protège les utilisateurs des développeurs ou de toute autre entité centrale de contrôle,
- **La chaîne de blocs privée :** c'est une chaîne de blocs avec permission où, les utilisateurs doivent avoir l'autorisation pour joindre le réseau. De même, pour lire, écrire ou valider les transactions, les utilisateurs doivent avoir la permission. Ce type de chaîne de blocs est centralisé, plus évolutif, moins exposé aux problèmes de confidentialité et aux attaques à

51%. Il est plus utilisé au sein d'organisations ayant le même objectif mais qui ne se font pas confiance entre eux,

- **La chaîne de blocs consortium :** une chaîne de blocs consortium est une chaîne de blocs avec permission qui est située sur la limite entre la chaîne de blocs privée et la chaîne de blocs publique. Elle est utilisée par des organisations indépendantes qui échangent des informations entre elles avec peu ou pas de confiance. Les valideurs sont des nœuds présélectionnés à l'avance. La chaîne de blocs consortium est partiellement centralisé

Tableau 1.1 Comparaison des types de chaînes de blocs

Chaîne de blocs	**Participation**	**Visibilité des membres**	**Sécurité**	**Centralisation**	**Évolutivité**	**Exemple**
Public	Sans permission	Inconnu	Excellent	Décentralisé	Faible	Bitcoin, Ethereum
Privée	Avec permission	Connu	Bon	Centralisé	Élevé	Blockstack, Multichain
Consortium	Avec permission	Connu	Très bon	Partiel	Modéré	Hyperledger, Quorum

1.2 Ethereum

Ethereum est un protocole d'échange décentralisé qui permet aux utilisateurs d'écrire des contrats intelligents et des applications décentralisées grâce à un langage de programmation Turing Complete (Buterin *et al.*, 2013). Ether est la monnaie cryptographique d'Ethereum. Elle est utilisée comme moyen de paiement des frais de transaction. En juin 2022, Ether est la deuxième plus grande cryptomonnaie au monde avec une capitalisation de plus de 200 milliards CAD (Forbes, 2022). Dans la suite de cette section, nous allons fournir plus de détails sur la chaîne de blocs Ethereum. Après avoir expliqué la notion de compte dans Ethereum, nous présenterons le composant d'exécution du code encore appelé Machine Virtuelle Ethereum. Ensuite nous parlerons des contrats intelligents, du langage de programmation Solidity et du concept de gaz.

1.2.1 Comptes Ethereum

La notion de compte Ethereum joue un rôle important dans le processus de transition d'état de la chaîne. Il existe deux types de comptes Ethereum : (1) les comptes utilisateurs ou encore comptes contrôlés en externe et (2) les comptes de contrats. Un compte utilisateur possède un pair de clé publique privée et un solde d'Ether. Il n'est pas associé à un code, mais il peut démarrer l'exécution d'une transaction avec d'autres comptes utilisateurs ou des comptes contrats. Un compte de contrat quant à lui est associé à un code, il peut échanger des transactions internes avec d'autres contrats, il peut être invoqué par des comptes utilisateurs. Le compte de contrat possède un solde d'Ether.

1.2.2 Machine virtuelle Ethereum

La machine virtuelle d'Ethereum (MVE) est un composant d'exécution présent dans chaque nœud complet. Pour exécuter une tâche spécifique, la MVE utilise un ensemble d'instructions-machine appelé « Opcodes ». Bien que cet ensemble d'instruction soit limité, il est assez suffisant pour qualifier la MVE de Turing complet. Afin d'optimiser le stockage des « Opcodes », ils sont encodés en bytecode. À chaque « Opcode » est alloué un bit, ainsi le nombre maximum « d'opcode » est 256 (16^2). La MVE fonctionne comme une machine virtuelle en forme de pile avec une profondeur de 1024 éléments. Chaque élément de la pile a une taille de 256 bits et seuls les 16 premiers éléments de la pile sont accessibles à un moment donné durant l'exécution. À cause de ces limitations, certains opcodes utilisent des contrats non persistants pour échanger les données.

1.2.3 Contrats intelligents et Solidity

Le concept de contrat intelligent est abordé pour la première fois en 1997 par Szabo (Szabo, 1997). Avec l'émergence des chaînes de blocs 2.0, ce concept va davantage se répandre. Les contrats intelligents sont des programmes qui s'exécutent automatiquement à la suite de certains

éléments déclencheurs préalablement remplis et qui sont déployés directement sur la chaîne de blocs. Les contrats intelligents aident la chaîne de blocs à maintenir la confiance entre plusieurs parties méfiantes.
Solidity est un des langages de programmation orientés contrat le plus actif et le plus maintenu. Solidity est un langage orienté objet, un langage d'accolade qui a été profondément influencé par le langage C++. Solidity est un langage statiquement typé qui supporte l'héritage, les librairies et permet la définition de types de données complexes par l'utilisateur.

1.2.4 Coûts d'exécution

Étant donné que chaque participant au réseau Ethereum peut exécuter un contrat intelligent, un utilisateur malicieux pourrait essayer de surcharger le réseau en créant des contrats contenant des opérations qui exigent d'énormes calculs. Afin d'éviter que ce type d'action ne se produise que ce soit de manière accidentelle ou intentionnelle, le code à exécuter de chaque transaction est soumis à une limite liée au coût de traitement. L'unité de base de ce coût de traitement est appelée gaz. Ainsi, chaque transaction contient une limite de gaz ; lorsque la transaction est exécutée et il y'a reste de gaz, ce gaz est retourné à l'utilisateur. Dans le cas où la transaction est exécutée avec un gaz déficitaire, la transaction est reversée, mais la quantité limitée de gaz fournie sera consommée.
Le coût de déploiement d'un contrat intelligent est fonction de la taille du contrat et est payable en gaz. Chaque contrat est associé à une adresse de hachage unique et une mémoire de travail. Tout utilisateur ou application externe qui connait l'adresse du contrat peut exécuter toutes les fonctions publiques déclarées par ce contrat. Ce qui contribuerait également à modifier l'état de la chaîne de blocs. Cependant toute transaction exécutée sur la chaîne de blocs est également soumise aux propriétés de transparence et d'inviolabilité offertes par la technologie des chaînes de bloc (Antonopoulos & Wood, 2018). Ainsi, l'exécution d'un contrat intelligent sur Ethereum peut contribuer grandement à maintenir de la confiance entre plusieurs parties méfiantes qui collaborent entre elles.

1.3 Gestion par processus d'affaires

La technologie des chaînes de blocs étant donné ses propriétés est une solution attractive pour la gestion collaborative des processus d'affaires entre plusieurs organisations sans confiance. Dans cette section, nous allons présenter en détail la notion de gestion par processus d'affaires.

1.3.1 BPM et BPMS

BPM (Business Process Management en anglais) signifie la gestion par processus d'affaires ou encore la gestion des processus métiers. Il s'agit d'un ensemble de techniques permettant la conception, l'exécution, le suivi et l'amélioration des processus d'affaires (Van Der Aalst, La Rosa & Santoro, 2016). Un processus d'affaires peut être défini comme une série d'activités qui sont exécutées au sein d'une organisation ou au sein d'un environnement technique dans le but de réaliser un objectif défini. Un processus d'affaires au sein de BPM passe par cinq phases qui caractérisent le cycle de vie d'un processus. Au début le processus est identifié, ensuite il est conceptualisé et modélisé. Ensuite le processus est implémenté et configuré. Après l'implémentation, le processus est exécuté et surveillé. À la dernière étape, le processus est amélioré suivant le résultat du diagnostic de surveillance (Sturm, Scalanczi, Schönig & Jablonski, 2019).

- **Phase d'identification :** à la phase d'identification, il faut identifier les processus existants, les énumérer, les prioriser. Ensuite, il faut découvrir la portée de chaque processus trouvé et les relations entre les processus,
- **Phase de conception et modélisation :** à la phase de conception et modélisation, il est question dans un premier temps de proposer une conception théorique du processus d'affaires futur en se basant sur les processus existants. Par la suite il sera question de modéliser le processus d'affaires. La modélisation du processus d'affaires va consister à représenter graphiquement le modèle. Le langage de modélisation le plus souvent utilisé ici est le BPMN,

- **Phase d'implémentation et de configuration :** à la phase d'implémentation et de configuration, il est question de configurer le processus afin de le rendre compréhensible par les moteurs d'exécution des processus d'affaires,
- **Phase d'exécution et de surveillance du processus :** après l'implémentation et la configuration, le processus est exécuté et surveillé. Ici, le processus est testé et déployé dans un environnement d'exécution. Les erreurs de conception, les goulots d'étranglement qui apparaissent durant l'exécution sont sauvegardés dans des logs et analysés dans la dernière phase appelée la phase de diagnostic et d'amélioration,
- **Phase de diagnostic et d'amélioration :** à la phase de diagnostic et d'amélioration, les logs précédemment sauvegardés sont analysés et des ajustements sont apportés au processus

L'environnement technique dans lequel le processus d'affaires est exécuté peut-être un logiciel. Ce type de logiciel appartient à la famille des applications centrées processus. Parmi les logiciels centrés processus, BPMS (Business Process Management System) encore appelé système de gestion des processus métier fait partie de la catégorie qui supporte toutes les étapes du cycle de vie d'un processus (Dumas, Rosa, Mendling & Reijers, 2018). Un BPMS est outil qui permet l'automatisation des processus métiers au sein d'une entreprise. Dans un environnement interorganisationnel avec peu ou pas de confiance entre les parties, l'utilisation de solutions traditionnelles centralisées BPMS présente un risque de violabilité, de transparence et de sécurité des données. Afin de remédier à ces limitations des solutions actuelles BPMS, de nouvelles solutions BPMS basées sur la technologie de chaîne de blocs ont émergé.

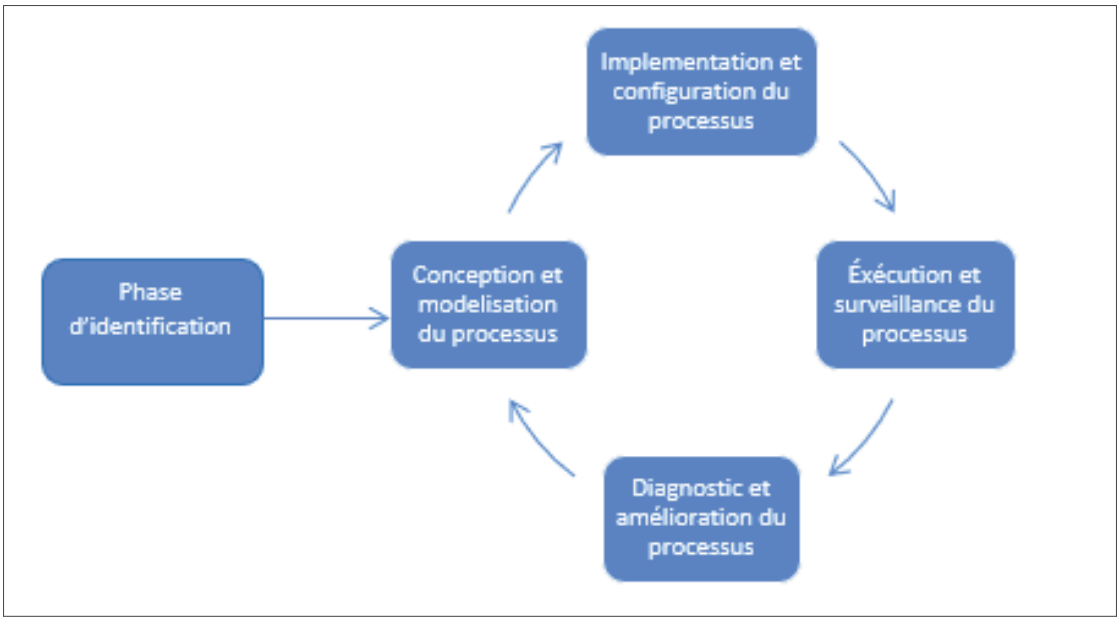


Figure 1.2 Cycle de vie de BPM
Adapté de Sturm et al.(2019, p.2)

1.3.2 BPMN

Afin de réaliser et exécuter un processus au sein d'un BPMS, le langage de modélisation BPMN a été créé. Le BPMN (Business Process Model and Notation en anglais) ou encore le modèle de procédé d'affaire et notation est un outil de communication utilisé pour représenter les séquences d'activités d'un processus d'affaires. C'est un standard de modélisation de processus d'affaires maintenu par l'Object Management Group. Les éléments constitutifs de BPMN sont classés en cinq catégories à savoir (Object Management Group, 2016) : les objets de flux, les données, les objets de connexions, les objets de couloirs et les artéfacts.

- Les objets de flux : ce sont les principaux éléments graphiques utilisés pour décrire le comportement d'un processus d'affaires. Dans cette catégorie nous avons les évènements, les activités et les branchements,
- Les données : cette catégorie est constituée de l'ensemble des données reçues ou produites par les objets de flux. Il s'agit notamment des données des objets, des données d'entrée, des données de sortie, du magasin des données,

- Les objets de connexions : ils sont utilisés pour connecter les objets de flux entre eux ou avec une autre information. Dans cette catégorie, il y'a quatre éléments : les flux de séquence, les flux de message, les associations, les associations de données,
- Les objets de couloir : ils sont utilisés pour représenter un participant dans un processus d'affaires. Il existe deux types d'objets de couloir : les pistes (pool en anglais) et les corridors (lane en anglais),
- Les artefacts : ils sont utilisés pour fournir des informations additionnelles aux processus d'affaires. L'ensemble d'artefacts comprend les annotations et les groupes.

Pour représenter les processus d'affaire, BPMN combine ces éléments suivants trois modèles différents : le modèle orchestration, le modèle collaboration et le modèle chorégraphie.

- Le modèle orchestration présente la coopération des processus à partir d'une seule position. Ce modèle est utilisé pour représenter les processus à l'intérieur d'une seule unité de travail ou les processus avec plusieurs participants, mais sous une autorité centrale de contrôle,
- Le modèle de collaboration présente des processus d'affaires impliquant plusieurs participants qui collabore par l'échange de message entre eux. Chaque participant a la possibilité d'implémenter son processus interne, le modifier sans impacter les interconnexions avec les autres participants,
- Le modèle de chorégraphie quant à lui, met plus l'accent sur les flux de message que sur les détails de chaque activité associée au processus. Les interactions entre les participants sont représentées en utilisant un format différent.

1.4 La minuterie et la décision inclusive de BPMN

Après avoir défini le concept de BPMN, nous nous attarderons sur deux éléments essentiels de ce langage de modélisation. Tout d'abord nous parlerons de l'évènement de minuterie de BPMN, de ses propriétés, ainsi que de sa typologie. Par la suite nous présenterons le branchement inclusif, ses propriétés et sa typologie.

1.4.1 L'évènement de minuterie

Un évènement de minuterie est une condition qui peut être utilisée pour influencer le début d'exécution d'une activité en se basant sur des contraintes temporelles.Un évènement de minuterie peut être utilisé soit au début d'un processus en tant qu'évènement de début, soit entre deux activités en tant qu'évènement intermédiaire. Il peut également être utilisé en bordure d'une activité en tant qu'évènement intermédiaire. (Object Management Group, 2011). Comme évènement de début, l'évènement de minuterie peut commencer une instance de processus d'affaires à un moment donné. Ceci signifie qu'un processus peut démarrer une seule fois suivant un intervalle de temps bien précis (par exemple un processus démarre chaque mardi ou alors chaque samedi) (Dumas, La Rosa, Mendling, Reijers *et al.*, 2013). Un évènement de minuterie intermédiaire situé entre deux activités fonctionne comme un chronomètre. Ceci signifie que lorsque l'exécution d'un processus traverse ce type d'évènement de minuterie, un chronomètre est déclenché. Après une durée définie, le chronomètre s'arrête et le processus se poursuit directement sur le flux de séquence suivant l'évènement de minuterie. Dans le cas d'un évènement intermédiaire de minuterie situé en bordure d'activité, l'évènement de minuterie se comporte comme une combinaison de chronomètre et d'alarme. Lorsque le processus arrive au niveau de l'activité à laquelle est associé en bordure l'évènement de minuterie, une minuterie est déclenchée. Après un montant de temps défini, l'activité peut être interrompue ou pas et le processus va poursuivre son exécution directement sur le flux de séquence qui suit directement cette activité. Lorsque l'exécution d'une activité est interrompue après un certain délai, l'évènement de minuterie est considéré comme un évènement interrupteur. Dans le cas contraire, il est considéré comme un évènement non interrupteur.

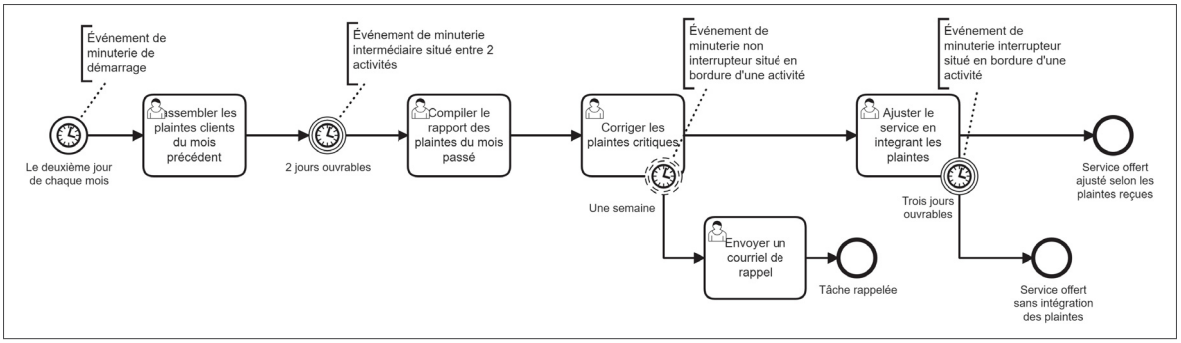


Figure 1.3 Types d'évènements de minuterie de BPMN

1.4.2 Le branchement inclusif

Un branchement inclusif encore appelé décision inclusive est utilisé afin de créer au sein d'un processus d'affaires des chemins qui sont à la fois alternatifs et parallèles. Le branchement inclusif est une combinaison du branchement exclusive et du branchement parallèle. Avec le branchement inclusif, toutes les conditions sont évaluées. Contrairement au branchement exclusif, l'évaluation d'une condition vraie n'exclut pas l'évaluation des autres conditions. Toutes les branches produites par les conditions vraies vont supporter l'exécution du processus. Ce branchement est conçu de sorte que toutes les branches peuvent être sélectionnées ou au moins une seule (Object Management Group, 2011). Le branchement inclusif supporte deux types de comportements : le branchement inclusif de divergence et le branchement inclusif de convergence. Dans un comportement de divergence, toutes les conditions sont évaluées et le processus ne va se poursuivre que sur les branches donc les conditions sont vraies. En ce qui concerne le comportement de convergence, pour toutes les branches entrantes dans le branchement, seules les branches exécutant le processus seront traitées. Une des particularités du branchement inclusif convergent est que, l'exécution du processus d'affaires qui traverse ce branchement ne progressera que si et seulement si toutes les branches convergentes vers le branchement et exécutant le processus sont complétées.

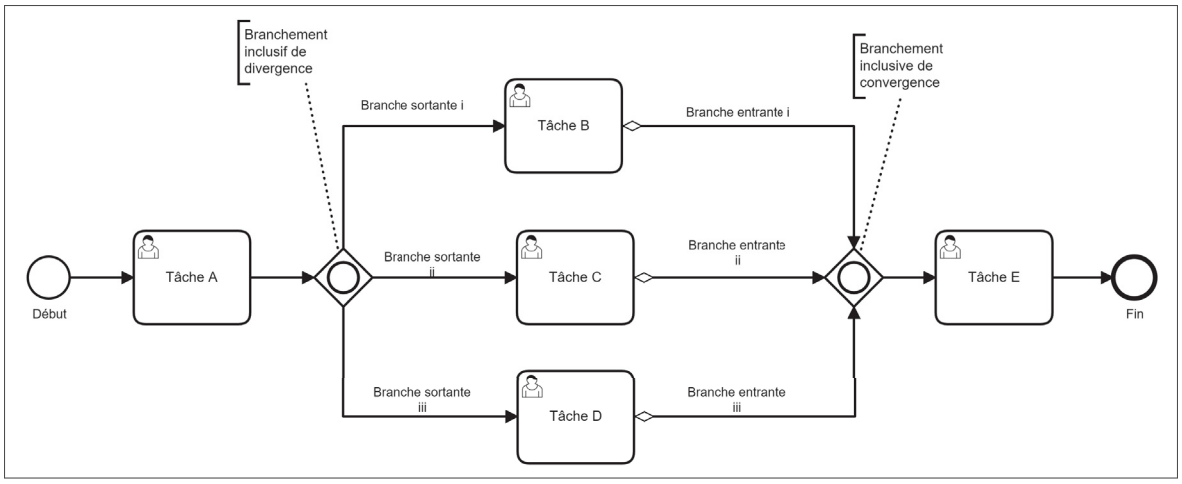


Figure 1.4 Exemple de branchement inclusif

1.5 Conclusion

Dans ce chapitre nous avons présenté les différentes technologies utilisées dans le cadre de notre travail. En résumé, nous avons présenté la technologie de chaîne de blocs, son architecture, le fonctionnement général des transactions sur la chaîne de blocs. Ensuite nous avons comparé les différents types de chaînes de blocs. À la suite de cette analyse, nous avons montré que les chaînes de blocs publiques offrent globalement plus d'avantages notamment en ce qui concerne la décentralisation et la transparence des données. Ensuite nous avons effectué une étude approfondie de la chaîne de bloc avec laquelle nous avons travaillé à savoir Ethereum. Nous avons expliqué le concept de contrat intelligent que nous utilisons dans notre solution pour transformer les éléments BPMN en code Solidity. Afin de mieux comprendre cette transformation, nous avons clarifié les notions telles que BPM, BPMS et BPMN. Enfin, une attention particulière a été portée sur les éléments BPMN suivants : les évènements de minuterie et les branchements inclusifs. En effet, la majorité des solutions existantes de conversion de BPMN en contrat intelligent ne supporte presque pas ces éléments.

CHAPITRE 2

REVUE DE LA LITTÉRATURE

La gestion des processus d'affaires est une discipline qui fournit un aperçu des processus métiers d'une organisation ainsi que des différentes interactions entre eux. L'objectif final est l'optimisation et l'automatisation au maximum des processus (Object Management Group, 2011). Cependant dans un contexte multiorganisationnel, le manque de confiance et de transparence entre les organisations constitue un goulot d'étranglement pour les systèmes centralisés de gestion des processus d'affaires. D'autre part, il y a la technologie de chaîne de blocs qui offre la possibilité de créer et d'exécuter des applications sans autorité centrale. En effet, cette technologie a le potentiel de pouvoir résoudre les problèmes de manque de confiance entre les parties et de transparence des données précédemment énoncés. Grâce à sa capacité à garantir l'intégrité et la persistance des données, la technologie de chaîne de bloc apparait comme une solution révolutionnaire pour la gestion des procédés d'affaires. Dans ce chapitre, nous allons passer en revue les recherches récentes de solution combinant la technologie de chaîne de blocs avec un système de gestion des processus métier. Ensuite, nous nous attarderons dans un premier temps sur les travaux récents qui proposent des moteurs d'exécution de processus métiers basés sur la chaîne de blocs implémentant des évènements de minuteries. Enfin, nous allons passer en revue les recherches récentes qui présentent des moteurs d'exécution de processus basés sur la chaîne de blocs capables d'interpréter les branchements inclusifs.

2.1 Moteurs de génération des processus d'affaires basés sur les chaînes de blocs

La technologie de chaîne de blocs en raison de ses propriétés, devient de plus en plus attractive dans le domaine de l'exécution des processus d'affaires entre plusieurs organisations ayant peu ou pas confiance entre elles. À cet effet, plusieurs travaux ont été effectués dans le but de faciliter l'intégration de la chaîne de blocs aux systèmes de gestions des processus d'affaires. Les critères tels que le type de chaînes de blocs, la méthode d'exécution des processus sur la

chaîne (Compilée ou Interprétée), le modèle de processus d'affaires supporté permettent de catégoriser ces solutions.

En ce qui concerne les chaînes de blocs publiques, les auteurs (Prybila, Schulte, Hochreiner & Weber, 2020) utilisent la chaîne de blocs Bitcoin pour mettre en œuvre et contrôler le flux des processus. Dans cette approche, la chaîne de blocs est utilisée pour sauvegarder l'exécution des tâches, sans spécifier quelle tâche devrait être exécutée à un moment donné. L'utilisation de la chaîne de bloc vient ainsi supprimer le besoin de confiance mutuelle dans les relations contractuelles. Ethereum quant à elle, constitue la plateforme la plus utilisée pour l'implémentation des systèmes de gestion des processus d'affaires basés sur la chaîne de blocs. Tout d'abord, il y a l'auteur de (Weber *et al.*, 2016) qui propose une solution collaborative d'exécution de processus basé sur la chaîne de blocs. Cette approche permet de transformer un diagramme chorégraphique BPMN en contrat intelligent écrit en Solidity. Les interactions entre les différents participants sont faites par échanges de messages ; ces messages sont enregistrés comme des transactions Ethereum. Cette approche présente un inconvénient majeur en termes de coût élevé de stockage des données sur la chaîne de blocs. Afin d'améliorer les travaux de (Weber *et al.*, 2016), les auteurs de (García-Bañuelos, Ponomarev, Dumas & Weber, 2017) vont proposer une solution optimisée d'exécution des processus basée sur la chaîne de blocs. Leur approche consiste à traduire dans un premier temps les processus d'affaires en réseau de Petri, ensuite le réseau de Petri est compilé en contrat intelligent écrit en Solidity. Bien que l'évaluation de la solution ait montré une réduction de gaz par rapport à la solution de (Weber *et al.*, 2016), elle couvre une faible palette d'élément BPMN. De plus, elle ne se préoccupe pas de la façon dont les différents participants collaborent durant l'exécution d'une instance de processus d'affaires ni du contrôle d'accès des participants au processus. Par la suite, il y a les auteurs de (López-Pintado *et al.*, 2017) qui proposent une solution plus mature appelée Caterpillar permettant de transformer des fichiers BPMN 2.0 en contrat intelligent sous forme de code Solidity. Cette recherche supporte une large gamme d'éléments BPMN, la gestion de l'état des processus, l'ordonnancement d'exécution. Dans (Lòpez-Pintado *et al.*, 2019), les auteurs offrent une version améliorée de Caterpillar qui permet une exécution compilée des

processus métiers avec la possibilité de définir une politique de contrôle d'accès des rôles des participants. Dans (López-Pintado, Dumas, García-Bañuelos & Weber, 2019b), la solution implémentée est une version de Caterpillar qui progresse de sa version compilée pour une version interprétée offrant plus de flexibilité dans l'exécution des processus. Il y'a également les auteurs de (Ladleif, Weske & Weber, 2019) et (Falazi, Hahn, Breitenbücher & Leymann, 2019) qui dans leur recherche, essaient de démontrer que les limitations du langage de modélisation BPMN peuvent être résolues grâce la technologie de chaîne de blocs. Après avoir proposé une amélioration de ce modèle qui supporte les propriétés de la chaîne de blocs et définit une syntaxe opérationnelle correspondante, ces auteurs vont implémenter une preuve de concept. Cette preuve de concept est basée sur Ethereum et les contrats intelligents sont écrits en Solidity. Son objectif est de valider la faisabilité de leur approche. Tandis que le premier groupe d'auteurs se concentre sur le modèle chorégraphique de BPMN, le second groupe quant à lui ajoute à sa solution une couche d'accès chaîne de blocs. Cette couche va agir comme intermédiaire entre les applications externes et la chaîne de bloc. Comme autres solutions supportées par Ethereum, nous avons les auteurs de (Sturm *et al.*, 2018) qui proposent un cadriciel léger (qui ne couvre pas assez de concept BPMN) permettant l'exécution de processus d'affaires interorganisationnel. Dans (Tran *et al.*, 2018), les auteurs utilisent les capacités de l'ingénierie dirigée par les modèles (IDM) pour faciliter le développement d'application de gestion des processus métiers basée sur la chaîne de blocs. Pour y arriver, les auteurs vont concevoir et développer un outil d'ingénierie dirigé par les modèles appelé Lorikeet. Par la suite, les auteurs vont implémenter les algorithmes de conversion de BPMN en contrats intelligents Solidity définis dans (García-Bañuelos *et al.*, 2017) et dans (Weber *et al.*, 2016). Cette solution est utilisée en industrie pour gérer et suivre l'état des processus métiers.

