Vers une approche d'identification automatique de microservices pour les besoins de migration de systèmes d'information

Mohamed Taoufik Daoud

THÈSE de DOCTORAT DE L'UNIVERSITÉ DE LYON

Opérée au sein de :

l'Université Claude Bernard Lyon 1

Ecole Doctorale N° UMR 5205
Laboratoire d'Informatique en Images et Systèmes d'information

Spécialité de doctorat : Informatique

Soutenue publiquement le 15/11/2021, par :

Mohamed Taoufik DAOUD

Vers une approche d'identification automatique de microservices pour les besoins de migration de systèmes d'information

---

Devant le jury composé de :

Rapporteur : M. BELLATRECHE Ladjel, Professeur des Universités, ENSMA
Rapporteur : M. CHBEIR Richard, Professeur des Universités, Université de Pau
Examinatrice : Mme GRIGORI Daniela, Professeure des Universités, Université Paris-Dauphine
Examinatrice : Mme SAVONNET Marinette, Maître de Conférences - HDR, Université de Bourgogne
Examinateur : M. BOUNEKKAR Ahmed, Maître de Conférences - HDR, Université Claude Bernard
Directeur de thèse : M. BENSLIMANE Djamal, Professeur des Universités, Université Claude Bernard
Co-directrice de thèse : Mme FACI Noura, Maître de Conférences - HDR, Université Claude Bernard

Concepts et définitions

.

MSA - Microservices Architecture
IoT - Internet of Things
OOP - Object-Oriented Programming
MVC - Model-View-Controller
SOA - Service Oriented Architecture
SOAP - Simple Object Access Protocol
ESB - Enterprise Service Bus
DLLs - Dynamic-link libraries
HTTP - Hypertext Transfer Protocol
REST - Representational State Transfer
API - Application Programming Interface
URL - Uniform Resource Locator
CRM - Customer Relationship Management
UI - User Interface
DevOps - Development and Operations
VM - Virtual Machine
DDD - Domain Driven Design
BC - Bounded Context
XML - Extensible Markup Language
BPEL - Business Process Execution Language
WSDL - Web Services Definition Language

Abstract

Microservices have emerged as an alternative solution to many existing technologies allowing to break monolithic applications into “small” fine-grained, highly-cohesive, and loosely-coupled components/modules. However, identifying microservices remains a challenge that could undermine this migration success. This thesis addresses this challenge and proposes an approach for microservices automatic identification from a set of business processes (BP). The proposed multi-model approach combines different independent dependency models that represent a BP’s control dependencies, data dependencies, semantic dependencies, respectively. The approach relies on collaborative clustering to put activities together into microservices. To illustrate the approach along with demonstrating its feasibility and efficiency, we adopt two case studies namely, bike rental and cargo tracking. In term of precision, the expermental results show how different types of dependencies between activities extracted from BP specifications as inputs enable generate microservices with high quality compared to other approaches proposed in the literature, as well.

Keywords - Business process, Control/Data/Semantic dependency, Clustering, Microservice.

Résumé

Les microservices se sont révélés être une solution alternative à de nombreuses technologies existantes, permettant de décomposer les applications monolithiques en “petits” composants/modules de granularité fine, hautement cohésifs et faiblement couplés. Cependant, l’identification des microservices reste un défi majeur pouvant remettre en cause le succès de ce type de migration. Cette thèse propose une approche pour extraire automatiquement les microservices à partir d’un ensemble de processus métier (en anglais, Business Process (BP)). L’approche combine différents modèles de dépendances d’un BP, indépendants des uns des autres et représentant respectivement des dépendances de contrôle, des dépendances de données et des dépendances sémantiques. Cette approche se base sur un clustering collaboratif afin de regrouper les activités en microservices, en utilisant comme données ces modèles de dépendances. Pour illustrer la démarche et démontrer sa faisabilité et ses performances, nous avons adopté deux études de cas, la location de vélos et le suivi de cargaison. En termes de précision, les résultats expérimentaux montrent que les différents types de dépendances entre activités extraites de spécification de BPs comme paramètres d’entrée permettent de générer des microservices de meilleure qualité par rapport aux autres approches proposées dans l’état de l’art.

Mots-clés - Processus métier, Dépendance Contrôle/Données/Sémantique, Clustering, Microservice.

Remerciements

Avant toute chose, je voudrais remercier l'intégralité des personnes m'ayant permis de réaliser ce projet.

Je souhaite exprimer mes plus sincères remerciements à mon directeur de thèse Djamal BENSLIMANE et ma co-directrice de thèse Noura FACI, qui m'ont permis de mener à bien ce travail, et pour m'avoir fait partager leurs expérience et expertise dans le domaine des microservices. Je souhaite aussi les remercier du temps consacré à mes travaux et de leur soutien.

Je souhaite exprimer mes plus sincères remerciements au Pr. Sabri ABDELOUAHED de la Faculté des Sciences, Université de Sidi Mohamed Ben Abdellah, Maroc.

Je remercie mes collègues de travail à SILAC et l'ensemble de l'équipe informatique du groupe EMERAUDE. Mes plus sincères remerciements à toute l'équipe de direction du groupe EMERAUDE, notamment Mr Thierry BARTHELET, Directeur Général m'ayant accordé l'opportunité d'effectuer cette thèse, ainsi que Mr MANGIN Luc, Directeur de SILAC, m'ayant encouragé et accordé le temps nécessaire pour avancer sur mon travail de recherche.

Un remerciement tout particulier à toute ma petite famille pour leur soutien et leurs encouragements tout au long de ce projet de thèse. Aucune dédicace ne pourrait exprimer mon amour et mon attachement à ma très chère épouse Oumaima DAOUD. Depuis que je t'ai connu, tu n'as cessé de me soutenir et de m'épauler. Tu me voulais toujours le meilleur. Ton amour ne m'a procuré que confiance et stabilité. Tu as partagé avec moi les meilleurs moments de ma vie, aux moments les plus difficiles de ma vie, tu étais toujours à mes cotés, Je te remercie de ne m'avoir jamais déçu. Aucun mot ne pourrait exprimer ma gratitude, mon amour et mon respect.

Table des matières

Abstract **iv**

Résumé **vi**

Remerciements **viii**

Table des figures **xiii**

Liste des tableaux **xv**

1 Introduction générale **1**

1.1 Cadre du travail 3

1.2 Motivation et problématique 4

1.2.1 Concept des architectures monolithiques 4

1.2.2 Limites de l'architecture monolithique 5

1.2.3 µServices et leurs avantages 7

1.2.4 DevOps et les µServices 10

1.3 Contributions 12

1.4 Organisation du manuscrit 14

2 État de l'art **15**

2.1 Comprendre les µServices 17

2.1.1 Principes des µServices 17

2.1.2 Déploiement des µServices 19

2.1.3 µServices *versus* SOA 23

2.2 Adopter les µServices 26

2.2.1 Réingénierie des applications 27

2.2.2 Migration vers une architecture µServices 28

2.3 Approches existantes de migration 30
2.4 Synthèse 37

3 Extraction de μservices basée sur les flux 40
3.1 Introduction 42
3.2 Fondements 43
3.3 Dépendances issues de flux de contrôle 48
3.3.1 Analyse préliminaire des flux de contrôle dans un BP 48
3.3.2 Analyse complémentaire des flux de contrôle 51
3.4 Dépendances issues de flux de données 52
3.4.1 Criticité des données 53
3.4.2 Stratégies de calcul 53
3.5 Conclusion 55

4 Identification de μServices basée sur la sémantique 56
4.1 Introduction 58
4.2 Stratégie dirigée par les termes 58
4.3 Stratégie dirigée par les concepts 60
4.4 Stratégie dirigée par les fragments d'une ontologie 62
4.5 Calcul des dépendances sémantiques entre activités 63
4.6 Conclusion 66

5 Validation Expérimentale 68
5.1 Introduction 70
5.2 Implémentation de l'architecture proposée 70
5.2.1 Architecture générale du prototype 70
5.2.2 Clustering collaboratif 72
5.3 Première expérimentation : Application de location de vélos *Bicing* 74
5.3.1 Description de l'application de localtion de vélos 74
5.3.2 Matrice de dépendances de contrôle 77
5.3.3 Matrice de dépendances de données 77
5.3.4 Matrices de dépendances sémantiques 78
5.3.5 Évaluation des performances 80
5.4 Deuxième expérimentation : Application de suivi de cargaisons *Cargo* 85
5.4.1 Description de l'application de suivi de cargaisons 85
5.4.2 Dépendances de contrôle et de données 86

5.4.3 Dépendances sémantiques 89
5.4.4 Indicateurs de performance 91
5.4.5 Discussions 92
5.5 Conclusion 95

6 Conclusion et Perspectives **96**
6.1 Conclusion 97
6.2 Perspectives 99

Bibliographie **101**

Liste des publications **109**

Table des figures

1.1 Architecture monolithique 5
1.2 μServices vs Monolithics 8
1.3 Cycle de Développement et Déploiement par DevOps. 11
1.4 Le processus de livraison continue 11

2.1 Monolithique *vs.* SOA *vs.* μServices 18
2.2 Architecture MSA 19
2.3 Conteneurs Docker *versus* machines virtuelles (72) 20
2.4 Avantages et inconvénients des SOA 24
2.5 Taxonomie de services dans SOA (73) 24
2.6 Exemple de refactoring de code (80). 27
2.7 Patterns μServices (17) 29
2.8 Identification des services candidats basées sur les fonctionnalités (51) 32
2.9 Critères de couplage (48) 33
2.10 Graphe de dépendance (59) 35
2.11 Méthode basée sur la liaison de données (75) 35
2.12 Concepts DDD pour une conception de μServices dirigée par domaine (24) . 36
2.13 Illustration d'une conception dirigée par domaine 37

3.1 Représentation générale de notre approche d'identification des μServices . . . 46
3.3 Les quatre connecteurs logiques utilisés dans notre approche 49

4.1 Annotation des activités dirigée par les termes 59
4.2 Annotation des activités dirigée par les concepts 60
4.3 Annotation des activités dirigée par les fragments 62

5.1 Architecture générale de notre approche. 71

5.2 Modèle de processus basé sur le BPMN du système de Location de vélos. . . 74
5.3 Plug-in Disco dans Protege 79
5.4 Ontologie de domaine pour la location de vélos et ses fragments F_i 81
5.5 Mesure de l'index Dunn du clustering collaboratif - Cas de l'application Bicing 83
5.6 Mesure de l'indicateur Temps du clustering collaboratif - Cas de l'application Bicing 84
5.7 Modèle du processus métier pour le suivi des cargaisons 86
5.8 Extrait de l'ontologie du domaine de l'application Cargo du suivi de cargaisons adapté de (83) 91

Liste des tableaux

2.1 Plateformes et Frameworks de développement des μServicess 22
2.2 Différences entre μServices et SOA 25
2.3 Analyse des approches via leur entrée/sortie et le technique utilisée 39

3.1 Exemples de propriétés relatives à la criticité 53

4.1 Extrait de valeurs de similitude via *les termes* 60
4.2 Extrait de valeurs de similitude via *les concepts* 61
4.3 Extrait des valeurs de similitude via *les fragments* 63
4.4 Extrait des dépendances sémantiques - technique d'annotation dirigée par les termes 64
4.5 Extrait des dépendances sémantiques - technique d'annotation dirigée par les concepts 65

5.1 Composants du système de location de vélos 76
5.2 Criticité des artefacts / attributs- Cas de l'application Bicing 76
5.3 Dépendances de contrôle entre les activités du processus métier Bicing 77
5.4 Valeurs attribuées aux opérations (Lecture (R) & Écriture (W)) 77
5.5 Extrait de valeurs de la criticité des artefacts 77
5.6 Extrait de dépendances de données - Cas de Bicing 78
5.7 Dépendances sémantiques entre les activités utilisant les annotations orientées termes de l'application Bicing 79
5.8 Extrait de similarité Concept-activité en utilisant $Disco_2 - Cas\ de\ Bicing$ . . . 80
5.9 Extrait des valeurs de similitude entre les activités et les fragments 81
5.10 Extrait de dépendances sémantiques utilisant la technique dirigée par les fragments 81
5.11 Contrôler les dépendances avec la probabilité d'occurrence (p) fixé à 0.5 . . . 86

5.12 Composants de l'application de suivi de cargaison. 87
5.13 Criticité des artefacts / attributs pour le cas de suivi de cargaisons. 88
5.14 Les valeurs attribuées aux activités pour les opérations de lecture et écriture. . 88
5.15 Les valeurs de la criticité des artifacts. 88
5.16 Extrait des *dépendances de données* - suivi de cargaisons. 89
5.17 Dépendances sémantiques entre activités en utilisant les annotation orientées termes - cas du suivi de cargaisons. 89
5.18 Extrait de similarité Concept-activité en utilisant $Disco_2$ 90
5.19 Dépendances sémantiques entre activités en utilisant les annotations orientées concepts - cas du suivi de cargaisons 90
5.20 Extrait de similarité Fragment-activité en utilisant $Disco_2$ 90
5.21 Dépendances sémantiques entre les activités en utilisant les annotations orientées fragments cas du suivi de cargaisons 92
5.22 Mesures de performance d'identification de microservices extraites de (44) . 92
5.23 Métriques des μServices candidats par approche de l'état de l'art - cas du suivi de cargaisons. 93
5.24 Métriques des μServices candidats en utilisant notre approche - Cas de suivi de cargaison. 93
5.25 Métriques du candidat μServices utilisant notre approche - Cas de Bicing . 94

1

Introduction générale

Table des matières

1.1 Cadre du travail 3
1.2 Motivation et problématique 4
1.2.1 Concept des architectures monolithiques 4
1.2.2 Limites de l'architecture monolithique 5
1.2.3 μServices et leurs avantages 7
1.2.4 DevOps et les μServices 10
1.3 Contributions 12
1.4 Organisation du manuscrit 14

Présentation générale

1.1 Cadre du travail

Cette thèse s'inscrit dans un programme CIFRE entre le Laboratoire d'Informatique en Image et Systèmes d'information (LIRIS) et l'entreprise SILAC du Groupe EMERAUDE.

Le Groupe EMERAUDE est la société Holding regroupant à ce jour 3 sociétés françaises :

- SFPI (Société Fougeraise de Peinture Industrielle), créée en 1984 à Fougères (35),
- TLV (Thermolaquage de Vendée), créée en 2008 à Chavagnes en Paillers (85),
- SILAC (Société Industrielle de Laquage), créée en 1981 à Champlitte (70), acquisition du groupe Emeraude en 2017.

Ces trois sociétés implantées dans l'Est et l'Ouest de la France sont spécialisées dans le thermo-laquage sur aluminium destiné principalement à la menuiserie architecturale. Elles proposent aussi des services complémentaires à leur cœur de métier : stockage de profilés, sertissage de profilés à rupture de pont thermique, laquage effet bois, laquage en peinture liquide et sont aussi présentes sur d'autres marchés industriels.

Le Chiffre d'Affaires du groupe Emeraude est d'environ 45M€, représentant environ 20% du marché du thermolaquage sur aluminium destiné à la menuiserie. Le groupe souhaite continuer sa croissance et rester un acteur majeur sur ce marché. Chaque société du groupe possède actuellement son propre système d'information et ses propres applications informatiques bâties sur des progiciels propriétaires et hétérogènes. Il devient alors nécessaire de créer un système d'information commun permettant d'optimiser les décisions et les performances du groupe.

1.2 Motivation et problématique

Cette thèse se situe dans le domaine des systèmes d'information et leur urbanisation. Elle vise à apporter des solutions scientifiques et opérationnelles à la problématique d'évolution de systèmes d'information dans un contexte de fusion d'entreprises. La problématique étudiée concerne la migration d'applications monolithiques existantes bâties sur des progiciels monolithiques et hétérogènes vers des applications bâties sur le concept de μServices pour mieux supporter les futures évolutions et en faciliter la maintenance. Il sera alors question de proposer une méthode d'identification automatique des μServices à partir d'applications existantes conçues de façon indépendante et donc présentant des hétérogénéités structurelles et sémantiques.

1.2.1 Concept des architectures monolithiques

Dans l'architecture monolithique, le programme est une grande unité exécutable, en général composée de milliers de lignes de code et d'une base de données centralisée et unique. Cette structure monolithique uniforme offre des avantages mais présente aussi des inconvénients. Parmi les avantages de l'architecture monolithique mentionnés dans (79), nous retenons les 3 suivants : (1) le développement en un seul bloc de code avec un seule artefact, (2) le recours à un seul test pour l'ensemble de l'application, et (3) l'utilisation d'un seul processus de déploiement facilitant ainsi l'utilisation de l'application. En revanche, les principaux inconvénients de l'architecture monolithique concernent (1) la lenteur d'exécution de requêtes dans le cas de bases de données volumineuses, (2) la difficulté de réaliser des déploiements continus puisqu'il est nécessaire de tout redéployer à chaque évolution, et (3) la difficulté de réaliser de la maintenance à cause de l'importante taille des programmes.

L'application monolithique est divisée en plusieurs couches comme illustrée dans la figure 1.1 : une couche d'interface utilisateur (UI), une couche de services avec la base de code, et une couche d'accès aux données (DB).

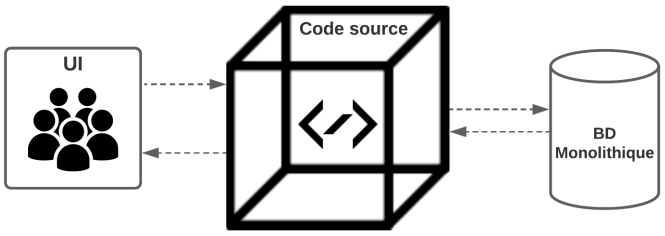


FIGURE 1.1: Architecture monolithique

Le déploiement d'une application monolithique est certes simple dans son processus. Généralement, il n'y a qu'un seul artefact à déployer réduisant sa complexité. Il est facile aussi de surveiller l'état de cet artefact. Cependant si une partie de l'application monolithique venait à tomber en panne, il est très probable que toute l'application dans son ensemble soit impactée. Ce type de panne augmente généralement les coûts de développement et de maintenance, et plus particulièrement lorsque le déploiement est réalisé dans des infrastructures d'exécution de type Cloud.

1.2.2 Limites de l'architecture monolithique

L'architecture monolithique ayant de nombreux composants partagés à grande échelle, ne peut être déployée que dans son ensemble. Cela peut demander d'exécuter plusieurs processus répartis sur différents processeurs, mais tous les processus partagent le même système d'exploitation et le même matériel (47). L'architecture monolithique doit passer par toutes les étapes de livraison continue, telles que le développement, les tests et la publication. Dans le cas de systèmes à grande échelle, ces processus prennent plus de temps, réduisant alors la flexibilité du système et augmentant son couplage et le coût de traitement (88).

Une base de code monolithique extrêmement complexe et incompréhensible peut empêcher les corrections et les ajouts de fonctionnalités. La défaillance d'un seul service entraînera immédiatement la défaillance de l'ensemble du système. Les applications monolithiques ralentissent ainsi la vitesse de développement et deviennent rapidement un obstacle en termes

d'évolutivité, de performances et au déploiement continu (16). La taille du code source augmente au fur et à mesure de l'ajout de nouvelles fonctionnalités. Le développement logiciel lui même devient complexe, et la moindre évolution d'un composant entraînera le redéploiement de l'ensemble de l'application (40).

Dans une application monolithique, tous les services sont exécutés comme un seul processus unique et sont donc fortement couplés. L'évolution d'un service, elle aussi, nécessitera le redéploiement de toute l'application.

Dans (76), De Santis et al. mettent en évidence la nécessité de changer et de faire migrer le système vers un autre paradigme pour les raisons suivantes :

- **Longs cycles de déploiement et réduction de la productivité des développeurs**. Il est certes assez facile de commencer à développer des applications avec une architecture monolithique, mais il est très difficile, voire impossible, de redéployer ces applications assez rapidement pour répondre à de nouvelles demandes métier, augmentant le temps de mise en production pour le déploiement des nouveaux services.
- **Couplage fort entre modules**. Le moindre changement dans un module peut impliquer des changements et des évolutions dans d'autres modules. En effet, les modules d'une architecture monolithique sont fortement couplés.
- **Exigence technologique**. La technologie utilisée dans un système monolithique peut s'avérer limitée à intégrer de nouveaux services développés avec de nouvelles technologies. Ceci limite grandement l'évolution des systèmes existants et leurs adaptation aux nouvelles technologies à adopter.
- **Problèmes d'évolutivité**. Les dépendances dans le code source sont difficiles à gérer dans la plupart des cas. Faire évoluer un code source au fil du temps devient problématique et source d'erreur.

1.2.3 μServices et leurs avantages

Ces dernières années, l'architecture des μServices est devenue de plus en plus populaire, que ce soit dans le domaine de la recherche ou dans le monde industriel. Elle est largement adoptée par des grandes et petites entreprises telles qu'Amazon, Netflix, LinkedIn, SoundCloud et bien d'autres. Le terme μService a été inventé en 2014 et constitue un style architectural pour développer des applications sous la forme d'une suite de petits services, ayant chacun un processus d'exécution et de communication séparé mais collaboratif (11). Les μServices sont des petits services modulaires, fonctionnels, autonomes et indépendants permettant de fournir des solutions plus agiles. Ils deviennent de plus en plus un moyen alternatif pour surmonter les défis architecturaux et organisationnels d'un système monolithique (53), en décomposant le système monolithe en un ensemble de petits services (16). Ces services communiquent entre eux via une couche proxy d'API, jouant un rôle important dans la manière dont les clients accèdent aux services, et interdisant au monde extérieur de connaître la structure exactes du service exposant une fonction spécifique.

Les μServices constituent un style architectural permettant de remédier aux limites des systèmes monolithiques (82; 28), et peuvent être étendus plus facilement dans l'environnement Cloud natif par rapport aux applications monolithiques. L'évolutivité vers une architecture μServices permet de définir des applications sous forme d'un ensemble de services autonomes, indépendants, et offre plusieurs avantages dont la réduction de la complexité des applications monolithiques. Le niveau de granularité du μService définit son champ d'action ou son domaine métier. Les μServices confèrent au système une architecture modulaire facilitant ainsi l'évolution des modules et l'utilisation éventuelle des technologies de développement et de conception diverses.

Les μServices présentent plusieurs avantages. Nous citons plus particulièrement les suivants :

- **Découplage**. Les unités fonctionnelles au sein d'un système sont faiblement couplées ;

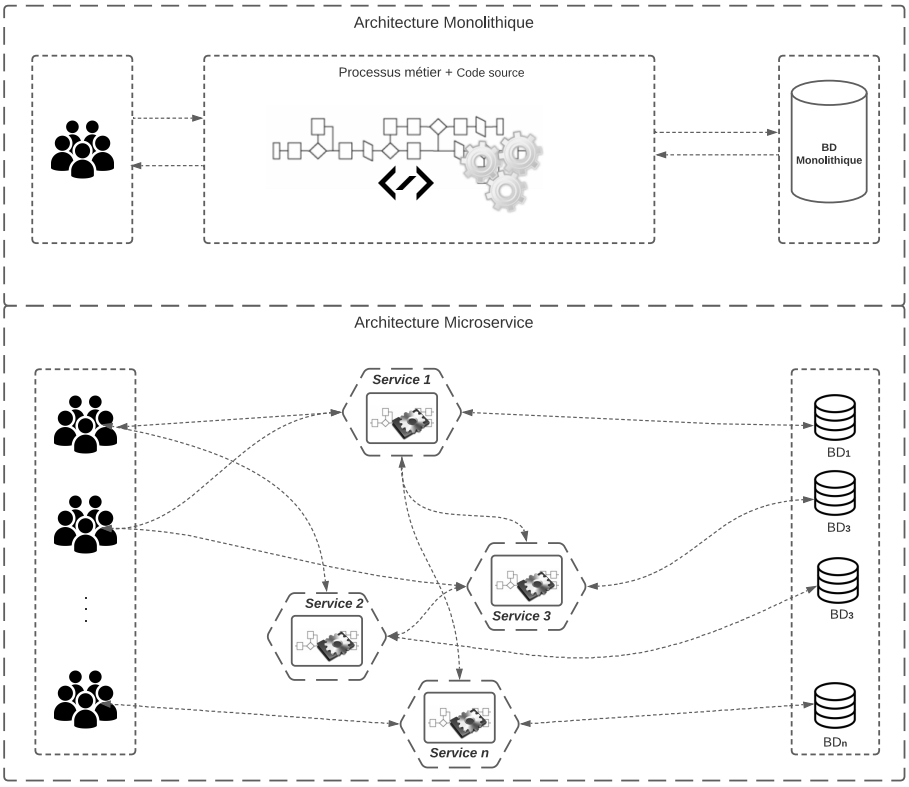


FIGURE 1.2: μServices vs Monolithics

en d'autres termes, les services sont indépendants en terme de code. Par conséquent, l'application basée sur les μServices est généralement facile à créer, à modifier et supporte le passage à l'échelle.

- **Agilité**. Les μServices prennent en charge le développement agile. Chaque nouvelle fonctionnalité est capable de s'adapter à l'environnement, et est souvent rapidement développée, facile à remplacer et plus simple à comprendre pour un développeur.
- **Indépendance**. Au niveau du développement, tous les μServices peuvent généralement être facilement étendus en prenant en charge leurs unités fonctionnelles individuelles. Au niveau du déploiement, ils seront déployés séparément dans n'importe quel environnement d'application.
- **Évolutivité**. Les composants individuels peuvent être mis à niveau selon les besoins de manière plus naturelle. En effet, les composants ont une taille relativement petite représentent une seule fonctionnalité métier avec un processus autonome et des APIs indépendants du langage. Tout ceci amène des avantages de maintenabilité et d'extensibilité.
- **Autonomie**. Les équipes de développement peuvent travailler indépendamment les unes des autres pour augmenter la vitesse de livraison. Chaque μService fournit une capacité métier unique pouvant être maintenue indépendamment.
- **Hétérogénéité technologique**. Différents technologies peuvent être utilisées au niveau programmation, SGBD, et IDE, et ce en fonction des choix considérés comme les plus appropriés par les programmeurs. Cela rendra le développement plus efficace et accélérera le déploiement.
- **Cycle de vie plus long**. Le cycle de vie d'une application à base de μServices peut être prolongé plus longtemps (peut-être indéfiniment), car n'importe quel service peut être remplacé par un meilleur service sans impacter considérablement l'application entière, optimisant ainsi le retour sur investissement.

- **Souplesse**. L'application à base de μServices est capable de s'adapter en fonction de son besoin métier.

1.2.4 DevOps et les μServices

DevOps est un ensemble des pratiques conçues pour réduire le temps entre les modifications et le changement du système et le déploiement des mises à jour et des modifications de production, tout en maintenant la qualité des logiciels en termes de code et de livraison. La livraison continue permet de livrer de manière incrémentale et continue des versions de logiciels fréquentes en automatisant systématiquement la création, les tests et les validations logiciels sans avoir à reconstruire et redéployer l'ensemble de l'application.

DevOps a remplacé l'ancien modèle de développement par une équipe de codeurs, une équipe de testeurs, une équipe de déploiement et une équipe d'exploitation. Selon l'enquête 2015 sur DevOps de Puppet Labs (36), les équipes utilisant les principes DevOps ont multiplié par 30 la fréquence des changements de déploiement et le temps de production a été réduit de 200 fois. Elles ne rencontrent pas trop de problèmes de qualité, subissent 60 fois moins d'échecs que les autres et résolvent les problèmes 168 fois plus rapidement que les autres. L'enquête montre également que ces résultats restent stables quels que soient la taille et le profil de l'entreprise.

Dans (88), le processus de mise en production et l'exécution des tests fonctionnels sont automatisés (Figure 1.4). Par rapport à l'architecture logicielle traditionnelle, DevOps associée aux μServices conduisent à une fréquence et une vitesse de livraison plus élevées (50).

En mettant le focus sur l'expérience utilisateur, les concepteurs sont à même de mieux identifier les μServices candidats, au lieu de se focaliser uniquement sur les fonctionnalités. Une autre caractéristique de DevOps est une adéquation entre l'équipe opérationnel et les développeurs en termes de multitudes technologies utilisées.

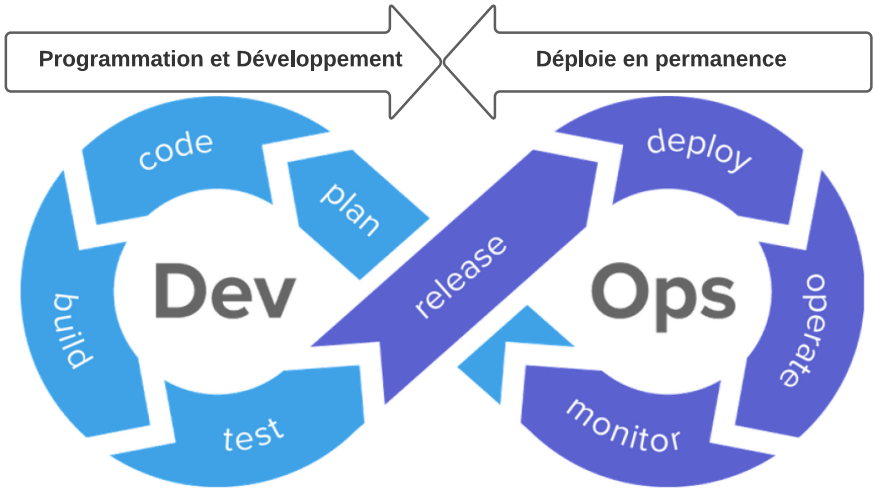


FIGURE 1.3: Cycle de Développement et Déploiement par DevOps.

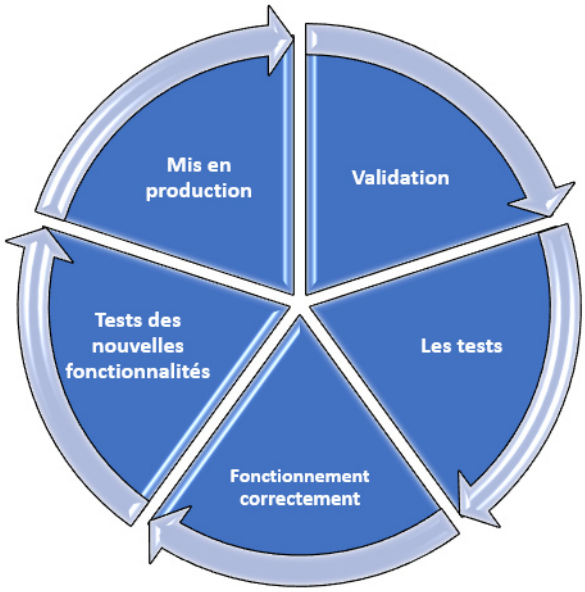


FIGURE 1.4: Le processus de livraison continue

1.3 Contributions

Cette thèse s'intéresse à la problématique de l'identification automatique des μServices dans un contexte de migration d'une architecture monolithique vers une architecture à base de μServices. Nous prenons comme point de départ un ensemble de processus métiers. Il s'agit alors de proposer une architecture qui, à partir de plusieurs processus métiers, est capable d'identifier les potentiels μServices.

Différentes contributions sont présentées dans cette thèse :

- **C1. Une architecture multi-modèles et basée sur des techniques de clustering**. Nous proposons une architecture basée sur la définition de plusieurs modèles de dépendances entre activités des processus métier et utilisant des techniques de clustering. Les modèles ont pour objectif de capturer les différents types de dépendances pouvant exister entre couples d'activités des processus métiers. Le résultat de chaque modèle est représenté séparément sous forme d'une matrice de dépendances entre couples d'activités. Les différentes matrices de dépendances sont ensuite exploitées par un algorithme de clustering collaboratif (développé par une autre doctorante) et permettant d'identifier automatiquement les différentes classes (clusters) d'activités. Ces différentes classes sont cohérentes (la distance entre les éléments d'une classe est petite) et cohésives (la distance enter les classes est petite). Chaque classe d'activités constitue une candidate pour représenter un potentiel μservice.

  Notre architecture intègre actuellement trois modèles de dépendances entre activités : modèle structurel (appelé aussi de contrôle), modèle de données, et un modèle sémantique. Cette architecture est flexible dans le sens où elle peut intégrer sans aucun coût de nouveaux modèles de dépendances à la seule condition que les dépendances entre activités restent représentées sous forme matricielle. L'architecture est flexible aussi sur les différentes façons de combiner les matrices de dépendances dans le cas de plusieurs processus métier. Dans cette thèse, nous proposons les combinaisons suivantes : par

modèle et par processus métier.

- **C2. Un modèle de dépendances de contrôle**. Il permet de mesurer à quel point deux activités sont dépendantes entre elles du point de vue exécution. Le calcul de dépendance structurelle entre deux activités est réalisé en fonction du lien direct ou indirect existant entre les activités et des différents opérateurs algébriques (séquence, OR, XOR, AND, etc.) composant le chemin reliant ses 2 activités. En règle générale, plus le chemin entre deux activités est long plus la dépendance entre ces activités est faible. Ce modèle de dépendances est formalisé et implémenté pour une extraction automatique des dépendances de contrôle à partir d'une spécification de processus métiers.
- **C3. Un modèle de dépendances de données**. Il permet de mesurer la dépendance entre les activités par rapport à la dimension partage de données. Le flux de données entre activités est pris en compte en mettant l'accent sur les opérations sur artefacts comme la lecture, l'écriture, et la création de données, ainsi que la criticité de ces artefacts. L'idée générale est que plus les activités partagent les données plus elles sont dépendantes entre elles.
- **C4. Un modèle de dépendances sémantiques**. Il permet de capturer les dépendances entre les activités selon une perspective sémantique. L'idée générale est que plus la similarité sémantique entre les noms d'activité est grande, plus les activités sont dépendantes, car elles traiteraient le même domaine. Différentes stratégies sont proposées pour mesurer cette similarité. La première stratégie est tout simplement basée sur la comparaison des noms d'activité. La deuxième stratégie a recours à une ontologie de domaine. Chaque nom d'activité est rattaché à un concept de l'ontologie. La similarité entre deux activités est calculée à partir de la similarité des concepts. La troisième stratégie a recours aussi à la même ontologie de domaine mais un nom d'activité est rattaché cette fois-ci à un fragment de l'ontologie (ensemble de concepts). La dépendance sémantique entre deux activités est calculée à partir de la similarité de ses fragments.

- **C5. Évaluation expérimentale**. Notre solution d'identification automatique de micro-services est évaluée sur deux études de cas. La première étude de cas concerne la location de vélo adaptée à partir d'un système réel. La deuxième étude de cas concerne l'application Cargo utilisée dans d'autres travaux. Elle nous a permis de comparer nos résultats à ceux obtenus dans certaines approches de la littérature scientifique. Pour chacune des deux cas d'étude, nous avons construit une ontologie pour les besoins du modèle de dépendances sémantiques.

1.4 Organisation du manuscrit

La suite de ce manuscrit est décrite comme suit. Le chapitre 2 est consacré à l'état de l'art en présentant d'abord les principaux concepts relatifs aux architectures monolithiques et µservices, et ensuite discutant l'essentiel des travaux relatifs à l'identification des µServices. Le chapitre 3 décrit les fondements de notre approche d'identification des µServices à partir d'un ensemble de processus métier et introduit les 2 modèles de dépendances structurelles et de données. Le chapitre 4 est dédié au modèle des dépendances sémantiques et discute les trois stratégies proposées de calcul des dépendances sémantiques : dirigée par les termes, dirigée par les concepts et dirigée par les fragments d'une ontologie. Le chapitre 5 évalue notre approche à travers des expérimentations. Les résultats obtenus sont présentés et discutés. Le chapitre 6 conclut ce manuscrit en rappelant les principales contributions et esquisse quelques perspectives de recherche.

2

État de l'art

Table des matières

2.1 Comprendre les μServices 17
2.1.1 Principes des μServices 17
2.1.2 Déploiement des μServices 19
2.1.3 μServices *versus* SOA 23
2.2 Adopter les μServices 26
2.2.1 Réingénierie des applications 27
2.2.2 Migration vers une architecture μServices 28
2.3 Approches existantes de migration 30
2.4 Synthèse 37

2.1 Comprendre les μServices

Comme le montre la figure 2.1, les applications monolithiques, généralement complexes, traitent d'un grand nombre de tâches connexes et diverses. Elles sont constituées de composants fortement couplés, difficiles à remplacer ou à maintenir, et nécessitant un gros investissement en termes de mise à niveau, de développement, et de correction. De plus, la majeure partie de la logique métier est déployée et exécutée dans un seul environnement d'exécution ou un seul serveur unique centralisé. L'approche monolithique nécessite ainsi plus d'efforts et de coordination entre les équipes de développement, mais aussi plus d'investissement pour effectuer des changements ou des déploiements.

Comparées aux applications monolithiques, les architectures orientée services (SOA) permettent plus de souplesse dans la gestion des systèmes d'information mais sont aussi un moyen de résoudre les problèmes d'interopérabilité. Face au problème d'évolutivité des SOA, les architectures μServices sont apparues, permettant une grande agilité dans les développements grâce à des principes de conception comme des composants faiblement couplés et fortement cohésifs avec une responsabilité unique. Cependant, l'architecture μServices apporte également son lot de défis comme la complexité des tests, la gestion complexe de dépendances entre μServices, et la nécessité d'une large palette de compétences techniques de la part des développeurs (39; 81).

2.1.1 Principes des μServices

L'architecture μServices est un sujet d'actualité dans le monde du logiciel et est devenue l'une des architectures logicielles les plus populaires, notamment avec l'arrivée du Cloud Computing. La figure 2.2 montre un aperçu d'une architecture des μServices. Les applications sont composées de μServices, se résumant à des composants déployables de manière indépendante, de portée limitée et prenant en charge l'interopérabilité via une communication basée sur les messages. Se devant d'être fortement cohésifs et faiblement couplés, chacun des

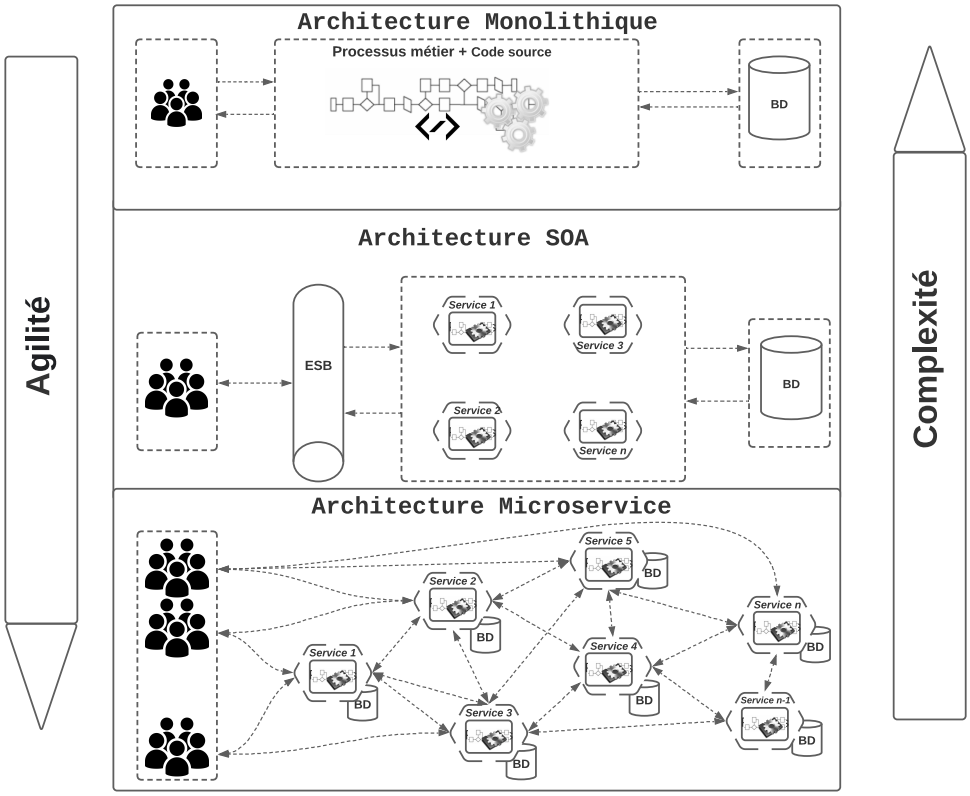


FIGURE 2.1: Monolithique *vs.* SOA *vs.* µServices

µServices possède sa propre base de données, mais peut éventuellement se voir partager une base de données avec un petit nombre d'autres µServices.

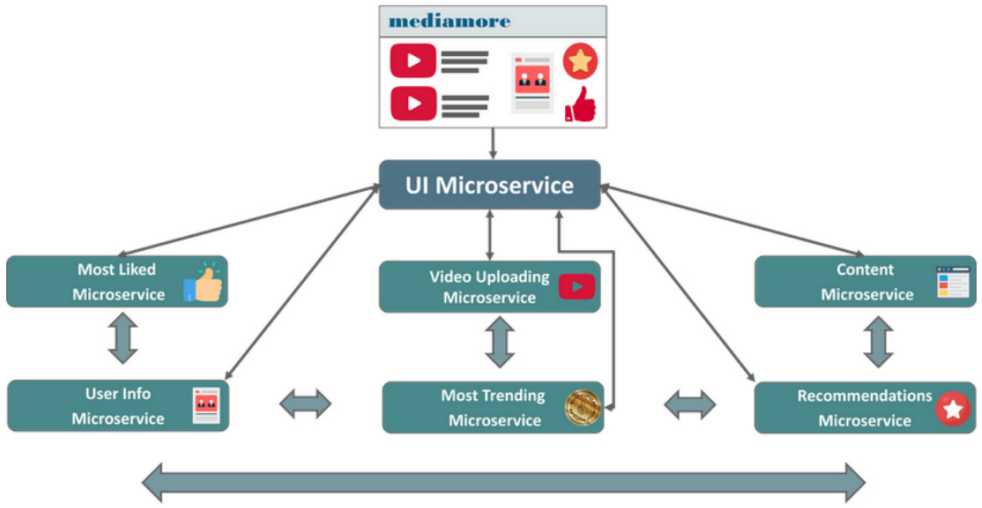


FIGURE 2.2: Architecture MSA

2.1.2 Déploiement des µServices

Il existe plusieurs techniques de déploiement des µServices, notamment les conteneurs (27). Le concept de conteneur, n'étant pas propre aux µServices mais s'y conformant bien, fournit une méthode d'encapsulation de fonctions dans un espace de processus à part entière (69). Le conteneur peut démarrer en quelques secondes et être stocké dans une bibliothèque pour une future réutilisation. En théorie, les applications s'exécutant dans des conteneurs sont portables sur n'importe quel système d'exploitation (78). Les conteneurs sont considérés comme l'avenir du développement d'une technologie de virtualisation pour résoudre de nombreux problèmes. Ils sont similaires aux machines virtuelles en termes de services fournis. Cependant, l'utilisation de conteneurs n'entraîne pas la surcharge supplémentaire liée à l'exécution d'un noyau séparé et à la virtualisation des composants matériels (22). La figure 2.3 montre une comparaison hiérarchique des conteneurs Docker et des machines virtuelles.

Dans (65), les auteurs évaluent les performances de Docker par rapport aux machines vir-

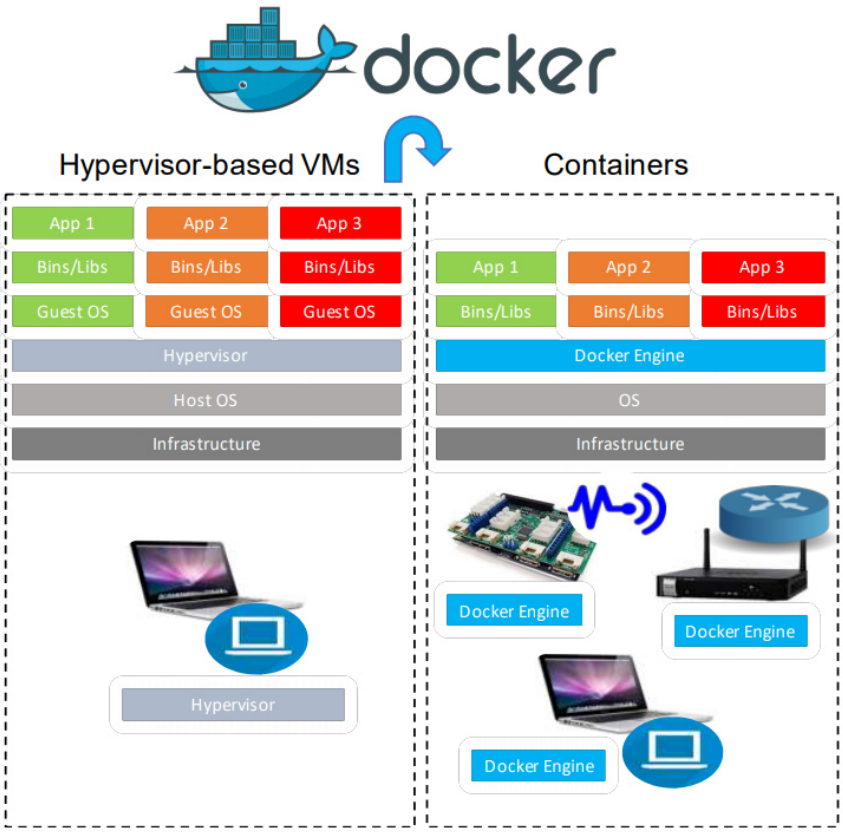


FIGURE 2.3: Conteneurs Docker *versus* machines virtuelles (72)

tuelles. Non seulement, Docker s'avère plus rapide au démarrage, mais il présente d'autres avantages comme sa légèreté lors d'un export. Il facilite aussi la décomposition d'un système complexe en une série d'éléments composables, permettant de restructurer l'application logicielle sans affecter l'ensemble.

Dans (67), les auteurs considère la contenairisation avec Docker et l'architecture de µServices comme deux voies complémentaires vers l'objectif ultime de livraison continue et d'efficacité opérationnelle. De nos jours, Docker est l'un des outils déploiement le plus largement déployé en production. Leur émergence est due principalement à des besoins réels des équipes techniques tels que des outils performants pour déployer des applications complexes de manière polyvalente et prévisible.

Dans (27), l'architecture Docker proposée se base sur la technologie des conteneurs Linux. Cette technologie vise à isoler les processus de leurs ressources via les mécanismes du système d'exploitation et à partager des applications multi-conteneurs.

Dans (72), les conteneurs orientés applications comme Docker sont plus adaptés aux frameworks de µServices. En effet, ils fournissent une Plateforme légère et stable pour rapidement créer et exécuter des tâches.

Le tableau 2.1 récapitule l'ensemble des outils et frameworks les plus populaires pour des développeurs de µServicess.

TABLE 2.1: Plateformes et Frameworks de développement des µServicess

Framework	Description
Spinnaker (Enterprise)	Plateforme open source avec divers outils pour la livraison continue.
Netflix OSS (Center)	Ensemble des outils open source permettant de construire et de déployer les services.
Spring (VMware)	Frameworks JAVA de développement de µServices.
Dropwizard (Dropwizard)	Framework largement utilisé pour développer des µServices en Java.
Vertx (Fox)	Boîte à outils pour la création des systèmes distribués réactifs sur la machine virtuelle Java.
Lagom (Odersky)	Framework open source pour la construction de systèmes de µServices réactifs en Java ou Scala.
OpenFaaS (Ellis)	Framework permettant aux développeurs de déployer facilement des fonctions et des µServices événementiels sur Kubernetes.
Azure Functions (Azure)	Outils de Microsoft de création d'une architecture de µServices.
Kubeless (framework)	Boîte à outils basée sur les ressources Kubernetes pour fournir une surveillance et un dépannage des services.
Goa (Design)	Framework pour la création de micro-services et d'API.
Kubernetes (Google)	Outils et ressources open source, permettant de gérer des applications conteneurisées sur plusieurs hôtes.
Drone (Drone.io)	Système de livraison continue basé sur la technologie des conteneurs.
Fuge (microservice shell)	Environnement d'exécution pour les développers des µServices.
Seneca (Framework)	Boîte à outils pour écrire des µServices et organiser la logique métier d'une application.
Bitbucket (Atlassian)	Environnement servant à planifier des projets, collaborer sur du code, le tester et le déployer.
Aws (Amazon)	Plateforme de création et de construction des µServices.