En ce qui concerne les chaînes de blocs privées, les auteurs (Nakamura *et al.*, 2018) proposent une approche basée sur la plateforme Hyperledger Fabric. Cette solution permet dans un premier temps de convertir un processus d'affaires en diagramme d'état pour la chaîne de blocs et les participants. Par la suite, ces diagrammes sont utilisés pour générer l'interface web des participants au processus et des contrats intelligents exécutables sur la chaîne de blocs. De

même, les auteurs de (Schinle, Erler, Andris & Stork, 2020) proposent une solution permettant l'intégration, l'exécution et le suivi des processus métiers sur la chaîne de blocs Hyperledger Fabric.

En prenant en compte la démarche d'exécution des systèmes de gestion des processus métiers basés sur la chaîne de bloc, nous retrouvons des solutions qui ont une approche compilée et des solutions qui ont une approche interprétée. Dans l'approche compilée, l'attention est portée sur la conversion des modèles de processus en contrats intelligents. Dans l'approche interprétée par contre, l'objectif consiste à déployer un contrat générique qui va permettre d'interpréter les modèles. Ce contrat va permettre d'éviter de redéployer chaque fois des instances de processus aussitôt qu'un changement survient sur ce dernier. Dans la catégorie de l'approche compilée on retrouve les solutions suivantes : (Falazi *et al.*, 2019; García-Bañuelos *et al.*, 2017; Nakamura *et al.*, 2018; Tran *et al.*, 2018; Weber *et al.*, 2016; López-Pintado *et al.*, 2017; Lòpez-Pintado *et al.*, 2019). Dans l'approche interprétée, on retrouve les approches suivantes : (López-Pintado *et al.*, 2019b; Prybila *et al.*, 2020; Sturm *et al.*, 2018). Bien que l'approche interprétée offre plus de flexibilité, elle présente le risque d'incohérence des états sur le processus modifié (Loukil, Boukadi, Abed & Ghedira-Guegan, 2021). L'approche compilée par contre permet d'éviter des situations d'impasses dues aux modifications fréquentes.

Après avoir parcouru cette documentation, nous constatons qu'Ethereum est la plateforme la plus utilisée pour les solutions de gestion des processus métiers basées sur la chaîne de blocs. De même, l'approche d'exécution des processus de manière compilée produit des modèles de processus avec des états plus stables. Un constat que nous avons également fait est que Caterpillar est le premier moteur d'exécution basé sur Ethereum des procédés d'affaires qui supportent une large gamme d'éléments BPMN. C'est une solution qui se démarque largement au-dessus des autres solutions. De plus, la grande majorité des solutions trouvées lors de nos recherches ne prend pas en compte certains éléments du standard BPMN 2.0 tels que les évènements de minuterie et les branchements inclusifs. Dans le cadre de ce travail, nous proposons une solution qui étend la solution Caterpillar avec des évènements de minuterie et des branchements inclusifs. Dans les prochaines sections de ce chapitre, nous allons passer en revue l'état de l'art sur les

approches existantes en matière de moteur d'exécution des processus basés sur la chaîne de blocs. L'accent sera mis sur les systèmes qui prennent en charge les deux éléments de BPMN précédemment cités.

2.2 Événement de minuterie dans la gestion par processus d'affaires basée sur la chaîne de bloc

Selon (Eder, Panagos & Rabinovich, 1999) et (Cheikhrouhou, Kallel, Guermouche & Jmaiel, 2015), la conception et l'implémentation des contraintes temporelles dans le domaine de la gestion par processus d'affaires, sont réellement un sujet critique de recherche. Dans le domaine des systèmes de gestion des processus métiers basés sur la chaîne de blocs, il existe très de peu de solutions qui prennent en charge des évènements de minuterie lors de la transformation des processus. Cependant, certaines solutions impressionnantes telles que (Tran *et al.*, 2018; López-Pintado *et al.*, 2017; Lòpez-Pintado *et al.*, 2019; Corradini *et al.*, 2020; Sturm *et al.*, 2018) et la version interprétée de Caterpillar (López-Pintado *et al.*, 2019b) ne supportent pas du tout les évènements de minuterie. En ce qui concerne la chaîne de bloc Ethereum, une des raisons majeures qui peut expliquer cette limitation provient des propriétés de la machine virtuelle d'Ethereum. En effet, la machine virtuelle n'offre pas la possibilité de déclencher l'exécution d'une transaction à un moment précis.

Néanmoins, nous avons noté que les auteurs de (Klinger & Bodendorf, 2020) font allusion à l'amélioration du temps d'exécution à partir des composants hors chaîne sans implémenter des évènements de minuterie de BPMN. Les auteurs de (Ladleif *et al.*, 2019) implémentent une preuve de concept qui combine la chaîne de bloc avec un langage de modélisation chorégraphique des processus métiers. À la fin de ce travail, les auteurs vont admettre certains challenges liés à l'‘implémentation des évènements de minuterie et vont proposer la résolution de ces difficultés comme une future amélioration. Dans la continuité du travail précédent, Ladleif et les auteurs de (Ladleif & Weske, 2020) vont présenter une bonne analyse concernant les limites d'intégration des évènements de minuterie dans les BPMS basés sur la chaîne de blocs. Par ailleursles auteurs vont montrer également une analyse des solutions alternatives existantes. Après avoir comparé

les différentes approches,ces auteurs vont proposer quelques astuces à exploiter afin de choisir la meilleure solution en fonction d'un scénario donné. La solution (Abid, Cheikhrouhou & Jmaiel, 2019) quant à elle, fournit une approche qui étend le moteur de génération Caterpillar. En effet, cette solution implémente des contrôleurs de temps tels que la durée des tâches, les temps absolus de début ou de fin au sein des méthodes des activités qui supportent les contraintes temporelles. Cependant, cette approche ne définit pas clairement le lien qui existe entre les contrôleurs de temps et les évènements de minuterie tels que spécifiés par les standards de BPMN 2.0 de l'organisme OMG. Nous avons également Mavridou et Laszka (Mavridou & Laszka, 2018) qui utilisent un langage de modélisation d'automate avec un nombre fini d'états pour générer des contrats intelligents sous forme de code Solidity. Cette génération prend en compte les retards d'exécution des processus et le temps du bloc sur Ethereum.

2.3 Branchement inclusif dans la gestion par processus d'affaires basée sur la chaîne de bloc

La notation BPMN 2.0 propose plusieurs éléments qui permettent de définir des comportements spécifiques d'un modèle de processus métier. Cependant, plusieurs moteurs d'exécution de processus échouent dans l'interprétation des éléments BPMN ou alors produisent des procédés d'affaire avec des réactions différentes des processus obtenus lors de la phase de modélisation. L'un des éléments de BPMN ayant un comportement délicat et capable de produire un résultat ambigu du processus d'affaires est le branchement inclusif (Corradini, Muzi, Re, Rossi & Tiezzi, 2022) . En ce qui concerne les moteurs d'exécution de BPMN basé sur la chaîne de blocs, une grande majorité des solutions proposées dans la littérature existante, ne supporte tout simplement pas les branchements inclusifs. Parmi ces solutions, nous retrouvons des frameworks remarquables tels que Caterpillar dans sa version compilée (Lòpez-Pintado *et al.*, 2019) et sa version interprétée (López-Pintado *et al.*, 2019b), lorikeet (Tran *et al.*, 2018) et (Prybila *et al.*, 2020). Cependant, les auteurs de (Sturm *et al.*, 2019) et (Sturm, Szalanczi, Jablonski & Schönig, 2020) ont effectué un grand travail concernant l'intégration des branchements inclusifs par les systèmes de gestion des processus métiers basés sur la chaîne de blocs. Leur approche consiste

à formaliser l'exécution de la sémantique du branchement inclusif du standard BPMN sur la chaîne de blocs. Cependant, l'implémentation de la solution fait face à certaines limitations notamment la déviation du standard BPMN 2.0 lors de l'exécution de la sémantique et également, l'absence de prise en charge des processus non structurés en blocs. Les auteurs de (Schinle *et al.*, 2020) présentent une approche qui permet l'intégration, l'exécution et le suivi des modèles de processus d'affaires sur la chaîne de bloc Hyperledger Fabric. Cette solution hérite des avantages et des inconvénients de la chaîne de bloc privée. De plus, bien qu'elle supporte conceptuellement l'intégration des branchements inclusifs, elle ne fournit pas une implémentation claire de cet élément. Dans (Loukil *et al.*, 2021), une autre solution décentralisée de gestion collaborative des procédés d'affaires est proposée. Cette solution est construite sur la chaîne de bloc Ethereum et supporte certains éléments de branchements du standard BPMN 2.0. Cependant, aucune implémentation claire du branchement inclusif n'est présentée.

2.4 Conclusion

Dans ce chapitre, nous avons passé en revue les travaux récents dans le domaine des systèmes de gestion des processus basés sur la chaîne de blocs. Plusieurs constats se sont dégagés de cette première analyse.Le premier concerne la maturité de la solution Caterpillar par rapport aux autres solutions. Ensuite nous avons l'importance de la version compilée par rapport à la version interprétée. Comme troisième observation nous avons l'absence d'évènements de minuterie et de branchements inclusifs au sein de la plupart des approches existantes. Par la suite, nous avons porté une attention particulière aux travaux de recherches abordant le support des évènements de minuterie et de branchements inclusifs. Ces différents constats provenant de la littérature ont contribué grandement à nourrir notre motivation dans la réalisation de notre travail. Notre objectif de recherche est de proposer une solution de gestion des processus métiers basée sur la chaîne de blocs et supportant les évènements de minuterie et les branchements inclusifs.

CHAPITRE 3

ARCHITECTURE ET CONCEPTION DE LA SOLUTION PUPA

Dans le chapitre présentant les notions de base, nous avons pu constater que la technologie des chaînes de blocs est une solution attractive dans le domaine de la gestion des processus métiers. Dans le chapitre portant sur l'état de la littérature, le constat selon lequel la majorité des solutions combinant la chaîne de blocs avec les BPMS ne supporte pas tous les éléments du standard BPMN a été fait. Plus spécifiquement, plusieurs moteurs d'exécution de processus basés sur la chaîne de blocs ne prennent pas en charge les évènements de minuterie ainsi que les branchements inclusifs. Dans le présent chapitre, nous allons présenter une nouvelle solution de gestion des processus d'affaires basée sur la chaîne de blocs dénommée Pupa. Pupa est un moteur d'exécution des processus d'affaires qui permet la transformation des modèles BPMN en contrats intelligents écrits en Solidity. L'une des particularités de Pupa est qu'elle permet l'interprétation et l'exécution des processus d'affaires qui implémentent les évènements de minuterie et les branchements inclusifs. Dans la suite de ce chapitre, nous allons présenter l'architecture de Pupa, ensuite nous parlerons des différentes étapes de conception de Pupa.

3.1 Architecture de la solution

Pupa est une extension de Caterpillar qui prend en charge de nouvelles fonctionnalités parmi lesquelles, la capacité à exécuter des processus d'affaires qui supportent les évènements de minuterie et les branchements inclusifs. De plus, la solution Pupa ne supporte pas le modèle chorégraphique, mais est conçu sur un modèle de collaboration spécifique. Un modèle de collaboration qui représente les processus comme s'ils sont exécutés au sein d'une même organisation, la présence du rôle dans la tâche permet de savoir quelle est le département/l'organisme concernée par l'activité. Dans cette section, nous allons vous présenter l'architecture de transformation des modèles BPMN en code Solidity de la solution Pupa adaptée de celle de Caterpillar. Cette architecture est constituée de trois couches : la couche on-chain, la couche off-chain et l'interface web.

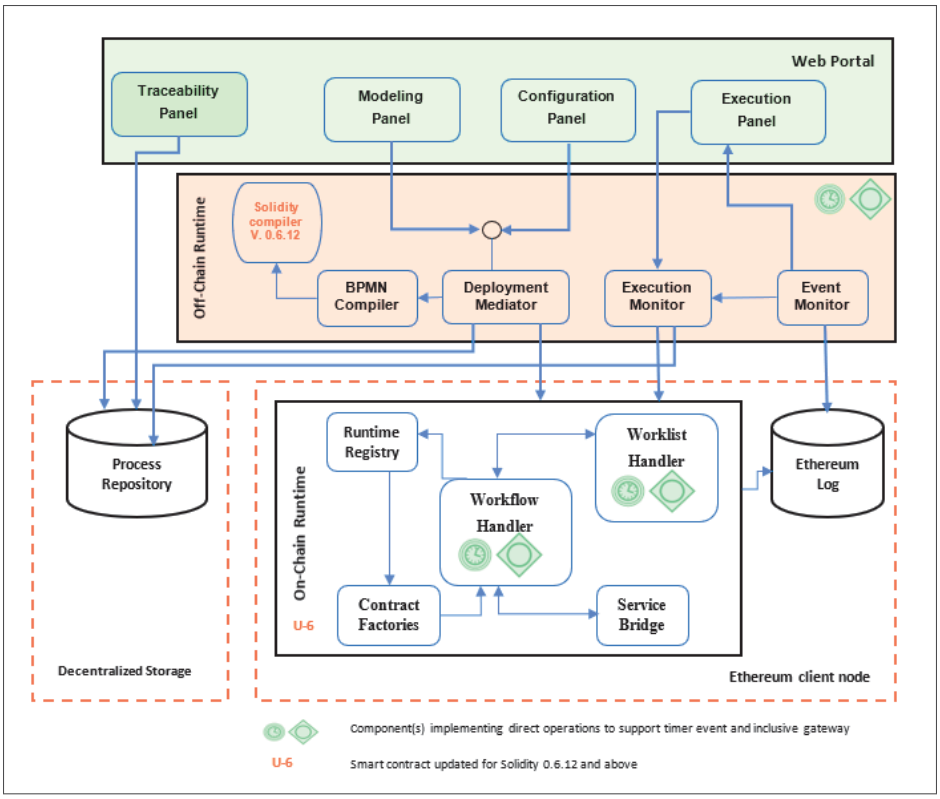


Figure 3.1 Architecture Pupa
Adaptée de Lòpez-Pintado et al.(2019, p.8)

3.1.1 Les composants on-chain

Dans cette couche on retrouve trois types de composants : le magasin des processus, les composants contrats intelligents et les logs.

3.1.1.1 Le magasin des processus

Il s'agit d'un système de stockage décentralisé, fait à base du protocole pair à pair IPFS. C'est une sorte d'entrepôt de données de processus d'affaires. Il stocke les données reçues, produites et requises par le système pour exécuter un modèle BPMN sur la chaîne de blocs. Ce composant permet d'avoir accès à toutes les données et métadonnées utilisées et produites tout au long du processus de transformation des modèles BPMN en contrats intelligents écrits en Solidity.

3.1.1.2 Les composants contrats intelligents

Ce sont des composants entièrement situés sur la chaîne de blocs. Ils sont répliqués sur tous les nœuds complets du réseau de la chaîne de blocs. Ils sont constitués d'un ensemble de contrats intelligents utilisés pour appliquer des opérations sur le modèle BPMN du processus. De plus, ils peuvent servir également pour extraire les données et les scripts du processus et pour gérer le contrôle d'accès et l'allocation des ressources auprès des utilisateurs. Dans cette catégorie, on retrouve cinq types de contrats intelligents :

1. **Workflow handler ou encore gestionnaire du flux de contrôle :** il est constitué d'un ensemble des contrats intelligents générés à partir d'un modèle BPMN de processus. Ces contrats sont utilisés pour contrôler le flux d'exécution du processus sur la chaîne. Dans ce composant, on retrouve des fonctions permettant de gérer le comportement de l'évènement de minuterie ainsi que de celui du branchement inclusif.
2. **Worklist handler ou encore gestionnaire des tâches utilisateurs :** il s'agit d'un ensemble de contrats intelligents qui supportent l'interaction avec l'utilisateur durant l'exécution du processus sur la chaîne. Les premières méthodes appelées lors de l'échange avec l'utilisateur durant l'exécution des évènements de minuterie ou des branchements inclusifs sont implémentées par ces contrats.
3. **Service Bridge ou encore gestionnaire des tâches de services (services applicatifs) :** ce composant est constitué des contrats intelligents permettant l'interaction avec les applications externes exposées en tant que service.
4. **Contract factories ou encore l'usine des instances de contrats intelligents :** il s'agit d'un ensemble de contrats intelligents responsables de l'instanciation de tous les contrats attachés à un processus donné.
5. **Runtime Registry ou encore le registre d'exécution :** c'est un contrat intelligent qui est chargé de garder l'historique et les liens existants entre les différentes instances de contrats associées à un processus d'affaires.

3.1.1.3 Les logs

Ce composant sert de support de communication entre les composants on-chain et les composants off-chain. Toute transaction insérée dans la chaîne de blocs déclenche un évènement. Cet évènement est écrit dans le composant log lequel, va notifier le composant off-chain qu'un changement est survenu sur la chaîne de blocs.

3.1.2 Les composants off-chain

Cette couche permet d'exposer les services des composants on-chain à des applications externes ou aux utilisateurs via le portail web. Elle permet de compiler et déployer les contrats intelligents, de suivre l'état d'exécution des instances de processus, d'écouter les évènements survenus sur la chaîne de blocs. Étant donné que les décisions d'exécution et l'intégrité du processus d'exécution sont assurées par les composants on-chain, la couche off-chain peut être optionnelle. Il est possible d'accéder aux données d'exécution du processus métier sans passer par les composants off-chain. Chaque acteur collaborant dans le processus d'affaires peut implémenter ses propres composants off-chain. Ainsi tout compromis qui apparaît au niveau de cette couche ne va impacter que les acteurs qui l'utilisent sans modifier les données de la chaîne de blocs. Dans cette catégorie, on retrouve quatre types de composants :

- **Le compilateur BPMN :** ce composant est chargé de compiler les modèles BPMN en contrats intelligents et de regrouper ces contrats intelligents générés autour du même processus métier. Toutes les données entrantes utilisées ainsi que les données produites durant le processus de compilation sont stockées dans l'entrepôt des données processus. La version de compilateur Solidity utilisé par Pupa est 0̂.6.12.
- **Le médiateur de déploiement :** il est chargé de déployer les contrats intelligents générés lors de la compilation. Une fois tous les contrats associés à un processus sont déployés, l'exécution du processus en question peut débuter.
- **Le contrôleur d'exécution :** il est chargé d'interroger le processus d'exécution et de fournir des informations concernant l'état d'exécution des éléments BPMN du processus d'affaires.

Lors de l'exécution de l'évènement de minuterie, le contrôleur d'exécution sera mis à profit pour informer l'utilisateur concernant le montant de temps restant afin que le délai de la minuterie soit achevé.

- **Le contrôleur d'évènement :** il est chargé d'écouter les évènements écrits dans les logs de la couche on-chain, de les analyser et de les partager avec le contrôleur d'exécution. Il peut éventuellement interagir avec les utilisateurs ou les applications externes.

3.1.3 Le portail web

Cette couche est l'interface entre l'utilisateur et les composants off-chain. Elle est constituée de quatre panneaux :

- **Le panneau d'exécution :** il permet à l'utilisateur de visualiser la liste des instances de processus, d'interagir avec le processus d'exécution. Il interfère avec le contrôleur d'exécution afin d'avoir toutes les données concernant les processus métiers déployés, les instances d'exécution. De même, à base des notifications reçues du contrôleur d'évènement, il met à jour l'affichage des transactions insérées dans la chaîne de blocs.
- **Le panneau de configuration :** il permet de créer ou importer le registre d'exécution pour un ensemble deprocessus d'affaires. Il permet également de créer ou d'importer la politique de rôle d'accès associée aux processus d'affaires.
- **Le panneau de modélisation :** il permet à l'utilisateur de dessiner les modèles BPMN des processus qui seront déployés plus tard sur la chaîne. L'utilisateur a également la possibilité d'importer des modèles BPMN faits à partir d'autres outils. En cas d'erreur dans les modèles BPMN, la compilation du code Solidity généré va échouer.
- **Le panneau de traçabilité :** ce panneau est propre à Pupa. Il permet à l'utilisateur de visualiser toutes les opérations exécutées on-chain sur un processus donné.

3.2 Conception de Pupa

Après avoir présenté l'architecture de Pupa, nous nous attarderons ici sur les qualités fondamentales qui distinguent notre solution. Tout d'abord nous présenterons les différentes propriétés associées à la gestion des évènements de minuterie, ainsi que celles associées à la gestion des branchements inclusifs. Par la suite, nous expliquerons comment Pupa interprète les évènements de minuterie et les branchements inclusifs afin de les rendre exécutables sur la chaîne de blocs. Enfin, nous allons présenter des spécifications non fonctionnelles mais tout aussi importantes qui ont été ajoutées à Pupa.

3.2.1 Spécifications fonctionnelles

Dans cette sous-section, nous parlerons des fonctionnalités que nous avons prises en compte lors de la phase de conception. Afin de produire le moteur de transformation de modèles BPMN en contrats intelligents supportant l'exécution des évènements de minuterie et des branchements inclusifs, nous avons défini certaines propriétés, variables et fonctions.

3.2.1.1 Événement de minuterie

Un évènement de minuterie peut être encore être défini comme étant un évènement qui est déclenché par un montant de temps prédéfini (Camunda.org, 2022). Dans la section 1.4.1, nous avons présenté les différents types d'évènements de minuterie. Pour notre solution, nous nous sommes focalisés sur l'évènement de minuterie intermédiaire situé entre deux tâches. Afin de faciliter son implémentation, trois règles de spécifications fonctionnelles ont été définies.

- **Un évènement de minuterie en tant que tâche utilisateur :** lors de la conception, il a été décidé que l'évènement de minuterie au sein de la solution Pupa agisse de manière similaire à l'activité tâche utilisateur de BPMN. Dans la solution de (Lòpez-Pintado *et al.*, 2019), les évènements supportés ne prennent pas en charge les données de saisie utilisateurs, les rôles d'accès, et encore moins les fonctions de références. Par contre, les évènements de minuterie

au sein de Pupa supportent la saisie des données par les utilisateurs, les rôles d'accès et les fonctions. Ainsi, un utilisateur va être capable d'entrer le temps d'attente de la minuterie en seconde lors de l'exécution du processus métier sur la chaîne de blocs. De plus, seuls les utilisateurs ayant les accès autorisés pourront effectuer des opérations sur l'évènement de minuterie durant l'exécution du processus. En effet, la solution Pupa hérite de la propriété de spécification et de vérification des rôles telle que défini dans (López-Pintado, Dumas, García-Bañuelos & Weber, 2019a). Cette fonctionnalité sera étendue sur les évènements de minuterie afin, de limiter l'accès à ces évènements qu'aux rôles préalablement définis. En ce qui concerne les fonctions, lorsque l'exécution du processus métier traverse l'évènement de minuterie, plusieurs méthodes sont exécutées chacune avec des mandats bien précis.

- **La vérification du temps effectuée par le nœud suivant :** étant donné qu'il n'est pas possible avec la machine virtuelle d'Ethereum de déclencher l'exécution du code Solidity à un moment précis, cette action sera effectuée par un utilisateur autorisé. En effet, l'exécution de la tâche qui succède l'évènement de minuterie est réalisée en deux étapes. La première étape consiste à vérifier que le temps défini dans la minuterie est bel et bien écoulé. Tant que ce temps n'est pas écoulé, l'exécution du processus ne progresse pas. La seconde étape consiste à l'exécution de la fonction réelle de la tâche. Cette étape n'est appelée que si le temps précédemment spécifié dans la minuterie a été achevé.
- **Les variables de minuterie et les fonctions de minuterie :** lorsqu'un évènement de minuterie est présent dans un processus métier, des variables et des fonctions de minuterie sont créées. Afin de ne pas surcharger les contrats intelligents qui sont générés, certaines variables et fonctions n'ont pas une croissance linéaire en fonction du nombre d'évènements de minuterie.

3.2.1.2 Branchements inclusifs

Les branchements inclusifs sont des items BPMN classifiés sous la catégorie des objets de flux. Ils sont utilisés pour implémenter des décisions à la fois alternatives et parallèles à l'intérieur d'un processus d'affaires. Dans la section 1.4.2, nous avons présenté les différents comportements qui sont effectués par les branchements inclusifs à savoir : les branchements

inclusifs de divergence et les branchements inclusifs de convergence. Pupa prend en charge ces deux types de comportements. Dans le but de faciliter l'implémentation de ces comportements, notre solution supporte les caractéristiques suivantes :

- **Configuration particulière de l'activité avant le branchement inclusif de divergence :** le branchement inclusif de divergence doit être précédé d'une tâche utilisateur. Cette tâche va permettre de spécifier lors de l'exécution du processus, les chemins à suivre après le branchement inclusif de divergence. Tous les chemins peuvent être sélectionnés. Au moins un chemin doit être sélectionné. Si aucun chemin n'est sélectionné, une erreur est produite et le flux d'exécution ne va pas continuer,
- **Les flux de séquence associés aux variables booléennes :** chaque flux de séquence sortant d'un branchement inclusif de divergence doit être associé à une variable booléenne. De même, chaque flux de séquence entrant dans un branchement inclusif de convergence doit être associé à une variable booléenne. En effet, un branchement inclusif de convergence doit être précédé de près ou de loin par un branchement inclusif de divergence. Ainsi, les flux de séquence sortants du branchement inclusif de divergence et les flux de séquence entrants du branchement inclusif de convergence doivent être respectivement associés à la même variable booléenne. La combinaison des variables booléennes avec les flux de séquence permet de gérer aisément le jeton d'exécution au niveau du branchement inclusif,
- **Les variables et fonctions du branchement inclusif :** la présence d'un branchement inclusif dans un processus métier va générer automatiquement des variables personnalisées dans le contrat intelligent. Par contre, les opérations concernant le branchement inclusif sont définies à l'intérieur du composant workflow. Ces instructions visent à réaliser deux objectifs majeurs. Le premier objectif consiste à s'assurer que dans un branchement inclusif de divergence, les choix effectués sont respectés. Autrement dit, le jeton d'exécution ne progresse que vers les flux de séquence qui ont été sélectionnés. Au moins un choix doit être fait. Si aucun choix n'est reçu, alors le jeton d'exécution ne bouge pas. Le second objectif consiste à s'assurer que le jeton d'exécution ne traverse le branchement inclusif de convergence que si et seulement si l'exécution du processus a été achevée sur tous les flux de séquence entrants.Il s'agit ici des flux de séquence entrants qui détiennent déjà le jeton.

3.2.2 Séquences d'exécution

Dans cette section, nous allons présenter les différentes étapes d'exécution que devra parcourir Pupa lorsque le processus traverse un évènement de minuterie et un branchement inclusif.

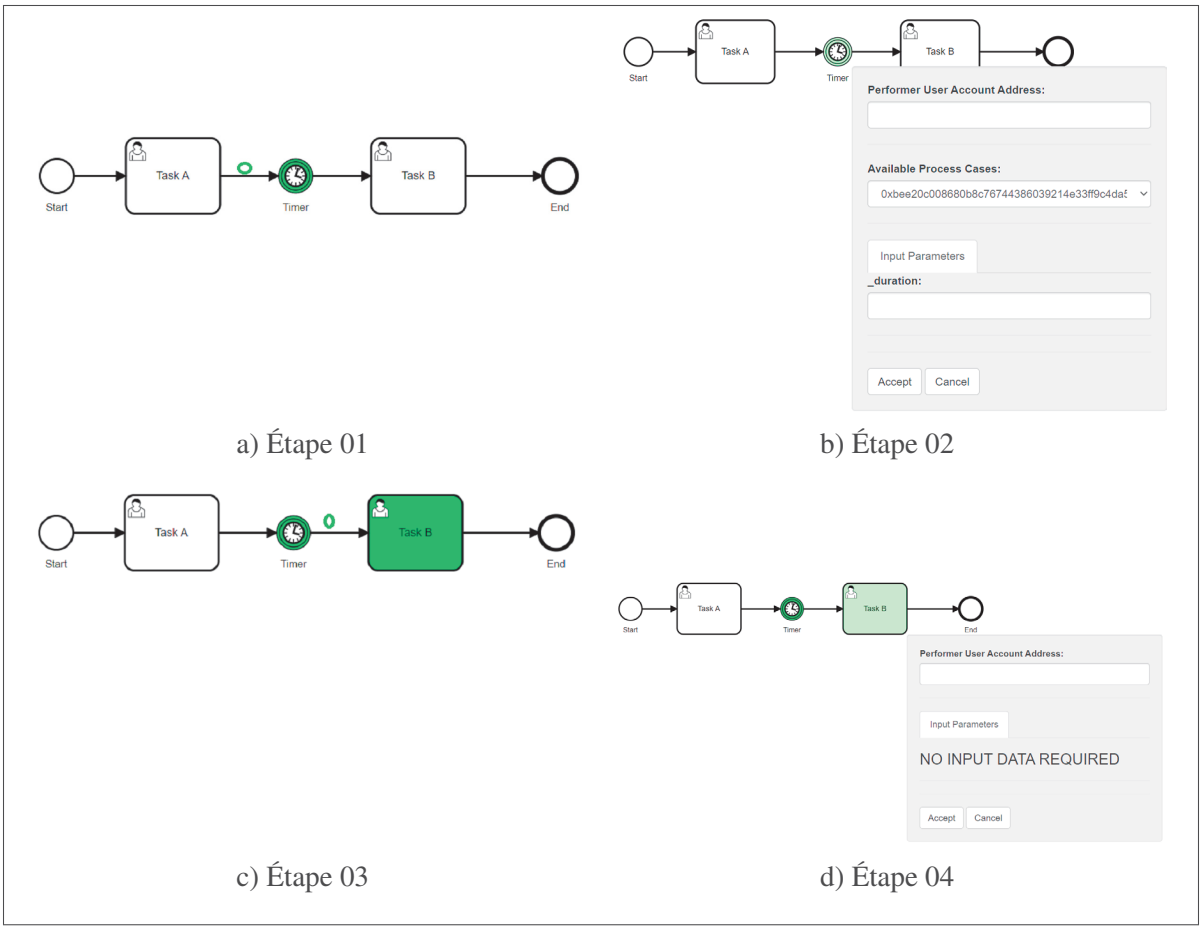
3.2.2.1 Événement de minuterie

L'implémentation de l'évènement de minuterie suit un mécanisme de jeton. Les différentes étapes d'exécution attendues lors de l'implémentation d'un évènement de minuterie sont les suivantes (voir Figure I-3) :

1. À la première étape, le flux de séquence entrant de la minuterie reçoit le jeton et le transmet à cette dernière. Dès que l'évènement de minuterie a le jeton, la couleur de l'évènement change et il passe à l'état prêt à être exécuté.
2. À la deuxième étape, l'activation de l'évènement de minuterie va proposer une boîte de dialogue où l'utilisateur doit saisir l'adresse du rôle autorisé à exécuter cette action. L'utilisateur devra également renseigner l'adresse du processus et la durée de la minuterie en seconde *(d)*. Aussitôt que l'utilisateur valide ses informations, un contrôle est fait en arrière-plan afin de vérifier les informations fournies. Si la vérification échoue, une erreur est affichée et le processus d'exécution ne progresse pas. Si par contre la vérification réussit, la transaction est sauvegardée dans la chaîne de blocs. L'horodatage du bloc qui vient d'être créé *(Tb)* est utilisé comme point de départ de la minuterie. Le choix de l'horodatage du bloc comme point de référence permet d'éviter tout problème issu de la variation du temps provenant des différents nœuds.
3. À la troisième étape, le jeton d'exécution va se déplacer sur le flux de séquence sortant de l'évènement de minuterie.
4. À la quatrième étape, le jeton d'exécution arrive à l'activité située juste après l'évènement de minuterie ce qui la met dans un état prêt à être exécuté. À l'activation de cette activité, l'utilisateur autorisé doit réaliser un test. Un formulaire lui est présenté où il doit remplir l'adresse associée à son rôle. À la validation du formulaire, un contrôle est effectué. Si

l'adresse proposée ne correspond pas à celle associée au rôle alors un message d'erreur est émis et le processus d'exécution ne progresse pas. Par contre, si l'adresse fournie est correcte, un nouveau test est effectué. Ce test va consister à vérifier si le temps actuel *(Ta)* valide l'équation A I-1. Si l'expression est validée, un message de confirmation est émis et la fonction spécifique attachée à cette activité est exécutée.

$$Ta >= Tb + d \tag{3.1}$$



a) Étape 01 b) Étape 02

c) Étape 03 d) Étape 04

Figure 3.2 Séquences d'exécution de l'évènement de minuterie

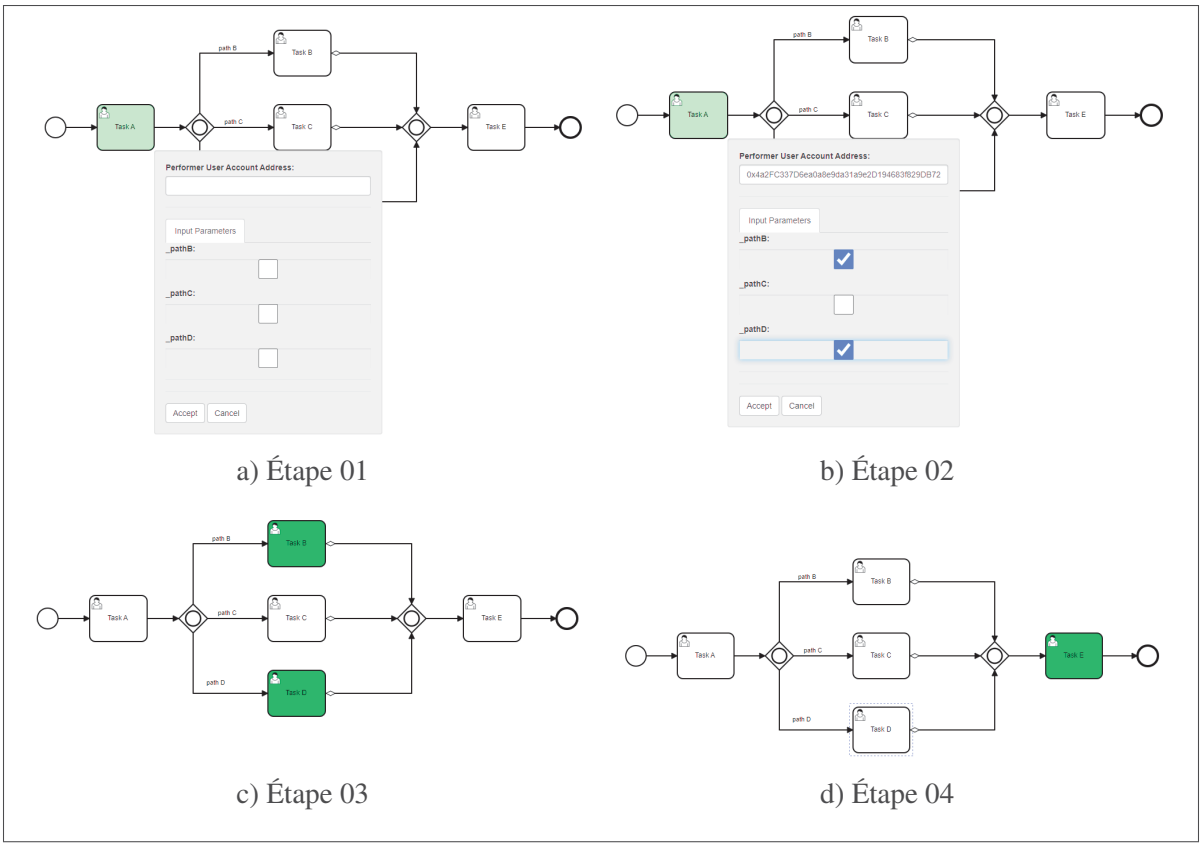
3.2.2.2 Branchements inclusifs

L'exécution du branchement inclusif se déroule en quatre phases (voir Figure I-4).

1. À la première phase, le jeton arrive à l'activité située juste avant le branchement inclusif divergent. Cette activité implémente une fonction qui permet d'effectuer des choix multiples parmi plusieurs variables. Chacune de ces variables est rattachée à un flux de séquence sortant du branchement inclusif divergent,
2. À la deuxième étape, l'utilisateur effectue un nombre de choix *(Nbrce)*. Il peut tout choisir sinon au moins une décision doit être selectionnée. Si aucun choix n'est effectué, rien ne se produit,
3. À la troisième étape, le jeton d'exécution se déplace vers les différents flux de séquence correspondant aux choix précédemment faits,
4. À la quatrième étape, lorsqu'un des chemins portant le jeton d'exécution transmet le jeton au branchement inclusif convergent, le branchement met à jour le nombre de jetons reçu *(Nbrjr)*. Le branchement inclusif convergent ne va déplacer le jeton vers son flux de séquence sortant que, si et seulement si l'expression suivante est vraie :

$$Nbrce = Nbrjr$$

Si le processus d'affaires ne contient que le branchement inclusif divergent, seules les étapes 01 à 3 seront effectuées



a) Étape 01 b) Étape 02

c) Étape 03 d) Étape 04

Figure 3.3 Séquences d'exécution du branchement inclusif

3.2.3 Spécifications non fonctionnelles

En plus des spécifications fonctionnelles, nous nous sommes intéressés à certaines caractéristiques non fonctionnelles à savoir la sécurité des contrats et la traçabilité des processus. Le but ici est de proposer une amélioration de ces propriétés. Dans cette section nous allons présenter les spécifications non fonctionnelles supportées par Pupa.

3.2.3.1 Sécurité des contrats intelligents

Les contrats intelligents jouent un rôle clé dans l'implémentation des applications décentralisées. Ils permettent de maintenir des accords entre des partenaires dans un environ fiable et sans conflit. Cependant la sécurité de ces contrats constitue un risque tant pour ces applications décentralisées que pour les utilisateurs. les auteurs de (Samreen & Alalfi, 2021) ont fait une

analyse des systèmes de gestion des processus basés sur la chaîne de blocs qui mettent un accent sur la sécurité des contrats. Selon ce rapport, plusieurs solutions ne mettent pas une emphase sur la sécurité des contrats parmi lesquelles Caterpillar. Par ailleurs, bien que Solidity soit le langage orienté contrat le plus utilisé, il présente des risques en matière de sécurité. Notamment en ce qui concerne la visibilité par défaut des fonctions et la taille des types de données. La visibilité par défaut des fonctions Solidity est public. Lorsque le développeur ne spécifie pas convenablement la visibilité des fonctions de certains groupes d'utilisateurs, il y'a un risque d'attaques externes. Un autre risque de sécurité est le fait que la machine virtuelle supporte seulement une taille fixe pour les données entières. On pourrait avoir des risques de débordement ou de sous-utilisation lors du stockage des données. À cet effet, les contrats générés par Pupa doivent être conçus avec une grande attention sur la sécurité.

3.2.3.2 Traçabilité des processus

La gestion des processus d'affaires entre plusieurs organisations nécessite la confiance entre les différentes parties. Grâce à son caractère vérifiable, la chaîne de blocs contribue à établir cette confiance. Pupa devra fournir une interface graphique qui va permettre de simplifier l'audit des processus métiers exécutés sur la chaîne. Ainsi, à partir d'un identifiant de processus, il doit être possible de savoir quelles opérations d'affaires ont été exécutées, quand elles ont été effectuées, dans quel bloc est contenue chaque transaction.

3.3 Conclusion

Dans ce chapitre nous avons présenté l'architecture de la solution Pupa. Nous avons expliqué les différentes couches constituantes de cette architecture, tout en précisant lorsqu'il y'a lieu les liens existants entre les composants de la couche et l'implémentation des évènements de minuterie et des branchements inclusifs. Ensuite nous avons parlé des spécifications fonctionnelles qui ont été prises en compte durant la phase de conception de Pupa. Plus spécifiquement, nous avons défini les différentes caractéristiques rattachées aux évènements de minuterie et aux branchements

inclusifs. Nous avons également présenté les étapes d'exécution attendue pour les évènements de minuterie, ainsi que pour les branchements inclusifs. Toujours dans le registre de la phase de conception, nous avons proposé des spécifications non fonctionnelles qui vont permettre d'accroitre la sécurité des contrats et, faciliter l'audit des processus exécutés sur la chaîne. Pour conclure, comparément à Caterpillar, notre solution offre une structure qui prend en charge plus d'éléments BPMN. De plus, notre solution offre plus de sécurité au niveau des contrats intelligents et fournit une interface de traçabilité des processus exécutés sur la chaîne de blocs.

CHAPITRE 4

IMPLÉMENTATION DE LA SOLUTION PUPA

Après avoir décrit l'architecture de Pupa ainsi que ces différents éléments de conception, nous allons présenter dans ce chapitre une implémentation d'un cas d'utilisation de processus d'affaires. Dans un premier temps, nous allons décrire le cas d'utilisation, ensuite nous allons produire le diagramme BPMN associé au processus d'affaires. Un accent sera porté sur les propriétés relatives aux branchements inclusifs et à l'évènement de minuterie à prendre en compte, lors de la conception du modèle BPMN. Par la suite, nous allons illustrer le procédé de conversion des processus d'affaires en contrats intelligents. Avant de conclure le chapitre, nous allons décrire les lignes de code des contrats intelligents générés à partir du cas d'utilisation. Un point central est mis sur le flux d'exécution de l'évènement de minuterie et des branchements inclusifs divergents et convergents.

4.1 Description étude de cas : gestion des commandes chez une entreprise fournisseur d'accès d'internet

Afin de valider les concepts théoriques associés à l'approche proposée Pupa, nous avons décidé d'implémenter un cas d'utilisation qui porte sur la gestion des commandes des clients au sein d'une entreprise fournisseur d'accès internet.

4.1.1 Diagramme BPMN

Lorsqu'un client sollicite le service internet auprès du fournisseur, un devis d'installation prenant en charge la localisation du client, le lui est retourné. Le client a trois jours pour accepter ou refuser le devis. Si le client refuse le devis, le processus de gestion de commande est annulé. Si le client valide le devis, la prochaine étape sera de préparer la commande. La préparation de la commande consiste à effectuer des décisions inclusives entre plusieurs options de facturation. Les différentes options sont les suivantes : facturation standard, facturation avec prise en charge

de la prime et facturation avec prise en charge par la compagnie de l'expédition du matériel d'installation. Au moins une option doit être choisie. Aux options sélectionnées, des opérations spécifiques seront appliquées. La dernière tâche du processus va consister à enregistrer les données d'installation du client. La figure 4.1 présente le diagramme réalisé avec le logiciel *Camunda Modeler 7*. Le choix de Camunda modeler est soutenu par le fait que c'est un outil à la fois conforme au standard BPMN 2.0 et qui permet d'enregistrer sur la chaîne de bloc, les transactions effectuées durant l'exécution du processus d'affaires.

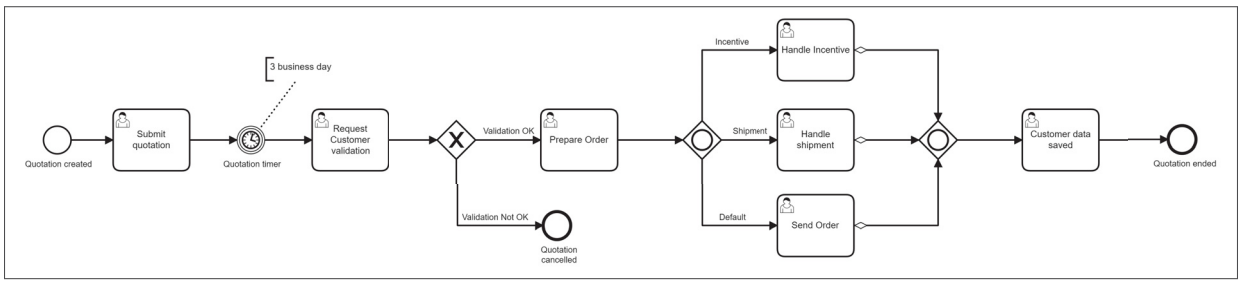


Figure 4.1 Diagramme BPMN du cas d'utilisation

4.1.2 Spécifications conceptuelles du modèle du processus d'affaires

Afin de faciliter l'interprétation de l'évènement de minuterie et des différents branchements inclusifs présents dans le cas d'utilisation, des indications particulières doivent être ajoutées au diagramme BPMN fait sur Camunda. Les indications sont effectuées à quatre niveaux notamment la racine du modèle BPMN, l'évènement de minuterie, la tâche placée juste avant le branchement inclusif de divergence. Le dernier niveau d'indication concerne les branches sortantes et entrantes du branchement inclusif divergent et du branchement inclusif convergent respectivement.

4.1.2.1 Au niveau de la racine du modèle

À cet étape, il faut déclarer dans la documentation du modèle BPMN, toutes les variables globales qui seront manipulées durant l'exécution du processus d'affaires sur la chaîne de blocs.

En ce qui concerne l'évènement de minuterie, l'expression *uint private duration = 0;* sera inscrite afin de déclarer la variable privée entière *duration* qui va permettre de stocker le délai de la minuterie en seconde qui sera fourni par l'utilisateur durant l'exécution du modèle. Pour ce qui est du branchement inclusif, les instructions de l'algorithme 4.1 sont présentées.
Ces instructions vont permettre de déclarer les variables privées de types booléens qui seront associées à chaque séquence sortante et entrante du branchement inclusif divergent et du branchement inclusif convergent respectivement.

Algorithme 4.1 Spécifications conceptuelles variables globales de marquage des branchements inclusifs

```
1 bool private handleInc;
2 bool private handleShip;
3 bool private sendOrder;
```

4.1.2.2 Au niveau de l'évènement de minuterie

La propriété de documentation de l'évènement de minuterie sera enrichie par deux spécifications. La première instruction @ *Customer* @, elle permet de définir le rôle autorisé à manipuler l'évènement de minuterie durant l'exécution. La seconde spécification est un groupe d'instructions qui permet de définir la fonction qui va implémenter le comportement précis attendu de la minuterie. Cette fonction ne retourne rien, mais elle prend en entrée une variable entière appelée $\_duration$ soit l'expression $(() : (uint\_duration))$. Lors de l'exécution de l'évènement de minuterie, une valeur est attendue de l'utilisateur. Cette valeur sera mise dans la variable $\_duration$. Le corps de la fonction quant à elle va servir à récupérer le contenu de la variable $\_duration$ pour la mettre dans la variable globale $duration$ ($duration = \_duration$). Voir l'algorithme 4.2 pour plus de détails

Algorithme 4.2 Spécifications conceptuelles de l'évènement de minuterie

```
1 @ Customer @
2 () : (uint _duration) -> {
3 duration = _duration ;}
```

4.1.2.3 Au niveau de la tâche qui précède le branchement inclusif divergent

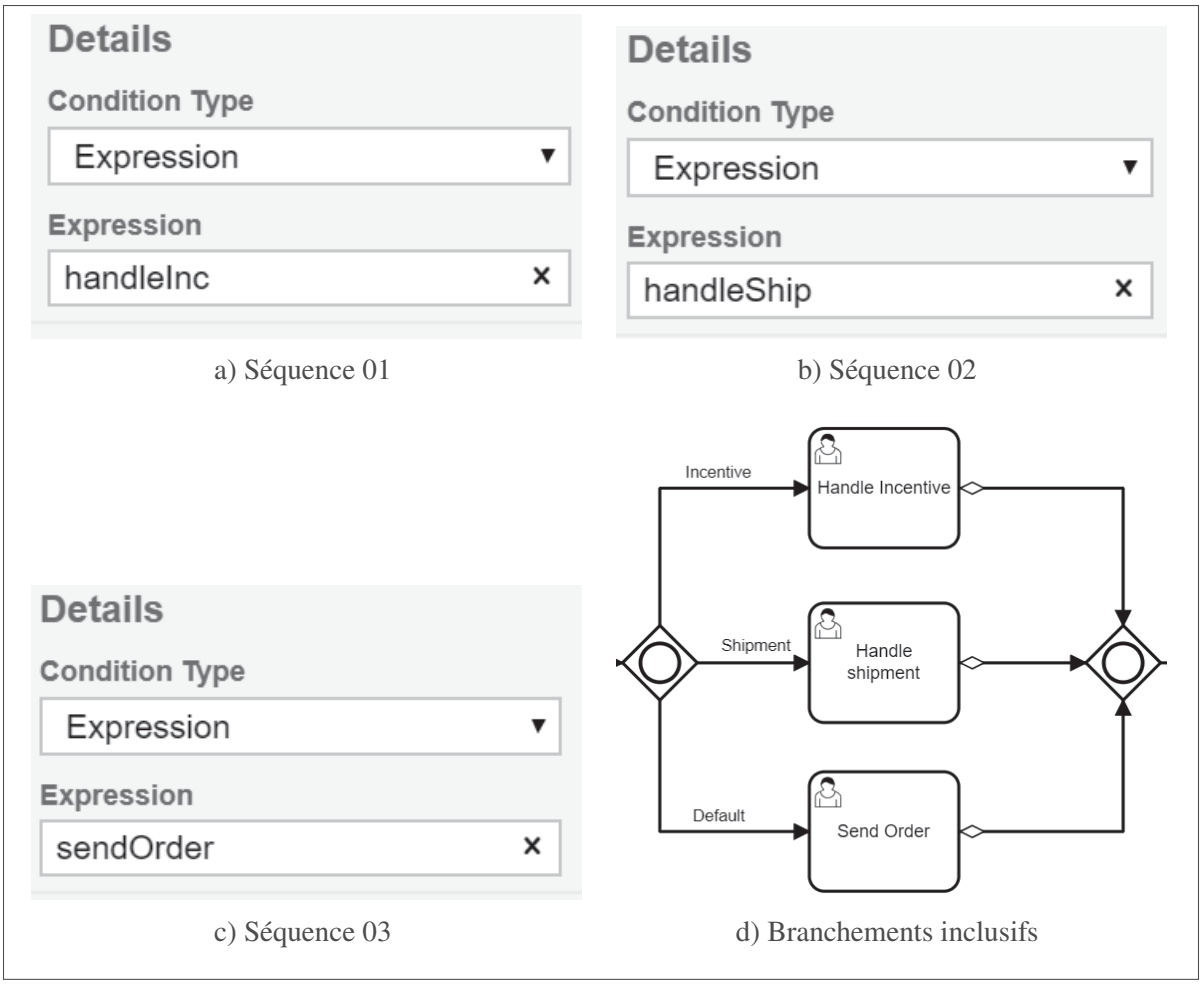
De manière similaire aux spécifications faites sur l'évènement de minuterie, les instructions définies ici (voir l'algorithme 4.3) vont permettre d'effectuer deux opérations. La première opération indiquer le rôle autorisé à manipuler cette tâche durant l'exécution. La seconde opération va implémenter une fonction qui ne retourne rien, attend les choix de l'utilisateur et transfère la valeur de ces choix dans les variables globales booléennes précédemment déclarées.