2.1.3 µServices *versus* SOA

Bien que les µServices et SOA soient deux styles architecturaux très différents, ils présentent néanmoins de nombreuses caractéristiques communes telles que les services comme composant principal de l'architecture (40). Dans la littérature, de nombreux travaux examinent les différences et similitudes entre SOA et l'architecture µServices. Alors que certains (e.g., (40)) considèrent l'architecture de µServices comme une représentation moderne de l'approche SOA, d'autres (e.g., (77; Martin Fowler)) estiment les µServices comme étant un sous-ensemble des SOA.

L'une des plus grandes différences entre µServices et SOA est la granularité des services. Plus précisément, les composants de service dans les µServices sont généralement des services à objectif unique pouvant accomplir une seule fonctionnalité. Avec SOA, la taille des composants peut varier, allant de petits services applicatifs à des grands services d'entreprise. Alors que le passage à un service à forte granularité peut éventuellement résoudre les problèmes de performance et de transaction, cela aura également un impact négatif sur le développement, les tests, le déploiement et la maintenance des applications orientées-services. La figure 2.4 montre les avantages et inconvénients de l'architecture SOA (25).

Une autre différence se situe dans la classification des services. Dans SOA, il existe une taxonomie de services très formalisée en termes de types de service et de leur rôle dans l'architecture globale, notamment celui de fournisseur et/ou consommateur de services, considéré comme des points de connexion d'un ESB (Enterprise Service Bus) (Figure 2.5). Ce bus permet ainsi à des services applicatifs composites d'accéder aux services disponibles de façon standardisée, flexible et transparente (Figure 2.1).

Avec l'essor de la virtualisation, du cloud computing, les pratiques de développement agiles et de la livraison continue de DevOps, l'architecture de µServices a suscité un intérêt grandissant. Quoique les architectures µServices et SOA préconisent la coopération de services grâce à leur intégration sur de nombreuses plates-formes indépendantes, elles adoptent

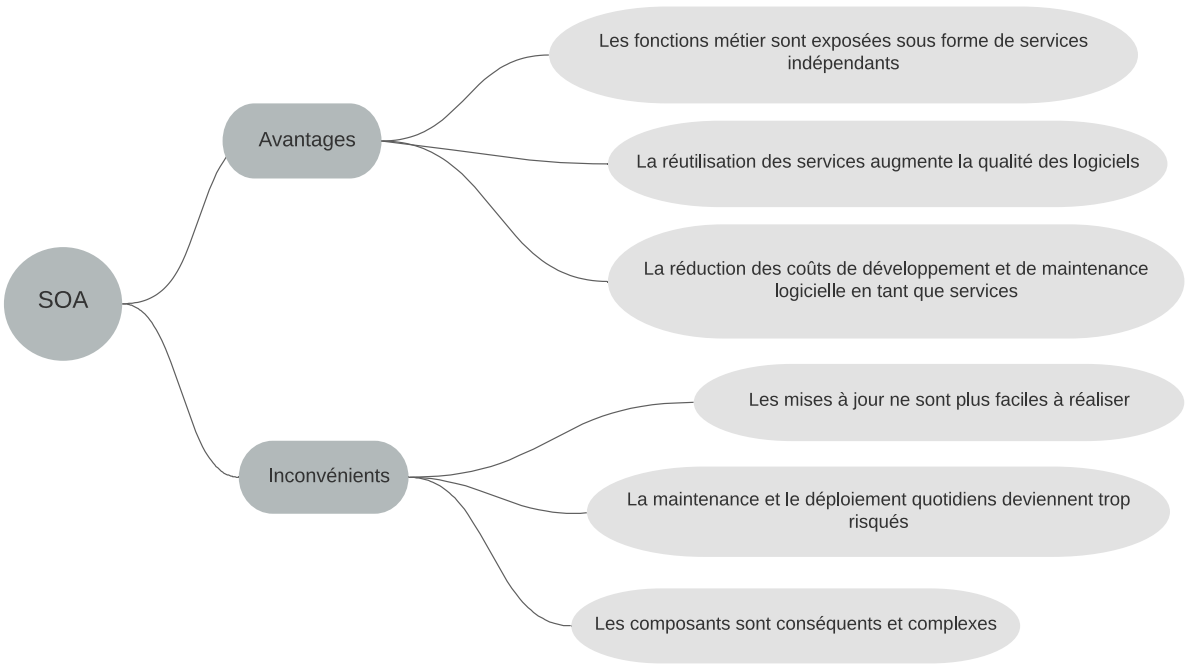


FIGURE 2.4: Avantages et inconvénients des SOA

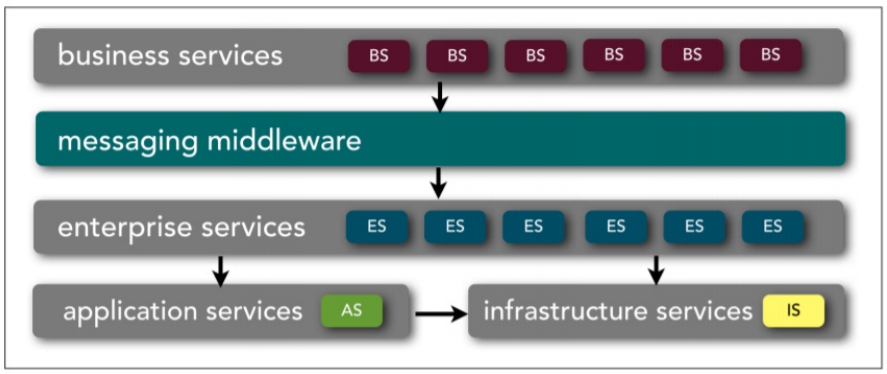


FIGURE 2.5: Taxonomie de services dans SOA (73)

de manière différente la réalisation de cette coopération.

Le Tableau 2.1.3 montre une comparaison entre SOA et µServices prenant compte des aspects techniques comme les mécanismes de communication ou encore de pratiques comme la réutilisabilité des services versus le découplage des µServices.

Malgré le fait que l'exécution d'applications basées sur des µServices nécessite une plus grande consommation de ressources comparés aux SOA (8), les µServices sont essentiels à l'approche native du Cloud. Les services sont mis en production indépendamment les uns des autres, étant une différence majeure avec la plupart des solutions SOA. Ces différences affectent non seulement le déploiement des services, mais aussi le travail d'évolution et de modification (13).

En résumé, les µServices présentent les avantages suivants :

1. `Meilleure évolutivité`. Les services peuvent être étendus indépendamment de l'utilisation d'autres services.
2. `Meilleure agilité`. Les services peuvent être modifiés sans affecter les autres services, avec une plus grande flexibilité.
3. `Meilleure disponibilité`. L'échec partiel d'un service peut ne pas affecter directement l'ensemble de l'application.
4. `Livraison continue`. En mode maintenance, les changements de service n'affectent pas l'ensemble de l'application,
5. `Augmentation de la productivité`. La structure de l'architecture logicielle de l'application s'apparente à celle des équipes de développement où chaque service est maintenu par une équipe (57).

TABLE 2.2: Différences entre µServices et SOA

Critère	µServices	SOA
Communication	Une couche de gestion API pour effectuer des communications simples, pris en charge par les µServices	S'appuie sur son middleware ESB servant pour la communication entre les composants connectés

Protocole	Utilisent des protocoles de messagerie légers tels que les protocoles comme HTTP/REST, JSON	Utilise plusieurs protocoles de messages tels que SOAP, JMS, XML, et HTTP
DevOps	Livraison continue et méthodes agiles	Livraison continue non prise en compte de manière naturelle
Taille des services	Petits services à granularité fine	Des petits services aux gros services
Indépendance	Tourne autour d'une fonctionnalité de services indépendants	Aucune exigence d'indépendance
Gestion	Gestion décentralisée et indépendante offrant une seule capacité métier, faiblement connectée et autonome	Gestion centralisée et dépendante des autres services
Déploiement	Pas de serveur d'applications dédié, déploiement individuel, simple et rapide	Utilisation d'une Plateforme commune, déploiement monolithique de plusieurs services
UI	Interface pour chaque µService	Une interface unique pour tous les services
Stockage	Bases de données indépendantes	Base de données partagée
Infrastructure	Petite taille	Grande taille
Rôle	Fonctionnement indépendant	Partage des ressources entre les services
Flexibilité	Flexibilité grâce à des développements et déploiements rapides et indépendants	Flexibilité par orchestration de services
Focus	Concept de contexte limité	Réutilisation des fonctions métiers
Architecture	Pour un projet	Á l'échelle de plusieurs projets
Conteneurs	Utilisation inhérente de conteneurs pour les µServices	Utilisation de conteneurs moins populaire
Méthode	Conception dirigée par domaine et pratiques agiles	Analyse et conception orientées objet

2.2 Adopter les µServices

L'adoption des µServices demande soit de développer à nouveau les applications à partir de zéro, tâche se révélant souvent laborieuse et source d'erreur, soit d'envisager du réingénierie d'applications (i.e., lifting - pas de changement de paradigme) ou de la migration d'un paradigme vers un nouveau (i.e., shifting - changement de paradigme), faisant l'objet de notre travail.

2.2.1 Réingénierie des applications

Il existe plusieurs techniques de réingénierie d'applications monolithiques vers les µServices, notamment l'analyse de code utilisant des fichiers log (52) ou encore des descriptions UML (**?**). Ces techniques constituent une transformation "superficielle" de l'application. En effet, les changements opérés devraient améliorer la qualité du code sans changer son comportement. Par la suite, nous mettons l'accent sur le refactoring de code permettant de retravailler et de restructurer du code applicatif. Cette technique est utilisée pour faire de l'optimisation de code, de l'injection de design patterns ou encore du raffinement des modèles. Par exemple, il peut s'agir de nettoyage et réorganisation de code (Figure 2.6). Cette technique permet de réduire la taille du code, et par effet de bord, d'éliminer de forts couplages entre les méthodes. Ainsi, le refactoring de code offre la possibilité de tester la clarté et la compréhension du code afin de faciliter les améliorations futures. Un avantage clé de cette technique est une amélioration de la qualité du code applicatif à moindre coût. En effet, elle s'appuie sur du code existant permettant ainsi une flexibilité dans l'ajout de nouvelles fonctionnalités. De nombreux langages de programmation ont des IDEs automatisant différentes méthodes de refactoring (Fowler).

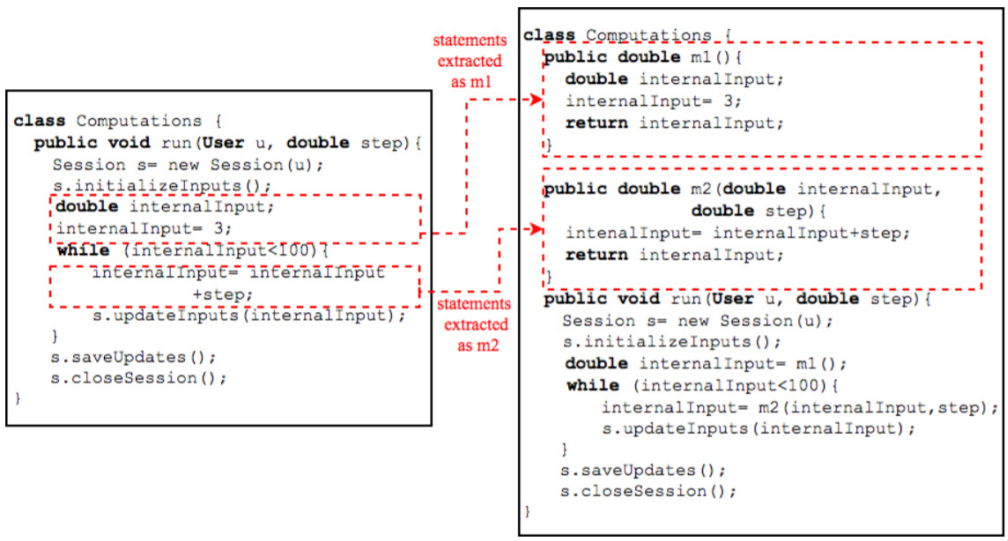


FIGURE 2.6: Exemple de refactoring de code (80).

A nos jours, le plus grand obstacle pour faire évoluer les applications monolithiques vers

des applications autonomes et indépendantes au niveau du code source, des fonctionnalités et de la base données, reste la décomposition en petits programmes de manière propre et faiblement liés (e.g., (38; 68)).

Plusieurs études avec différentes méthodes ont été mises en place pour confirmer la possibilité de refactoring d'un monolithe vers les architectures µServices (e.g., (44; 45; 15)). Par exemple, Fritzsch et al. proposent de retravailler le code source du monolithe par des concepteurs et des architectes logiciels au lieu de développer à nouveau le monolithe (44).

2.2.2 Migration vers une architecture µServices

La migration d'une architecture monolithique vers une architecture de µServices n'est pas forcément une tâche simple, ceci est causé par de nombreux facteurs qui compliquent le basculement. Dans (17), Richardson présente un ensemble de patterns guidant le développement d'applications distribuées orientées µServices. Cet ensemble couvre divers aspects conceptuels et techniques comme par exemple, la décomposition par capacité métier et/ou par sous-domaine (Domain-Driven Design), les modèles de déploiement et les tests (Figure 2.7).

Dans la littérature, les motivations derrière une migration vers une architecture µServices sont multiples comme une facilité de maintenance, une évolutivité, une délégation de responsabilité vers des équipes indépendantes. Balalaie et al. identifient plusieurs défis relatifs à la migration vers les µServices comme la méconnaissance des concepts d'amélioration en continue de la part des développeurs et la complexité de déploiement dans des environnements de développement (7). D'autres défis, un peu plus techniques, sont en lien avec l'exécution d'applications basées sur les µServices comme le problème d'allocation et de consommation de ressources (8).

Dans (45), Furda et al. identifient trois principaux défis dans une migration d'applications SOA vers les µServices :

- `Multi-tenant`. La capacité du système à répondre aux besoins de différents clients. Par

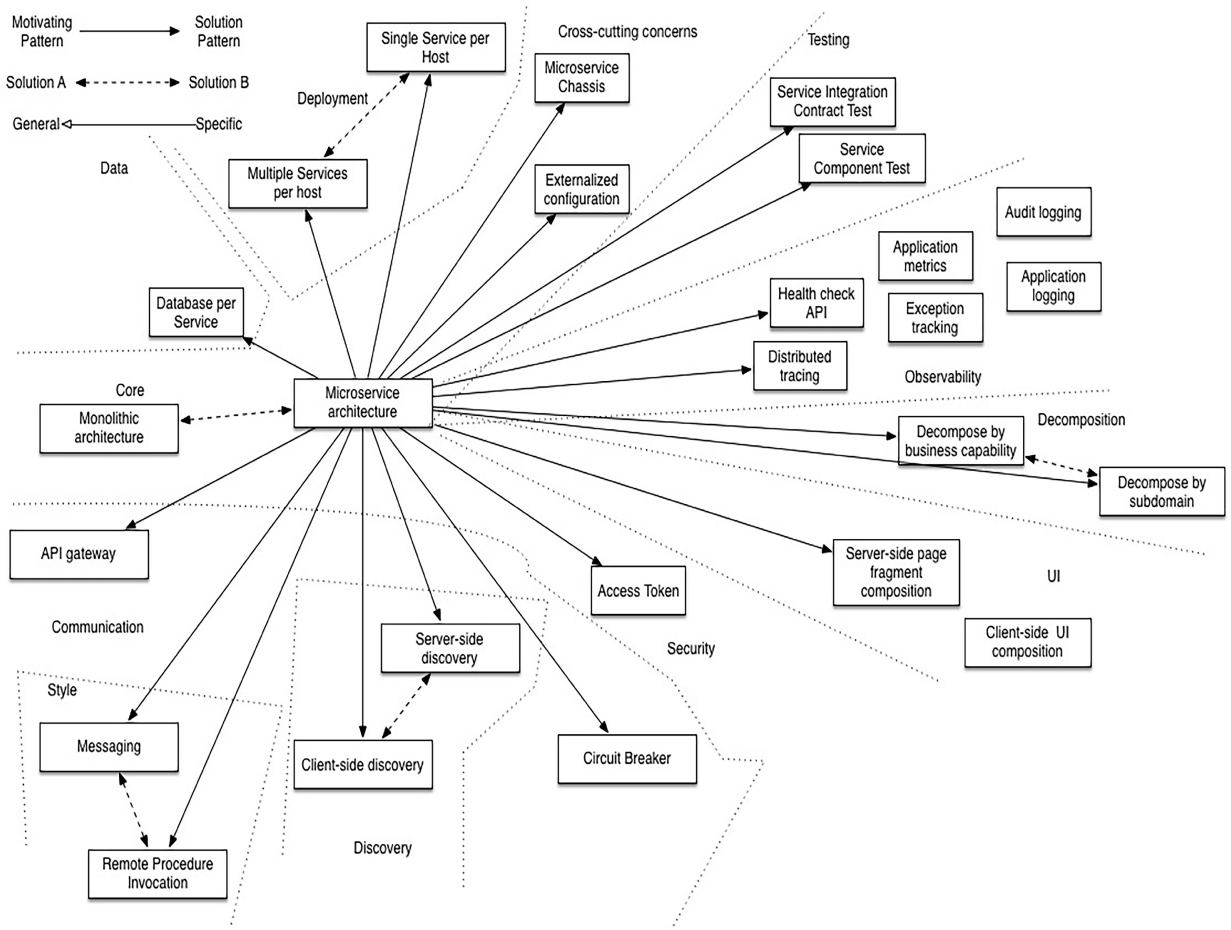


FIGURE 2.7: Patterns µServices (17)

exemple, il faut assurer une répartition uniforme des différentes instances de μServices.

- `Etat du code`. La qualité du code affecte considérablement les exigences non fonctionnelles d'évolutivité, de fiabilité, et de disponibilité de la future application basée sur les μServices.
- `Cohérence des données`. Ce problème peut entraîner une violation des exigences fonctionnelles de la future application basée sur les μServices.

D'autres études (e.g., (90; 18)) se sont intéressées quant à elles, aux méthodes de vérification de la migration du monolithique vers les μServices en terme de taux de couverture fonctionnelle, par exemple.

2.3 Approches existantes de migration

Dans (54), Knoche and Hasselbring proposent une méthode incrémentale de migration vers les μServices, basé sur les expériences issues d'une étude de cas industrielle. Cette méthode consiste à d'abord implémenter les nouvelles fonctionnalités sous forme de μServices, puis de faire migrer de manière incrémentale les services existants en commençant par ceux composant les applications clientes. Cette méthode de migration permet d'établir une interface bien définie et indépendante de la plateforme existante. Elle permet également d'améliorer l'évolutivité de l'application et de déplacer les fonctionnalités non-clientes (i.e., internes) dans des composants séparés, et d'éliminer les parties redondantes et obsolètes de l'application. Cependant, certaines parties de l'application ne peuvent pas être modernisées en utilisant cette méthode, comme celles basées sur une technologie propriétaire.

Le travail de Amiri (3) préconise d'extraire des informations pertinentes de processus métier pour identifier les μServices. De manière plus précise, l'approche proposée se base sur la notion de dépendances entre activités et consiste en plusieurs étapes. La première modélise un système comme un ensemble de processus métier, tenant en compte des dépendances struc-

turelles et de données entre activités. Ainsi, chaque activité du processus métier joue un rôle important dans l'identification des μServices. Cette approche présente une base intéressante pour de futures recherches. En effet, il reste d'autres aspects non traités, tels que les exigences de domaine ou encore les propriétés des activités.

Dans (82), Taibi et al. mènent une étude permettant d'identifier les différentes étapes de migration adoptées par des concepteurs/développeurs avérés, notamment une analyse des dépendances, la définition d'une architecture logicielle et la prise en compte des outils et protocoles de communication entre les futurs μServices. Les auteurs soulignent eux-ci de nombreux défis à relever d'un point de vue technique et organisationnel. Par contre, l'un des arguments à l'encontre d'une adoption des μServices reste néanmoins la couche de données identifiée comme particulièrement compliquée.

Dans (51), Jin et al. proposent une méthode d'identification des services candidats basée sur les fonctionnalités. La méthode comprend trois étapes : l'extraction de traces d'exécution, l'identification des services candidats à l'aide d'un algorithme de clustering fonctionnel, et l'identification des interfaces pour les services candidats. Les auteurs comparent leur travail avec des méthodes classiques de décomposition prenant compte des relations/dépendance structurelles et statiques extraites du code source, comme WCA (Weighted Clustering Algorithm) (14), MEM (μService Extraction Model) (63), et LIMBO (scaLable InforMation BOttleneck) (4). LIMBO est un algorithme de décomposition logicielle traditionnelle, basé sur le clustering hiérarchique évolutif pour la minimisation de la perte d'informations lors de la mise en cluster d'un produit logiciel. LIMBO applique l'algorithme agglomeratif pour obtenir un clustering des artifacts, puis le transforme en un ensemble de décompositions possibles, et finalement sélectionne celles les plus pertinentes. Quoique intéressant, ce travail souffre de la forte dépendance à la qualité des traces d'exécution générées.

Dans (48), Gysel et al. définissent une méthode de décomposition des services, appelée Service Cutter. Cette méthode prend un ensemble de spécifications et de critères de couplage

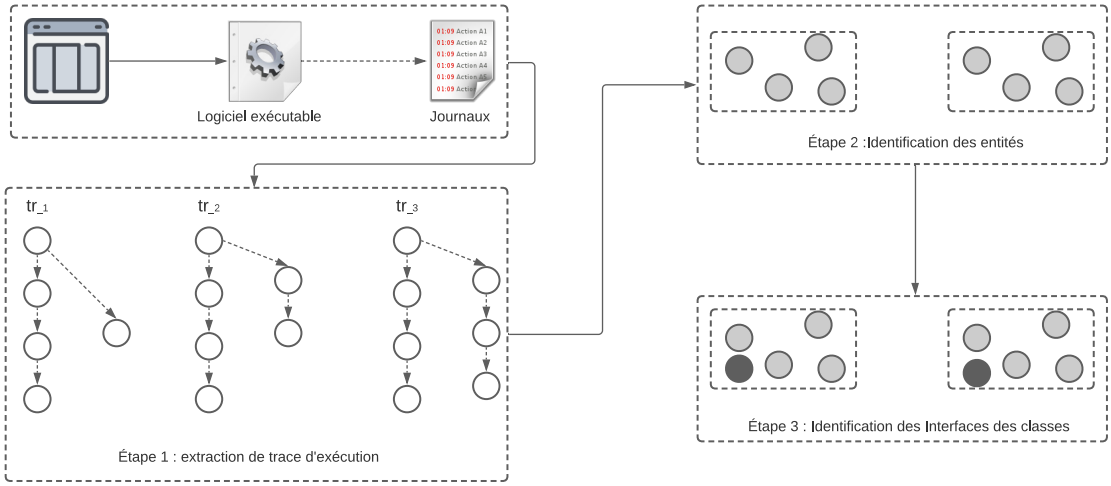


FIGURE 2.8: Identification des services candidats basées sur les fonctionnalités (51)

comme paramètres d'entrée et utilise un algorithme de clustering en tenant compte des critères de couplage regroupés en quatre catégories suivantes (Fig. 2.10) :

- `Cohésion`. Cette catégorie décrit les attributs communs de composants (e.g., proximité sémantique), pouvant expliquer les raisons pour lesquelles ces composants devraient appartenir au même service.
- `Compatibilité`. Cette catégorie comprends les différentes caractéristiques de composants incompatibles (e.g., sécurité critique) pour lesquelles un regroupement en μServices s'avérait inapproprié voire impossible.
- `Contraintes`. Cette catégorie définit des exigences à fort impact lors de regroupement de composants.
- `Communication`. Cette catégorie permet de mesurer les coûts techniques associés au modèle de communication entre services.

Dans (9), Baresi et al. proposent une méthode de décomposition basée sur une analyse sémantique de termes contenus dans des spécifications OpenAPI. Cette similarité est calculé par DISCO (56), en fonction des co-occurrences de termes dans de grands corpus de texte. L'idée est de regrouper les concepts partageant les mêmes propriétés dans le même service candidat.

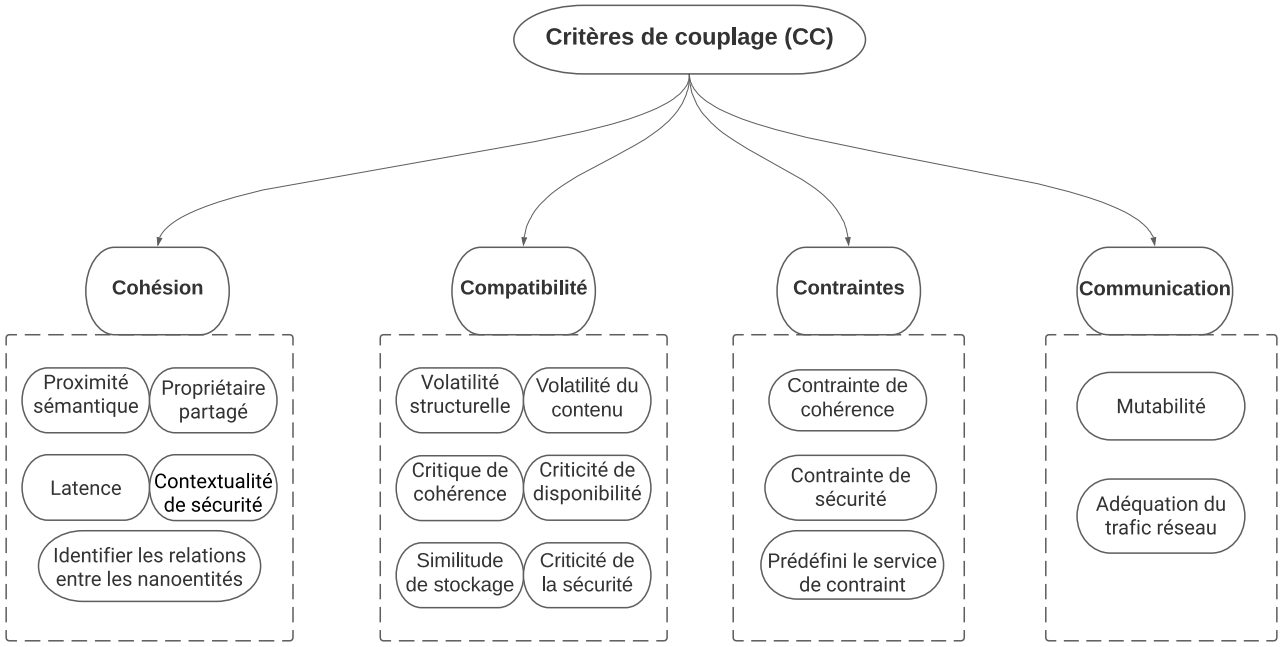


FIGURE 2.9: Critères de couplage (48)

La méthode proposée fait une correspondance entre les interfaces décrites en termes d'opérations et de ressources en des spécifications OpenAPI. Les résultats expérimentaux montrent que leur méthode est capable de trouver des décompositions appropriées dans environ 80% des cas, et fournir des premières indications sur la granularité des μServices et leur cohésion.

Dans (74), les auteurs proposent quatre stratégies pour résoudre les problèmes de partitionnement de l'application en μServices :

- `Décomposition par capacité métier (Business Capability)`. Cette stratégie définit les services correspondant à des fonctions métier, en se basant sur une analyse des objectifs et de la structure des processus métiers. Les services construits autour de la création de valeur métier plutôt que des fonctions techniques se sont avérés cohérents et faiblement couplés.
- `Décomposition par conception pilotée par domaine (Domain-Driven Design)`. Cette stratégie partitionne l'application monolithique selon différents sous-domaines, chacun représentant un service de l'organisation.

- `Décomposition par cas d'utilisation`. Cette stratégie définit le service responsable d'une fonctionnalité spécifique décrivant un use case.
- `Décomposition par ressource`. Cette stratégie permet de définir le service responsable d'un ensemble d'opérations partageant des ressources communes.

Il est à noter que les deux premières stratégies offrent une plus grande stabilité à la nouvelle architecture de µServices car les fonctions métier et les limites de domaine ne subissent généralement pas de changements majeurs. Par contre, comme les cas d'utilisation et les ressources nécessaires risquent d'évoluer dans le temps, les deux dernières stratégies peuvent ne pas être stables.

Dans (59), Levcovitz et al. proposent une méthodologie d'urbanisation d'un système bancaire en termes de µServices. La méthodologie consiste en cinq étapes. La première fait la correspondance d'un ensemble de tables de la base de données (tb_i) en fonctions métier (bf_j). L'étape suivante crée des graphes de dépendance entre différents niveaux de l'application (client, serveur, base de données), où les sommets représentent des interfaces (fc_k) et des fonctions métier (bf_j). La troisième étape identifie les paires $< bf_j, tb_i >$. L'avant-dernière étape consiste à extraire les règles métier dépendant des tables de la base données et ce afin d'identifier les candidats à transformer vers des µServices. Finalement, la dernière étape permet de créer des passerelles API, couche intermédiaire entre côté-client et côté-serveur, rendant ainsi la migration vers les µServices transparente pour les clients. Cependant, cette approche n'est pas automatique car elle nécessite de diviser manuellement les tables de base de données en groupes, restant une tâche fastidieuse.

Dans (16), Chen et al. introduisent une méthode de décomposition des applications monolithiques en µServices, guidée par les flux de données en lien avec la logique métier. Cette méthode consiste d'abord à faire nettoyer et détailler des flux de données manuellement par les analystes, afin d'apporter des éclaircissements/précisions sur ces flux. L'étape suivante consiste en une décomposition dirigée par les flux de données afin d'extraire le diagramme

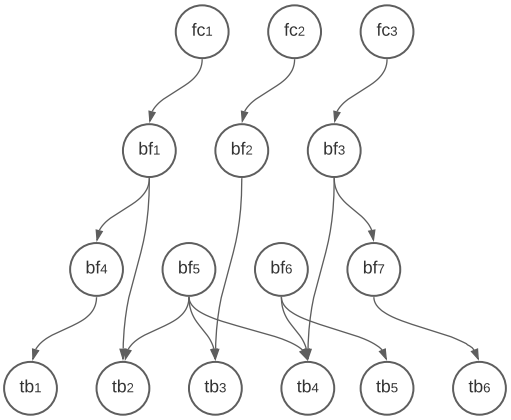


FIGURE 2.10: Graphe de dépendance (59)

de flux de données raffiné ensuite en un diagramme décomposable permettant d'améliorer la qualité des futurs µServices. Finalement, la dernière étape permet d'extraire et d'identifier de manière automatique des modules orientés µServices, en fonction des opérations du diagramme de flux de données. Néanmoins, la méthode proposée souffre de plusieurs limitations, comme l'identification des opérations de données similaires nécessitant une certaine expertise.

Dans (75), Salvadori et al. propose le framework LINKDATOR traitant de la problématique de liaison de données sémantique pour développer les µServices. Ce framework combine la description sémantique et le lien entre les données en tenant compte l'ontologie de domaine (Figure 2.11). Le framework est constitué de 2 composants principaux :

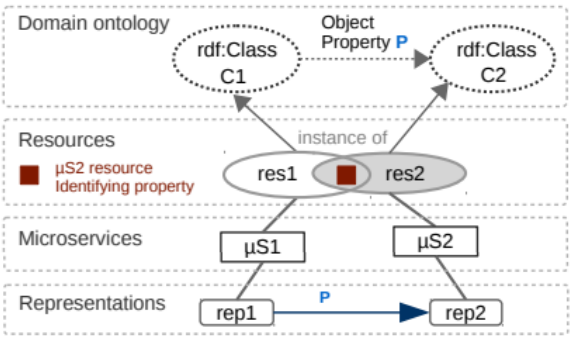


FIGURE 2.11: Méthode basée sur la liaison de données (75)

1) LINKDATOR-Core permet de créer des liens JSON-LD entre les entités fournies par les µServices ; il fait référence à des descriptions de µServices sémantiques, à un modèle d'onto-

logie contenant des informations sur les classes sémantiques et les propriétés des ressources et à un moteur de liens déterminant les propriétés des objets ; 2) LINKDATOR-API permet d'encapsuler des fonctionnalités de base dans une API Web, d'enregistrer la description sémantique des μServices et de créer un lien vers la représentation donnée.

Dans (24), Diepenbrock et al. proposent une méthode de décomposition où le concept de domaine joue une rôle prépondérant dans le DDD (Domain-driven design). Le principe consiste à découper le domaine en sous-domaines, et considérer chaque sous-domaine comme un service. Il est important de noter que le concept de contexte borné/délimité (limited context, BC) correspond naturellement aux μServices (Figure 2.12).

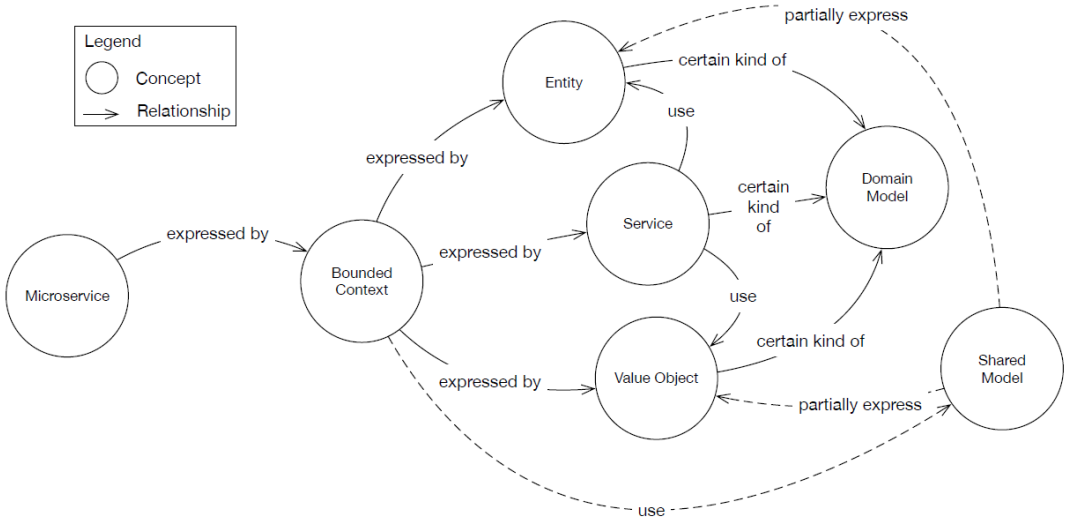


FIGURE 2.12: Concepts DDD pour une conception de μServices dirigée par domaine (24)

Afin de concevoir convenablement des μServices, les experts du domaine et les développeurs doivent nécessairement collaborer et ce de manière étroite. La portée des μServices se définit en terme de contexte borné. L'idée générale est de rester cohérent lors de la conception de μServices avec le contexte borné. En effet, tout domaine est composé de plusieurs contextes bornés où chacun encapsule les fonctions associées dans le modèle. Les μServices représentent, par conséquent, ces contextes bornés, étant une bonne indication sur un couplage faile et une forte cohésion des μServices. La figure 2.13 montre l'exemple de conception diri-

gée par domaine pour la location de vélo.

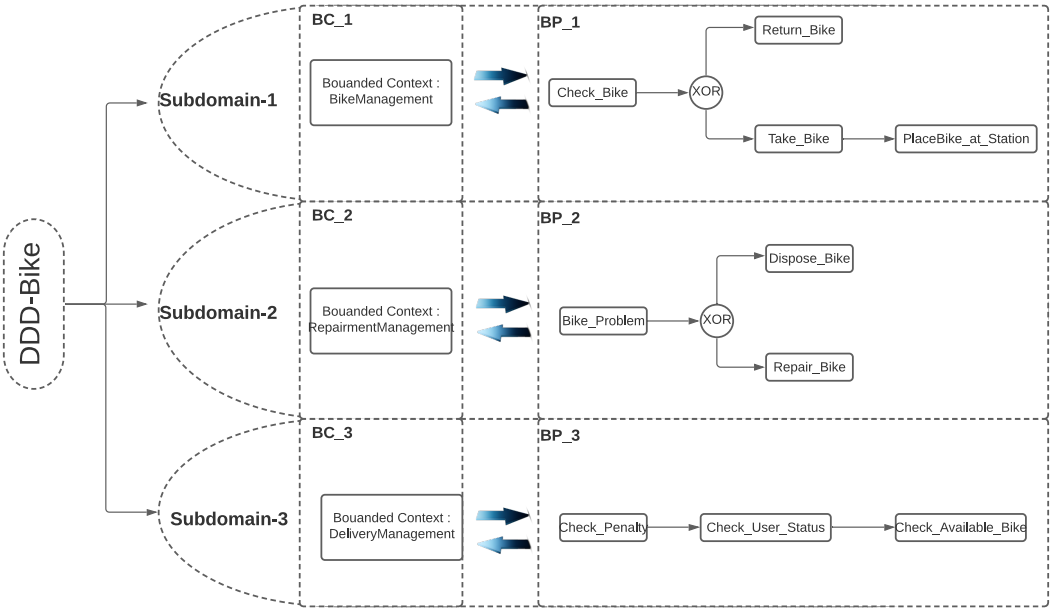


FIGURE 2.13: Illustration d'une conception dirigée par domaine

2.4 Synthèse

Le tableau 2.3 présente une analyse synthétique des approches présentées en Section 2.3. Nous utilisons quatre critères de comparaison : principaux paramètres d'entrée pour analyse, algorithme d'identification, méthode d'évaluation indiquant si les expérimentations ont été effectuées sur une application industrielle ou bien sur une étude de cas, et les métriques de performance pour le benchmarking. Nous avons prêté une attention particulière aux paramètres d'entrée, pouvant être des processus métier, des spécifications d'API, des diagrammes UML, des ontologies de domaine, etc. En effet, cela donne une forte indication sur le type d'approches (i.e., forward, backward, ou hybride). Ces paramètres d'entrée font ensuite l'objet d'une analyse afin d'extraire des informations utiles telles que des dépendances entre artefacts ou encore entre activités d'un processus métier, comme fût notre cas. Nous pouvons également noter que les techniques de clustering soient largement utilisées pour identifier les µServices.

Contrairement aux approches étudiées adoptant une technique unique de clustering, notre approche consistera à définir un clustering collaboratif de manière à mettre en lumière d'éventuelles corrélations entre différentes analyses. Enfin, la plupart des approches s'appuient sur des études de cas pour mettre en œuvre leurs solutions. Peu travaux ont implémenté la solution proposée sur des applications industrielles dans la mesure où cela nécessiterait une mobilisation importante de ressources humaines. Dans nos travaux, nous avons considéré deux études de cas, dont l'une fût utilisée pour comparer les performances de notre approche avec celles de 3 autres. Différentes métriques ont été définies dans notre travail comme l'indice de Dunn, le couplage afférent, le couplage efférent, l'instabilité et la cohésion.

TABLE 2.3: Analyse des approches via leur entrée/sortie et le technique utilisée

	Données	Résultats	Technique utilisée	Évaluation
(10)	Code source JAVA	Génération automatique et déploiement des µServices dans un conteneur Docker	La méthode a été implémentée à l'aide de JetBrains MPS (framework de méta-modélisation textuelle)	Déploiement dans un conteneur Docker.
(59)	Bases données monolithique relationnelles avec un source code en langage C, plus le modèle structurel de données.	Paire (fci, tbj) comme candidat de µServices	Basé sur la mise en correspondance des dépendances entre bases de données, fonctions métier et interfaces.	Système bancaire gèrant les transactions effectuées par les clients sur plusieurs canaux bancaires (Internet Banking, Call Center, DAB, POS, etc..).
(63)	Les relations sémantiques formelles	Générer des recommandations pour des candidats potentiels de µServices	Stratégies de couplage formelles s'appuient sur des (méta) informations issues de bases de code monolithiques et les intégrons dans un algorithme de clustering basé sur les couplage logique, sémantique.	Interface Web dans Angular et appliction JAVA.
(52)	Journal des traces d'exécution des programmes	µServices Candidats	Analyse des traces et les liens d'exécution entre le code source.	Etudes de cas via l'applications web : e-commerce, système de forum de discussion, système de blogging, sites de blog.
(4), (1	Utilise les relations structurelles du code source et les dépendances entre les artefacts logiciels	Identification des artifacts	Mesure de la distance entre une décomposition de système en sous système et l'utilisation d'algorithme de clustering	
(87)	Interface utilisateur avec un outil basé sur le système de gestion de construction Maven.	La génération sélective de composants µServices	MAGMA est un système basé sur Java permettant aux utilisateurs d'utiliser les archétypes Maven pour accélérer le développement d'architectures µServices via une interface utilisateur graphique.	Utilisé avec succès dans l'enseignement et dans des projets de recherche, une interface graphique pour configurer et invoquer graphiquement les builds.
(1)	Spécifications des exigences fonctionnelles et non fonctionnelles	Poids de dépendance entre deux fonctions	Algorithme basé sur des règles	Etude de cas : système de streaming vidéo en ligne
(3)	Processus métier	Dépendances des matrices de données et de contrôle	Clustering centralisé	Etude de cas : Approbation du plan Processus BPMN.
(16)	Diagramme de flux de données	graphe de dépendance entre processus, magasins de données et entités externes	Algorithme basé sur des règles	Etude de cas : suivi des marchandises.
(55)	Code source, modèle de domaine	————	Analyse statique	Application industrielle : application de gestion personnalisée d'une compagnie d'assurance.
(48)	Cas d'utilisation, modèles d'entité-relation, groupes d'accès de sécurité, zones de sécurité séparées.	Graphique de dépendance entre nanoentités	Clustering centralisé	Etudes de cas : suivi des cargaisons et système d'échange.

3

Extraction de μservices basée sur les flux

Table des matières

3.1 Introduction 42

3.2 Fondements 43

3.3 Dépendances issues de flux de contrôle 48

3.3.1 Analyse préliminaire des flux de contrôle dans un BP 48

3.3.2 Analyse complémentaire des flux de contrôle 51

3.4 Dépendances issues de flux de données 52

3.4.1 Criticité des données 53

3.4.2 Stratégies de calcul 53

3.5 Conclusion 55

3.1 Introduction

En règle générale, la modernisation des systèmes monolithiques en μServices reste une tâche difficile. En effet, les décisions prises sur comment décomposer l'architecture logicielle existante en μServices ne sont pas simples dans la mesure où il faudra établir de nouvelles dépendances entre services tout en préservant les fonctionnalités d'origine (33). Compte tenu des travaux présentés dans le chapitre 2, les différentes recherches sur l'identification des μServices portent soit sur l'utilisation de fichiers log (26), de codes sources (34), de diagrammes de classes (9), ou encore de bases de données existantes (16) comme paramètres d'entrée. Par contre aucun d'entre eux n'exploite les processus métier/d'affaires (Business Process). Nous sommes partis du constat qu'étant donné le rôle majeur des processus métier dans l'organisation d'entreprise, ils peuvent devenir une source importante d'informations pour concevoir des applications à base de μServices à partir de celles existantes. À notre connaissance, seul Amiri préconise l'adoption des BPs pour identifier des μServices (3).

Selon Weske, *"A business process consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations"* (85). Afin d'identifier des groupements indépendants d'activités hautement interdépendantes, cohésifs, faiblement couplés et à granularité fine en tant que μServices, nous avons, dans un premier temps, exploré 2 types de dépendances entre activités, communément appelés flux de contrôle et flux de données, puis dans un second temps, les dépendances sémantiques, faisant l'objet du chapitre 4. Comme les activités de domaine sont reliées entre elles avec des opérateurs logiques comme (XOR, AND, OR) et s'échangent éventuellement des données/documents, leur formalisation permettrait de donner une indication sur la corrélation entre les activités. Pour illustrer les modèles de dépendances proposés dans ce chapitre, nous représentons le BP pour la location de vélo (Chapitre 5, Fig. 5.2).

3.2 Fondements

Bien que la méthode d'Amiri soit intéressante comme l'adoption de BPs et les notions de dépendances entre activités, la technique proposée souffre en revanche, de nombreuses limitations. Par exemple, il manque un modèle de dépendances structurelles d'activités pour capturer formellement les dépendances en fonction des opérateurs/connecteurs du langage de modélisation de processus métier utilisé. Aussi, le modèle de dépendances de données est très simple en se limitant aux opérations de lecture et d'écriture. Enfin, la solution présentée considère uniquement la pertinence de la structure et celle des données, tout en ignorant toute autre pertinence, notamment la sémantique entre les tâches.

Dans notre approche, les processus métier et les mesures de dépendance entre activités sont le cœur dans notre contribution d'identification automatique des µServices. Selon la définition de Davenport (21), un processus métier est un ensemble d'activités logiquement reliées entre elles afin d'atteindre des objectifs. Un modèle de processus métier se compose ainsi d'un ensemble d'activités et de contraintes d'exécution logiques entre elles. Nous identifions 3 types de dépendance comme suit :

- **Dépendance de contrôle** fait référence à la fois à l'ordre d'exécution entre activités (par exemple, *finish-to-start* et *start-to-start*) mais également aux connecteurs logiques (par exemple, XOR et AND) entre activités. Nous partons du postulat que si les activités sont directement connectées via une dépendance de contrôle, elles formeraient probablement un µservice hautement cohésif. Dans le cas contraire, elles seraient très probablement utilisés pour former des µservices séparés auxquels chacune serait affiliée.
- **Dépendance des données** fait référence à des flux d'entrées/sorties entre activités illustrant la circulation d'une activité à une autre. Ces entrées/sorties peuvent aussi bien correspondre à des artefacts (ou documents) qu'à leurs attributs pouvant être manipulés par différentes opérations comme illustré dans le tableau 5.1 (e.g., créer (c) et écrire (w)). En plus aux flux entrée/sortie, la dépendance des données pourrait aussi indiquer dans

quelle mesure les attributs des artefacts et/ou les artefacts sont obligatoires ou facultatifs pour une exécution BP. Nous préconisons que les activités travaillant sur les mêmes attributs d'artefacts devraient faire partie du même μservice afin d'éviter de retarder l'échange d'informations pertinentes.

- **Dépendance sémantique** fait référence à des similitudes entre activités devant être exécutées. Ces similitudes peuvent être extraites en utilisant le nom de l'activité et l'ontologie de domaine du BP. Nous préconisons que les activités avec des noms présentant des similitudes sémantiques et/ou faisant référence au même concept d'ontologie pourraient être interdépendants et feraient probablement partie du même μService. Ce type de dépendance fait l'objet du chapitre 4.

Sur la base des dépendances entre les activités, nous regroupons les activités dans μservices en utilisant des techniques de clustering. Dans la littérature, le clustering est soit centralisé soit collaboratif (49). Dans le premier, un seul composant gère le clustering en utilisant les caractéristiques des activités en termes de dépendances de *contrôlel*, *données* et *sémantiques* dépendances. Dans ce dernier, plusieurs composants, chacun en charge d'un type de fonctionnalité (e.g., par modèle ou par BP), échangent certains détails lors de la mise en cluster afin que des clusters appropriés soient construits conjointement. Les performances et la pertinence des techniques de clustering sont discutées amplement dans la littérature (19) et (86). De nombreux travaux comme (19) et (49) préconisent le clustering collaboratif pour identifier μservices. Il fournit des résultats précis contrairement au clustering centralisé où les caractéristiques des activités doivent être agrégées avant de lancer un clustering.