Algorithme 4.3 Spécifications tâche précédant le branchement inclusif divergent

```
1 @ Customer @
2 () : (bool _handleIncentive, bool _handleShipment, bool _sendOrder,) -> {
3 handleInc = _handleIncentive ;
4 handleShip = _handleShipment ;
5 sendOrder = _sendOrder ;}
```

4.1.2.4 Au niveau des flux de séquences associés aux branchements inclusifs

À cet étape, la spécification est effectuée sur la propriété de détail de chaque séquence. Elle consiste à définir le type de condition supportée par la séquence et la valeur de cette condition. Dans ce cas de figure, le type de condition est une expression et la valeur de la condition correspond à la valeur de la variable booléenne associée à la séquence. La figure 4.2 donne un aperçu de ces configurations.



a) Séquence 01

b) Séquence 02

c) Séquence 03

d) Branchements inclusifs

Figure 4.2 Spécifications conceptuelles des séquences des branchements

4.2 Conversion des processus d'affaires en contrats intelligents

La traduction d'un processus métier en contrat intelligent passe par une série d'étapes. Dans cette section nous présentons les différentes étapes de compilation du modèle BPMN en contrats intelligents déployables sur Ethereum. Par la suite, nous allons exposer le mécanisme de contrôle de l'exécution du processus d'affaire sur la chaîne de bloc.

4.2.1 Les étapes de conversion des diagrammes BPMN en contrats intelligents

La conversion d'un processus d'affaires en contrats intelligents avec *Pupa* passe par plusieurs étapes. Les contrats intelligents générés, à la fin de ces étapes, doivent pouvoir être déployés sur le réseau Ethereum. En d'autres termes, traduire un processus métier en contrats intelligents exécutables sur Ethereum revient à produire des contrats compilés par la Machine Virtuelle d'Ethereum. Le processus de compilation décrit ici implémente la même logique définie par les auteurs de (Lòpez-Pintado *et al.*, 2019). Tout d'abord il est important de dessiner le processus sous forme de diagramme BPMN. À la fin de cette étape, un fichier BPMN en format *XML* sera produit. La prochaine étape va consister à analyser ce fichier BPMN afin de produire un ensemble de contrats intelligents et une représentation en mémoire d'objets BPMN facilement manipulables. Grâce à la combinaison du moteur de génération de gabarit de JavaScript EJS, du langage de programmation *Typescript* et de la librairie *bpmn-moddle*, le fichier *XML* est analysé syntaxiquement. À la suite de cela, des objets *Typescript* correspondants aux différents items BPMN du processus d'affaires sont créés ainsi que les métadonnées associées à ces éléments. Enfin, les contrats intelligents précédemment produits sont combinés avec les contrats intelligents de base présentés dans l'architecture et les données de flux de contrôle afin d'être parsés par la machine virtuelle d'Ethereum. L'objectif ici de produire le code en octets et l'interface binaire programme (ABI en anglais) qui seront utilisés pour déployer les contrats sur Ethereum. La figure 4.3 donne un aperçu de la séquence de compilation des modèles BPMN en contrats intelligents écrits en Solidity sous Pupa.

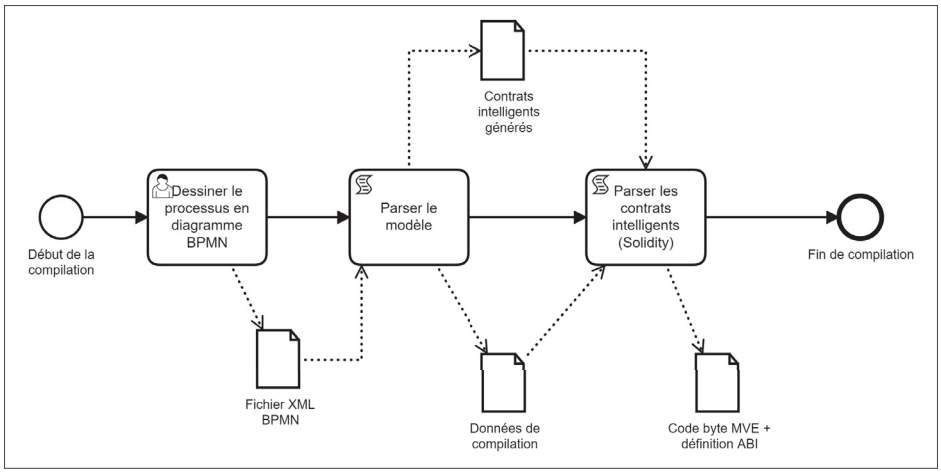


Figure 4.3 Étapes de compilation de Pupa
Adaptées de Lòpez-Pintado et al.(2019, p.12)

4.2.2 Les composants d'exécution d'un processus d'affaires

La dynamique d'exécution d'un processus d'affaires par Pupa sur la chaîne de blocs est similaire à un jeu de jeton. Lors de conversion du fichier *XML* de BPMN en objet, des indexes sont associés à chaque élément BPMN du modèle suivant un mécanisme de représentation similaire au modèle mathématique du réseau de pétri (Mutunda, 2017). Cette dynamique est la même utilisée par les travaux suivants : (García-Bañuelos *et al.*, 2017; López-Pintado *et al.*, 2017; Lòpez-Pintado *et al.*, 2019). Lorsqu'un processus est instancié, un jeton est placé sur le flux de séquence sortant de l'évènement de démarrage du processus. Une activité passe à l'état actif ou prêt à être exécuté aussitôt qu'un jeton est présent dans un de ces flux de séquence entrants. Lorsque l'exécution de l'activité est terminée, le jeton d'exécution est retiré du flux entrant et est placé ou dupliqué sur le ou les flux sortants de l'activité. Afin de réaliser cet objectif d'exécution des modèles BPMN sur chaîne, trois composants essentiels sont utilisés : la fonction *step*, la variable *marking* et les opérations bit à bit.

4.2.2.1 La fonction *step*

C'est une fonction interne utilisée pour mettre à jour l'état d'exécution du processus d'affaires sur la chaîne de bloc. Pour atteindre cet objectif, la fonction *step* utilise la variable *marking* et les opérations de bit à bit.

4.2.2.2 La variable *marking*

La variable globale *marking* est un tableau de bit coder sous forme d'entier non signé de 256 bits. Elle est utilisée pour implémenter la distribution du jeton d'exécution sur les flux de séquence. Chaque séquence est associée avec 1 bit dans cette variable (1 si la séquence a le jeton et 0 sinon). La valeur de cette variable peut prendre la forme de 2^i avec i comme la position de la séquence au sein du modèle BPMN du processus d'affaires. La première séquence du modèle est initialisée à 1. Par exemple lorsque le jeton est sur la séquence numéro 3 la variable *marking* = $2^3 = 8$. Afin de réduire la consommation de gaz, la variable *marking* n'est pas directement mise à jour, mais c'est une copie locale de cette variable encore appelée *tmpMarking* qui est manipulée. Par ailleurs, il y'a également la variable *StartedActivities* qui est un tableau de bit chargé de coder les objets de flux (activité, évènement, branchement). La valeur que peut contenir cette variable peut être une représentation en système décimal d'un tableau de bit dont le dernier bit est 1. Le reste des bits de cette variable est composé d'une série de n bits 0 avec n égale à l'index de l'objet de flux dans le modèle. La valeur d'initialisation de *StartedActivities* est égale à 0. Pour la première activité, la valeur de la variable *StartedActivities* sera égale à la valeur en base 10 du nombre binaire $10_{(2)}$ soit $10_{(2)} = 2_{(10)}$. Tout comme la variable *marking*, la variable *StartedActivities* est manipulée au travers d'une variable locale appelée *tmpStartedActivities*. Les valeurs des variables *marking* et *StartedActivities* sont influencées par l'ordre de création des éléments BPMN dans le modèle. La figure 4.4 donne un aperçu des index des éléments BPMN du modèle. Les nombres de couleurs vertes représentent les index des flux de séquence tandis que les nombres de couleurs bleues représentent les index des objets de flux.

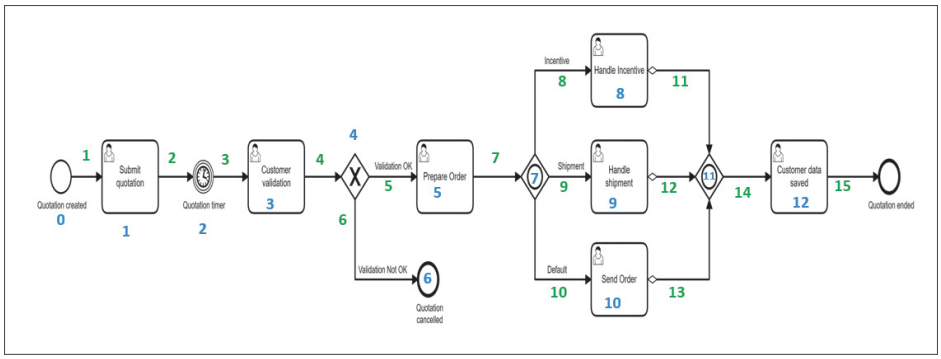


Figure 4.4 Index des éléments BPMN du processus d'affaires

4.2.2.3 Les opérations de bit à bit

En ce qui concerne les opérations de bit à bit, elles sont utilisées pour gérer les requêtes faites sur l'état d'exécution du processus. *AND* (&) est utilisé pour vérifier si un élément est actif ou pas, de même il permet l'inclusion des groupes d'expression durant les tests. L'opérateur *OR* (|) quant à lui fournit une méthode permettant de coder l'union des ensembles sous forme d'entier. Enfin, la combinaison des opérateurs *NOT* (~) et *AND* (&) permet de remplacer un ancien jeton de la variable marking par un nouveau. Le tableau 4.1 présente quelques exemples d'opérations bit à bit.

Tableau 4.1 Exemple d'expressions qui réalisent des opérations de bit à bit

Expressions	Signification
uint public marking = uint (2)	Initialisation de la variable marking. Le jeton est à la position 1 car $2^1 = 2$
tmpMarking & uint (2) ! = 0	Cette expression vérifie si le jeton est à la position 1, en d'autres termes est ce que l'activité 'Submit quotation' est active
tmpMarking &= uint (~32) ;	Enlever le jeton de la séquence située à la position 5 ($2^5 = 32$)
tmpMarking = tmpMarking & uint (~128) | uint (1024) ;	Enlever le jeton présent sur le flux de séquence situé à la position 7 ($2^7 = 128$) et le mettre sur le flux de séquence de la position 10, car $2^{10} = 1024$
tmpStartedActivities |= uint (8)	Mettre le jeton d'exécution sur l'activité d'index 3, car $8 = 1000_{(2)}$

4.3 Génération des contrats intelligents de l'étude de cas

Dans cette section, nous allons expliquer les lignes de codes Solidity générées qui contribuent à réaliser les comportements fonctionnels attendus de l'évènement de minuterie et des branchements inclusifs

4.3.1 Intégration de l'évènement de minuterie

L'extrait de code Solidity 4.1 présente la partie de la fonction *step* qui implémente la gestion du jeton d'exécution au niveau de l'évènement de minuterie. Dans les lignes 1 et 2, la fonction *step* est déclarée et la boucle *'Tant que'* est invoquée. À l'intérieur de cette boucle, une série de

tests de valeur de la variable *marking* est effectuée afin, de déterminer si le jeton d'exécution est sur un flux de séquence particulier. Dans un scénario d'exécution concurrentielle, le jeton peut se retrouver sur plusieurs séquences à la fois. À cet effet, la fonction *step* va relancer sa boucle *'Tant que'* jusqu'à ce que tous les éléments actifs soient traités. La ligne 3 traduit la série de tests effectués sur les éléments précédents l'évènement de minuterie. Dans les lignes 4-10, un test est effectué pour savoir si le jeton d'exécution est à la séquence 2. Lorsque le test réussit, la fonction de démarrage de l'évènement de minuterie est appelée. La fonction de démarrage est associée à chaque activité. Elle est utilisée pour enregistrer l'index de l'activité, l'adresse du rôle autorisé à exécuter cette activité et, émettre un évènement qui va déclencher l'exécution des opérations réalisées par cette activité. À la ligne 6, l'index de la minuterie est utilisé pour initialiser la correspondance entre la minuterie et ses flux de séquence entrants et sortants. Entre les lignes 7 à 10, le jeton est enlevé sur le flux de séquence 2, ensuite il est placé sur l'activité 2 et la prochaine instruction à l'intérieur de la boucle *'Tant que'* est effectuée. Dans les lignes 11 à 18, le jeton d'exécution est au niveau de la séquence 3. Dans le cas où l'évènement de minuterie n'a pas été initialisé, seule la fonction de démarrage du nœud 3 sera appelée. Dans le cas contraire, le nœud est démarré et le jeton est retiré de la séquence et l'activité 3 prend le jeton. Il est important de préciser ici que la fonction de démarrage de l'activité située juste après l'évènement de minuterie a une implémentation particulière. Elle vérifie si le jeton d'exécution a bel et bien traversé l'évènement de minuterie, si oui elle continue, sinon rien ne se passe.

4.3.2 Intégration des branchements inclusifs divergents et convergents

L'extrait de code 4.2 présente la portion du code Solidity pour les branchements inclusifs de divergence. Les lignes 1 à 9 permettent de dupliquer le jeton d'exécution et de positionner les copies du jeton uniquement sur les flux sortants de séquences dont la valeur de la variable booléenne associée est = 1 (vrai). Dans les lignes 10 à 27, les opérations des activités situées sur les différentes branches sortantes du branchement inclusif sont implémentées. Durant l'exécution, seules les activités précédées du flux de séquence contenant le jeton d'exécution seront effectuées.

L'extrait de code 4.3 quant à lui présente la portion de code de la fonction *step* implémentant le branchement inclusif de convergence. Lorsque le jeton est au niveau du branchement inclusif, un test est effectué afin de savoir si le branchement a été initialisé. L'initialisation du branchement consiste à connaitre le nombre de branches entrantes attendues du branchement inclusif qui contient le jeton d'exécution (lignes 1 à 11). Lorsque le branchement inclusif a été initialisé, le nombre de branches attendues contenant le jeton d'exécution est comparé avec le nombre de branches entrantes ayant déjà terminées leur exécution (lignes 12 à 21). Pour vérifier cette comparaison, la variable *inclusiveGatewayCounter* est utilisée. En effet lorsque la branche entrante qui porte le jeton a terminé son exécution, le jeton est positionné sur une séquence entrante du branchement. Ensuite la variable *inclusiveGatewayCounter* est incrémentée, le jeton est enlevé de la séquence et placé au niveau du branchement inclusif (voir lignes 22- 39)

```
1  function step(uint tmpMarking, uint tmpStartedActivities) internal {
2        while (true) {
3           ...
4           if (tmpMarking & uint(4) != 0) {
5              Use_case_AbstractWorlist(worklist).Quotation_timer_start(2);
6              timerFlows[uint(2)] = timerFlow(uint(4), uint(8));
7              tmpMarking &= uint(~4);
8              tmpStartedActivities |= uint(4);
9              continue;
10          }
11          if (tmpMarking & uint(8) != 0) {
12             if (TimerNodes[uint(3 -1)].timerRoleAddr == address(0)) {
                     Use_case_AbstractWorlist(worklist).Customer_validation_start(3);
13             } else {                    Use_case_AbstractWorlist(worklist).Customer_validation_start(3);
14                    tmpMarking &= uint(~8);
15                    tmpStartedActivities |= uint(8);
16              }
17             continue;
18          }
```

Extrait 4.1 fonction *step* - Opérations de l'évènement de minuterie

```
1           if (tmpMarking & uint(128) == uint(128)) {
2               if (handleInc)
3                   tmpMarking = tmpMarking & uint(~128)| uint(256);
4               if (handleShip)
5                   tmpMarking = tmpMarking & uint(~128)| uint(512);
6               if (sendOrder)
7                   tmpMarking = tmpMarking & uint(~128)| uint(1024);
8               continue;
9           }
10          if (tmpMarking & uint(256) != 0) {
11              Use_case_AbstractWorlist(worklist).Handle_Incentive_start(8);
12              tmpMarking &= uint(~256);
13              tmpStartedActivities |= uint(256);
14              continue;
15          }
16          if (tmpMarking & uint(512) != 0) {
17              Use_case_AbstractWorlist(worklist).Handle_shipment_start(9);
18              tmpMarking &= uint(~512);
19              tmpStartedActivities |= uint(512);
20              continue;
21          }
22          if (tmpMarking & uint(1024) != 0) {
23              Use_case_AbstractWorlist(worklist).Send_Order_start(10);
24              tmpMarking &= uint(~1024);
25              tmpStartedActivities |= uint(1024);
26              continue;
27          }
```

Extrait 4.2 fonction *step* - Opérations du branchement inclusif divergent

```
1          if (tmpMarking & uint(14336) == uint(14336)) {
2                 if(SelecIncomInclNodes[uint(11)] == 0) {
3                    uint internalcter = 0;
4                    if (handleInc)
5                       internalcter +=1;
6                    if (handleShip)
7                       internalcter +=1;
8                    if (sendOrder)
9                       internalcter +=1;
10                   SelecIncomInclNodes[uint(11)] = internalcter;
11                }
12                else {
13                   if(SelecIncomInclNodes[uint(11)] == inclusiveGatewayCounter) {
14                      tmpMarking = tmpMarking & uint(~14336) | uint(16384);
15                      inclusiveGatewayCounter = 0;
16                   }
17                   else
18                      tmpMarking = tmpMarking & uint(~14336);
19                }
20             continue;
21          }
22          if (tmpMarking & uint(2048) != 0 && (handleInc)) {
23             inclusiveGatewayCounter +=1;
24             tmpMarking &= uint(~2048);
25             tmpMarking |= uint(14336);
26             continue;
27          }
28          if (tmpMarking & uint(4096) != 0 && (handleShip)) {
29             inclusiveGatewayCounter +=1;
30             tmpMarking &= uint(~4096);
31             tmpMarking |= uint(14336);
32             continue;
33           }
34          if (tmpMarking & uint(8192) != 0 && (sendOrder)) {
35             inclusiveGatewayCounter +=1;
36             tmpMarking &= uint(~8192);
37             tmpMarking |= uint(14336);
38             continue;
39          }
```

Extrait 4.3 fonction *step* - Opérations du branchement inclusif convergent

4.4 Conclusion

Dans ce chapitre nous avons présenté le cas d'étude que nous avons utilisé afin d'implémenter la solution proposée Pupa. Après avoir illustré le diagramme BPMN du processus d'affaires du cas d'étude, nous avons présenté les différentes contraintes conceptuelles qui ont été intégrées dans le modèle BPMN. Par la suite nous avons expliqué comment la compilation du modèle BPMN en contrat intelligent écrit en Solidity est effectuée. À ce niveau, une emphase a été mise sur les composants utilisés dans l'implémentation du contrôle du flux d'exécution du processus

d'affaires sur la chaîne de blocs. À cet effet, nous avons présenté la méthode *step* qui permet de mettre à jour l'état du processus d'exécution, ainsi que la variable marking et les opérations de bit à bit qui sont utilisées par cette méthode. Par la suite, nous avons présenté des extraits de code de la méthode *step* qui réalisent les besoins fonctionnels de l'évènement de minuterie et des branchements inclusifs divergents et convergents.

CHAPITRE 5

ÉVALUATION DE LA SOLUTION ET DISCUSSION DES RÉSULTATS

Dans ce chapitre, nous allons en premier lieu décrire l'environnement de travail utilisé pour concevoir la solution proposée. Ensuite, nous allons présenter les mésures de performance que nous avons choisies. Après avoir effectué une comparaison entre notre solution et la solution Caterpillar, nous allons présenter une analyse détaillée de la solution proposée nommée Pupa.

5.1 Environnement de travail

Avant de commencer l'évaluation de notre solution, nous allons présenter dans cette section les outils que nous avons utilisés pour réaliser et tester notre solution. Dans un premier temps, nous parlerons des outils utilisés hors chaîne de blocs et ensuite nous nous attarderons sur l'environnement de chaîne de blocs.

5.1.1 Composants off-chain

Dans cette catégorie, nous retrouvons les composants tels que le moteur de compilation des contrats intelligents, le fichier d'analyse syntaxique du modèle BPMN. Ces composants sont programmés en JavaScript. Nous utilisons *node.js* pour développer ces fichiers. Les fichiers de gabarit utilisés pour le formatage des contrats intelligents sont écrits en *EJS*, un langage de modèle pour JavaScript. Les diagrammes BPMN sont réalisés avec l'application *Camunda modeler*. Une version légère est intégrée dans la solution.

Afin d'améliorer la sécurité des contrats tel qu' énoncé dans les spécifications non fonctionnelles, nous les avons, faits passer des tests de vulnérabilité. Pour cela, nous avons utilisé trois outils : *Slither* (Feist, Grieco & Groce, 2019), *smartcheck* (Tikhomirov *et al.*, 2018) et *Remix IDE* (Rem, 2022). Les tests consistaient à exécuter les outils sur du code Solidity afin de savoir s'il existe des lignes de code qui présentent des risques. Toutes les lignes potentiellement à risque qui

sont détectées vont subir une série d'opérations manuelles. Les traitements manuels effectués consistaient à classifier la sévérité du risque, identifier des éventuels faux positifs, ajuster les lignes de codes et exécuter à nouveau les outils. Cette opération sera répétée jusqu'à l'obtention d'un test positif avec les outils. Le test positif consiste tout simplement à avoir un résultat de test sans ligne de code de sévérité élevé ou moyen. Les lignes de code de sévérité faible sont acceptées uniquement si elles sont identifiées comme faux positifs.

5.1.2 Composants on-chain

Les composants de chaînes de blocs utilisés pour déployer les contrats intelligents sont les suivants :

- Ganache : est un outil que nous utilisons pour réaliser notre réseau local de chaîne de blocs Ethereum afin de déployer et tester nos contrats intelligents,
- Solidity et l'unité de gaz : les contrats intelligents sont écrits en langage Solidity. La version du compilateur Solidity est ^0.6.12. Nous avons choisi deux standards comme prix du gaz. 37 gwei pour les transactions rapides et 30 gwei pour les transactions normales. Gwei est une unité de mesure de la cryptomonnaie Ether qui veut dire giga-Wei avec Wei comme étant l'unité de base de l'Ether. En d'autres mots, 1 gwei = 1 000 000 000 wei = 10^{-9} Ether. Les choix de prix de gaz sont des montants convenables pour un temps de confirmation moyen de 15 secondes selon Eth Gas Station (https ://ethgasstation.info/),
- IPFS : les liens vers les objets exécutés sur la chaîne de blocs sont stockés et accessibles hors chaîne de blocs par l'implémentation du protocole IPFS sur la base de données MongoDB. Ainsi sur une interface web réalisée en HTML, JavaScript et Angular, les utilisateurs peuvent auditer les processus exécutés sur la chaîne de blocs. Sur le formulaire d'audit, l'utilisateur doit fournir le hash d'une transaction ou d'un processus déjà exécuté sur la chaîne de blocs (voir la figure 5.1).

Cette section vise à évaluer notre proposition de solution. Cette évaluation est effectuée sur un ordinateur portable équipé d'un processeur Intel (R) Core (TM) i7 2,00 GHz fonctionnant

sous Windows 10 édition Entreprise. Nous avons au préalable installé le composant d'exécution JavaScript Node.js version 14.16.1, l'application de base de données distribuée MongoDB, Ganache et le cadriciel Angular.

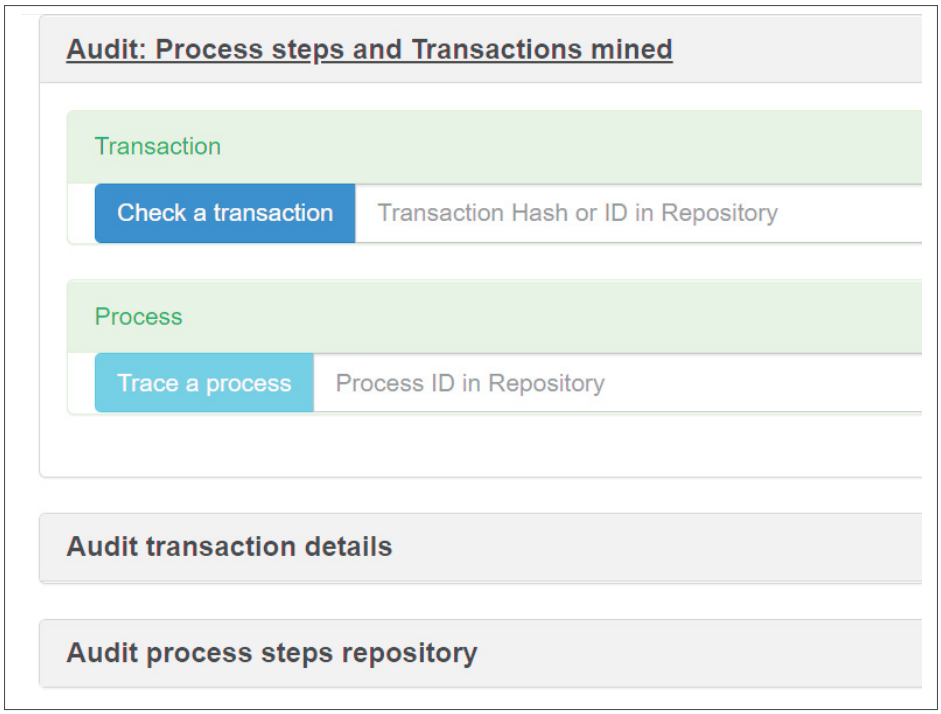


Figure 5.1 Interface de traçabilité des processus

5.2 Métriques de performance

Pour évaluer notre solution, nous avons choisi trois mesures de performance : la consommation du gaz, la limite de gaz par transaction et le nombre de lignes du contrat généré.

5.2.1 Consommation du gaz

Ici nous faisons varier le nombre d'activités utilisateurs d'un processus afin de connaitre la consommation gaz correspondante. L'objectif est de connaître l'impact de la taille du processus sur la consommation du gaz. Après avoir comparé cette performance avec celle de la solution Caterpillar, nous présenterons les détails de cette performance au sein de Pupa.

5.2.2 Taille de la transaction

Nous faisons varier le nombre de tâches utilisateurs d'un processus afin de parvenir à la limite de gaz supportée par la transaction. L'objectif ici est de connaitre le nombre maximum de tâches utilisateurs contenues dans un processus, pouvant être supportées par une transaction. Nous avons choisi la limite de transaction par défaut proposée par Ganache (soit 6721975). Avant de détailler cette performance au sein de Pupa, nous allons tout d'abord comparer cette performance avec celle de la solution Caterpillar.

5.2.3 Le nombre de lignes de code générées

Ici nous faisons varier le nombre de tâches utilisateurs ou le nombre d'évènements de minuterie afin de connaître le nombre de lignes de code du contrat intelligent qui sont générées. L'objectif est de connaître l'impact des évènements de minuterie sur le nombre de lignes de code généré, analyser cet impact par rapport à celui des tâches utilisateurs. Lors de la conception, nous avons décidé de représenter un évènement de minuterie comme une activité utilisateur. En essayant de comparer le nombre de lignes de code générées, nous voulons évaluer cette décision de conception. Par la suite nous allons essayer de comparer cette performance avec la solution Caterpillar. Après cela, nous allons analyser en détails les résultats de cette performance au sein de Pupa.

5.3 Comparaison avec la solution Caterpillar

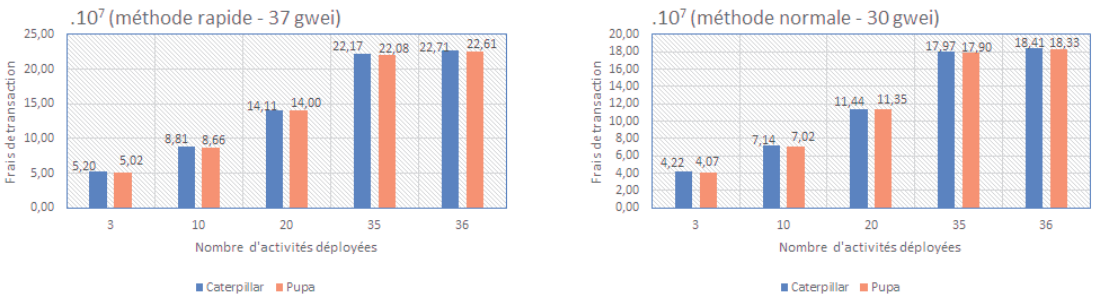
Le tableau 5.1 présente le détail du gaz utilisé lors du déploiement des contrats statiques et dynamiques par les solutions Caterpillar et Pupa. Dans ce qui va suivre, nous allons comparer la consommation du gaz entre la solution Caterpillar et notre solution. Par la suite, nous allons apprécier l'impact de la limite de gaz d'une transaction entre les deux solutions. Enfin, nous allons confronter les lignes de code de l'évènement de minuterie de Pupa avec celles de la tâche utilisateur de la solution Caterpillar.

Tableau 5.1 Comparaison du gaz utilisé pour déployer les solutions Caterpillar et Pupa

Contrats	**Gaz utilisé**		
	Caterpillar	Pupa	Difference
Contrats statiques			
Process registry	597895	599947	(2052)
BindingPolicy (pour 2 rôles)	159859	158875	984
Contrats dynamiques			
Processus avec 3 activités	1405839	1357311	48528
Processus avec 10 activités	2379973	2341349	38624
Processus avec 20 activités	3812180	3784715	27465
Processus avec 35 activités	5991255	5967356	23899
Processus avec 36 activités	6136861	6111278	25583
Processus avec 40 activités	N/A	6696979	
Processus avec 45 activités	N/A	7435751	
Processus avec 46 activités	N/A	7583547	
Processus avec 50 activités	N/A	8191320	
Processus avec 53 activités	N/A	8646270	
Processus avec 54 activités	N/A	8797993	
Processus avec 55 activités	N/A	8941850	

5.3.1 Consommation de gaz

La figure 5.2 nous donne un aperçu de la consommation de gaz de notre solution et de celle de la solution Caterpillar. Nous constatons que de manière générale notre solution consomme moins de gaz que la solution Caterpillar. Le fait d'avoir éliminé les données de flux de contrôle inutilisées lors de la génération de contrats intelligents nous a permis de réduire la consommation de gaz lors de la phase de déploiement sur la chaîne de blocs.



a) Méthode rapide

b) Méthode standard

Figure 5.2 Consommation de gaz - Caterpillar vs Pupa

5.3.2 La limite de gaz autorisée par une transaction

Nous avons analysé le nombre maximum de tâches supportées par les différentes solutions sans rencontrer le dépassement de la limite de gaz de la transaction. Nous constatons que notre solution supporte plus d'activités que la solution proposée par Caterpillar soit 55 activités contre 36. L'une des raisons majeures à cette différence est l'amélioration apportée à l'opération d'analyse syntaxique du fichier XML de BPMN. Nous avons revu les éléments extraits des fichiers de gabarit afin de ne prendre en compte que ceux qui sont étroitement liés au processus d'affaires manipulé.

5.3.3 Lignes de code générées

La figure 5.3 montre que le nombre de lignes de code générées croît linéairement pour les tâches utilisateurs de la solution Caterpillar. De plus, un évènement de minuterie de Pupa génère moins de lignes de code qu'une tâche utilisateur de Caterpillar. Lors de l'implémentation de l'évènement de minuterie comme une tâche utilisateur, nous avons revu la façon dont les éléments associés à la minuterie sont générés. Après avoir séparé les données variables des données statiques, nous avons maximisé le nombre de fonctions génériques. À l'aide des types de données personnalisées, nous avons reproduit certains comportements de la tâche utilisateur chez l'évènement de minuterie sans surcharger le code.

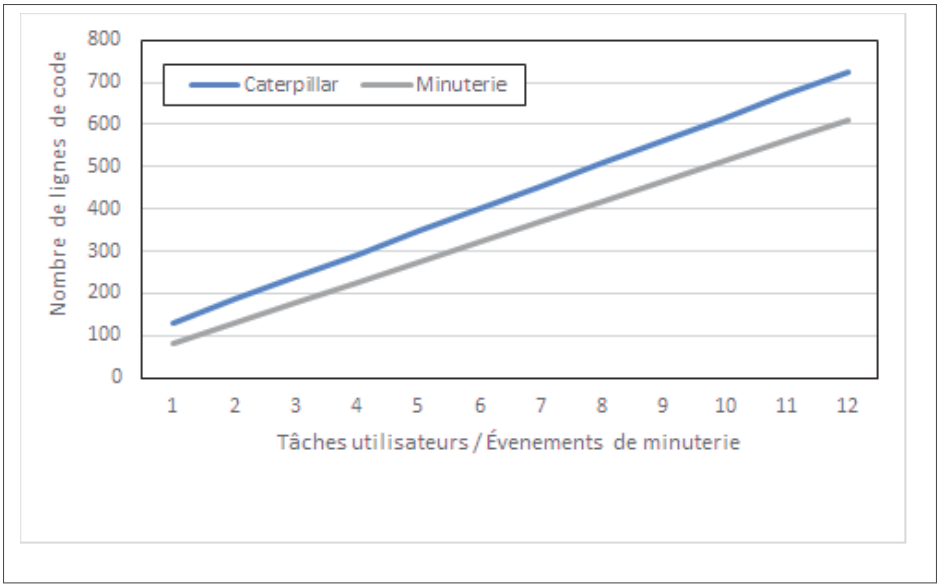


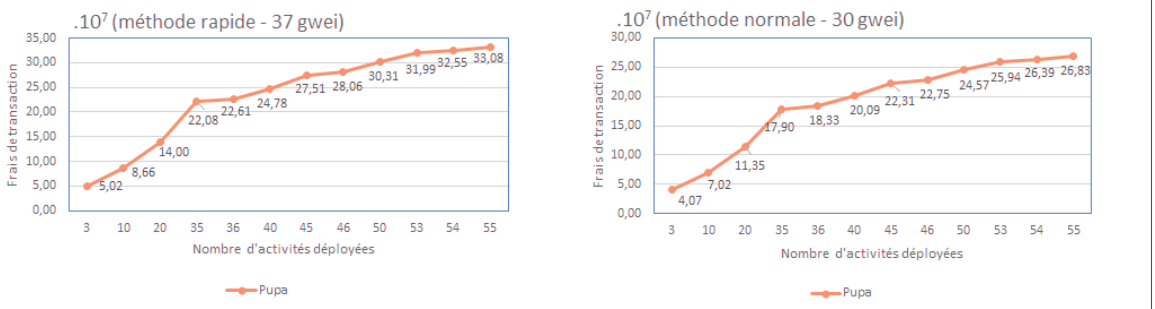
Figure 5.3 Taille processus/nombre de lignes de code générées - Caterpillar vs Pupa

5.4 Évaluation de la solution Pupa

Nous analysons ici l'impact de la taille d'un processus sur la consommation de gaz, sur la limite de gaz de la transaction. Par la suite nous allons évaluer le nombre de lignes de code générées par les évènements de minuterie et les tâches utilisateurs.

5.4.1 Impact de la taille du processus sur la consommation de gaz

Nous observons ici l'impact de la taille d'un processus sur la consommation de gaz. La figure 5.4 montre l'évolution de la consommation de gaz en fonction du nombre d'activités contenues dans le processus déployé. Nous constatons que la consommation de gaz augmente lorsque le nombre d'activités déployées augmente. Nous remarquons également que la consommation de gaz suit une croissance linéaire en fonction du nombre d'activités contenues dans le modèle BPMN déployé. Ceci s'explique par le fait que plus il y'a des activités dans un processus, plus le nombre d'instructions du contrat intelligent à compiler augmente ce qui entraîne l'augmentation de la consommation de gaz par la solution.



a) Méthode rapide b) Méthode standard

Figure 5.4 Impact de la taille du processus sur la consommation du gaz

5.4.2 Impact de la taille du processus sur la limite de gaz de la transaction

Nous avons évalué notre solution afin de connaître l'impact de la taille du processus sur la limite de gaz d'une transaction. Nous constatons que la solution Pupa peut déployer un processus comportant au maximum 55 activités sans dépasser la limite de gaz autorisée par par la transaction. Au-delà de 55, le déploiement est interrompu.

5.4.3 Impact de l'évènement de minuterie et des tâches utilisateurs sur le nombre de lignes du contrat

La figure 5.5 montre que le nombre de lignes de code générées augmente linéairement aussi bien pour le nombre de tâches utilisateurs du processus, que pour le nombre d'évènements de minuterie présents dans le processus. De plus, l'évènement de minuterie génère moins de lignes de code Solidity que la tâche utilisateur. En effet, l'utilisation des types de données personnalisées dénommées *Struct* permet de manipuler plusieurs évènements de minuterie sans avoir à surcharger le code

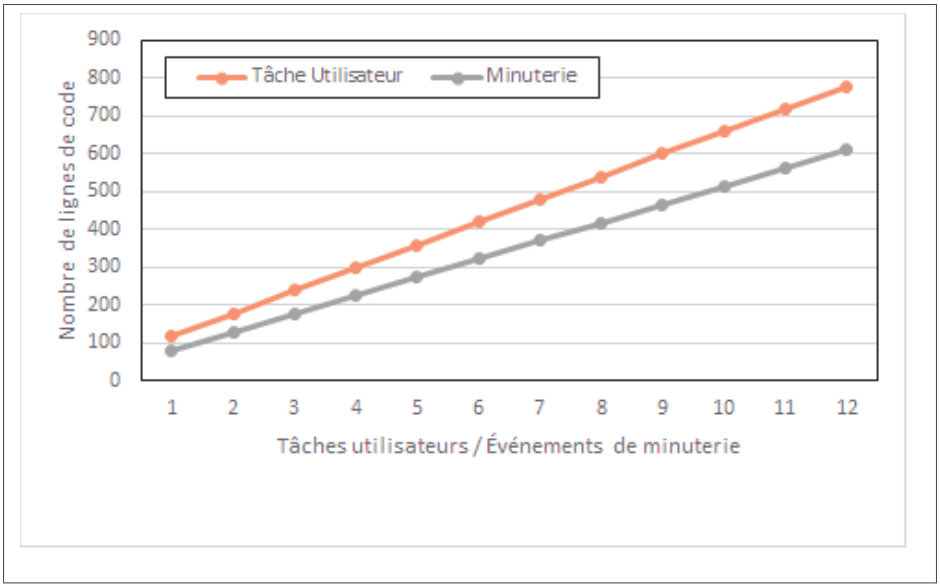


Figure 5.5 Impact de la taille du processus sur le nombre de lignes de code générées

5.5 Conclusion

Dans ce chapitre, nous avons présenté l'environnement de travail que nous avons utilisé pour concevoir et développer notre solution et pour effectuer nos tests. Ensuite, nous avons parlé des éléments de mesure de performance que nous avons utilisés. Par la suite, nous nous sommes attelés à évaluer notre solution afin de connaitre certaines performances. Ces performances concernent l'impact de la taille du processus sur la consommation de gaz, la limite de gaz de la transaction et sur le nombre de lignes de code générées. Après l'évaluation de notre solution, nous avons comparé ces performances avec celle de la solution Caterpillar. Les conclusions de l'analyse comparative montrent que notre solution consomme moins de gaz que Caterpillar. Notre solution supporte des processus plus larges que la solution proposée par Caterpillar suivant la limite de gaz de la transaction. Enfin, pour une minuterie implémentant les mêmes propriétés et fonctions qu'une tâche utilisateur, la transformation d'un évènement de minuterie génère moins de lignes de code que celle d'une activité utilisateur de Caterpillar.

CHAPITRE 6

EXPÉRIENCE PERSONNELLE

Ce chapitre, décris mon cheminement tout au long de la réalisation de la maîtrise avec mémoire à l'École de technologie supérieure. En premier lieu, je vais décrire les étapes parcourues afin de fixer et atteindre les objectifs de la recherche. Par la suite, je vais présenter les défis que j'ai surmontés afin de réaliser ce projet. Enfin, je vais énumérer les leçons apprises tout au long de mon expérience, aussi bien sur le plan professionnel que sur le plan personnel.

Au départ de ma maîtrise, le premier arrêt consistait à identifier la problématique générale de recherche, choisir les cours qui seront utiles pour la réalisation de l'objectif de recherche et définir le plan de maîtrise. Durant le second semestre de mon parcours de maîtrise notamment en automne 2020, j'ai eu l'occasion d'être sélectionné pour participer à stage au sein du département de Recherche et Développement de l'entreprise *VadimUS*. À cet effet, une proposition de projet avait été énoncée ce qui a fortement influencé le domaine de recherche. Le cours de lecture dirigé a été d'une grande importance pour commencer l'activité de revue littéraire. J'ai commencé à parcourir les travaux de recherche effectués dans le domaine de la gestion des processus dans un environnement décentralisé. En parallèle, j'essayais d'étudier les processus métiers de *VadimUS*. L'objectif était de produire une preuve de concept capable de démontrer que la gestion des processus d'affaires entre plusieurs organisations peut être effectuée de manière transparente, sécuritaire et immuable grâce à la chaîne de blocs. À cet effet, un prototype fonctionnel a été produit durant l'été 2021. Cependant, pour des raisons d'utilité interne cette piste de recherche a été abandonnée. Dans le but de demeurer agile, j'ai décidé de m'intéresser à l'amélioration des approches existantes en matière de système de gestion des processus d'affaires basé sur la chaîne de bloc. J'ai découvert qu'il existe plusieurs approches de systèmes de gestion des processus métiers basés sur la chaîne de blocs. Certains sont faits sur des chaînes de blocs privées tandis que d'autres sont faits sur des chaînes de blocs publiques. J'ai observé que l'approche la plus utilisée par ces solutions consistait à, convertir les modèles BPMN des processus métiers en contrats intelligents. Cependant, j'ai remarqué que la majeure partie de ces solutions ne

prennent pas en charge deux composants clés du langage de modélisation de BPMN à savoir : les évènements de minuterie et les branchements inclusifs. Fort de ce constat, j'ai pu définir ma proposition de recherche qui consiste à implémenter un outil de conversion de modèle BPMN en contrats intelligent avec prise en charge des évènements de minuterie et des branchements inclusifs.

À l'automne 2021, dans le cadre de mon stage, j'ai été amené à développer une nouvelle version du module de génération de scénario de prescription client. Il était question de produire des prescriptions optimales aux clients en fonction de certaines données reçues en entrée. Cette expérience m'a permis de compléter un des livrables qui était attendu de moi en entreprise.

Après cela, j'ai commencé à approfondir mes connaissances sur les concepts tels que BPM, BPMS, BPMN. Ensuite, grâce aux compétences acquises dans le cours de planification d'un projet de recherche (MTR 801), j'ai effectué une étude comparative des solutions existantes de conversion de modèle BPMN en contrat intelligent. L'objectif était de connaitre les points communs, les points forts et les limites de chaque solution. Cette analyse m'a permis de constater que la solution *Caterpillar* est une solution remarquable qui se distingue largement des autres travaux publiés. Bien que j'aie eu à utiliser *Caterpillar* pour réaliser le prototype de preuve de concept précédemment énoncé, l'analyse comparative réalisée m'a permis de valider la maturité de la solution *Caterpillar* par rapport aux autres solutions. Ensuite, j'ai effectué une inspection profonde de la solution *Caterpillar*. Cet examen m'a permis d'ausculter l'architecture, le code des contrats intelligent de base, le code du moteur de génération, le code des fichiers de gabarits utilisés lors de la génération des contrats. Cette étude minutieuse m'a permis de déceler les points faibles de la solution *Caterpillar* et partir de là pour proposer une solution améliorée. Parmi ces points faibles, on peut citer la non-prise en charge des évènements de minuterie et des branchements inclusifs lors de la conversion de BPMN en contrats intelligents. Nous avons également les faiblesses de sécurité des contrats intelligents, la consommation élevée de gaz. Comme autre limite, il y'a également l'absence d'auditabilité hors-chaîne des processus d'affaires exécutés sur Ethereum Après cette étude, la prochaine tâche a consisté à proposer une architecture pour la nouvelle solution. Ensuite, grâce aux compétences reçues du cours

intitulé systèmes et applications décentralisées ainsi que des lectures d'autres communications, j'ai commencé à modifier les contrats intelligents de Caterpillar. La modification a consisté à faire évoluer les contrats écrits en *Solidity 0.4.26* vers une version plus récente du compilateur à savoir *0.6.12* et plus. Par la suite, j'ai amélioré les contrats de base dans le but de leur permettre de supporter les évènements de minuterie et les branchements inclusifs. Après, j'ai appliqué des modèles de sécurité sur les contrats intelligents que j'ai testés avec des outils de contrôle de sécurité tels que : *Slither*, *SmartCheck* et *Remix IDE*. Après cette étape, je me suis focalisé à développer une nouvelle version du moteur de génération de contrats intelligents, de nouveaux fichiers gabarits de génération et une nouvelle interface offrant la fonctionnalité de traçabilité hors chaîne. Les différents tests effectués durant cette phase m'ont permis de définir les spécifications conceptuelles à prendre en compte durant la réalisation du modèle BPMN. En dernier lieu, je me suis attelé à optimiser les contrats générés en termes de coût de gaz, de nombre de lignes de code générées et j'ai évalué les performances par rapport à la solution *Caterpillar*. Grâce aux résultats obtenus, j'ai rédigé une communication scientifique qui met en œuvre notre plus-value et publié dans la conférence internationale *Business Process Management Conference 2022*. À la suite de l'acceptation de notre travail, nous nous préparons à le présenter et à discuter des résultats le moment venu. La dernière étape fut la rédaction de ce mémoire afin de décrire mon projet de recherche et détailler le travail réalisé durant ma maîtrise.

Afin de concrétiser ce projet, j'ai bravé plusieurs challenges :

1. Tout d'abord, l'incapacité pour Ethereum de planifier l'exécution d'une transaction m'a amené à implémenter une première version hybride qui supporte une partie de l'exécution sur chaîne et une autre partie hors chaîne. Cependant, cette option présentait des risques en matière de sécurité, de plus elle s'éloignait du standard d'exécution entièrement sur chaîne de bloc déjà implémenté par *Caterpillar*. Pour compenser cela, j'ai changé d'approche en transférant l'opération qui devait se faire automatiquement hors chaîne durant l'exécution à une interaction avec l'utilisateur durant l'exécution du processus sur la chaîne de blocs,
2. Ensuite, la sécurisation des contrats que j'ai implémentée a constitué à certains endroits un frein pour la réduction de la consommation du gaz. En fait durant les tests, un dilemme s'est

posé entre l'option de privilégier la sécurité des contrats ou la réduction du gaz. Afin de contourner cet obstacle, nous avons optimisé l'analyse syntaxique du fichier *XML* de BPMN. En effet, le contrat intelligent final est généré à la suite d'une compilation des contrats de base et des données de contrôle de flux. Ainsi, j'ai décidé d'améliorer la génération des données de contrôle pour générer uniquement les informations qui ont un lien étroit avec le processus d'affaires qui est manipulé. Ceci a permis de réduire la consommation du gaz des contrats générés tout en conservant l'aspect sécurité des contrats,

3. En dernier lieu, l'implémentation de l'interface de traçabilité avec *Angular* a connu quelques difficultés. En effet, l'accès à l'entrepôt des données de processus (*Process repository*) hors chaîne n'offrait pas la présentation attendue. J'avais des transactions qui étaient associées aux mauvaises données et vice-versa. Après un examen approfondi, je me suis rendu compte que cela était dû aux appels synchrones qui étaient effectués. Pour y remédier, j'ai étudié la programmation asynchrone avec *NodeJS* et *Angular* et je l'ai implémenté.

Pour terminer, ma maîtrise est une expérience profitable aussi bien sur le plan professionnel que sur le plan personnel. J'ai découvert le domaine de registres distribués, des plateformes de chaînes de blocs et j'ai acquis de nouvelles compétences techniques. La maîtrise avec mémoire m'a fait découvrir le monde de la recherche. La recherche m'a permis de développer un esprit innovant, un regard critique dans les domaines scientifiques. Par ailleurs, j'ai pu développer durant mes diverses interactions avec mon superviseur, mes collègues de stage, mes camarades du laboratoire FUSÉE LAB, des compétences en matière de communication, de rédaction scientifique et de travail en équipe.

CHAPITRE 7

CONCLUSION ET RECOMMANDATIONS

Conformément aux propriétés telles que la transparence, la sécurité et l'immutabilité, la chaîne de bloc est de plus en plus une technologie attractive dans le domaine de la gestion des processus d'affaires interorganisationnels. À cet effet, afin de tirer avantage de cette innovation, plusieurs travaux de recherche de conversion de modèles BPMN en contrats intelligents sur la plateforme Ethereum ont été initiés. Malgré la maturité de certaines solutions proposées, l'absence d'intégration de l'évènement de minuterie et des branchements inclusifs durant le procédé de transformation des processus d'affaires en contrats intelligents réduit la performance de ces solutions. De plus, cela participe à produire des contrats qui n'implémentent pas le comportement attendu durant l'exécution sur la chaîne de blocs.

Dans ce mémoire, nous présentons une solution de conversion de modèles BPMN en contrats intelligents, conçue en language Solidity qui prend en charge les évènements de minuterie et les branchements inclusifs. Cette proposition de solution améliore l'approche de la solution Caterpillar qui jusque-là apparaissait comme étant la solution la plus mature de transformation de modèles BPMN en contrats intelligents exécutés sur Ethereum. Pour se faire, nous transformons le comportement de base de l'évènement de minuterie pour le rendre capable de supporter des interactions avec l'utilisateur. En ce qui concerne le branchement inclusif, nous associons une variable de marquage à chaque flux de séquence rattaché au branchement ce qui nous permet d'implémenter le comportement divergent et convergent du branchement inclusif. Par ailleurs, nous proposons une interface permettant d'auditer les processus exécutés sur la chaîne de blocs. Cette solution a été expérimentée à l'aide d'un cas d'utilisation tout en spécifiant les contraintes conceptuelles à prendre en compte par le modèle BPMN. Enfin, nous avons évalué les performances de la solution proposée. Les résultats démontrent que notre solution consomme moins de gaz que la solution Caterpillar. Par ailleurs, le nombre de lignes de code générées par l'évènement de minuterie est aussi inférieur au nombre de lignes généré par la tâche utilisateur

de Caterpillar. De même, la limite de gaz d'une transaction est atteinte plus lentement par notre solution que par celle proposée par Caterpillar.

Notre solution, bien que prenant en charge plus d'éléments BPMN pourrait être encore améliorée. Comme travaux futurs, nous comptons implémenter le comportement de minuterie de démarrage et les comportements de minuterie interrupteurs et non-interrupteurs situés en bordure d'activité. En ce qui concerne les branchements inclusifs nous envisageons d'optimiser la solution afin de permettre la prise en charge des cas complexe. Avec la nouvelle version du protocole Ethereum Eth 2.0, il serait interessant d'adapter la solution Pupa afin de la rendre fonctionnelle sur Eth 2.0. Par ailleurs, nous souhaiterons offrir plus de flexibilité aux utilisateurs en leur donnant la possibilité de choisir la plateforme de chaîne de blocs sur laquelle les contrats intelligents générés de la conversion du modèle BPMN devraient être déployés. En d'autres mots, nous souhaiterons fournir une implémentation fonctionnelle de Pupa sur des plateformes de chaîne de blocs d'entreprise telles que Hyperledger Fabric, Quorum.

ANNEXE I

PUPA: SMART CONTRACTS FOR BPMN WITH TIME-DEPENDENT EVENTS AND INCLUSIVE GATEWAYS

Rodrigue Tonga Naha[1] , Kaiwen Zhang[1]

[1] Département de génie logiciel et des TI, École de technologie supérieure,
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3

Article publié à la conférence « International Conference on Business Process Management 2022. (BPM 2022, Münster, Germany) », juillet 2022

1. Abstract

The digital transformation of business processes faces a major hindrance due to the lack of trust and transparency. As blockchain and other distributed ledger (DLT) are considered key enabling technologies, there is a need for supporting tools which can deploy business models over smart contracts in order to leverage these decentralized platforms. Existing blockchain-based Business Process Management (BPM) solutions support various blockchain platforms and different types of modelling language, i.e., Ethereum and Business Process Model Notation (BPMN). However, the majority of these methods do not support processes with time events and inclusive gateways due to severe limitations imposed by smart contract programming languages. In other words, mainstream blockchain platforms do not offer a straightforward way to execute a transaction at a later time. To overcome these aforementioned issues, we propose an engine called Pupa, a blockchain-based decentralized protocol to translate business processes with time events and inclusive gateways to smart contracts. Pupa accomplishes this by adding task feature to time events, check function on top of activities succeeding time events and, listening variables to sequence flow forking or joining inclusive gateways. We implemented Pupa by extending Caterpillar, an existing BPMN solution using Solidity and Ethereum, and evaluated the performance of our proposed engine and its generated smart contracts with a baseline solution. Our results show that Pupa is competitive with baseline solutions in terms of cost and

performance, while offering additional advantages in terms of decentralization and supporting additional BPMN semantics.

keyword : Blockchain, Business Process Management, Smart Contract, BPMN, Timer, Inclusive gateway

2. Introduction

In the context of collaborative business processes, blockchain and distributed ledger technologies (DLTs) allow mutually untrusted parties to cooperate without need of a central authority. In order to leverage DLTs to provide transparency and traceability to business processes, smart contracts must be deployed which can manage and validate information generated by the business processes. While these contracts can be implemented manually using specialized programming languages, it is desirable to rely on a translation tool to automatically translate models (e.g., written in BPMN) to smart contracts (e.g., written in Solidity), effectively bridging the gap between the two domains.