La figure 3.1 illustre notre approche sur les trois axes d'identification des μServices (i.e., contrôle, données, et sémantique). Principalement basées sur l'étude des différentes dépendances entre activités d'un ou plusieurs processus métier, les activités sont regroupées en μServices potentiellement à grain fin, hautement cohésifs et faiblement couplés, à travers un clustering collaboratif. Les étapes de collecte, d'analyse des dépendances et de clustering collaboratif se

déroulent comme suit :

- L'étape de collecte utilise la spécification de BP ainsi que des entrées provenant du concepteur de BP pour réunir tous les détails nécessaires à l'étape suivante et ce par type de dépendance (c'est-à-dire contrôle, données et sémantique).
- L'étape d'analyse des dépendances examine ces détails afin de quantifier les corrélations entre activités sous forme de mesures.
- L'étape de clustering collaboratif applique différentes techniques de clustering (une par type de dépendance) sur les mesures obtenues à l'étape précédente.

Par la suite, nous donnerons plus de détails sur la formalisation des dépendances de *contrôle* et de *données*, ainsi que *sémantiques*.

L'une des principales caractéristiques de notre approche est son algorithme de clustering collaboratif pouvant être appliqué à de nombreux BP lors de l'identification de µServices. Supposons un ensemble de n BP $\{BP_1, BP_2, \cdots, BP_n\}$ et un ensemble de m modèles de dépendance $\{M_1, M_2, \cdots, M_m\}$ liés à ces BP. Notre approche génère n matrices de dépendances à partir de chaque modèle, revenant au total à $n * m$ matrices de dépendances. Ensuite, ces matrices sont utilisées selon l'une des options suivantes :

- `Aucune fusion`. Cette option considère chaque matrice indépendante du reste, puis applique l'algorithme de clustering collaboratif aux $m * n$ nœuds où chaque nœud est associé aux $m * n$ matrices indépendantes.
- `Fusion par modèle`. Cette option fusionne toutes les matrices provenant du même modèle appliqué aux différents BP, puis applique l'algorithme de clustering collaboratif aux m nœuds où chaque nœud est associé à une matrice fusionnée.
- `Fusion par BP`. Cette option fusionne toutes les matrices qui proviennent des différents modèles de dépendance appliqués à un BP, puis applique l'algorithme de clustering collaboratif aux m nœuds où chaque nœud est associé à une matrice fusionnée.

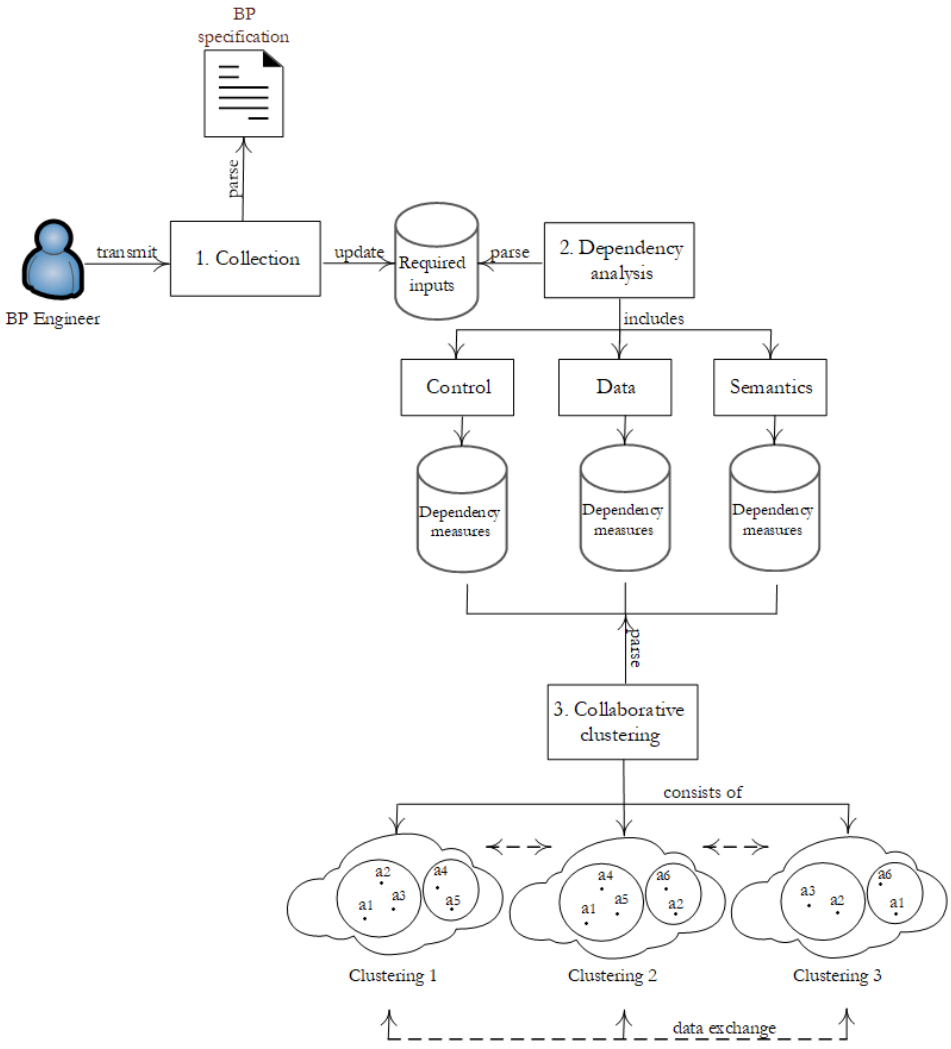
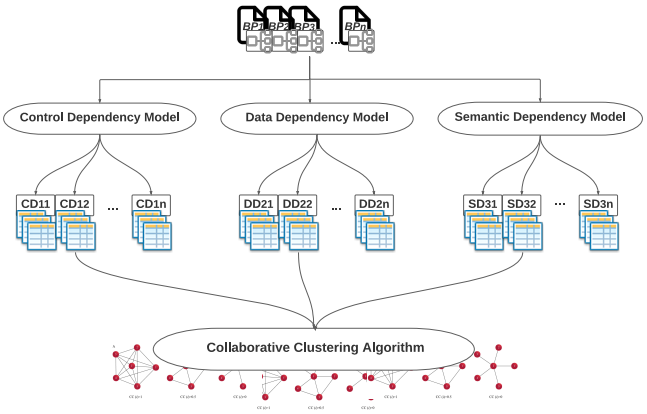
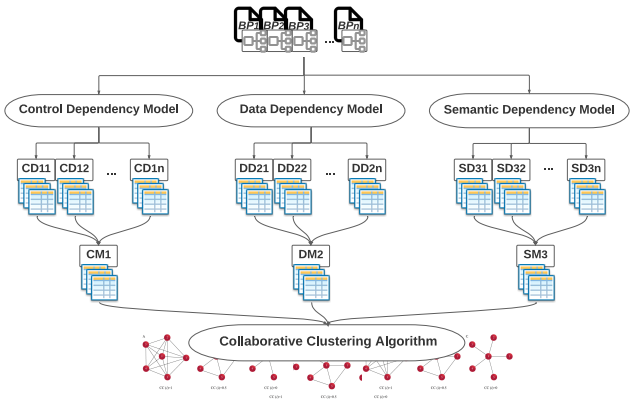


FIGURE 3.1: Représentation générale de notre approche d'identification des μServices

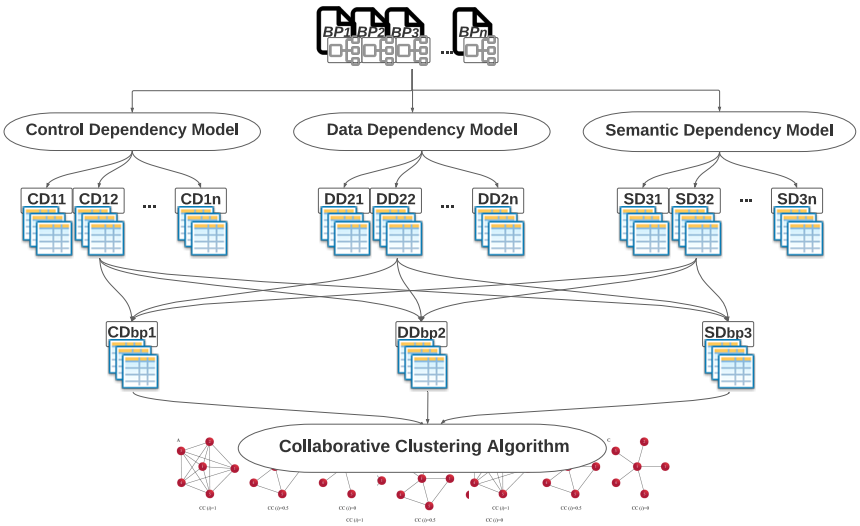
La figure 3.2c illustre ces trois options pour le cas de *n* BP et nos trois modèles de dépendance. Il est important de noter que la première option ne fera pas appel à l'algorithme de clustering collaboratif. Néanmoins, l'utilisateur peut, pour n'importe quelle raison, choisir les autres options ou encore définir de nouvelles telles que la fusion de matrices de manière aléatoire. Une fois les μservices identifiés, leur choix d'implémentation peut être soit propriétaire, soit basée sur des ressources externes. Le choix retenu pourrait être basé sur certaines exigences, notamment la sécurité et la confidentialité.



(a) Aucune Fusion



(b) Fusion par modèle



(c) Fusion par BP

3.3 Dépendances issues de flux de contrôle

Pour mesurer une *dépendance de contrôle* entre 2 activités (a_i, a_j), nous considérons la probabilité d'occurrence de a_j après l'exécution de a_i. Cette probabilité dépend de l'ordre d'exécution et / ou des opérateurs logiques entre a_i et a_j. Considérons la *dépendance de contrôle* entre a_5 et a_{10} étant reliée à a_6 via XOR (Fig. 5.2). Après avoir exécuté a_5, la probabilité d'occurrence de a_{10} dépend de la décision prise à XOR (c'est-à-dire soit a_6 ou a_{10}). Nous notons que la probabilité d'occurrence de toute activité est calculée dans le temps en utilisant le journal d'exécution du BP.

3.3.1 Analyse préliminaire des flux de contrôle dans un BP

Il existe 2 types de *dépendances de contrôle* à savoir, direct et indirect. Le premier fait référence à un certain ordre d'exécution (ExecOrder) entre a_i et a_j **pouvant** être connecté à son tour, à d'autres activités a_k via un certain Operator. Un ordre d'exécution entre 2 activités peut être illustré avec soit *finish-to-start* (noté SEQ), *finish-to-finish*, *start-to-start*, ou *start-to-finish*. Ces derniers font référence à un chemin d'exécution (ExecPath) avec un certain ordre d'exécution entre a_i et a_j impliquant d'autres activités a_k connectées via 2 ou plusieurs opérateurs.

Cas de la séquence

Soit CD(a_i, a_j[Operator $\{a_k\}$])$_{\text{ExecOrder}}$ une *dépendance de contrôle* (en anglais, *C*ontrol *D*ependency) directe. Nous commençons par une *dépendance de contrôle* simple, c'est-à-dire $CD(a_i, a_j)_{\text{SEQ}}$ (i.e., $a_k = \emptyset$). SEQ entre a_i et a_j signifie que a_j ne pourra s'exécuter qu'après une exécution réussie de a_i ; autrement dit, l'exécution de a_j reste incertaine. $CD(a_i, a_j)_{\text{SEQ}}$ indique une probabilité d'occurrence de a_j (p) après la terminaison de a_i

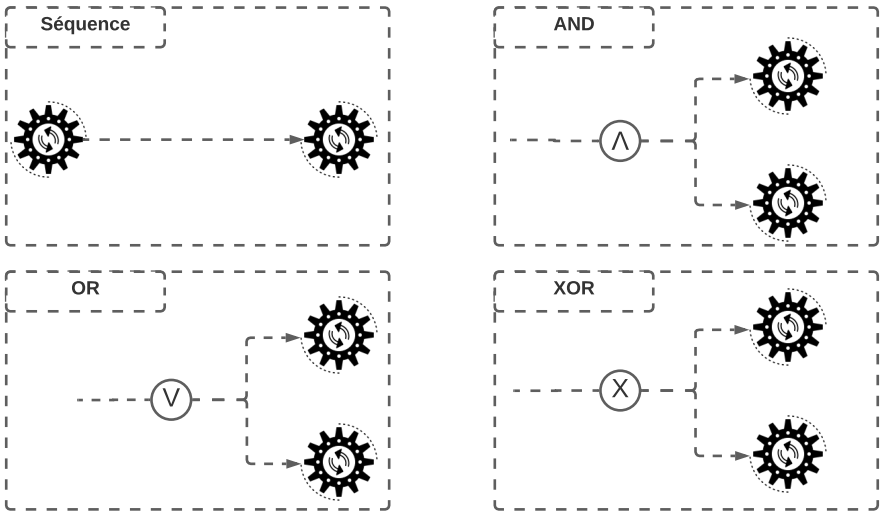


FIGURE 3.3: Les quatre connecteurs logiques utilisés dans notre approche

selon l'équation 3.1 :

$$CD(a_i, a_j)_{\mathsf{SEQ}} = p \tag{3.1}$$

where $p \in]0,1]$.

Nous examinons maintenant la *dépendance de contrôle* $CD(a_i, a_j \,\mathsf{Operator}\, \{a_k\})_{\mathsf{SEQ}}$ (c'est-à-dire $\{a_k\} \neq \emptyset$)). Selon la sémantique de Operator, nous supposons que certaines activités r de $\{a_k\} \cup a_j$ seront sélectionnées pour l'exécution. L'équation 3.2 définit le nombre d'ensembles contenant r activités qui seront sélectionnés pour l'exécution comme une combinaison $C(n,r)$ où n correspond à $card(\{a_k\} \cup a_j)$.

$$C(n,r) = \frac{n\,!}{r\,! \times (n-r)\,!} \tag{3.2}$$

Selon la sémantique de Operator (i.e., AND, XOR, ou OR), $CD(a_i, a_j \,\mathsf{Operator}\, \{a_k\})_{\mathsf{SEQ}}$ est calculé comme suit :

Cas de AND

$CD(a_i, a_j \text{ AND } \{a_k\})_{\text{SEQ}}$. Cette dépendance signifie que l'exécution de a_j ne pourra démarrer qu'après celle de a_i ayant réussie indépendamment de $\{a_k\}$. Formellement, l'équation 3.3 permet de calculer $CD(a_i, a_j \text{ AND } \{a_k\})_{\text{SEQ}}$ comme suit :

$$CD(a_i, a_j \text{ AND } \{a_k\})_{\text{SEQ}} = C(n,n) * CD(a_i, a_j)_{\text{SEQ}} \tag{3.3}$$

où $p \in]0,1]$ & $C(n,n) = 1$, selon l'équation 3.2.

Cas de XOR

$CD(a_i, a_j \text{ XOR } \{a_k\})_{\text{SEQ}}$. Cette dépendance signifie qu'une seule activité sera sélectionnée parmi $\{a_k\} \cup a_j$ après que a_i se soit terminée avec succès. Formellement, l'équation 3.4 calcule $CD(a_i, a_j \text{ XOR } \{a_k\})_{\text{SEQ}}$ comme suit :

$$CD(a_i, a_j \text{ XOR } \{a_k\})_{\text{SEQ}} = \frac{1}{C(n,1)} * CD(a_i, a_j)_{\text{SEQ}} \tag{3.4}$$

où $C(n,1)$ est le nombre de possibilités pour sélectionner une activité parmi $\{a_k\} \cup a_j$. Selon l'équation 3.2, $C(n,1)$ est égal à n.

Cas de OR

$CD(a_i, a_j \text{ OR } \{a_k\})_{\text{SEQ}}$. Cette dépendance signifie qu'un ensemble d'activités r de $2^{\{a_k\} \cup a_j}$ (c'est-à-dire tous les choix multiples possibles) sera sélectionné après que a_i se soit terminé avec succès. Par souci de simplicité, nous supposons que toute activité dans $\{a_k\} \cup a_j$ a la même probabilité d'occurrence sur $2^{\{a_k\} \cup a_j}$, étant égale à $\frac{r}{n}$ où r varie de 1 à n. Formellement, l'équation 3.5 calcule $CD(a_i, a_j \text{ OR } \{a_k\})_{\text{SEQ}}$ comme suit.

$$CD(a_i, a_j \text{ OR } \{a_k\})_{\text{SEQ}} = \frac{\sum_{r=1,n}(\frac{r}{n} \times C(n,r))}{\sum_{r=1,n} C(n,r)} * CD(a_i, a_j)_{\text{SEQ}} \tag{3.5}$$

où

- $\sum_{r=1,n}(\frac{r}{n} \times C(n,r))$ représente le nombre d'occurrences de a_j parmi les combinaisons possibles d'activités [1].
- $\sum_{r=1,n} C(n,r)$ correspond au nombre total de combinaisons possibles d'activités [2].

3.3.2 Analyse complémentaire des flux de contrôle

Nous voulons maintenant calculer la dépendance structurelle ci-dessus entre une activité de début (a_i) et une activité de fin (a_j) du BP, composée de multiples connecteurs logiques différents.

Soit $CD(a_i, a_j)_{\text{ExecPath}}$ une dépendance de *contrôle indirecte* entre a_i et a_j. Nous commençons par une *dépendance de contrôle* simple $CD(a_i, a_j)_{\text{Path}_{i,j}}$ où $\text{Path}_{i,j}$ fait référence à un ensemble d'autres activités ($\{a_k\}$) connectés avec des opérateurs. $CD(a_i, a_j)_{\text{Path}_{i,j}}$ désigne alors la probabilité d'une série d'événements **conjonctifs** où chaque événement se réfère à l'occurrence de a_k dans $\text{Path}_{i,j}$ selon l'équation 3.6 :

$$CD(a_i, a_j)_{\text{Path}_{i,j}} = \prod_{a_l, a_m \in \text{Path}_{i,j}} CD(a_l, a_m \text{ Operator } \{a_{k_m}\})_{\text{SEQ}} \tag{3.6}$$

Nous examinons maintenant la *dépendance de contrôle* $CD(a_i, a_j)_{\text{Paths}_{i,j}}$ où plusieurs chemins d'exécution ($\{\text{Path}^q_{i,j}\}$) **possibles** existent entre a_i et a_j. $CD(a_i, a_j)_{\text{Paths}_{i,j}}$ désigne alors une agrégation de toutes les *dépendances de contrôle* simples. L'équation 3.7 permet de calculer $CD(a_i, a_j)_{Path_{i,j}}$ comme suit :

$$CD(a_i, a_j)_{\text{Paths}_{i,j}} = \text{Agg}(CD(a_i, a_j)_{\{\text{Path}^q_{i,j}\}_{q=1,\ldots}}) \tag{3.7}$$

1. Soit n 3, $\sum_{r=1,n}(\frac{r}{n} \times C(n,r))$ a la valeur suivante : $(\frac{1}{3} \times 3 + \frac{2}{3} \times 3 + \frac{3}{3} \times 1 = 4)$.
2. Soit n 3, $\sum_{r=1,n} C(n,r)$ a la valeur suivante : (3+ 3+ 1) = 7.

où Agg fait référence à une fonction d'agrégation couramment utilisée comme le maximum dans nos expériences.

La Section 5.3.2 contient un exemple de matrice de dépendances de contrôle entre activités du BP pour la location de vélo.

3.4 Dépendances issues de flux de données

Pour mesurer une *dépendance de données* entre 2 activités (a_i, a_j), nous considérons le niveau de criticité d'un artefact/attribut reflétant l'importance des informations partagées entre ces activités. Ce niveau indique dans quelle mesure l'indisponibilité de l'artefact/attribut aurait un impact sur la continuité des activités dans le BP, à un niveau plus global. Nous considérerons la *criticité* d'un artefact/ attribut comme un moyen de mesurer la *dépendance de données* entre 2 activités. Dans (71), Paulsen et al. fournissent une analyse de criticité complète au profit des organisations ayant besoin d'identifier et de hiérarchiser les actifs (par exemple, artefacts et processus) vitaux pour atteindre leurs objectifs. En puisant de cette analyse, nous distinguons 2 types de criticité : *fonctionnel* (F) faisant référence à l'indisponibilité d'un artefact/attribut pouvant entraver la bonne exécution du BP, et *non fonctionnel* (NF) faisant référence à la corruption d'un artefact/attribut pouvant nuire à la qualité de service du BP (QoS).

Pour aider les concepteurs de BP à classer, dans une certaine mesure, les artefacts/attributs comme critiques ou non critiques, nous identifions les propriétés F - et NF de l'attribut/artefact en terme de criticité.

Tout d'abord, nous considérons 3 niveaux d'information notamment, *stratégique*, *tactique* et *opérationnel* comme propriétés fonctionnelles en terme de criticité. Par exemple, un attribut *stratégique* devrait être plus critique que des attributs *tactiques* ou *opérationnels*. Aussi, nous considérons 3 niveaux de sécurité, notamment *confidentialité*, *intégrité* et *disponibilité*,

comme propriétés non-fonctionnelles en terme de criticité. Par exemple, certaines activités de *BP* peuvent ne pas être concernés par les questions de *confidentialité* mais plus par celles relatives à la *disponibilité* et l'*intégrité*. La table 3.1 donne des exemples de propriétés de manière générale.

TABLE 3.1: Exemples de propriétés relatives à la criticité

Criticité	Propriétés	Exemple
Fonctionnel	Opérationnelles	Données décisionnelles
	Tactiques	Données de concurrence
	Stratégiques	Données d'expérience client
Non fonctionnel	Sécurité	Données personnelles protégées
	Intégrité	Données d'identité
	Confidentialité	Données financières

3.4.1 Criticité des données

Pour définir le degré de criticité d'un artefact (*ar*)/attribut (at) ($DC(ar|at)$), le concepteur de BP doit travailler sur la priorité entre propriétés fonctionnelles et non-fonctionnelles en lien avec la criticité en utilisant 3 classes de valeur, Faible (L), Moyen (M), et Élevé (H). Ensuite, il devra faire une correspondance entre ces classes (L, M et H) et les intervalles de valeurs suivants : [0, k [, [k, k' [, et [k',k''], respectivement, et ce après avoir défini les valeurs de k, k' et k". Finalement, le concepteur de BP définit $DC(ar|at)$ en fonction de l'intervalle associé à la classe de valeur à laquelle la propriété $ar|at$ appartient. Puisque les artefacts et les attributs peuvent être associés à des proprités F et / ou NF, nous spécialisons $DC(ar|at)$ en $DC^{\mathsf{F}}(ar|at)$ et $DC^{\mathsf{NF}}(ar|at)$.

3.4.2 Stratégies de calcul

Pour calculer $DC(ar|at)$, nous définissons deux stratégies. La première calcule $DC(ar|at)$ comme une somme pondérée de $DC^{\mathsf{F}}(ar|at)$ et $DC^{\mathsf{NF}}(ar|at)$ (Équation 3.8).

$$DC(ar|at) = w_1 \times DC^{\mathsf{F}}(ar|at) + w_2 \times DC^{\mathsf{NF}}(ar|at), \quad w_1 + w_2 = 1 \tag{3.8}$$

où w_1 et w_2 sont des poids (ie, importance) associés à $DC^F(ar|at) \mid DC^{NF}(ar|at)$, respectivement, et définis par le concepteur du BP .

Quant à la seconde, elle considère $DC(ar|at)$ comme un tuple $< DC^{\mathsf{F}}(ar|at), DC^{\mathsf{NF}}(ar|at) >$. On spécifie donc les *dépendances de données* ($\mathscr{DD}^2$) entre a_i et a_j (Equation 3.9) :

$$\mathscr{DD}^2(a_i, a_j) = \sum_{ar_{i,j}|at_{i,j} \in DATA_{i,j}} \mathscr{F}(pair(ar_{i,j}|at_{i,j}), DC^{\mathsf{F}}(ar_{i,j}|at_{i,j}), DC^{\mathsf{NF}}(ar_{i,j}|at_{i,j})) \tag{3.9}$$

où $\mathscr{F}$ renvoie la valeur de la *dépendance de données* spécifiée par le concepteur du BP pour $< pair(ar_{i,j}|at_{i,j}), DC^{\mathsf{F}}(ar_{i,j}|at_{i,j}), DC^{\mathsf{NF}}(ar_{i,j}|at_{i,j}) >$.