To this end, several approaches have been proposed, such as Caterpillar (López-Pintado *et al.*, 2017) and Lorikeet (Tran *et al.*, 2018), and others (Corradini *et al.*, 2020; Sturm *et al.*, 2018; Prybila *et al.*, 2020). These tools are able to translate many of the core functionalities of the BPMN language directly into smart contracts that can be readily deployed into a blockchain. However, they do not support two key BPMN features : timer events and inclusive gateways. The former is used to represent event triggered by a defined time while, the latter is used to create a combination of alternative and parallel paths of process flow where all paths are evaluated.

These two BPMN expressions are particularly challenging to implement in smart contracts because of temporal constraint restriction coming from smart contracts. On the other hand, the combination of parallel and exclusive gateway behaviour covered by inclusive gateway complexifies his usage and request a higher flexibility while designing.

In this paper, we want to address these limitations with our proposed solution, Pupa, which is a new BPMN engine capable of generating Solidity code which can be deployed on Ethereum. Our contributions in this paper are :

1. We provide support for time events generation in smart contracts. Our solution allows the user to specify a time duration input, which is verified while the smart contract is executed.
2. We provide support for inclusive gateways by employing a marking variable inside the generated smart contract which can properly execute the forking and joining behaviours according to BPMN semantics.
3. We implemented our solution by extending Caterpillar. We evaluated Pupa and its generated sample code and compared it against baseline solutions in terms of cost and performance.

The remainder of this paper is organized as follows : Sec. 3 introduces background concepts and related works for blockchains and BPMN. Sec. 4 provides details of our solution Pupa. Sec. 5 illustrates our approach with an order management use case. Sec. 6 contains the results of our experimental evaluation. And finally, Sec. 7 concludes the paper.

3. Background and related works

This section is subdivided in two parts. First, we describe background knowledge regarding smart contracts and BPMN. Then, we review works related to supporting timer events and inclusive gateways in BPMN smart contracts.

3.1 Background on smart contracts

The idea of smart contracts was presented initially in 1997 (Szabo, 1997) and popularized with Ethereum blockchain. Smart contracts are computer programs deployed directly on the blockchain, which execute autonomously in response to certain triggers or transactions (Savelyev, 2017). Smart contracts help blockchain technology in establishing trust between untrusted parties. Among the blockchain platforms implementing Smart contract, Ethereum is the most globally

used (Xu *et al.*, 2017). Ethereum Smart contracts are code written in the Solidity language, and executed on the Ethereum Virtual Machine (EVM) once deployed. EVM is runtime component embedded within each full Ethereum node. To execute specific tasks, EVM uses a small set of low-level machine instructions (also called opcodes) but, sufficient enough to allow EVM to be Turing-complete. In order to store efficiency opcodes, they are encoded to bytecode. Every opcode is allocated a byte ; therefore, the maximum number of opcodes is 256 (16^2). The EVM works as a stack-based virtual machine with a depth of 1024 items. Each item is a 256-bit word and only the top 16 items are accessible at given moment in the execution. Due to these limitations some opcodes use contract memory which is a not persistent memory, to retrieve or pass data. Since everyone running an Ethereum node can perform contract execution, an attacker could try to spam the network by creating contracts including lots of computationally expensive operations. In order to avoid accidental or hostile computation wastage code, a limit on computational steps of code execution to use for each transaction is required. The base unit of computation is called gas. Since a transaction include a gas limit, any gas not used in a transaction is returned to the user ; however, any transaction with missing gas will be reverted, and the limited gas provided is consumed. The deployment cost is based on the Smart contract size, and measured in gas. The truthful execution of the code on Ethereum can serve to establish trust among untrusted parties.

3.2 Background on BPMN

To execute and perform processes inside a business process management system (BPMS), a standard language called BPMN, have been defined. BPMN is a communication tool used to represent business process flows. BPMN elements are classified in five basic categories (Object Management Group, 2016) : flow Objects, data, connecting Objects, swimlanes, and artifacts. BPMS and BPMN have been used amply by companies to simplify and automate intra-organizational processes. But for inter-organizational processes, one major challenge remains : the lack of mutual trust (Mendling *et al.*, 2018). Some existing BPMS have been able to combine blockchain and BPMN in order to, support the execution of collaborative business

processes, between mutually untrusted participants. However, the vast majority of these solution does not support business process with timer event or inclusive gateway.

3.3 Background on timer events

A timer event is a type of event which can be used to influence the commencement of an activity execution based on temporal constraints. Timer events can be used as start event, intermediate event or boundary event. Fig. I-1 gives an overview of timer events. A timer start event serves to create process instance at a given time this means that a process should start only once and should start in specific time intervals (example : every Tuesday) (Dumas *et al.*, 2013). A timer intermediate catching event works as a stopwatch. This means that, when an execution is triggered by a timer intermediate catching event, a timer begins. When the time duration is over, the sequence flow outgoing of the timer intermediate catching event is followed. A timer boundary event acts as a combination of stopwatch and an alarm clock. When an execution reaches the activity to which the boundary event is attached, a timer is started. After a specified duration, the activity is interrupted and the sequence flow leaving the timer event is followed. When the execution of an activity is interrupted after a deadline, the timer event is considered as an interrupting event. in the other hand, when the original task or sub process is not interrupted, the timer event is a non-interrupting event.

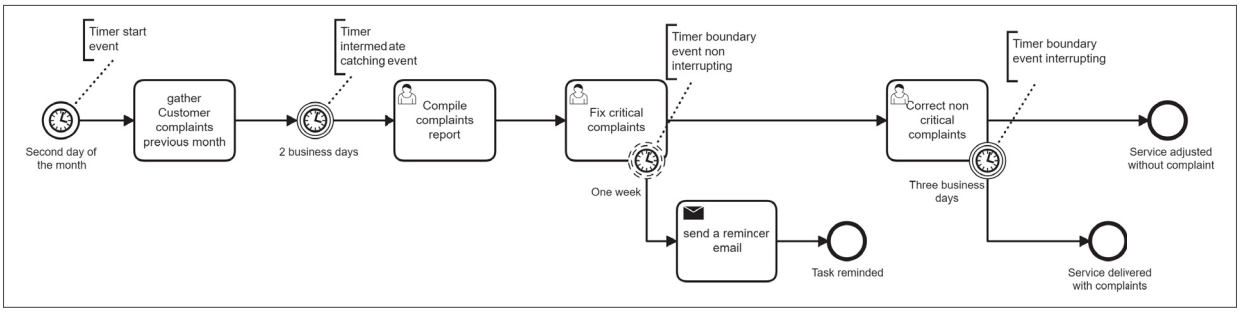


Figure-A I-1 Different Types of Timer Events

3.4 Background on inclusive gateways

Inclusive gateway, equally called inclusive decision, is used to create alternative and also, parallel paths inside a process flow. It is the combination of exclusive and parallel gateway. With inclusive gateway, all conditions are evaluated, Unlike the exclusive gateway, the true test of one condition expression does not exclude the test of other condition. All paths produced by sequence flows with a true evaluation, are taken. It should be designed such that all paths may be taken (similarly to parallel gateway) or at least one path is taken (Object Management Group, 2011) (OMG 2011). Inclusive gateway can support fork behaviour and join behaviour (Camunda.org, 2022). In the fork behaviour, all sequence flows going out of the gateway are evaluated ; the sequence flows with a true evaluation will be followed in parallel and the activities attached to these sequence flows will be executed concurrently. In the join behaviour, all the concurrent sequence flows incoming to the inclusive gateway will be checked and the inclusive gateway will only wait for the incoming sequence flows that are executed. After that, the process execution can move to the next step. Fig. I-2 shows the fork and join concept.

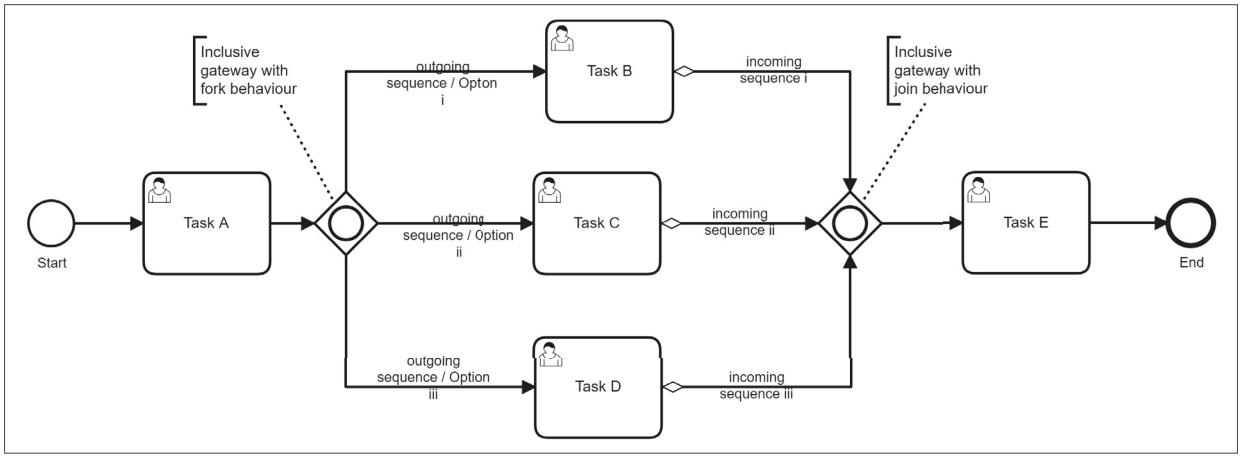


Figure-A I-2 Inclusive Gateway - Fork & Join View

3.5 Related works on process-oriented smart contract Solution

Thanks to its properties, Blockchain technology becomes more and more attractive in the field of process-centric solution. Therefore, there exist many approaches of Blockchain-based business process management systems. These approaches can be classified according to the type of blockchain used. Concerning the public blockchain, the majority of blockchain-based BPMS are made on top of Ethereum. (Sturm *et al.*, 2018) provides a light framework supporting the process execution on Ethereum. (Tran *et al.*, 2018) uses the features of model-driven engineering to simplify the development of blockchain-based process-oriented applications. (Weber *et al.*, 2016) has presented a collaborative approach of blockchain-based process execution with , a major drawback about expensive gas cost for data storage on chain. In order to improve the previous work, (García-Bañuelos *et al.*, 2017) has presented an optimized solution consisting of converting BPMN process into petri net and, compiling the output petri net into a smart contract written in Solidity. However, this solution covers a small set of BPMN items and does not care about access control of process participants on chain. Next, (López-Pintado *et al.*, 2017) designed a mature solution called Caterpillar which implements the translation of BPMN 2.0 constructs into smart contract. Caterpillar supports a large set of BPMN item, process state and execution handling. In order to improve this first version of Caterpillar, (Lòpez-Pintado *et al.*, 2019) and (López-Pintado *et al.*, 2019a), provide a compiled version of Caterpillar with integration of role-based access policy. Despite the fact that Caterpillar is a remarkable solution, it does not accept timer events and inclusive gateways

3.6 Related Works on timer events

According to (Eder *et al.*, 1999), and (Cheikhrouhou *et al.*, 2015), design and management of temporal specifications in the business process area is definitely a critical topic of research. In blockchain-based BPMS field, the integration of temporal constraints is supported by a very limited research works. However, some remarkable framework such as Lorikeet (Tran *et al.*, 2018), Caterpillar (Lòpez-Pintado *et al.*, 2019), Chorchain (Corradini *et al.*, 2020), (Sturm *et al.*, 2018), and the interpreted version of Caterpillar (López-Pintado *et al.*, 2019b), do not

support at all time events. (Klinger & Bodendorf, 2020) discuss about improving execution time from off chain component without implementing timer events. (Ladleif *et al.*, 2019) admits some challenges on timer implementation and, put it as a potential future improvement. (Abid *et al.*, 2019) provides an approach which extends Caterpillar and consists of adding time-guards inside the functions of activities implementing temporal constraints such as task duration, absolute start/end times. However, this approach doesn't provide a clear link between temporal constraints implemented and the different variants of timer events as described by OMG 11 (Object Management Group, 2011). Mavridou and Laszka (Mavridou & Laszka, 2018) use finite-state machine modelling language to generate Solidity code while taking into consideration delayed processes, and block timestamps on Ethereum. Ladleif et al (Ladleif & Weske, 2020) present a good discussion paper about challenges and alternatives solutions available regarding the integration of time constraints in blockchain-based BPMS. After comparing properties of these solutions, they present some hints which can be helpful to choose the right answer for specific scenarios.

3.7 Related works on inclusive gateways

Large number of works regarding blockchain-based BPMN engine, do not support inclusive gateway. Among them we can find some notable frameworks like Caterpillar (Lòpez-Pintado *et al.*, 2019), (López-Pintado *et al.*, 2019b), Lorikeet (Tran *et al.*, 2018), (Prybila *et al.*, 2020). On the other hand, (Sturm *et al.*, 2019) and (Sturm *et al.*, 2020) have integrated inclusive gateways in a blockchain-based business process. Their approach consists of formalizing execution of semantics of BPMN inclusive gateways on blockchain. However, these implementations face some limitations such as the deviation of the execution semantics from the BPMN standard, and the lack of support of non-block-structured process. Schinle et al (Schinle *et al.*, 2020) present an approach for the integration, execution and monitoring of modelled business processes based on Hyperledger Fabric's chaincode. This approach supports inclusive gateways in theory, but no implementation is provided. Loukil et al (Loukil *et al.*, 2021) provide a decentralized collaborative business process solution, builds on top of Ethereum, which supports gateway elements. However, no clear implementation of inclusive gateway is presented.

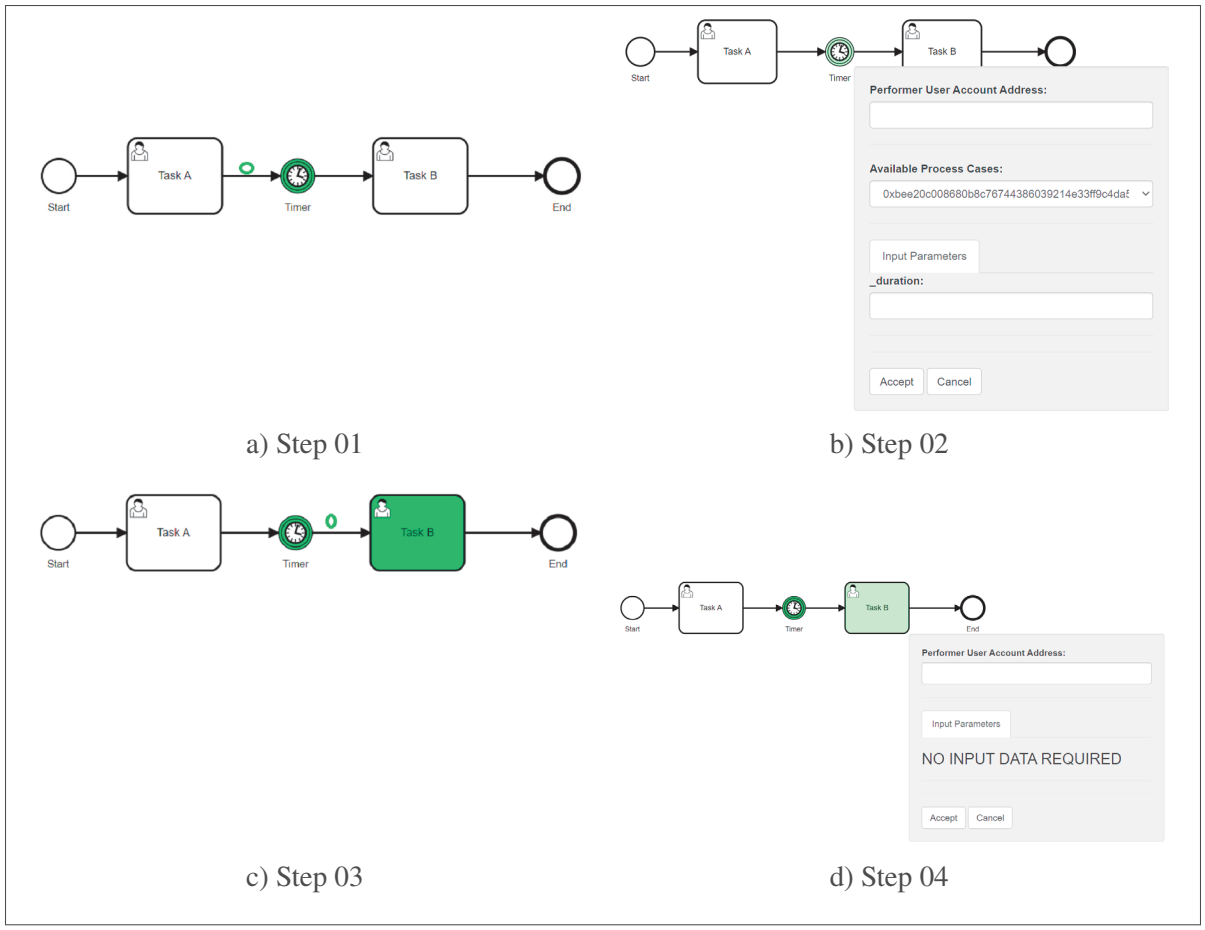
4. Details of the proposed solution

In this section, we present details of our proposed solution Pupa. First, we provide details of our solution for support timer events. Then, we explain how to support inclusive gateways. Finally, we demonstrate how Pupa functions using a case study modelled after customer order management process.

4.1 Handling time-dependent events

Our proposed timer solution has the following properties :

1. **Timer event as user task.** Timer event acts similarly as User task or Service task. Unlike other implementations (Lòpez-Pintado *et al.*, 2019), the timer event supports input data, function, and role access. Therefore, a user can input delay time in second on timer event while executing the deployed business process smart contract. Only users having the same role than the role linked to timer, can perform operation on timer.
2. **Time check done on next node.** Since it is not possible to trigger execution of solidity code at a given time with EVM, authorized user will perform this check. Execution of activity succeeding timer event in business process model is done in two steps. The first step is to check if the duration time previously specified on timer event is over. If the duration is completed then execution flow will call the second step which is the real execution function of the activity.
3. **Timer variables and timer functions.** Timer variables and timer functions are created dynamically meaning that for a business process with no timer, a smart contract with no timer variables and no timer functions is generated.



a) Step 01 b) Step 02

c) Step 03 d) Step 04

Figure-A I-3 User Flow for Timer Events

The implementation of timer event follows a token mechanism previously used by Caterpillar. Firstly, the incoming sequence flow of timer receives process token, after that, timer event is executed. At this step, an authorized user will provide the timer duration in second (d) and validate. This transaction is saved on blockchain and the timestamp of associated block (T_b) is used as the starting point of the timer stopwatch. Choosing block timestamp as time reference, helps to prevent time variation coming from various Ethereum node. The third step will move process token to the outgoing sequence flow of timer. In the last step, the token arrives on node following timer event. At this phase, a test condition consisting to verify authorized role attached to node. During this transaction, the timestamp of the current block (T_{cb}) is captured and If the duration previously set is over then the step is completed. In order words, if the remaining time is equal or less than 0, then the execution process will move if not, the check will take

place again. In order to ease the interaction with user and reduce gas, in case of the result of test condition is false, the remaining waiting time (T_r) is output. Expression A I-1 describes the value of remaining time (T_r). Fig. I-3 illustrates the entire process for a user.

$$T_r = (T_b + d) - T_{cb} \tag{A I-1}$$

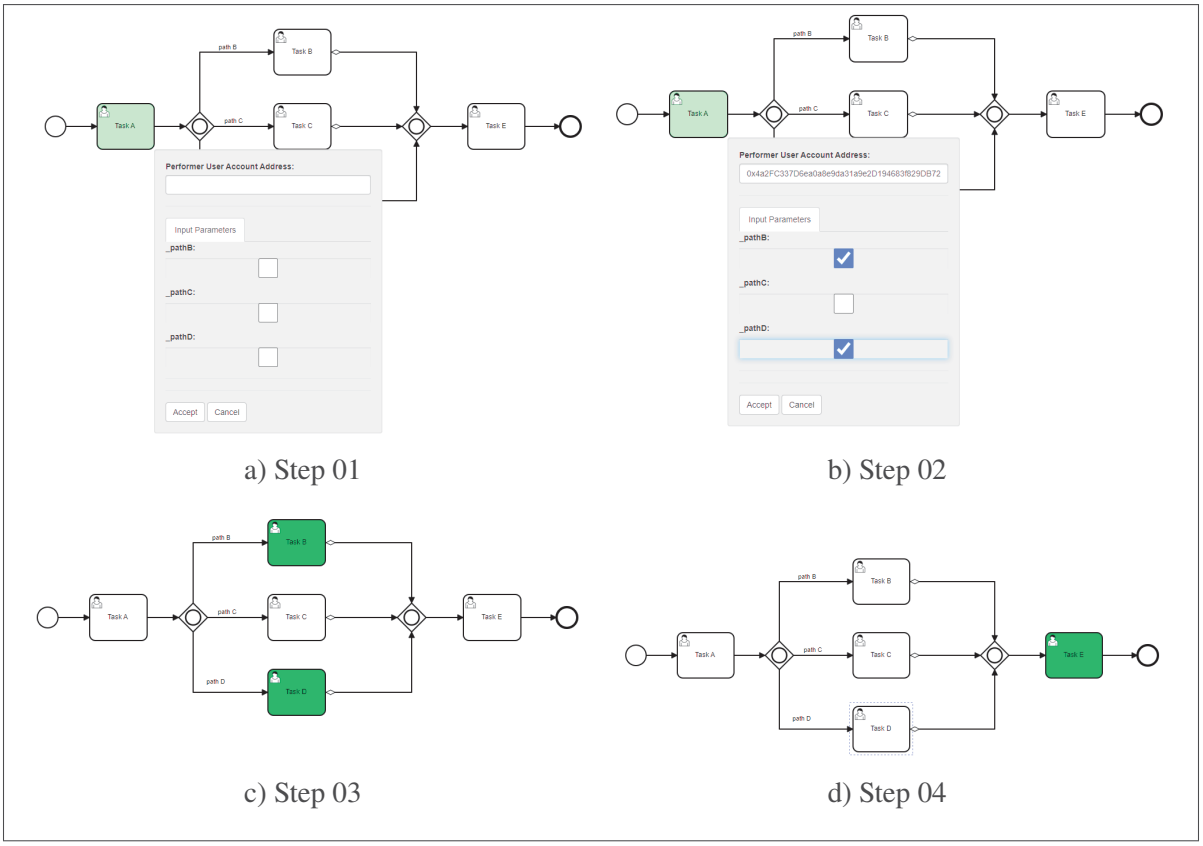
4.2 Supporting inclusive gateways

Inclusive gateway is a BPMN element classified under the category Flow objects. It is used to create an alternative and parallel path inside a process flow. Inclusive gateway can support multiple outgoing sequences flow (fork), multiple incoming sequences flow (join) or, both fork and join behaviours. Our implementation covers inclusive gateways supporting fork behaviour, and inclusive gateways supporting join behaviour.

Our proposed solution has the following characteristics :

1. **User task before inclusive gateway.** Inclusive gateway with fork behaviour should be preceded by a user task. The user task will offer the opportunity to choose the path to follow. All available choices can be selected or at least one. If no choice is selected, an error is raised and the process token will not move.
2. **Sequence flow associated to Boolean expression.** Inclusive gateway with fork behaviour should have each outgoing sequence flow linked to a Boolean expression. Also, inclusive gateway with join view and which is preceded near or far by an inclusive gateway with fork behaviour, should have respectively incoming sequence flow and outgoing sequence flow linked to the same Boolean variable.
3. **Inclusive gateway variables and functions.** The presence of inclusive gateway in a business process will create customized variables in Smart contracts. There are not specific functions for Inclusive gateway. Operations about inclusive decisions are handled inside the workflow component. These instructions have two major purposes. Firstly, to ensure that in fork behaviour, process token moves only to the selected sequence flow. If no choice is made, nothing will happen. Secondly, these instructions should ensure that in join behaviour,

process token moves to the next node following inclusive gateway only if, all the incoming chosen paths of this gateway have been completed.



a) Step 01 b) Step 02

c) Step 03 d) Step 04

Figure-A I-4 User Flow for Inclusive Gateways

Our solution consists of four steps, as shown in Fig. I-4. In Step 1 and Step 2, an activity is inserted before inclusive gateway forking. This activity is used to choose path to follow. As soon as paths are chosen, the process token will move to the corresponding path and, variables associated to outgoing and incoming sequence flows are updated. In Step 3, activities found in the paths holding process token, are performed. When process token arrives to the inclusive gateway joining, the gateway will compare the number of tokens received with the number of paths previously chosen. If there are equals, process execution flow will move to activity next to inclusive gateway. This move represents Step 4.

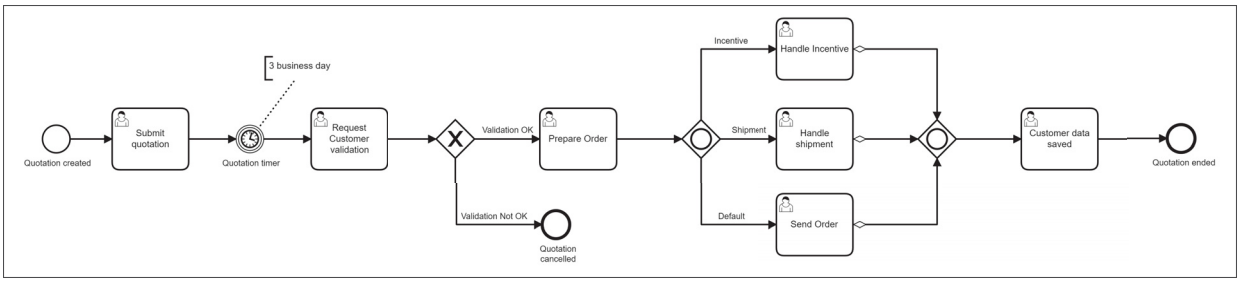


Figure-A I-5 Use Case - Customer Order

5. Use case study

We demonstrate our approach with a walk-through example shown in Fig. I-5, which models a business process managing a customer order for an Internet Service Provider company. When a customer asks for service from ISP, a quotation including the installation equipment and based on customer location, is sent to customer. Customer has three days to accept or refuse the quotation. After accepting the quotation, the next step is the preparation of customer order. Order can support at least one of the following services : incentive, shipment and, sending order only. After all the selected service have been completed, data from customer installation are recorded.

Timer events : The timer contract code generated by Pupa is presented in Listing I.1. In order to better understand this code, we will explain briefly the use of *step* function, *marking* variable and bit-wise operations.

step is an internal function used to update the business process state. It handles step of sequence flow during the whole execution of a business process. To achieve this, *step* function uses *marking* variable and bit-wise operations.

marking is a global variable (256-bits unsigned integers) in charge of the distribution of process token across the sequence flows. Each sequence flow is mingled with one bit in this variable : 1 if a token is present in the sequence flow, 0 otherwise Values supported by this variable is equal to 2^i (where *i* is the position of a sequence flow inside the business process flow starting at 0).

Regarding bit-wise operations, they are used to handle queries on the process state. *AND* (&) is used to verify if an element is started or enabled and allows testing set inclusion. *OR* (|) operator provides a method to encode the set union as an integer. Finally, the combination of *NOT* (~) and *AND* (&) serves to replace the old token from the variable *marking* by a new one.

In line 1 and 2, the step function is declared and while loop is invoked. After crossing tasks before timer event (line 3), the process token arrives at timer event node. After a successful check done on *marking* variable, the *start* function of timer node is called, the index of corresponding timer will be used to initialize the mapping between timer node and its incoming and outgoing sequence flow id and, the token will be replaced (Lines 4 - 10). The *start* function is associated to each activity and is used to register activity index and the authorized role address for this activity. In lines 11 - 20, the token moves to the node just after timer. If the timer event has not yet been initialized, only the *start* function of this node is called otherwise, the *start* function is called and the process token is removed from the node. Note that the start function of this node will check always if node is already registered to avoid duplicated records.

Inclusive gateways : Listing I.2 shows the generated code for forking inclusive gateways. The *marking* variable will move token only on sequence flow with a Boolean variable set true (Lines 1-9). In lines 10-27 code of activity located on outgoing paths of the fork inclusive gateway is represented. Only activities located on path holding process token will be executed.

When the token arrives at the joining inclusive gateway (Listing I.3), an initializing check is done. A test is done on each incoming path to know if it holds or not process token. After this test, the number of expected incoming path holding token is linked with the index of inclusive gateway (Lines 1-11). If the gateway have been already initialized as shown in lines 12 - 21, the number of expected incoming path holding process token is compared to the number of incoming path already completed. If they are equals token will move to activity next to the gateway and the global variable called *inclusiveGatewayCounter* representing counter of incoming path performed, is set to 0. Otherwise, the token will only be removed on this inclusive gateway. Lines 22 - 39 represent the generated code of all incoming sequence flow joining

to the inclusive gateway. For each sequence, the Boolean variable attached to is tested. If it is true, *inclusiveGatewayCounter* is incremented and the token move to inclusive gateway position.

```
1  function step(uint tmpMarking, uint tmpStartedActivities) internal {
2        while (true) {
3           ...
4           if (tmpMarking & uint(4) != 0) {
5              Use_case_AbstractWorlist(worklist).Quotation_timer_start(2);
6              timerFlows[uint(2)] = timerFlow(uint(4), uint(8));
7              tmpMarking &= uint(~4);
8              tmpStartedActivities |= uint(4);
9              continue;
10          }
11          if (tmpMarking & uint(8) != 0) {
12             if (TimerNodes[uint(3 -1)].timerRoleAddr == address(0)) {
                     Use_case_AbstractWorlist(worklist).Customer_validation_start(3);
13             } else {                    Use_case_AbstractWorlist(worklist).Customer_validation_start(3);
14                    tmpMarking &= uint(~8);
15                    tmpStartedActivities |= uint(8);
16              }
17             continue;
18          }
```

Extrait I.1 Step Function - Timer Instructions

```
1           if (tmpMarking & uint(128) == uint(128)) {
2              if (handleInc)
3                 tmpMarking = tmpMarking & uint(~128)| uint(256);
4              if (handleShip)
5                 tmpMarking = tmpMarking & uint(~128)| uint(512);
6              if (sendOrder)
7                 tmpMarking = tmpMarking & uint(~128)| uint(1024);
8              continue;
9           }
10          if (tmpMarking & uint(256) != 0) {
11             Use_case_AbstractWorlist(worklist).Handle_Incentive_start(8);
12             tmpMarking &= uint(~256);
13             tmpStartedActivities |= uint(256);
14             continue;
15          }
16          if (tmpMarking & uint(512) != 0) {
17             Use_case_AbstractWorlist(worklist).Handle_shipment_start(9);
18             tmpMarking &= uint(~512);
19             tmpStartedActivities |= uint(512);
20             continue;
21          }
22          if (tmpMarking & uint(1024) != 0) {
23             Use_case_AbstractWorlist(worklist).Send_Order_start(10);
24             tmpMarking &= uint(~1024);
25             tmpStartedActivities |= uint(1024);
26             continue;
27          }
```

Extrait I.2 Step Function - Inclusive Gateway Fork Behaviour

```
        if (tmpMarking & uint(14336) == uint(14336)) {
               if(SelecIncomInclNodes[uint(11)] == 0) {
                  uint internalcter = 0;
                  if (handleInc)
                     internalcter +=1;
                  if (handleShip)
                     internalcter +=1;
                  if (sendOrder)
                     internalcter +=1;
                  SelecIncomInclNodes[uint(11)] = internalcter;
               }
               else {
                  if(SelecIncomInclNodes[uint(11)] == inclusiveGatewayCounter) {
                     tmpMarking = tmpMarking & uint(~14336) | uint(16384);
                     inclusiveGatewayCounter = 0;
                  }
                  else
                     tmpMarking = tmpMarking & uint(~14336);
               }
            continue;
        }
        if (tmpMarking & uint(2048) != 0 && (handleInc)) {
            inclusiveGatewayCounter +=1;
            tmpMarking &= uint(~2048);
            tmpMarking |= uint(14336);
            continue;
        }
        if (tmpMarking & uint(4096) != 0 && (handleShip)) {
            inclusiveGatewayCounter +=1;
            tmpMarking &= uint(~4096);
            tmpMarking |= uint(14336);
            continue;
         }
        if (tmpMarking & uint(8192) != 0 && (sendOrder)) {
            inclusiveGatewayCounter +=1;
            tmpMarking &= uint(~8192);
            tmpMarking |= uint(14336);
            continue;
        }
```

Extrait I.3 Step Function - Inclusive Gateway Join Behaviour

The source code of Pupa can be downloaded under the BSD 3-clause License from *https ://github.com/rodrigueNTprojects/Pupa*

6. Result and evaluation

This section presents an experimental evaluation of Pupa. The goal here is to compare execution cost between our solution and Caterpillar.

Tableau-A I-1 Deployment Cost Comparison Between Pupa and Caterpillar

Contracts	Deployment cost		
	Caterpillar	Pupa	Difference
Statics			
Process registry	597895	599947	(2052)
BindingPolicy (For 2 role)	159859	158863	996
Dynamic			
Process with 3 tasks	1405839	1380197	25642
Process with 10 tasks	2379973	2368627	11346
Process with 20 tasks	3812180	3797209	14971
Process with 35 tasks	5991255	5967344	23911
Process with 40 tasks	N/A	6742984	
Process with 45 tasks	N/A	7435775	

6.1 Compiling and deploying new smart contracts

Contracts generated from Pupa are written in Solidity 0.6.12. Security aspect have been taken in consideration. In particular, we run a smart contract security profiler tool called Slither against statics and dynamics contracts, in order to reduce any risk. Contracts are generated, compiled and deployed thanks to JavaScript components such as EJS(Embeded JavaScript template) and TS(TypeScript).

6.2 Gas and performance evaluation

To run our evaluation, we tested in series many processes. Our goal was to analyze gas consumption, and the application load. Because Ethereum blocks have limited sizes, we tried to understand how many basics tasks a process can support when it is generated once with Pupa. We conducted gas evaluation in order to ensure that Pupa is optimized and cheaper in use. We compared deployment cost of Pupa with Caterpillar. Tests are performed on local blockchain network called Ganache. Table I-1 demonstrates that Pupa consumes less gas than Caterpillar. Also, processes with more than 35 tasks cannot be executed once on Caterpillar while Pupa is able to do so.

7. Conclusion

Time events and inclusive gateways are core features of BPMN which are difficult to support in smart contract generation tools due to the limitations of blockchain environments. In this paper, we present Pupa, a Solidity generator for BPMN which supports the missing features. Timer events are handled by adding check functions on top of the execution of activity succeeding to time event. Pupa supports inclusive gateways by attaching a marking variable to properly support joining and forking behaviours. We implemented our solution by extending Caterpillar and compared against a known baseline. For future work, we aim to implement the deferred choice timer event in order to provide a listenable feature which triggers when some events have not been selected. Furthermore, we plan to extend Pupa across multiple platforms, to facilitate the adoption of our tool on popular enterprise blockchain systems such as Hyperledger Fabric and Quorum.

ANNEXE II

CONTRAT INTELLIGENT GÉNÉRÉ POUR LE PROCESSUS D'AFFAIRE DU CAS D'UTILISATION

```
pragma solidity ^0.6.12;
import "./AbstractFactory.sol";
import "./AbstractProcess.sol";
import "./AbstractRegistry.sol";
contract Use_case_Factory is AbstractFactory {
function newInstance(address parent, address processRegistry) override public returns(address) {
Use_case_Contract newContract = new Use_case_Contract(parent, worklist, processRegistry);
return address(newContract);
}
function startInstanceExecution(address processAddress) override public {
Use_case_Contract(processAddress).startExecution();
}
}
contract Use_case_Contract is AbstractProcess {
uint public marking = uint(2);
uint public startedActivities = 0;
// Process Variables
bool private okValidation;
bool private handleInc;
bool private handleShip;
bool private sendOrder;
uint private duration = 0;struct timerNode {
address timerRoleAddr;
uint timeNowCaptured;
uint duration;
bool timeCompleted;
}
struct timerFlow {
uint flowNodeTimerIndex;
uint postMarkingTimer;
}
mapping(uint => timerNode) private TimerNodes;
mapping(uint => timerFlow) private timerFlows;
uint inclusiveGatewayCounter=0;
//mapping(uint => multiIncomInclNode) private MultiIncomInclNodes;
mapping(uint => uint) private SelecIncomInclNodes;
constructor(address _parent, address _worklist, address _processRegistry) public AbstractProcess(_parent, _worklist, _processRegistry) {
}
function startExecution() override public {
require(marking == uint(2) && startedActivities == 0);
step(uint(2), 0);
}
function handleEvent(bytes32 code, bytes32 eventType, uint _instanceIndex, bool isInstanceCompleted) override public {
// Process without calls to external contracts.
// No events to catch !!!
}
function killProcess() override public {
(marking, startedActivities) = killProcess(0, marking, startedActivities);
}
function killProcess(uint processElementIndex, uint tmpMarking, uint tmpStartedActivities) override internal returns(uint, uint) {
if(processElementIndex == 0)
tmpMarking = tmpStartedActivities = 0;
return (tmpMarking, tmpStartedActivities);
```

```
}
function broadcastSignal() override public {
(uint tmpMarking, uint tmpStartedActivities) = broadcastSignal(marking, startedActivities, 0);
step(tmpMarking, tmpStartedActivities);
}
function Submit_quotation_complete(uint elementIndex) external {
(uint tmpMarking, uint tmpStartedActivities) = (marking, startedActivities);
if(elementIndex == uint(1)) {
require(msg.sender == worklist && tmpStartedActivities & uint(2) != 0);
step(tmpMarking | uint(4), tmpStartedActivities & uint(~2));
return;
}
}
function Customer_validation_complete(uint elementIndex, bool _okValidation) external {
(uint tmpMarking, uint tmpStartedActivities) = (marking, startedActivities);
if(elementIndex == uint(3)) {
require(msg.sender == worklist && tmpStartedActivities & uint(8) != 0);
{okValidation= _okValidation;}
step(tmpMarking | uint(16), tmpStartedActivities & uint(~8));
return;
}
}
function Prepare_Order_complete(uint elementIndex, bool _handleIncentive,bool _handleShipment,bool _sendOrder) external {
(uint tmpMarking, uint tmpStartedActivities) = (marking, startedActivities);
if(elementIndex == uint(5)) {
require(msg.sender == worklist && tmpStartedActivities & uint(32) != 0);
{ handleInc = _handleIncentive;
handleShip = _handleShipment;
sendOrder = _sendOrder;}
step(tmpMarking | uint(128), tmpStartedActivities & uint(~32));
return;
}
}
function Handle_Incentive_complete(uint elementIndex) external {
(uint tmpMarking, uint tmpStartedActivities) = (marking, startedActivities);
if(elementIndex == uint(8)) {
require(msg.sender == worklist && tmpStartedActivities & uint(256) != 0);
step(tmpMarking | uint(2048), tmpStartedActivities & uint(~256));
return;
}
}
function Handle_shipment_complete(uint elementIndex) external {
(uint tmpMarking, uint tmpStartedActivities) = (marking, startedActivities);
if(elementIndex == uint(9)) {
require(msg.sender == worklist && tmpStartedActivities & uint(512) != 0);
step(tmpMarking | uint(4096), tmpStartedActivities & uint(~512));
return;
}
}
function Send_Order_complete(uint elementIndex) external {
(uint tmpMarking, uint tmpStartedActivities) = (marking, startedActivities);
if(elementIndex == uint(10)) {
require(msg.sender == worklist && tmpStartedActivities & uint(1024) != 0);
step(tmpMarking | uint(8192), tmpStartedActivities & uint(~1024));
return;
}
}
function Customer_data_saved_complete(uint elementIndex) external {
(uint tmpMarking, uint tmpStartedActivities) = (marking, startedActivities);
if(elementIndex == uint(12)) {
```

```
require(msg.sender == worklist && tmpStartedActivities & uint(4096) != 0);
step(tmpMarking | uint(32768), tmpStartedActivities & uint(~4096));
return;
}
}
function step(uint tmpMarking, uint tmpStartedActivities) internal {
while (true) {
if (tmpMarking & uint(2) != 0) {
Use_case_AbstractWorlist(worklist).Submit_quotation_start(1);
tmpMarking &= uint(~2);
tmpStartedActivities |= uint(2);
continue;
}
if (tmpMarking & uint(4) != 0) {
Use_case_AbstractWorlist(worklist).Quotation_timer_start(2);
timerFlows[uint(2)] = timerFlow(uint(4), uint(8));
tmpMarking &= uint(~4);
tmpStartedActivities |= uint(4);
continue;
}
if (tmpMarking & uint(8) != 0) {
if (TimerNodes[uint(3 -1)].timerRoleAddr == address(0)) {
Use_case_AbstractWorlist(worklist).Customer_validation_start(3);
} else {
Use_case_AbstractWorlist(worklist).Customer_validation_start(3);
tmpMarking &= uint(~8);
tmpStartedActivities |= uint(8);
}
continue;
}
if (tmpMarking & uint(16) != 0) {
tmpMarking &= uint(~16);
if (okValidation) tmpMarking |= uint(32);
else tmpMarking |= uint(64);
continue;
}
if (tmpMarking & uint(32) != 0) {
Use_case_AbstractWorlist(worklist).Prepare_Order_start(5);
tmpMarking &= uint(~32);
tmpStartedActivities |= uint(32);
continue;
}
if (tmpMarking & uint(64) != 0) {
tmpMarking &= uint(~64);
if (tmpMarking & uint(65534) == 0 && tmpStartedActivities & uint(5930) == 0) {
(tmpMarking, tmpStartedActivities) = propagateEvent("Quotation_cancelled", "Default", tmpMarking, tmpStartedActivities, uint(64));
}
continue;
}
if (tmpMarking & uint(128) == uint(128)) {
//uint ctFalseSelected = uint(3);
if (handleInc)
tmpMarking = tmpMarking & uint(~128)| uint(256);
//ctFalseSelected -=1;
if (handleShip)
tmpMarking = tmpMarking & uint(~128)| uint(512);
//ctFalseSelected -=1;
if (sendOrder)
tmpMarking = tmpMarking & uint(~128)| uint(1024);
//ctFalseSelected -=1;
```

```
174 //if(ctFalseSelected == uint(3))
175 //tmpMarking = tmpMarking & uint(128);
176 continue;
177 }
178 if (tmpMarking & uint(256) != 0) {
179 Use_case_AbstractWorlist(worklist).Handle_Incentive_start(8);
180 tmpMarking &= uint(~256);
181 tmpStartedActivities |= uint(256);
182 continue;
183 }
184 if (tmpMarking & uint(512) != 0) {
185 Use_case_AbstractWorlist(worklist).Handle_shipment_start(9);
186 tmpMarking &= uint(~512);
187 tmpStartedActivities |= uint(512);
188 continue;
189 }
190 if (tmpMarking & uint(1024) != 0) {
191 Use_case_AbstractWorlist(worklist).Send_Order_start(10);
192 tmpMarking &= uint(~1024);
193 tmpStartedActivities |= uint(1024);
194 continue;
195 }
196 if (tmpMarking & uint(14336) == uint(14336)) {
197 if(SelecIncomInclNodes[uint(11)] == 0) {
198 //if(MultiIncomInclNodes[uint(11)].selectedIncomingCounter == 0) {
199 uint internalcter = 0;
200 if (handleInc)
201 internalcter +=1;
202 if (handleShip)
203 internalcter +=1;
204 if (sendOrder)
205 internalcter +=1;
206 //MultiIncomInclNodes[uint(11)].selectedIncomingCounter = internalcter;
207 SelecIncomInclNodes[uint(11)] = internalcter;
208 }
209 else {
210 if(SelecIncomInclNodes[uint(11)] == inclusiveGatewayCounter) {
211 //if(MultiIncomInclNodes[uint(11)].selectedIncomingCounter == inclusiveGatewayCounter) {
212 tmpMarking = tmpMarking & uint(~14336) | uint(16384);
213 inclusiveGatewayCounter = 0;
214 }
215 else
216 tmpMarking = tmpMarking & uint(~14336);
217 }
218 continue;
219 }
220 if (tmpMarking & uint(2048) != 0 && (handleInc)) {
221 inclusiveGatewayCounter +=1;
222 tmpMarking &= uint(~2048);
223 tmpMarking |= uint(14336);
224 continue;
225 }
226 if (tmpMarking & uint(4096) != 0 && (handleShip)) {
227 inclusiveGatewayCounter +=1;
228 tmpMarking &= uint(~4096);
229 tmpMarking |= uint(14336);
230 continue;
231 }
232 if (tmpMarking & uint(8192) != 0 && (sendOrder)) {
233 inclusiveGatewayCounter +=1;
```

```
tmpMarking &= uint(~8192);
tmpMarking |= uint(14336);
continue;
}
if (tmpMarking & uint(16384) != 0) {
Use_case_AbstractWorlist(worklist).Customer_data_saved_start(12);
tmpMarking &= uint(~16384);
tmpStartedActivities |= uint(4096);
continue;
}
if (tmpMarking & uint(32768) != 0) {
tmpMarking &= uint(~32768);
if (tmpMarking & uint(65534) == 0 && tmpStartedActivities & uint(5930) == 0) {
(tmpMarking, tmpStartedActivities) = propagateEvent("Quotation_ended", "Default", tmpMarking, tmpStartedActivities, uint(8192));
}
continue;
}
break;
}
if(marking != 0 || startedActivities != 0) {
marking = tmpMarking;
startedActivities = tmpStartedActivities;
}
}
function getWorklistAddress() external view returns(address) {
return worklist;
}
function getInstanceIndex() external view returns(uint) {
return instanceIndex;
}
function broadcastSignal(uint tmpMarking, uint tmpStartedActivities, uint sourceChild) override internal returns(uint, uint) {
return (tmpMarking, tmpStartedActivities);
}
function addTimer (uint _TimerIndex, address _timerRoleAddr, uint _timeNowCaptured, uint _duration) internal {
TimerNodes[_TimerIndex] = timerNode(_timerRoleAddr, _timeNowCaptured, _duration, false);
}
function getEndTime (uint TimerIndex) public view returns(uint) {
uint endTime = TimerNodes[TimerIndex].timeNowCaptured + TimerNodes[TimerIndex].duration * 1 seconds;
return endTime;
}
function isTimerCompleted (uint TimerIndex) public view returns(bool) {
return TimerNodes[TimerIndex].timeCompleted;
}
function worklist_Timer_Update(uint TimerIndex, uint blockTime) external {
TimerNodes[TimerIndex].timeNowCaptured = blockTime;
}
function check_timer_duration_complete(uint TimerIndex) external {
uint captureTimeNow = block.timestamp;
if (captureTimeNow > (TimerNodes[TimerIndex].timeNowCaptured + TimerNodes[TimerIndex].duration)) {
TimerNodes[TimerIndex].timeCompleted = true;
}
}
function worklist_timer_handling_complete(uint TimerIndex, uint duration) external {
(uint tmpMarking, uint tmpStartedActivities) = (marking, startedActivities);
require(msg.sender == worklist && tmpStartedActivities & uint(timerFlows[TimerIndex].flowNodeTimerIndex) != 0);
addTimer(TimerIndex, msg.sender, 0, duration);
step(tmpMarking | uint(timerFlows[TimerIndex].postMarkingTimer), tmpStartedActivities & uint(~timerFlows[TimerIndex].flowNodeTimerIndex));
return;
}
}
```

```
pragma solidity ^0.6.12;
import "./AbstractWorklist.sol";
abstract contract Use_case_AbstractWorlist {
function Submit_quotation_start(uint) virtual external;
function Quotation_timer_start(uint) virtual external;
function Customer_validation_start(uint) virtual external;
function Prepare_Order_start(uint) virtual external;
function Handle_Incentive_start(uint) virtual external;
function Handle_shipment_start(uint) virtual external;
function Send_Order_start(uint) virtual external;
function Customer_data_saved_start(uint) virtual external;
function Submit_quotation_complete(uint) virtual external;
function Customer_validation_complete(uint, bool) virtual external;
function Prepare_Order_complete(uint, bool, bool, bool) virtual external;
function Handle_Incentive_complete(uint) virtual external;
function Handle_shipment_complete(uint) virtual external;
function Send_Order_complete(uint) virtual external;
function Customer_data_saved_complete(uint) virtual external;
function worklist_Timer_Update(uint, uint) virtual external;
function check_timer_duration_complete(uint) virtual external;
function worklist_timer_handling_complete(uint, uint) virtual external;
}
contract Use_case_Worklist is AbstractWorklist {
// Events with the information to include in the Log when a workitem is registered
event Submit_quotation_Requested(uint);
event Quotation_timer_Requested(uint);
event Customer_validation_Requested(uint);
event Prepare_Order_Requested(uint);
event Handle_Incentive_Requested(uint);
event Handle_shipment_Requested(uint);
event Send_Order_Requested(uint);
event Customer_data_saved_Requested(uint);
function Submit_quotation_start(uint elementIndex) external {
workitems.push(Workitem(elementIndex, msg.sender));
emit Submit_quotation_Requested(workitems.length - 1);
}
function Quotation_timer_start(uint elementIndex) external {
workitems.push(Workitem(elementIndex, msg.sender));
emit Quotation_timer_Requested(workitems.length - 1);
}
function Customer_validation_start(uint elementIndex) external {
uint lastIndex = workitems.length - 1;
if (workitems[lastIndex].elementIndex != elementIndex) {
workitems.push(Workitem(elementIndex, msg.sender));
emit Customer_validation_Requested(workitems.length - 1);
}
}
function Prepare_Order_start(uint elementIndex) external {
workitems.push(Workitem(elementIndex, msg.sender));
emit Prepare_Order_Requested(workitems.length - 1);
}
function Handle_Incentive_start(uint elementIndex) external {
workitems.push(Workitem(elementIndex, msg.sender));
emit Handle_Incentive_Requested(workitems.length - 1);
}
function Handle_shipment_start(uint elementIndex) external {
workitems.push(Workitem(elementIndex, msg.sender));
emit Handle_shipment_Requested(workitems.length - 1);
}
function Send_Order_start(uint elementIndex) external {
```

```
354 workitems.push(Workitem(elementIndex, msg.sender));
355 emit Send_Order_Requested(workitems.length - 1);
356 }
357 function Customer_data_saved_start(uint elementIndex) external {
358 workitems.push(Workitem(elementIndex, msg.sender));
359 emit Customer_data_saved_Requested(workitems.length - 1);
360 }
361 function Submit_quotation(uint workitemId) external {
362 require(workitemId < workitems.length && workitems[workitemId].processInstanceAddr != address(0) && canPerform(msg.sender,
        workitems[workitemId].processInstanceAddr, workitems[workitemId].elementIndex));
363 Use_case_AbstractWorlist(workitems[workitemId].processInstanceAddr).Submit_quotation_complete(workitems[workitemId].elementIndex);
364 workitems[workitemId].processInstanceAddr = address(0);
365 }
366 function Customer_validation(uint workitemId, bool _okValidation) external {
367 require(workitemId < workitems.length && workitems[workitemId].processInstanceAddr != address(0) && canPerform(msg.sender,
        workitems[workitemId].processInstanceAddr, workitems[workitemId].elementIndex));
368 Use_case_AbstractWorlist(workitems[workitemId].processInstanceAddr).Customer_validation_complete(workitems[workitemId].elementIndex,
        _okValidation);
369 workitems[workitemId].processInstanceAddr = address(0);
370 }
371 function Prepare_Order(uint workitemId, bool _handleIncentive, bool _handleShipment, bool _sendOrder) external {
372 require(workitemId < workitems.length && workitems[workitemId].processInstanceAddr != address(0) && canPerform(msg.sender,
        workitems[workitemId].processInstanceAddr, workitems[workitemId].elementIndex));
373 Use_case_AbstractWorlist(workitems[workitemId].processInstanceAddr).Prepare_Order_complete(workitems[workitemId].elementIndex,
        _handleIncentive, _handleShipment, _sendOrder);
374 workitems[workitemId].processInstanceAddr = address(0);
375 }
376 function Handle_Incentive(uint workitemId) external {
377 require(workitemId < workitems.length && workitems[workitemId].processInstanceAddr != address(0) && canPerform(msg.sender,
        workitems[workitemId].processInstanceAddr, workitems[workitemId].elementIndex));
378 Use_case_AbstractWorlist(workitems[workitemId].processInstanceAddr).Handle_Incentive_complete(workitems[workitemId].elementIndex);
379 workitems[workitemId].processInstanceAddr = address(0);
380 }
381 function Handle_shipment(uint workitemId) external {
382 require(workitemId < workitems.length && workitems[workitemId].processInstanceAddr != address(0) && canPerform(msg.sender,
        workitems[workitemId].processInstanceAddr, workitems[workitemId].elementIndex));
383 Use_case_AbstractWorlist(workitems[workitemId].processInstanceAddr).Handle_shipment_complete(workitems[workitemId].elementIndex);
384 workitems[workitemId].processInstanceAddr = address(0);
385 }
386 function Send_Order(uint workitemId) external {
387 require(workitemId < workitems.length && workitems[workitemId].processInstanceAddr != address(0) && canPerform(msg.sender,
        workitems[workitemId].processInstanceAddr, workitems[workitemId].elementIndex));
388 Use_case_AbstractWorlist(workitems[workitemId].processInstanceAddr).Send_Order_complete(workitems[workitemId].elementIndex);
389 workitems[workitemId].processInstanceAddr = address(0);
390 }
391 function Customer_data_saved(uint workitemId) external {
392 require(workitemId < workitems.length && workitems[workitemId].processInstanceAddr != address(0) && canPerform(msg.sender,
        workitems[workitemId].processInstanceAddr, workitems[workitemId].elementIndex));
393 Use_case_AbstractWorlist(workitems[workitemId].processInstanceAddr).Customer_data_saved_complete(workitems[workitemId].elementIndex);
394 workitems[workitemId].processInstanceAddr = address(0);
395 }
396 function Worklist_Timer_global(uint workitemId, uint blockTime) external {
397 require(workitems[workitemId].processInstanceAddr != address(0));
398 Use_case_AbstractWorlist(workitems[workitemId].processInstanceAddr).worklist_Timer_Update(workitems[workitemId].elementIndex, blockTime);
399 }
400 function check_timer_duration(uint workitemId) external {
401 require(workitemId < workitems.length && workitems[workitemId].processInstanceAddr != address(0) && canPerform(msg.sender,
        workitems[workitemId].processInstanceAddr, workitems[workitemId].elementIndex));
402 Use_case_AbstractWorlist(workitems[workitemId].processInstanceAddr).check_timer_duration_complete(workitems[workitemId].elementIndex);
403 }
```

```
404 function worklist_timer_handling(uint workitemId, uint _duration) external {
405 require(workitemId < workitems.length && workitems[workitemId].processInstanceAddr != address(0) && canPerform(msg.sender,
        workitems[workitemId].processInstanceAddr, workitems[workitemId].elementIndex));
406 Use_case_AbstractWorlist(workitems[workitemId].processInstanceAddr).worklist_timer_handling_complete(workitems[workitemId].elementIndex,
        _duration);
407 }
408 }
```