Une fois que tous les $DC(ar|at)$ sont établis, nous spécifions maintenant les *dépendances de données* ($\mathscr{DD}^1$) entre a_i et a_j (Équation 3.10).

$$\mathscr{DD}^1(a_i, a_j) = \sum_{ar_{i,j}|at_{i,j} \in DATA_{i,j}} pair(ar_{i,j}|at_{i,j}) \times DC(ar_{i,j}|at_{i,j}) \tag{3.10}$$

où

- $ar_{i,j}|at_{i,j}$ représente l'artefact /attribut échangé entre a_i et a_j,
- $DATA_{i,j}$ est l'ensemble de $ar_{i,j}|at_{i,j}$, et
- $pair(ar_{i,j}|at_{i,j})$ désigne la valeur associée à la paire d'activités (par exemple, r/w, w/w, et c/r) entre a_i et a_j, proposé par Amiri (3).

La Section 5.3.3 contient un exemple de matrice de dépendances de données entre activités du BP pour la location de vélo.

3.5 Conclusion

Pour répondre aux limites des systèmes monolithiques, nous avons exploré comment identifier les μServices nécessaires à partir d'un ensemble de BP. Contrairement à d'autres approches de refactoring basées sur des codes sources et/ou des logs, les BP constituent une meilleure alternative. Les BPs permettent de définir qui fait quoi, quand, où et pourquoi. Nous avons proposé 2 modèles de dépendances, appelées contrôle et données, pour capturer ces détails. Le premier modèle permet notamment d'établir des dépendances directes et indirectes entre activités en fonction des connecteurs les reliant et utilisant les probabilités d'occurrence. Le second modèle permet quant à lui de considérer la criticité des données échangées entre activités comme indicateur de leur cohésion et (de)couplage.

Dans le chapitre suivant, nous étendons notre champ d'investigation à un autre type de dépendances, notamment capitaliser sur les liens sémantiques entre activités formant de potentiels μServices.

4

Identification de μServices basée sur la sémantique

Table des matières

4.1 Introduction 58
4.2 Stratégie dirigée par les termes 58
4.3 Stratégie dirigée par les concepts 60
4.4 Stratégie dirigée par les fragments d'une ontologie 62
4.5 Calcul des dépendances sémantiques entre activités 63
4.6 Conclusion 66

4.1 Introduction

Ce chapitre présente un modèle pour la représentation des dépendances sémantiques qui peuvent exister entre les activités d'un processus métier. Ce modèle permet de mesurer automatiquement le niveau de similarité entre deux activités soit directement en comparant leurs noms (*stratégie dirigée par les termes*), soit indirectement en comparant les concepts d'une ontologie auxquels les activités peuvent être rattachées (*stratégie dirigée par les concepts*), ou encore en comparant les fragments d'une ontologie auxquels les activités peuvent être rattachées (*stratégie dirigée par les fragments*). Le recours à l'ontologie permet de nous affranchir des conventions d'écriture pouvant exister dans le nommage des activités. En outre, les relations sémantiques pouvant exister dans une ontologie permettront de mieux mesurer la similarité entre les activités.

4.2 Stratégie dirigée par les termes

La stratégie dirigée par les termes, notée ($\mathscr{W}$), est simple et vise à mesurer à quel point deux termes sont similaires. Dans notre cas, le terme fait référence au nom, simple ou composé, d'une activité comme par exemple "vérifier les informations d'identification de l'utilisateur" dans notre étude de cas Bicing. Un des avantages de la stratégie *dirigée par les termes* est qu'elle ne nécessite pas l'utilisation d'une ontologie de domaine. Elle repose sur le principe de co-occurrence de mots [1]. La co-occurrence fait référence à la fréquence d'occurrence au-dessus du hasard de deux termes et est utilisée comme indicateur pour mesurer la proximité sémantique de deux termes.

Le fonctionnement de l'annotation des activités dirigée par les termes est illustré par la figure 4.1. Les termes (noms) des activités d'un processus métier (BP) sont d'abord extraits. Ils sont ensuite utilisés par paire pour calculer le niveau de la co-occurence de chaque paire.

1. en.wikipedia.org/wiki/Co-occurrence.

Cette co-occurence permet ensuite de calculer la dépendance sémantique (orientée termes) de chaque paire d'activités. L'ensemble de ces dépendances est stocké sous forme d'une matrice.

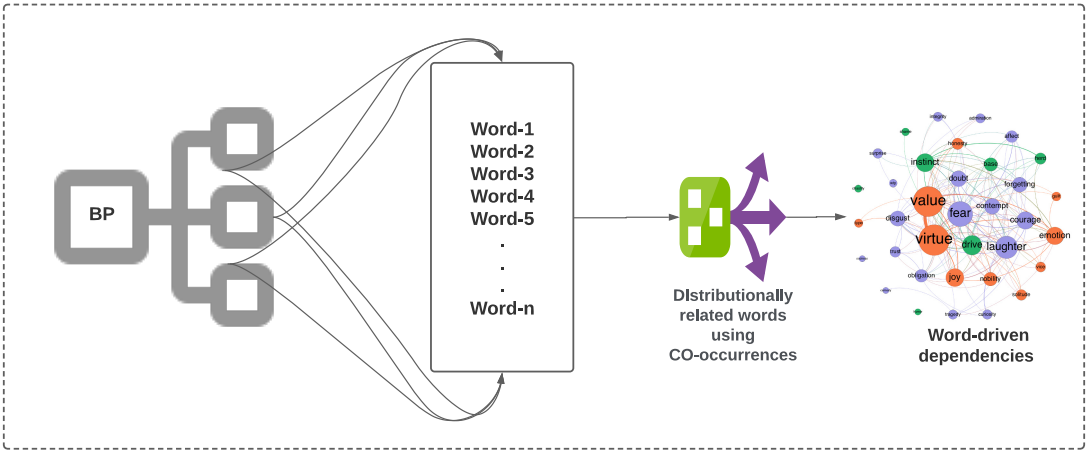


FIGURE 4.1: Annotation des activités dirigée par les termes

L'algorithme 1 décrit comment une activité a_i est annotée en utilisant un ensemble de n mots ($\mathscr{W}_{a_i}$). Pour calculer $\mathscr{W}_{a_i}$, nous adoptons la solution de Kolb et al., *DISCO* (DIStributionally related words using CO-occurences) (56), qui suppose que des mots ayant une signification similaire (co-) apparaissent dans des contextes similaires de sacs de mots. En adoptant *DISCO*, l'ensemble des n termes les plus similaires à a_i sur le plan de la distribution ($\{w_k\}$) ainsi que leurs degrés de similitude respectifs ($sd_{i,k}$) (c'est-à-dire $\mathscr{W}_{a_i}$) sont retournés.
Outre *DISCO*, toute autre technique de similitude textuelle sémantique peut être adoptée dans notre approche. Ainsi, une activité a_i sera annotée par $\mathscr{W}_{a_i} = \{< w_k, sd_{i,k} >\}$.

Algorithm 1: Annotation dirigée par les termes

Input: a_i, n
Output: $\mathscr{W}_{a_i}$
1 **begin**
2 $\quad \mathscr{W}_{a_i} \leftarrow DISCO_2(a_i, n)$
3 $\quad$ **return** $\mathscr{W}_{a_i}$

TABLE 4.1: Extrait de valeurs de similitude via *les termes*

Activity / Activity	a_1	a_2	a_3	a_4	a_5	a_6
a_1	1	0.180	0.034	0.015	0.010	0.041
a_2	0.180	1	0.042	0.012	0.009	0.020
a_3	0.034	0.042	1	0.0019	0.266	0.120
a_4	0.015	0.012	0.0019	1	0.0043	0.0021
a_5	0.010	0.009	0.266	0.0043	1	0.0593
a_6	0.041	0.020	0.120	0.0021	0.0593	1

4.3 Stratégie dirigée par les concepts

La stratégie dirigée par les concepts, notée ($\mathscr{C}$), est basée sur l'utilisation d'une ontologie de domaine pour annoter les noms d'activités des BPs avec des concepts de l'ontologie. La dépendance sémantique entre deux activités d'un BP est alors déduite des similitudes qui pourraient exister entre les différents concepts annotant les activités en question.

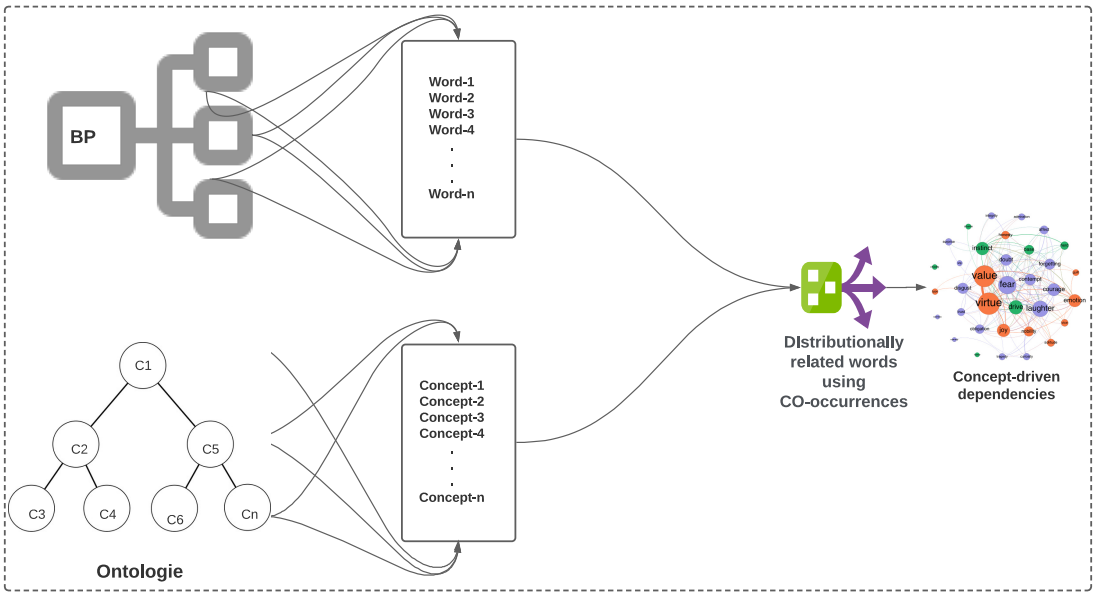


FIGURE 4.2: Annotation des activités dirigée par les concepts

La figure 5.4 présente un exemple d'une ontologie de domaine $\mathscr{O}_{BP}$ que nous avons créée pour les besoins d'une application de location de vélos.

L'algorithme 2 décrit comment une activité a_i est annotée en utilisant un ensemble de

concepts ($\mathscr{C}_{a_i}$) qui appartiennent à l'ontologie de domaine ($\mathscr{O}_{BP}$) relatif à un BP donné.

Pour chaque concept c_j de l'ontologie $\mathscr{O}_{BP}$, l'algorithme appelle d'abord la mesure de similarité sémantique de Kolb et al. $DISCO_2$ [2] entre les différents termes a_i d'un BP et le concept c_j. Les différentes similarités ainsi calculées sont stockées dans une matrice $D[a_i, c_j]$. Le tableau 4.2 décrit les valeurs de similarité associées à notre exemple Bicing. Pour calculer l'ensemble $\mathscr{C}_{a_i}$ des concepts annotant une activité a_i, la fonction max_s conservera les concepts (c_k) ayant les valeurs de similarité les plus élevées dans $D[a_i]$ en tenant en compte d'une certaine précision σ. $D[a_i]$ est la projection de la matrice D sur la dimension a_i. Ainsi, si la valeur la plus élevée des similarités est de $0,5$ et que σ vaut 0,1, alors max_s inclura tous les concepts possédant avec l'activité de nom a_i une similarité comprise entre 0,4 et 0,5.
Il est important de remarquer que les activités sont donc annotées avec un nombre de concepts variable. Ainsi, une activité a_i sera annotée par $(\mathscr{C}_{a_i}) = \{< c_k, D[i,k] >\}$.

Algorithm 2: annotation dirigée par les concepts

Input: a_i, $\mathscr{O}_{BP}$, σ
Output: $\mathscr{C}_{a_i}$
1 **begin**
2 **foreach** $c_j \in \mathscr{O}_{BP}$ **do**
3 $\mathscr{D}[a_i, c_j] \leftarrow DISCO_2(a_i, c_j)$
4 $\mathscr{C}_{a_i} \leftarrow max_s(\{\mathscr{D}[a_i, c_j]\}, \sigma)$
5 **return** $\mathscr{C}_{a_i}$

TABLE 4.2: Extrait de valeurs de similitude via *les concepts*

Concept / Activité	*Bicycle*	*BicycleRental*	*User*	*AchorPoint*	*Station*
a_1	0.01818	0.02632	0.18076	0.04190	0.03621
a_2	0.01924	0.04481	0.99997	0.05231	0.03667
a_3	0.00421	0.01784	0.04266	0.12018	0.14423
a_5	0.00718	0.00476	0.00962	0.05937	0.02105
a_9	0.00968	0.01392	0.15772	0.0614	0.03038
a_{10}	0.12016	0.06755	0.04724	0.04081	0.10965

2. $DISCO_2$ calcule la similitude sémantique (c.-à-d., relation entre concepts) tandis que *DISCO* calcule la similarité distributionnelle (c'est-à-dire la relation entre les mots).

4.4 Stratégie dirigée par les fragments d'une ontologie

La stratégie dirigée par les fragments d'une ontologie, notée $\mathscr{F}$, vise à annoter une activité non pas par des concepts pris individuellement, mais plutôt par des fragments définis comme un ensemble compact de concepts. Le recours à une telle annotation par fragments permettrait de tenir compte des différents liens sémantiques pouvant exister entre des concepts. Ainsi, les dépendances sémantiques entre deux activités seront fortement dépendantes des similitudes qui pourraient exister entre les couples de fragments.

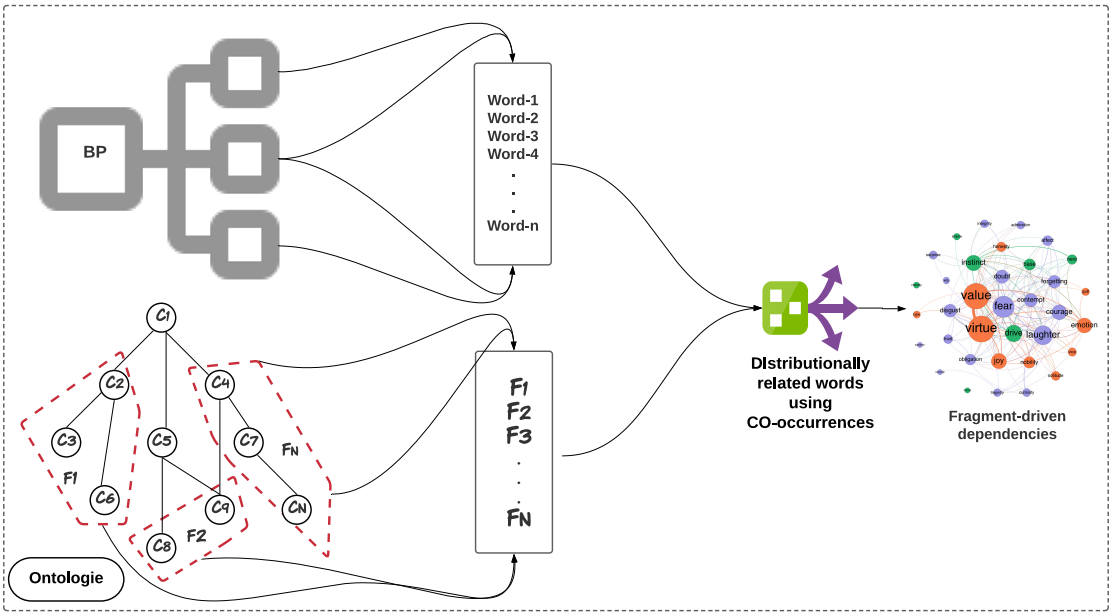


FIGURE 4.3: Annotation des activités dirigée par les fragments

L'algorithme 3 décrit comment une activité a_i est annotée à l'aide d'un ensemble de **fragments** ($\mathscr{F}_{a_i}$) les plus similaires englobant chacun des concepts appartenant à $\mathscr{O}_{BP}$. Pour développer $\mathscr{F}_{a_i}$, l'algorithme 3 calcule d'abord les degrés d'appartenance de a_i ($\mathscr{M}[i]$) à chaque fragment $\in \mathscr{O}_{BP}$ basé sur l'ensemble des concepts communs (c_j) entre ce fragment et $\mathscr{C}_{a_i}$ avec $D[i,j]$ obtenu dans l'algorithme 2, ligne 2. Ensuite, la fonction max_m considère les fragments (F_k) avec le plus haut degré d'appartenance ($\mathscr{M}[i]$) par rapport à une certaine précision σ. Ainsi, a_i sera annoté par $\mathscr{F}_{a_i} = \{< F_k, \mathscr{M}[i,k] >\}$.

Le tableau 5.9 représente un extrait des valeurs de similitude associées à Bicing.

Algorithm 3: Annotatation dirigée Fragments

Input: $\mathscr{C}_{a_i}$, $\mathscr{O}_{BP}$, σ
Output: $\mathscr{F}_{a_i}$
1 **begin**
2 **foreach** $\mathscr{F}_k \in \mathscr{O}_{BP}$ **do**
3 $\mathscr{M}[i,k] \leftarrow \sum_{c_j \in \mathscr{F}_k \cap \mathscr{C}_{a_i}} \mathscr{D}[i,j]$
4 $\mathscr{F}_{a_i} \leftarrow max_m(\mathscr{M}[i], \sigma)$ **return** $\mathscr{F}_{a_i}$

TABLE 4.3: Extrait des valeurs de similitude via *les fragments*

Fragment / Activité	F_1	F_2	F_3
a_1	0.98182	0.97368	0.9581
a_2	0.98076	0.95519	0.94769
a_3	0.99579	0.98216	0.95734

4.5 Calcul des dépendances sémantiques entre activités

Dans les sections précédentes, nous avons montré comment annoter chaque activité en utilisant une des stratégies suivantes : dirigée par les termes, dirigée par les concepts, et dirigée par les fragments. Nous présenterons dans cette section la méthode de calcul des dépendances sémantiques entre deux activités à partir d'une des annotations précédentes.

Etant donné une annotation des activités, la dépendance sémantique (*SeD*) entre deux activités a_i et a_j est formellement définie par l'équation 4.1.

$$SeD(a_i, a_j) = 1 - d_{\mathscr{X}}(\mathscr{X}_{a_i}, \mathscr{X}_{a_j}) \tag{4.1}$$

où $\mathscr{X}_{a_i}$ correspond au résultat de l'annotation des activités (soit $\mathscr{W}_{a_i}$, ou $\mathscr{C}_{a_i}$, ou $\mathscr{F}_{a_i}$) et $d_{\mathscr{X}}$ représente la distance entre deux annotations $\mathscr{X}_{a_i}$ et $\mathscr{X}_{a_j}$. Nous définissons ci-après $d_{\mathscr{X}}$ en fonction de la technique d'annotation utilisée.

- *Distance entre deux annotations dirigées par les termes*. La distance $d_{\mathscr{X}}$ est notée $d_{\mathscr{W}}$. Celle-ci aura pour but de comparer tous les mots de $\mathscr{W}_{a_i}$ à ceux de $\mathscr{W}_{a_j}$. Ainsi, l'équation 4.1 se déclinera sous la forme de l'équation 4.2 dans le cas des annotations dirigées par les termes.

$$d_{\mathscr{W}}(\mathscr{W}_{a_i}, \mathscr{W}_{a_j}) = \sum_{w_k \in \mathscr{W}_{a_i}^p} sd_{i,k} + \sum_{w_k \in \mathscr{W}_{a_j}^p} sd_{j,k} \tag{4.2}$$

où $\mathscr{W}_{a_i}^p$ représente l'ensemble de mots *privés* de $\mathscr{W}_{a_i}$ (c'est-à-dire $\mathscr{W}_{a_i} - \mathscr{W}_{a_i} \cap \mathscr{W}_{a_j}$). Un mot privé est un mot appartenant exclusivement à un ensemble.

L'équation 4.3 normalise le calcul de la distance $d_{\mathscr{W}}$.

$$d_{\mathscr{W}}^{norm}(\mathscr{W}_{a_i}, \mathscr{W}_{a_j}) = \frac{d_{\mathscr{W}}(\mathscr{W}_{a_i}, \mathscr{W}_{a_j})}{|\mathscr{W}_{a_i}^p| + |\mathscr{W}_{a_j}^p|} \tag{4.3}$$

Un extrait des dépendances sémantiques entre certaines activités de notre exemple de location de vélos est présenté dans le tableau 4.4. Les dépendances sont calculées en utilisant l'équation 4.3.

TABLE 4.4: Extrait des dépendances sémantiques - technique d'annotation dirigée par les termes

Activité / Activité	a_1	a_2	a_3	a_4	a_5	a_6
a_1	1	0.180	0.034	0.015	0.010	0.041
a_2	0.180	1	0.042	0.012	0.009	0.020
a_3	0.034	0.042	1	0.0019	0.266	0.120
a_4	0.015	0.012	0.0019	1	0.0043	0.0021
a_5	0.010	0.009	0.266	0.0043	1	0.0593
a_6	0.041	0.020	0.120	0.0021	0.0593	1

- *Distance entre deux annotations dirigées par les concepts*. La distance $d_{\mathscr{X}}$ est notée $d_{\mathscr{C}}$. Celle-ci aura pour but de comparer tous les concepts de l'ensemble $\mathscr{C}_{a_i}$ associé à l'activité a_i avec ceux d'un autre autre ensemble $\mathscr{C}_{a_j}$ associé à l'activité a_j. Ainsi,

l'équation 4.1 se déclinera sous la forme de l'équation 4.4 dans le cas des annotations dirigées par les concepts.

$$d_{\mathscr{C}}(\mathscr{C}_{a_i},\mathscr{C}_{a_j}) = \sum_{c_k \in \mathscr{C}_{a_i}} (1-\mathscr{D}[i,k]) * \sum_{c_l \in \mathscr{C}_{a_j}} (1-\mathscr{D}[j,l]) * WU(c_k,c_l) \tag{4.4}$$

où $WU(c_k,c_l)$ représente la distance entre deux concepts c_k et c_l (89).

L'équation 4.5 normalise le calcul de la distance $d_{\mathscr{C}}$.

$$d^{norm}_{\mathscr{C}}(\mathscr{C}_{a_i},\mathscr{C}_{a_j}) = \frac{d_{\mathscr{C}}(\mathscr{C}_{a_i},\mathscr{C}_{a_j})}{|\mathscr{C}_{a_i}| * |\mathscr{C}_{a_j}|} \tag{4.5}$$

Un extrait des dépendances sémantiques entre certains couples d'activités de notre exemple Bicing en utilisant l'équation 4.5 est présenté dans le tableau 4.5.

TABLE 4.5: Extrait des dépendances sémantiques - technique d'annotation dirigée par les concepts

Activité / Activité	a_1	a_2	a_3	a_4	a_5
a_1	-	0,488573	0,488843	0,390858	0,320864
a_2	0,488573	-	0,488315	0,320517	0,320517
a_3	0,488843	0,488315	-	0,495525	0,325429
a_4	0,390858	0,320517	0,495525	-	0,275969
a_5	0,320864	0,320517	0,325429	0,275969	-

- *Distance entre deux annotations dirigée par les fragments*. La distance $d_{\mathscr{X}}$ est notée $d_{\mathscr{F}}$. Celle-ci aura pour but de comparer tous les fragments de $\mathscr{F}_{a_i}$ associés à l'activité a_i avec ceux de $\mathscr{F}_{a_j}$ associées à l'activité a_j. Ainsi, l'équation 4.1 se déclinera sous la forme de l'équation 4.6 dans le cas des annotations dirigées par les fragments.

$$d_{\mathscr{F}}(\mathscr{F}_{a_i}, \mathscr{F}_{a_j}) = \prod_{\mathscr{F}_k \in \mathscr{O}_{BP}} (1 - |\mathscr{M}[i,k] - \mathscr{M}[j,k]|) \tag{4.6}$$

Soit $\mathscr{P}_2(BP)$ l'ensemble de toutes les paires d'activités distinctes construites à partir de BP (c'est-à-dire $a_k \neq a_l$). Formellement, l'équation 4.7 calcule le $d_{\mathscr{F}}$ normalisé comme suit.

$$d^{norm}_{\mathscr{F}}(\mathscr{F}_{a_i}, \mathscr{F}_{a_j}) = \frac{d_{\mathscr{F}}(\mathscr{F}_{a_i}, \mathscr{F}_{a_j})}{d^{max}_{\mathscr{F}}} \tag{4.7}$$

Où

- $d^{max}_{\mathscr{F}} = max(\{d_{\mathscr{F}}(\mathscr{F}_{a_k}, \mathscr{F}_{a_l})\}_{<a_k,a_l> \in \mathscr{P}_2(BP)})$

4.6 Conclusion

En complément des modèles de dépendances entre activités proposés dans les chapitres précédents, le modèle de dépendances sémantiques entre activités présenté dans ce chapitre se base sur trois techniques d'annotation dirigées : par les termes, par les concepts d'une ontologie, et par les fragments d'une ontologie. Ce modèle exploite l'algorithme de mesure de similarité sémantique DISCO , et peut bien évidemment utiliser tout autre algorithme et technique de calcul de similarités. Notre modèle de dépendances sémantiques entre activités se résume à deux principales étapes. La première étape consiste à annoter chaque activité d'un processus métier pour lui associer soit des termes similaires, soit un ensemble de concepts similaires, soit encore un fragment de concepts d'une ontologie. Cette annotation est ensuite exploitée pour calculer sous forme d'une matrice les différentes dépendances sémantiques qui peuvent exister entre chaque couple d'activités. Nous gardons à l'esprit qu'une telle matrice de dépendances sémantiques reflète une dimension des dépendances, et que celle-ci sera combinée avec les autres matrices calculées avec les modèles présentés dans les chapitres précédents.

Il sera ainsi possible de capitaliser sur chacun des trois modèles pour extraire des μServices fortement cohésifs et faiblement couplés.

5

Validation Expérimentale

Table des matières

5.1 Introduction 70

5.2 Implémentation de l'architecture proposée 70

5.2.1 Architecture générale du prototype 70

5.2.2 Clustering collaboratif 72

5.3 Première expérimentation : Application de location de vélos *Bicing* 74

5.3.1 Description de l'application de localtion de vélos 74

5.3.2 Matrice de dépendances de contrôle 77

5.3.3 Matrice de dépendances de données 77

5.3.4 Matrices de dépendances sémantiques 78

5.3.5 Évaluation des performances 80

5.4 Deuxième expérimentation : Application de suivi de cargaisons *Cargo* 85

5.4.1 Description de l'application de suivi de cargaisons 85

5.4.2 Dépendances de contrôle et de données 86

5.4.3 Dépendances sémantiques 89

5.4.4 Indicateurs de performance 91

5.4.5 Discussions 92

5.5 Conclusion 95

5.1 Introduction

Ce chapitre a pour objectif de démontrer la faisabilité technique de notre approche d'identification des μServices et de comparer nos résultats à ceux des approches existantes traitant de la même problématique. Nous présenterons d'abord l'architecture générale de notre prototype et décriverons succinctement l'algorithme de clustering collaboratif utilisé pour identifier les μServices. Nous présenterons ensuite les résultats obtenus à partir de deux études expérimentales menées sur une application de location de vélos et une application de gestion de cargaisons. La première application est expérimentée pour les besoins de calcul d'un certain nombre d'indicateurs de performance pour montrer l'intérêt de l'approche proposée. La deuxième application est expérimentée dans un but de comparer nos résultats à ceux de l'état de l'art.

5.2 Implémentation de l'architecture proposée

5.2.1 Architecture générale du prototype

La figure 5.1 illustre l'architecture technique de notre approche. Les modèles de processus métiers (BP) sont d'abord convertis au format *XML* à l'aide du plugin Camunda . L'architecture comprend également deux modules de base, *analyse des dépendances*, et *extraction des microservices*. Le premier module analyse un processus métier BP représenté en *XML* pour calculer les trois types de dépendances entre des couples d'activités. Il implémente ainsi les trois modèles de dépendance. Les trois stratégies du modèle sémantique (orientées termes, concepts et fragments) sont toutes implémentées. Une ontologie de domaine associée à l'application en question développée sous OWL de l'ontologie associée au BP en question et développée sous Protégé 5.5 [1] est ainsi utilisée pour les besoins du modèle sémantique. Pour calculer les différents types de dépendance entre activités, le module *analyse des dépendances*

1. protege.stanford.edu.

implémente toutes les équations et algorithmes rapportés dans les sections 3.3, 3.4, et 4. Le résultat obtenu est représenté sous forme de documents *XML*. Le deuxième module, *microservice extraction*, implémente l'algorithme *c*HAC (Section 5.2.2). Il importe d'abord l'ensemble des dépendances entre actvités à partir des fichiers XML produits par le premier module. Pour ce faire, il utilise l'outil Java DOM Parser [2], une API Java pour les modèles d'objets de documents. Ensuite, l'algorithme de clustering collaboratif *c*HAC est exécuté après avoir choisi le type d'annotation utilisée pour le modèle de dépendances sémantiques : annotation dirigée par les termes, les concepts ou les fragments. Le résultat de ce deuxième module est un ensemble de clusters (c'est-à-dire μServices).

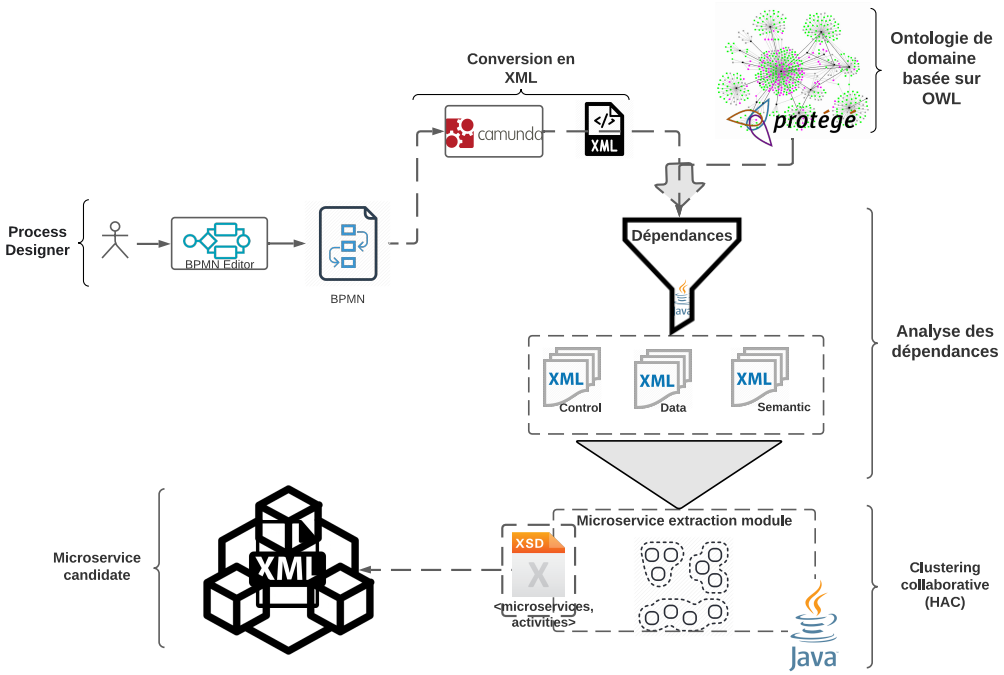


FIGURE 5.1: Architecture générale de notre approche.

2. `xerces.apache.org/xerces2-j/javadocs/xerces2/org/apache/xerces/parsers/DOMParser.html`.

5.2.2 Clustering collaboratif

Dans cette section, nous présentons succinctement les principes de l'algorithme de clustering collaboratif utilisé et proposé par El Mezouari et al. dans le cadre des travaux de thèse de Asmae El Mezouari sous la direction de D. Benslimane et A. Elfazziki. Les détails de l'algorithme sont présentés dans (20).

Le clustering est l'une des techniques bien connues de l'apprentissage automatique (Machine Learning). Étant donné un ensemble d'objets, il consiste à classer chaque objet dans un groupe spécifique appelé cluster. Dans notre travail, nous considérons chaque activité a_i d'un BP comme un objet distinct. On s'attend à ce que les activités d'un même cluster soient aussi homogènes que possible pour garantir la propriété de cohésion d'un groupe d'activités. Au contraire, les activités appartenant à différents groupes sont censées être aussi distinctes que possible pour assurer le couplage faible des groupes. Chaque groupe d'activités pourrait être un potentiel microservice.

L'algorithme de clustering *c*ollaboratif (*c*HAC) utilisé dans l'architecture proposée étend celui dit hiérarchique classique (HAC) (66). *c*HAC est effectué par N nœuds de clustering homogènes $(CN_1, CN_2, \ldots, CN_n)$ exécutant le même programme. Cependant, ces nœuds diffèrent en termes d'entrées. Chaque nœud *CN* gère une et une seule matrice de dépendances. Le nombre choisi k de groupes associés à chaque *CN* n'est pas nécessairement le même ; il peut être différent d'un *CN* à un autre.

L'algorithme *c*HAC favorise la collaboration entre les noeuds *CN*. En effet, chaque *CN* a sa propre matrice de dépendances et " garde un œil sur ce que font les autres *CN* en partageant avec eux certains scores de distance entre activités, si cela est jugé nécessaire . Ainsi, avant chaque nouvelle itération de clustering HAC, un noeud *CN* utilise à la fois sa matrice de score local (*LSM*) qui stocke les scores de distance locale entre deux activités et une matrice de score partagé (*SSM*) qui stocke un score de distance globale entre chaque couple d'activités. La distance locale est calculée à partir des données initiales d'un noeud alors que la distance

globale tient compte de l'ensemble des données dont dispose les différents noeuds à chaque itération. L'algorithme *c*HAC se déroule en trois principales phases :

- `Phase d'initialisation`. Tous les nœuds *CN* sont lancés avec leur nombre respectif de clusters k, et leurs matrices de score de dépendances locales respectives. Une matrice partagée de dépendance globale vide est également créée pour stocker le score global de dépendance des activités. Chaque activité constitue un cluster. Une variable partagée est également introduite pour synchroniser l'itération des nœuds. Une nouvelle itération est lancée sur un nœud donné si-et-seulement-si les autres nœuds ont déjà terminé leurs itérations en cours.
- `Phase d'itérations collaboratives`. Un clustering HAC classique est étendu pour le rendre collaboratif. La paire de clusters C_u et C_v la plus proche est calculée en utilisant à la fois la matrice des distances locales et la matrice des distances globales. Au niveau d'un noeud *CN*, deux clusters C_u et C_v sont fusionnés si et seulement si leur distance calculée à partir de la matrice locale est supérieure ou égale à celle calculée en utilisant la matrice globale. En cas de fusion, la matrice locale est bien entendu mise à jour.
- `Phase de sélection`. Une fois que les différents résultats de clustering sont produits par les différents *CN*, des métriques de distance leur sont appliquées pour choisir le meilleur clustering, celui qui favorise à la fois la cohésion et le couplage faible des groupes.

  Il est important de noter que l'algorithme *c*HAC peut fonctionner soit dans une stratégie de collaboration uniforme où chaque *CN* collabore avec tous les autres *CN*, soit dans une stratégie de collaboration diversifiée où chaque nœud *CN* a ses propres collaborateurs. Pour ce dernier cas, différentes matrices partagées sont nécessaires, une par nœud *CN*.

5.3 Première expérimentation : Application de location de vélos *Bicing*

5.3.1 Description de l'application de localtion de vélos

Dans notre première étude de cas, nous considérons une application *Bicing* de location de vélos dans une grande métropole. Elle comprend plus de 400 stations d'ancrage de vélos réparties à travers Barcelone et environ 6000 vélos que les utilisateurs louent moyennant des frais. Une description plus détaillée de cette application est présentée dans (35).

Pour répondre aux besoins de notre travail, nous proposons dans la figure 5.2 une représentation de haut niveau d'un processus métier du système Bicing. Nous avons eu recours au modèle et à la notation de processus métier standard (BPMN) pour illustrer cette représentation. Nous avons identifié différentes activités (par exemple (a_1) pour la demande d'un vélo et (a_9) pour le démontage d'un vélo), différents liens entre les activités via l'utilisation d'opérateurs logiques (XOR entre (a_6) et (a_{10}) et OR entre (a_2) et (a_3)). Différents artefacts sont aussi identifiés comme par exemple *vélo* et *utilisateur* et leurs attributs respectifs (par exemple, *ID* et *statut*).

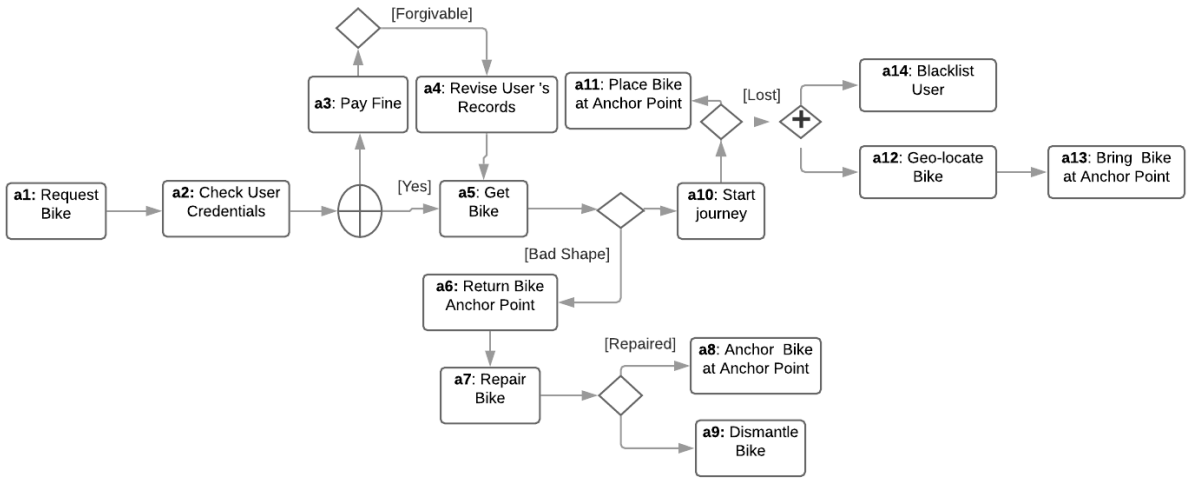


FIGURE 5.2: Modèle de processus basé sur le BPMN du système de Location de vélos.

Tout commence par l'activité (a_1) lorsqu'un utilisateur demande un vélo dans une certaine station d'ancrage. Après avoir vérifié les informations d'identification de l'utilisateur par l'activité (a_2) et des éventuels paiements tardifs (a_3), le système Bicing met à jour les enregistrements de l'utilisateur (a_4), puis approuve la demande (a_5). S'il s'avère que le vélo est défectueux, l'utilisateur le remet (a_6) et en demande éventuellement un autre. Sinon, l'utilisateur commence son parcours (a_{10}). Régulièrement, tous les vélos sont révisés (a_7) conduisant soit à les remettre en location (a_8), soit à les disposer (a_9) de côté. Lorsque l'utilisateur arrive à destination, il retourne le vélo à un certain point d'ancrage (a_{11}). Dans le cas contraire, le système Bicing met l'utilisateur sur une liste noire (a_{14}) en raison d'un retour inapproprié du vélo et géo-localise le vélo (a_{12}) pour qu'il soit collecté par les services compétents puis mis à disposition des autres utilisateurs (a_{13}).

Du point de vue des spécifications, les activités ($\{a_i\}$) peuvent nécessiter des entrées ($\{i_i\}$) et produire des sorties ($\{o_i\}$). Nous relions les entrées et les sorties aux attributs spécifiques des artefacts. Une activité agit sur à la fois les attributs et les artefacts utilisant respectivement les opérations *read*(r)/*write*(w) et *update*(u)/*create*(c)/*delete*(d). Table 5.1 présente le répertorie les activités, les artefacts, les attributs des artefacts et les opérations auxquelles les artefacts / attributs sont soumis.

Ainsi par exemple, l'activité a_5 (get bike) applique l'opération *write* sur les attributs User_Destination et Rent_Date, ce qui conduit à exécuter l'opération *create* dont le résultat est l'artéfact Rental.

La description des différentes activités est résumée dans la table 5.1. Table 5.2 décrit la criticité des artefacts/attributs nécessaires au calcul des dépendances d'activités en terme de partage de données.

TABLE 5.1: Composants du système de location de vélos

Activity	Artefacts	Attributes of artefacts
a_1	Bike (u)	Anchor_Point (r), Bike_ID (r), Bike_Status (w)
	User (u)	User_ID (r), User_Destination (r)
a_2	User (u)	User_ID (r), User_Credit (r), User_Destination (r)
a_3	User (u)	User_ID (r), User_History (r), User_Validity (r)
a_4	User (u)	User_ID (r), User_History (w)
a_5	Bike (u)	Bike_ID (r), Bike_Status (w)
	User (u)	User_ID (r), User_Status (w)
	Rental (c)	User_Destination (w), Rent_Date (w)
a_6	Bike (u)	Anchor_Point (r), Bike_ID (r), Bike_Status (w)
	User (u)	User_ID (r), User_Status (w)
	Rental (d)	Rent_ID (r)
a_7	Bike (u)	Bike_Status (w)
	Repair (c)	estimated_Repair_Cost (w), agree_Repair (w)
a_8	Bike (u)	Anchor_Point (r), Bike_Status (w)
a_9	Bike (d)	Bike_ID (r)
a_{10}	User (u)	User_ID (r), User_Status (w)
a_{11}	Bike (u)	Anchor_Point (r), Bike_ID (r), Bike_Status (w)
	User (u)	User_ID (r)
	Rental (u)	Rent_ID (r), Rent_Cost (w), User_History (w)
a_{12}	Bike (u)	Bike_ID (r), Bike_Location (w)
a_{13}	Bike (u)	Bike_ID (r), Anchor_Point (r), Bike_Status (w)
a_{14}	User (u)	User_ID (r), User_Status (w), User_History (w)

TABLE 5.2: Criticité des artefacts / attributs- Cas de l'application Bicing

Artefact	Attributes	DC^F
Bicycle	Anchor_Point	M (k'$_1$)
	Bike_Status	H (k"$_1$)
User	User_Status	H (k"$_2$)
	User_Destination	L (k_2)
	User_History	H (k"$_2$)
Rental	Rent_ID	H (k"$_3$)
	Rent_Cost	M (k'$_3$)
Repair	agree_Repair	M (k'$_4$)

Artefact	Attributes	DC^{NF}
Bicycle	Bike_ID	H (k"$_1$)
User	User_ID	H (k"$_2$)
	User_Validity	M (k'$_2$)
Repair	estimated_Repair_Cost	H (k"$_4$)

5.3.2 Matrice de dépendances de contrôle

Le modèle du processus métier décrit précédemment est utilisé pour générer automatiquement les dépendances de contrôle qui existent entre chaque couple d'activités. Le résultat de ce calcul est décrit dans la table 5.3

TABLE 5.3: Dépendances de contrôle entre les activités du processus métier Bicing

Activité / Activité	a_1	a_2	a_3	a_4	a_5	a_6	a_7
a_1		0,5	0,166	0,041	0,020	0,005	0,0026
a_2	0,5		0,25	0,062	0,031	0,007	0,003
a_3	0,166	0,25		0,25	0,125	0,031	0,015
a_4	0,041	0,062	0,25		0.5	0,125	0,062
a_5	0,020	0,031	0,125	0.5		0,25	0,125
a_6	0,005	0,007	0,031	0,125	0,25		0.5
a_7	0,0026	0,003	0,015	0,062	0,125	0.5	

5.3.3 Matrice de dépendances de données

La table 5.6 représente un extrait de *dépendances de données* pour Bicing utilisant l'équation 3.8 et s'appuyant sur les tables 5.4 et 5.5 donnant respectivement les valeurs attribuées aux opérations et celles de criticité des artefacts.

Mode	**Valeur**
R&R	0,25
R&W	0,5
W&W	1
C&R	1
C&W	1
C&D	1

TABLE 5.4: Valeurs attribuées aux opérations (Lecture (R) & Écriture (W))

Artefact	**C**	**SC**	**NC**
Bicycle	1	0,5	0,25
User	1	0,5	0,25
Rental	1	0,5	0,25
Repair	1	0,5	0,25
Destination	1	0,5	0,25
Validity	1	0,5	0,25

TABLE 5.5: Extrait de valeurs de la criticité des artefacts

TABLE 5.6: Extrait de dépendances de données - Cas de Bicing

Activité	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
a_1	0	0,203125	0	0	0,4375	0,46875	0,25	0,25	0,1875	0
a_2	0,203125	0	0	0	0	0	0	0	0	0
a_3	0	0	0	0,125	0	0	0	0	0	0
a_4	0	0	0,125	0	0	0	0	0	0	0
a_5	0,4375	0	0	0	0,25	0,5	0	0	0	0
a_6	0,46875	0	0	0	0,5	0	0	0	0	0,25
a_7	0,25	0	0	0	0	0	0	0	0	0
a_8	0,25	0	0	0	0	0	0	0	0	0
a_9	0,1875	0	0	0	0	0	0	0	0	0
a_{10}	0	0	0	0	0	0,25	0	0	0	0

5.3.4 Matrices de dépendances sémantiques

Pour les besoins de calcul des dépendances sémantiques entre noms d'activités, nous avons eu recours au système *DISCO*. Ce système est donc utilisé dans les différentes stratégies que nous avons proposées : annotation orientée termes, annotation orientée concepts, et annotation orientée fragments. *DISCO* est une méthode de calcul de la similarité de distribution entre les mots en utilisant le contexte. DISCO propose deux mesures principales de similarité $DISCO_1$ et $DISCO_2$, Nous nous intéressons dans ce travail à $DISCO_2$, qui calcule la similarité de second ordre entre deux mots d'entrée en fonction de la distribution des ensembles de mots similaires. La figure 5.3 illustre le plugin **DISCO4Protege** [3] qui fonctionne avec le **Word space** [4] de **Disco API**.