ANNEXE III

LOG D'EXÉCUTION DU CAS D'UTILISATION

```
--------------------------------------------------------------------------------------------------
POST /resources/task-role 200 1248.577 ms - 105
OPTIONS /models/62f18638e0310182a8d910c5 204 0.173 ms - 0
TRYING TO CREATE INSTANCE OF CONTRACT:  Process_0f7tlwe
BindingAccessControl Contract DEPLOYED and RUNNING at 0x017f277d6fc712b79dc17b770bbe86b30aebda69
Gas Used:  1101325
.....................................................................
Root Process Contract DEPLOYED and RUNNING !!! AT ADDRESS:  0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14
GAS USED:  1206519
.....................................................................
Case-creator nominated
--------------------------------------------------------------------------------------------------
POST /models/62f18638e0310182a8d910c5 200 2257.258 ms - 240
QUERYING ALL ACTIVE CONTRACTS
{
  id: '62f18638e0310182a8d910c5',
  name: 'Use_case',
  address: '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14'
}
GET /processes/ 200 327.594 ms - 108
QUERYING ACTIVATION FOR CONTRACT: 0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14
CHECKING STARTED ELEMENTS
Element ID:  Activity_0xxx0dc
Element Name:  Submit_quotation
Input Parameters:  []
bundleId:  62f18638e0310182a8d910c5
pCases:  [ '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14' ]
hrefs:  [ '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/0' ]
.....................................................................
--------------------------------------------------------------------------------------------------
GET /processes/0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14 200 749.686 ms - 15532
OPTIONS /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/0 204 0.264 ms - 0
WANT TO EXECUTE TASK: Submit_quotation, ON WORKLIST: 0x5edfb06f0e812a3304405a47b863f1a28f12dd6e
TRANSACTION: 0xeb5b7aad7913c4bcbc7664af11df09184a6cf7bcb1db3748cee36c0a7d40ed09, PENDING !!!
--------------------------------------------------------------------------------------------------
POST /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/0 200 698.693 ms - 88
QUERYING ACTIVATION FOR CONTRACT: 0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14
Transaction of Submit_quotation updated in repository with id 62f1b38ae0310182a8d910c7
CHECKING STARTED ELEMENTS
Successfull operation while updating tx ID list:  62f1b38ae0310182a8d910c7
Element ID:  Event_1y5scx8
Element Name:  Quotation_timer
Input Parameters:  [ { internalType: 'uint256', name: '_duration', type: 'uint256' } ]
bundleId:  62f18638e0310182a8d910c5
pCases:  [
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14',
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14'
]
hrefs:  [
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/0',
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/1'
]
.....................................................................
--------------------------------------------------------------------------------------------------
```

Figure-A III-1 Log 01 - Exécution du cas d'utilisation sur Pupa

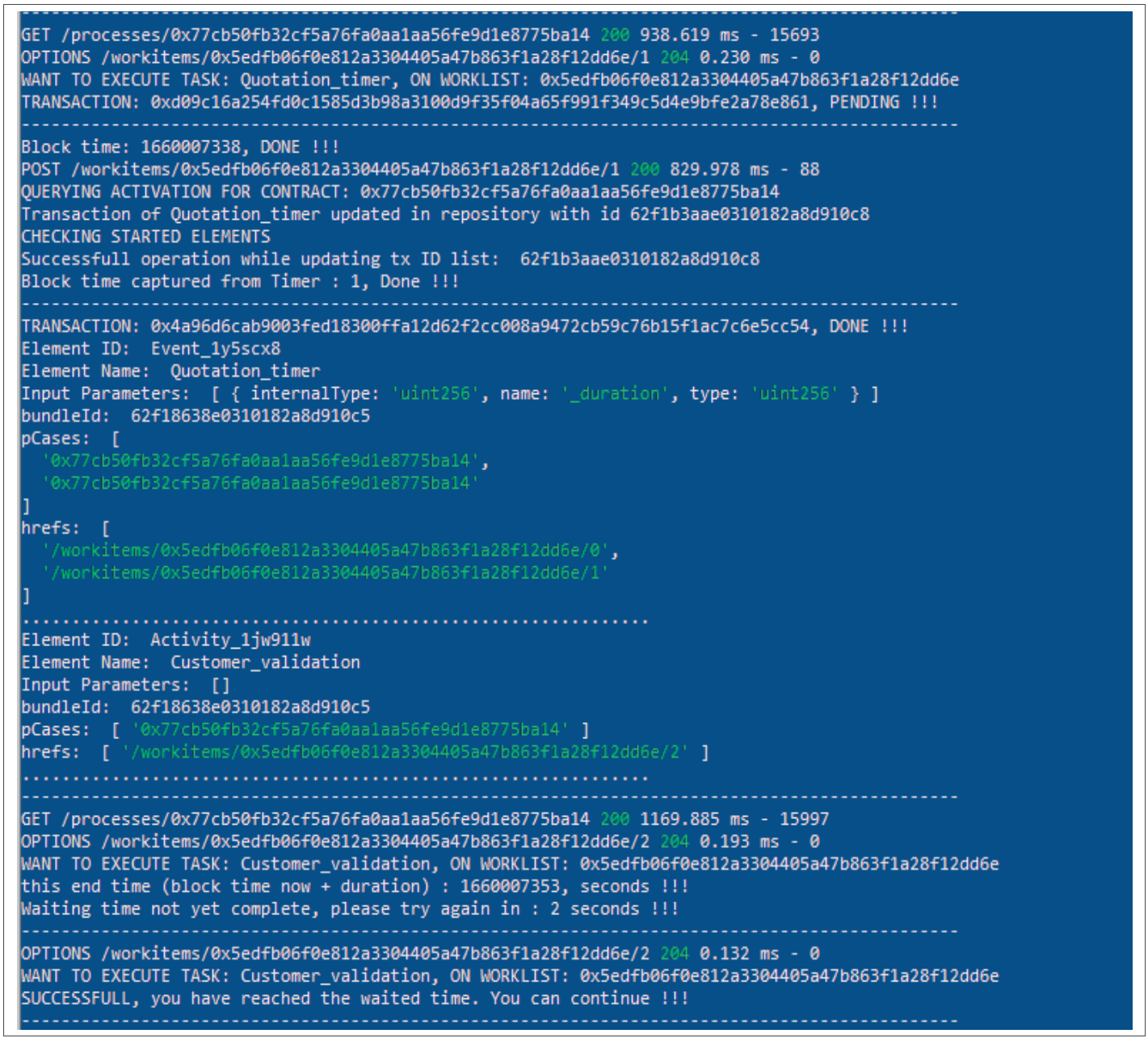


Figure-A III-2 Log 02 - Exécution du cas d'utilisation sur Pupa

```
--------------------------------------------------------------------------------------------
POST /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/2 200 869.610 ms - 88
QUERYING ACTIVATION FOR CONTRACT: 0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14
CHECKING STARTED ELEMENTS
Element ID:  Event_1y5scx8
Element Name:  Quotation_timer
Input Parameters:  [ { internalType: 'uint256', name: '_duration', type: 'uint256' } ]
bundleId:  62f18638e0310182a8d910c5
pCases:  [
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14',
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14'
]
hrefs:  [
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/0',
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/1'
]
............................................................
Element ID:  Activity_1jw911w
Element Name:  Customer_validation
Input Parameters:  [ { internalType: 'bool', name: '_okValidation', type: 'bool' } ]
bundleId:  62f18638e0310182a8d910c5
pCases:  [ '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14' ]
hrefs:  [ '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/2' ]
............................................................
--------------------------------------------------------------------------------------------
GET /processes/0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14 200 828.168 ms - 16057
OPTIONS /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/2 204 0.238 ms - 0
WANT TO EXECUTE TASK: Customer_validation, ON WORKLIST: 0x5edfb06f0e812a3304405a47b863f1a28f12dd6e
TRANSACTION: 0x8c9f37094ed635f560574935b1d4fdec2eeb96d0e242b4338336a41134067e97, PENDING !!!
--------------------------------------------------------------------------------------------
POST /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/2 200 615.980 ms - 88
QUERYING ACTIVATION FOR CONTRACT: 0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14
Transaction of Customer_validation updated in repository with id 62f1b3d7e0310182a8d910c9
CHECKING STARTED ELEMENTS
Successfull operation while updating tx ID list:  62f1b3d7e0310182a8d910c9
Element ID:  Event_1y5scx8
Element Name:  Quotation_timer
Input Parameters:  [ { internalType: 'uint256', name: '_duration', type: 'uint256' } ]
bundleId:  62f18638e0310182a8d910c5
pCases:  [
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14',
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14'
]
hrefs:  [
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/0',
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/1'
]
............................................................
Element ID:  Activity_1omhbd9
Element Name:  Prepare_Order
Input Parameters:  [
  { internalType: 'bool', name: '_handleIncentive', type: 'bool' },
  { internalType: 'bool', name: '_handleShipment', type: 'bool' },
  { internalType: 'bool', name: '_sendOrder', type: 'bool' }
]
bundleId:  62f18638e0310182a8d910c5
pCases:  [ '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14' ]
hrefs:  [ '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/3' ]
............................................................
--------------------------------------------------------------------------------------------
```

Figure-A III-3 Log 03 - Exécution du cas d'utilisation sur Pupa

```
--------------------------------------------------------------------------------------------
GET /processes/0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14 200 892.711 ms - 16175
OPTIONS /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/3 204 0.135 ms - 0
WANT TO EXECUTE TASK: Prepare_Order, ON WORKLIST: 0x5edfb06f0e812a3304405a47b863f1a28f12dd6e
TRANSACTION: 0x3adfcec8bae9c17eef741ef8c49dc5aafe78c08dc10a69376944efd4d358c5cd, PENDING !!!
--------------------------------------------------------------------------------------------
POST /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/3 200 564.429 ms - 88
QUERYING ACTIVATION FOR CONTRACT: 0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14
Transaction of Prepare_Order updated in repository with id 62f1b3e9e0310182a8d910ca
CHECKING STARTED ELEMENTS
Element ID:  Event_1y5scx8
Element Name:  Quotation_timer
Input Parameters:  [ { internalType: 'uint256', name: '_duration', type: 'uint256' } ]
bundleId:  62f18638e0310182a8d910c5
pCases:  [
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14',
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14'
]
hrefs:  [
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/0',
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/1'
]
.............................................................
Element ID:  Activity_01uak04
Element Name:  Handle_Incentive
Input Parameters:  []
bundleId:  62f18638e0310182a8d910c5
pCases:  [ '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14' ]
hrefs:  [ '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/4' ]
.............................................................
Element ID:  Activity_0k9p255
Element Name:  Send_Order
Input Parameters:  []
bundleId:  62f18638e0310182a8d910c5
pCases:  [ '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14' ]
hrefs:  [ '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/5' ]
.............................................................
--------------------------------------------------------------------------------------------
GET /processes/0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14 200 1286.858 ms - 16289
Successfull operation while updating tx ID list:  62f1b3e9e0310182a8d910ca
OPTIONS /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/4 204 0.137 ms - 0
WANT TO EXECUTE TASK: Handle_Incentive, ON WORKLIST: 0x5edfb06f0e812a3304405a47b863f1a28f12dd6e
TRANSACTION: 0x38a8e918ac66addf4ad8180f1e86cf1498d8c46118f04e3ac99722907df9c307, PENDING !!!
--------------------------------------------------------------------------------------------
POST /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/4 200 498.500 ms - 88
QUERYING ACTIVATION FOR CONTRACT: 0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14
Transaction of Handle_Incentive updated in repository with id 62f1b3f3e0310182a8d910cb
CHECKING STARTED ELEMENTS
Successfull operation while updating tx ID list:  62f1b3f3e0310182a8d910cb
Element ID:  Event_1y5scx8
Element Name:  Quotation_timer
Input Parameters:  [ { internalType: 'uint256', name: '_duration', type: 'uint256' } ]
bundleId:  62f18638e0310182a8d910c5
pCases:  [
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14',
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14'
]
hrefs:  [
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/0',
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/1'
]
.............................................................
```

Figure-A III-4 Log 04 - Exécution du cas d'utilisation sur Pupa

```
.............................................................
Element ID:  Activity_0k9p255
Element Name:  Send_Order
Input Parameters:  []
bundleId:  62f18638e0310182a8d910c5
pCases:  [ '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14' ]
hrefs:  [ '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/5' ]
.............................................................
-------------------------------------------------------------------------------------------
GET /processes/0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14 200 941.601 ms - 15988
OPTIONS /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/5 204 0.393 ms - 0
WANT TO EXECUTE TASK: Send_Order, ON WORKLIST: 0x5edfb06f0e812a3304405a47b863f1a28f12dd6e
TRANSACTION: 0x5ae79e841de76930a183be54b3451b845c400bd6b967a238d5d00233a2df005d, PENDING !!!
-------------------------------------------------------------------------------------------
POST /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/5 200 507.848 ms - 88
QUERYING ACTIVATION FOR CONTRACT: 0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14
Transaction of Send_Order updated in repository with id 62f1b404e0310182a8d910cc
CHECKING STARTED ELEMENTS
Element ID:  Event_1y5scx8
Element Name:  Quotation_timer
Input Parameters:  [ { internalType: 'uint256', name: '_duration', type: 'uint256' } ]
bundleId:  62f18638e0310182a8d910c5
pCases:  [
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14',
  '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14'
]
hrefs:  [
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/0',
  '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/1'
]
.............................................................
Element ID:  Activity_0deh1pi
Element Name:  Customer_data_saved
Input Parameters:  []
bundleId:  62f18638e0310182a8d910c5
pCases:  [ '0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14' ]
hrefs:  [ '/workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/6' ]
.............................................................
-------------------------------------------------------------------------------------------
GET /processes/0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14 200 955.036 ms - 15997
Successfull operation while updating tx ID list:  62f1b404e0310182a8d910cc
OPTIONS /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/6 204 0.210 ms - 0
WANT TO EXECUTE TASK: Customer_data_saved, ON WORKLIST: 0x5edfb06f0e812a3304405a47b863f1a28f12dd6e
TRANSACTION: 0xe34279ebcc6ed628937474404a0253f8ca195333abcd13616481ece82ae77415, PENDING !!!
-------------------------------------------------------------------------------------------
POST /workitems/0x5edfb06f0e812a3304405a47b863f1a28f12dd6e/6 200 489.063 ms - 88
QUERYING ACTIVATION FOR CONTRACT: 0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14
Transaction of Customer_data_saved updated in repository with id 62f1b416e0310182a8d910cd
CHECKING STARTED ELEMENTS
Successfull operation while updating tx ID list:  62f1b416e0310182a8d910cd
No started elements ...
-------------------------------------------------------------------------------------------
GET /processes/0x77cb50fb32cf5a76fa0aa1aa56fe9d1e8775ba14 200 495.431 ms - 15232
```

Figure-A III-5 Log 05 - Exécution du cas d'utilisation sur Pupa

LISTE DE RÉFÉRENCES

(2022). Remix - Ethereum IDE. Repéré le 2022-10-17 à https://remix.ethereum.org/.

Abid, A., Cheikhrouhou, S. & Jmaiel, M. (2019). Modelling and executing time-aware processes in trustless blockchain environment. *International Conference on Risks and Security of Internet and Systems*, pp. 325–341.

Antonopoulos, A. M. & Wood, G. (2018). *Mastering ethereum : building smart contracts and dapps*. O'reilly Media.

Buterin, V. (2015). On public and private blockchains, Aug 2015. *URL : https ://blog. ethereum. org/2015/08/07/on-public-and-private-blockchains*.

Buterin, V. et al. (2013). Ethereum white paper (2013). *URL https ://github. com/ethereum/wiki/wiki/White-Paper*.

Camunda.org. (2022). *Camunda 7 Docs, Inclusive Gateway*. Repéré à https://docs.camunda.org/manual/7.16/reference/bpmn20/gateways/inclusive-gateway/.

Cheikhrouhou, S., Kallel, S., Guermouche, N. & Jmaiel, M. (2015). The temporal perspective in business process modeling : a survey and research challenges. *Service Oriented Computing and Applications*, 9(1), 75–85.

Corradini, F., Marcelletti, A., Morichetta, A., Polini, A., Re, B. & Tiezzi, F. (2020). Engineering trustable choreography-based systems using blockchain. *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pp. 1470–1479.

Corradini, F., Muzi, C., Re, B., Rossi, L. & Tiezzi, F. (2022). BPMN 2.0 OR-Join Semantics : Global and local characterisation. *Information Systems*, 105, 101934.

Dumas, M., La Rosa, M., Mendling, J., Reijers, H. A. et al. (2013). *Fundamentals of business process management*. Springer.

Dumas, M., Rosa, M. L., Mendling, J. & Reijers, H. A. (2018). *Fundamentals of Business Process Management*. Springer Berlin Heidelberg. doi : 10.1007/978-3-662-56509-4.

Eder, J., Panagos, E. & Rabinovich, M. (1999). Time constraints in workflow systems. *International Conference on Advanced Information Systems Engineering*, pp. 286–300.

Falazi, G., Hahn, M., Breitenbücher, U. & Leymann, F. (2019). Modeling and execution of blockchain-aware business processes. *SICS Software-Intensive Cyber-Physical Systems*, 34(2), 105–116.

Feist, J., Grieco, G. & Groce, A. (2019). Slither : a static analysis framework for smart contracts. *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, pp. 8–15.

Forbes. (2022). 10 Best Cryptocurrencies Of June 2022. *Forbes*. Repéré à https://www.forbes.com/advisor/investing/cryptocurrency/top-10-cryptocurrencies/.

García-Bañuelos, L., Ponomarev, A., Dumas, M. & Weber, I. (2017). Optimized execution of business processes on blockchain. *International conference on business process management*, pp. 130–146.

Klinger, P. & Bodendorf, F. (2020). Blockchain-based Cross-Organizational Execution Framework for Dynamic Integration of Process Collaborations. *Wirtschaftsinformatik (Zentrale Tracks)*, pp. 1802–1817.

Ladleif, J. & Weske, M. (2020). Time in blockchain-based process execution. *2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*, pp. 217–226.

Ladleif, J., Weske, M. & Weber, I. (2019). Modeling and enforcing blockchain-based choreographies. *International Conference on Business Process Management*, pp. 69–85.

López-Pintado, O., García-Bañuelos, L., Dumas, M. & Weber, I. (2017). Caterpillar : A Blockchain-Based Business Process Management System. *BPM (Demos)*, 172.

López-Pintado, O., Dumas, M., García-Bañuelos, L. & Weber, I. (2019a). Dynamic role binding in blockchain-based collaborative business processes. *International Conference on Advanced Information Systems Engineering*, pp. 399–414.

López-Pintado, O., Dumas, M., García-Bañuelos, L. & Weber, I. (2019b). Interpreted execution of business process models on blockchain. *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*, pp. 206–215.

Lòpez-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I. & Ponomarev, A. (2019). Caterpillar : a business process execution engine on the Ethereum blockchain. *Software : Practice and Experience*, 49(7), 1162–1193.

Loukil, F., Boukadi, K., Abed, M. & Ghedira-Guegan, C. (2021). Decentralized collaborative business process execution using blockchain. *World Wide Web*, 24(5), 1645–1663.

Mavridou, A. & Laszka, A. (2018). Designing secure ethereum smart contracts : A finite state machine based approach. *International Conference on Financial Cryptography and Data Security*, pp. 523–540.

Mendling, J., Weber, I., Aalst, W. V. D., Brocke, J. V., Cabanillas, C., Daniel, F., Debois, S., Ciccio, C. D., Dumas, M., Dustdar, S. et al. (2018). Blockchains for business process management-challenges and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, 9(1), 1–16.

Mutunda, F. L. (2017). *Timer Service for an Ethereum BPMN Engine*. (Thèse de doctorat, Master Thesis at University Of Tartu).

Nakamoto, S. (2008). Bitcoin : A peer-to-peer electronic cash system. *Decentralized Business Review*, 21260.

Nakamura, H., Miyamoto, K. & Kudo, M. (2018). Inter-organizational business processes managed by blockchain. *International Conference on Web Information Systems Engineering*, pp. 3–17.

Norman, A. T. (2017). *Blockchain Technology Explained : The Ultimate Beginner's Guide About Blockchain Wallet, Mining, Bitcoin, Ethereum, Litecoin, Zcash, Monero, Ripple, Dash, IOTA And Smart Contracts*. by Amazon Distribution.

Object Management Group, B. T. C. (2011). Business Process Model and Notation, version 2.0. *OMG Document Number Formal/2011-01-03*.

Object Management Group, B. P. M. (2016). Notation (BPMN) Version 2.0. 2, Object Management Group, 2013. *URL http ://www. omg. org/spec/BPMN/2.0*.

Prybila, C., Schulte, S., Hochreiner, C. & Weber, I. (2020). Runtime verification for business processes utilizing the Bitcoin blockchain. *Future generation computer systems*, 107, 816–831.

Samreen, N. F. & Alalfi, M. H. (2021). A survey of security vulnerabilities in ethereum smart contracts. *arXiv preprint arXiv :2105.06974*.

Savelyev, A. (2017). Contract law 2.0 :'Smart'contracts as the beginning of the end of classic contract law. *Information & communications technology law*.

Schinle, M., Erler, C., Andris, P. N. & Stork, W. (2020). Integration, Execution and Monitoring of Business Processes with Chaincode. *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pp. 63–70.

Sturm, C., Szalanczi, J., Schönig, S. & Jablonski, S. (2018). A lean architecture for blockchain based decentralized process execution. *International conference on business process management*, pp. 361–373.

Sturm, C., Scalanczi, J., Schönig, S. & Jablonski, S. (2019). A blockchain-based and resource-aware process execution engine. *Future Generation Computer Systems*, 100, 19–34.

Sturm, C., Szalanczi, J., Jablonski, S. & Schönig, S. (2020). Decentralized control : a novel form of interorganizational workflow interoperability. *IFIP Working Conference on The Practice of Enterprise Modeling*, pp. 261–276.

Szabo, N. (1997). Formalizing and securing relationships on public networks. *First monday*. Repéré à https://journals.uic.edu/ojs/index.php/fm/article/view/548.

Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E. & Alexandrov, Y. (2018). Smartcheck : Static analysis of ethereum smart contracts. *Proceedings of the 1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, pp. 9–16.

Tran, A. B., Lu, Q. & Weber, I. (2018). Lorikeet : A Model-Driven Engineering Tool for Blockchain-Based Business Process Execution and Asset Management. *BPM (Dissertation/Demos/Industry)*, pp. 56–60.

Van Der Aalst, W. M., La Rosa, M. & Santoro, F. M. *(2016)*. Business process management. Springer.

Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A. & Mendling, J. (2016). Untrusted Business Process Monitoring and Execution Using Blockchain. Dans *Lecture Notes in Computer Science* (pp. 329–347). Springer International Publishing. doi : 10.1007/978-3-319-45348-4_19.

Xu, X., Weber, I., Staples, M., Zhu, L., Bosch, J., Bass, L., Pautasso, C. & Rimba, P. (2017, apr). A Taxonomy of Blockchain-Based Systems for Architecture Design. *2017 IEEE International Conference on Software Architecture (ICSA)*. doi : 10.1109/icsa.2017.33.