3. DISCO4Protege
4. Word-Spaces

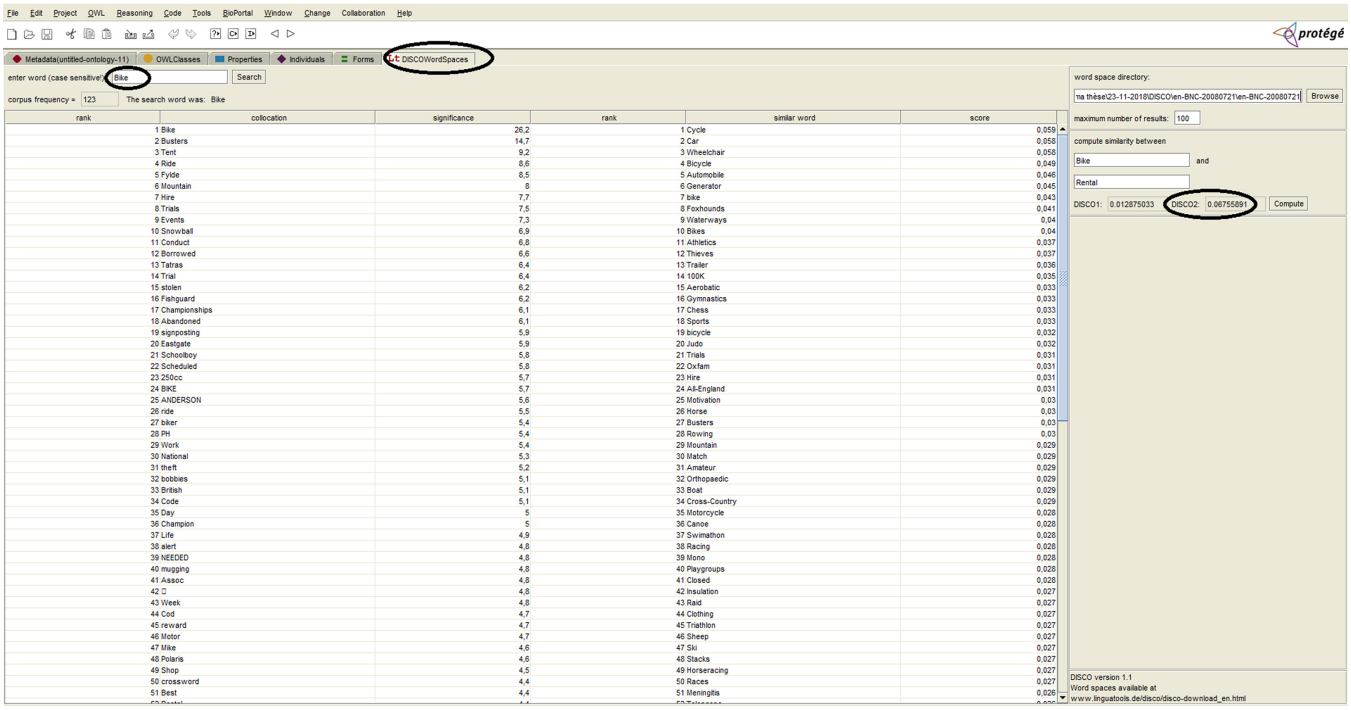


FIGURE 5.3: Plug-in Disco dans Protege

Matrice de dépendances des activités basée sur les annotations orientées termes

Le calcul de la dépendance entre deux noms d'activités en utilisant les annotations orientées termes est simple à déduire du système disco. La matrice de dépendance des activités de notre processus métier est illustrée par le tableau 5.7

TABLE 5.7: Dépendances sémantiques entre les activités utilisant les annotations orientées termes de l'application Bicing

Activité / Activité	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}
a_1	1	0.180	0.034	0.015	0.010	0.041	0.053	0.022	0.053	0.004	0.029	0.014	0.029	0.013
a_2	0.180	1	0.042	0.012	0.009	0.020	0.0472	0.0186	0.0472	0.061	0.034	0.37	0.034	0.046
a_3	0.034	0.042	1	0.002	0.001	0.266	0.1201	0.0680	0.0244	0.0680	0.004	0.002	0.04	0.0027
a_4	0.015	0.012	0.002	1	0.004	0.002	0.0089	0.0122	0.0089	0.061	0.37	0.37	0.034	0.046
a_5	0.010	0.009	0.001	0.004	1	0.059	0.0160	0.0041	0.0160	0.0076	0.024	0.024	0.0243	0.018
a_6	0.041	0.020	0.266	0.002	0.059	1	0.0408	0.0581	0.0408	0.0061	0.022	0.0047	0.0225	0.0142

Matrice de dépendances des activités basée sur les annotations orientées concepts

Dans le cas d'une annotation orientée concepts, le calcul des dépendances entre activités est moins évident. En effet, il est d'abord question de mesurer la distance sémantique qui sépare un nom d'activité à chacun des concepts de l'ontologie utilisée. La table 5.8 illustre ce calcul. Ces distances sémantiques ainsi calculées sont ensuite utilisées pour calculer la dépendance sémantique entre chaque couple d'activités. La table 5.8 présente un extrait des dépendances sémantiques entre activités en utilisant les annotations orientées concepts.

TABLE 5.8: Extrait de similarité Concept-activité en utilisant $Disco_2 - Cas\ de\ Bicing$

Concept / Activité	C_1	C_2	C_3	C_4	C_5	C_6
a_1	0.01818	0.02632	0.18076	0.04190	0.03621	0.253
a_2	0.01924	0.04481	0.99997	0.05231	0.03667	0.067
a_3	0.00421	0.01784	0.04266	0.12018	0.14423	0.00279
a_5	0.0381	0.093	0.0608	0.01833	0.0625	0.0213
a_6	0.021	0.2028	0.04131	0.0940	0.335	0.325

Matrice de dépendances des activités basée sur les annotations orientées fragments

Comme pour les concepts, DISCO est utilisé pour mesurer la distance sémantique des activités aux différents fragments de l'ontologie. La table 5.9 présente un extrait de ces distances. Ces différentes distances sont ensuite utilisées pour mesurer les dépendances sémantiques entre activités via les annotations orientées fragments. La table 5.10 illustre un extrait de ce dépendances sémantiques.

5.3.5 Évaluation des performances

Nous avons évalué notre approche d'identification automatique des microservices en utilisant les différentes matrices de dépendances à travers l'algorithme du clustering collaboratif *c*HAC et en explorant les différentes stratégies d'annotation proposées dans le modèle de dépendances sémantiques. Les performances de notre approche sont évaluées à la fois par la me-

TABLE 5.9: Extrait des valeurs de similitude entre les activités et les fragments

Fragment / Activité	F_1	F_2	F_3	F_4	F_5
a_1	0.98182	0.97368	0.9581	0.98182	0.97368
a_2	0.98076	0.95519	0.94769	0.98076	0.96333
a_3	0.99579	0.98216	0.95734	0.99579	0.98216
a_5	0.99524	0.99524	0.99038	0.99282	0.99524
a_9	0.99032	0.98608	0.9386	0.99032	0.98608
a_{10}	0.93245	0.95276	0.95919	0.95919	0.93245

TABLE 5.10: Extrait de dépendances sémantiques utilisant la technique dirigée par les fragments

Activité / Activité	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	a_{11}	a_{12}	a_{13}	a_{14}
a_1	-	0.07	0.05	0.0082	0.04	0.009	0.031	0.0421	0.065	0.032	0.04	0.025	0.05	0.032
a_2	0.07	-	0.021	0.07	0.059	0.007	0.049	0.028	0.036	0.071	0.0232	0.081	0.045	0.039
a_3	0.05	0.021	-	0.042	0.067	0.034	0.023	0.074	0.033	0.071	0.0453	0.028	0.076	0.062
a_5	0.0082	0.07	0.042	-	0.028	0.02	0.062	0.03	0.051	0.037	0.0323	0.037	0.056	0.045
a_6	0.04	0.059	0.067	0.028	-	0.025	0.07	0.057	0.067	0.069	0.045	0.021	0.012	0.023

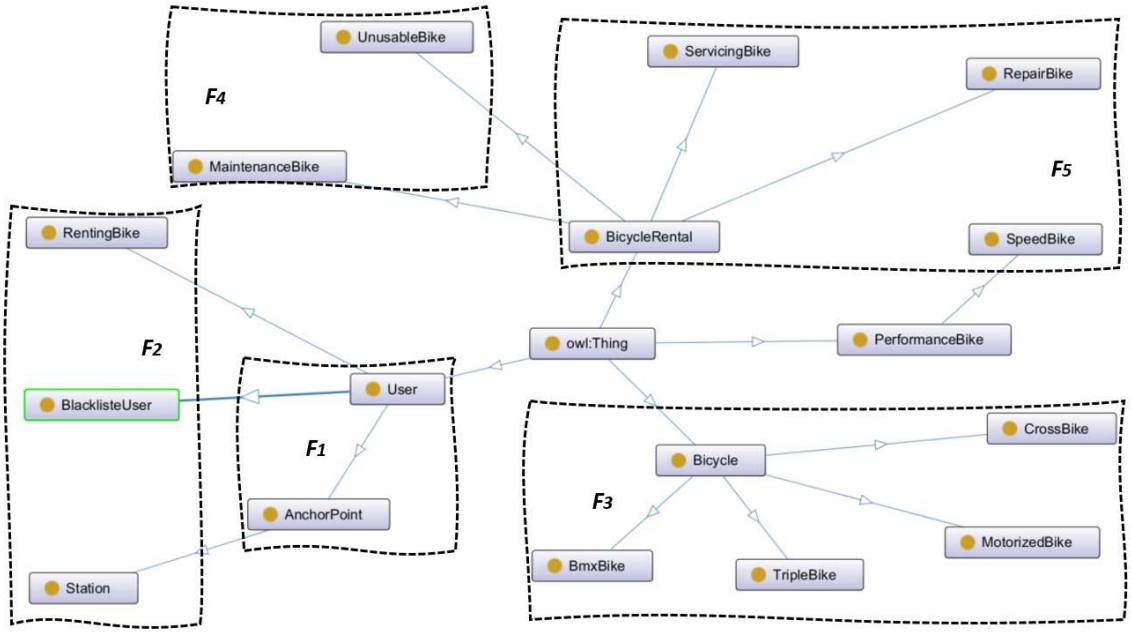


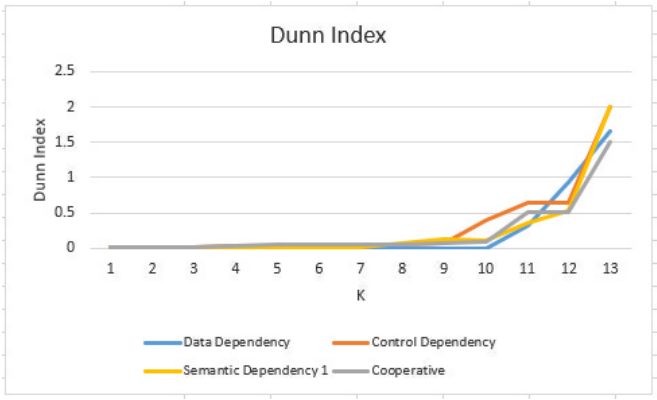
FIGURE 5.4: Ontologie de domaine pour la location de vélos et ses fragments F_i

sure de l'indice de Dunn, et par la mesure du temps de convergence de l'algorithme. L'indice de Dunn est une métrique de qualité permettant d'identifier les clusters qui sont compacts (c.-à-d., la variance est mineure entre les activités appartenant au même cluster) et séparés (c'est-à-dire la distance entre les clusters est majorée). Ainsi, un indice de Dunn plus élevé indique un meilleur regroupement.

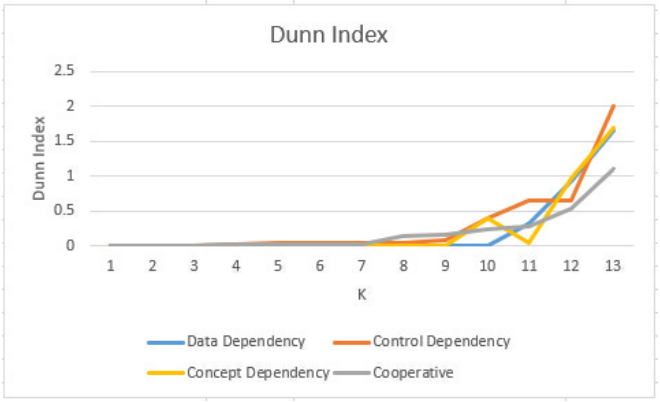
Disposant de trois types de matrices de dépendances (contrôle, données, et sémantique), l'algorithme du clustering utilise alors trois noeuds. Chaque noeud prend en charge un type de matrice. Nous avons évalué les performances de notre approche en combinant les deux premiers types de matrice d'abord avec la matrice de dépendance issue d'une annotation orientée termes, ensuite avec celle issue de l'annotation orientée concepts et enfin avec celle issue de l'annotation orientée fragments. Les différentes évaluations obtenues sont aussi comparées au cas de clustering classique basé sur l'utilisation d'un seul type de matrice. Les figures 5.5 et 5.6 illustrent les résultats obtenus pour le mesure de la qualité du clustering (Indice de Dunn) et du temps de convergence pour les différents cas précédemment exposés.

Ces résultats montrent clairement que l'indice de Dunn obtenu au niveau du noeud associé à la dépendance de contrôle est presque 10 fois meilleur que celui obtenu au niveau des 2 autres noeuds. Ainsi, le modèle de contrôle serait plus riche et fournirait plus d'informations sur les activités que les deux autres modèles. Ce constat montre aussi que l'agrégation de toutes les matrices issues des différents modèles de dépendance n'est pas nécessairement la meilleur option à choisir puisque certains modèles dégraderaient d'autres. Ceci renforce donc noter idée de recourir à un algorithme de clustering collaboratif dont l'idée principale est de permettre à noeud de n'utiliser que sa matrice de dépendance mais de garder en même temps un oeil sur ce qui se déroule dans les autres noeuds.

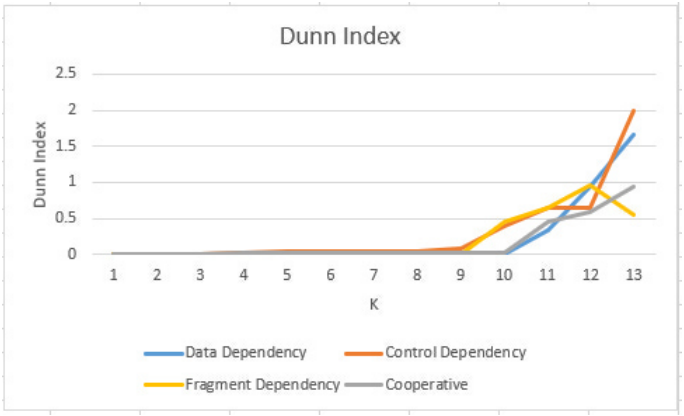
Ces résultats montrent aussi que le clustering collaboratif surpasse les 3 autres modèles utilisés séparément en terme de temps de convergence tout en conservant un indice de Dunn raisonnable (Fig. 5.5).



(a) Dépendances de contrôle, de données et sémantique orientée termes

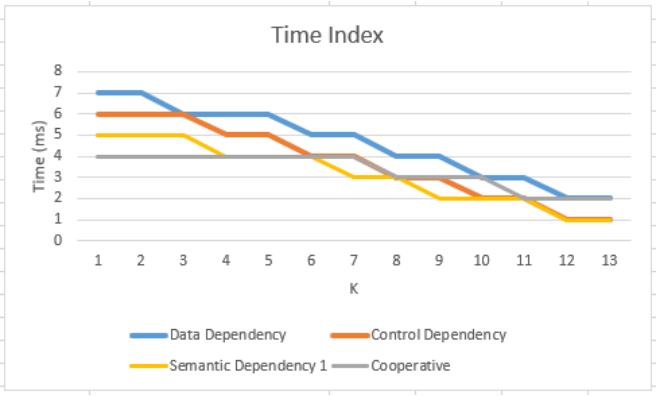


(b) Dépendances de contrôle, de données et sémantique orientée concepts

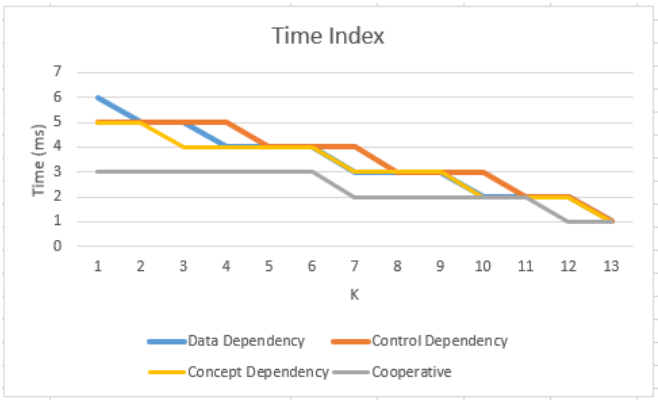


(c) Dépendances de contrôle, de données et sémantique orientée fragments

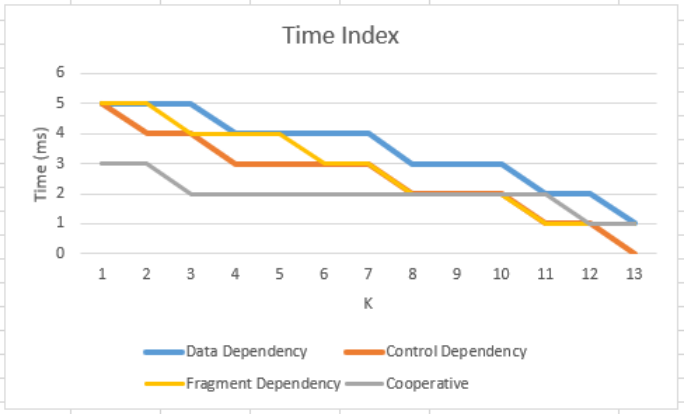
FIGURE 5.5: Mesure de l'index Dunn du clustering collaboratif - Cas de l'application Bicing



(a) Dépendances de contrôle, de données et sémantique orientée termes



(b) Dépendances de contrôle, de données et sémantique orientée concepts



(c) Dépendances de contrôle, de données et sémantique orientée fragments

FIGURE 5.6: Mesure de l'indicateur Temps du clustering collaboratif - Cas de l'application Bicing

5.4 Deuxième expérimentation : Application de suivi de cargaisons *Cargo*

5.4.1 Description de l'application de suivi de cargaisons

Cette section est dédiée à la présentation de la comparaison des performances de notre approche à celles de certaines approches qui traitent de la même problématique et présentées dans (9), (48), et (60). Ces différentes approches utilisent toutes une étude de cas du suivi des cargaisons de Gysel et al. (48).

Pour les besoins de la deuxième expérience, nous avons conçu un modèle de processus métier basé sur le BPMN pour le suivi des cargaison. Ce processus métier est représenté dans le formalisme BPMN comme illustré par la figure 5.7.

Le processus métier *Cargo* démarre lorsque la compagnie maritime expédie les conteneurs d'un client (a'_1) par voie terrestre et maritime. Après avoir vérifié les informations d'identification du client (a'_2) et la faisabilité des options d'expédition (a'_3), l'entreprise envoie une facture au client pour paiement (a'_5) ou lui notifie le rejet (a'_4). Lors de la confirmation du paiement (a'_6), des activités supplémentaires sont effectuées. Tout d'abord, l'activité (a'_7) organise l'acheminement des conteneurs par route et par mer tandis que l'activité (a'_8) suit les flux entrants et sortants des entrepôts. Ensuite, l'activité (a'_9) charge les conteneurs sur le navire tandis que l'activité (a'_{10}) suit le dédouanement des conteneurs lors des éventuelles escales. À l'arrivée de la cargaison à destination, l'activité (a'_{11}) décharge les conteneurs du navire sur des camions. Enfin, l'activité (a'_{12}) signale des éventuelles irrégularités émises par les douanes. En cas d'irrégularités, l'expéditeur des conteneurs serait passible d'amendes mettant fin au processus (a'_{13}). A l'inverse, l'activité (a'_{14}) organise l'acheminement des marchandises par la route tandis que l'activité (a'_{15} suit l'acheminement des conteneurs entre les entrepôts avant la livraison.

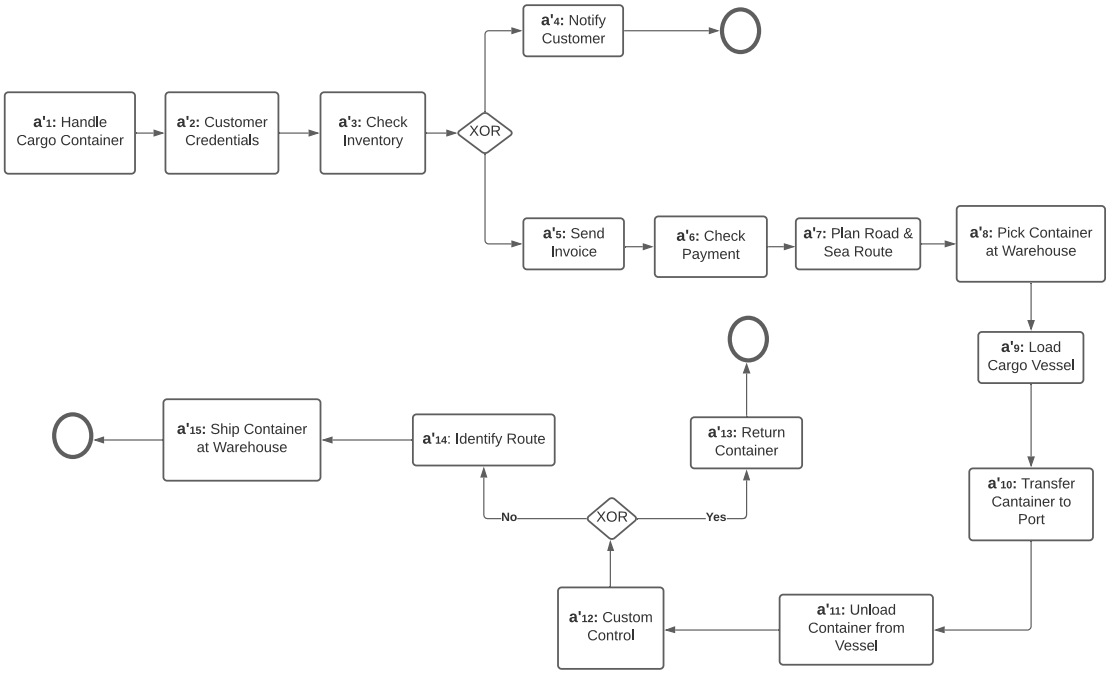


FIGURE 5.7: Modèle du processus métier pour le suivi des cargaisons

5.4.2 Dépendances de contrôle et de données

Matrice de dépendances de contrôle

La matrice de dépendances entre les activités sont extraites automatiquement à partir de la représentation *XML* du processus métier *Cargo*. Le tableau 5.11 illustre un extrait de la matrice de dépendances obtenue.

TABLE 5.11: Contrôler les dépendances avec la probabilité d'occurrence (p) fixé à 0.5

Activité / Activité	a'_1	a'_2	a'_3	a'_4	a'_5	a'_6	a'_7	a'_8	a'_9	a'_{10}	a'_{11}
a'_1	-	0.5	0.25	0.062	0.062	0.0312	0,015625	0,0078125	0,00390625	0,00195313	0,00097
a'_2	0.5	-	0.5	0.125	0.125	0.0625	0,03125	0,015625	0,0078125	0,00390625	0,00195313
a'_3	0.25	0.5	-	0.25	0.25	0.125	0,0625	0,03125	0,015625	0,0078125	0,00390625
a'_4	0.062	0.125	0.25	-	0.25	0.125	0,03125	0,0156	0,0078	0,0039	0,0019
a'_5	0.062	0.125	0.25	0.25	-	0.5	0,25	0,125	0,0625	0,03125	0,015625
a'_6	0.0312	0.0625	0.125	0.125	0.5	-	0,5	0,25	0,125	0,0625	0,03125

Matrice de dépendances de données

Comme cela a été fait pour la première expérimentation, nous relions les entrées et les sorties des activités aux attributs spécifiques des artefacts pour le suivi du cargaison. Table 5.12 répertorie les activités, les artefacts, les attributs des artefacts et les opérations auxquelles les artefacts / attributs sont soumises . Ainsi par exemple, une opération a'_{10} (transfer container to port) applique l'opération *write* à Event_ID et Destination, ce qui conduit à l'exécution de l'opération *create* dont le résultat est l'artefact Événement.

TABLE 5.12: Composants de l'application de suivi de cargaison.

Activity	Artefacts	Attributes of artefacts
a'_1	Container (u)	Container_ID (r), Container_Destination (r), Container_Status (w)
	Customer (u)	Customer_ID (r), Customer_Status (w)
a'_2	Customer (u)	Customer_ID (r), Customer_Credit (r) , Customer_Status (w)
a'_3	Storage (u)	Storage_ID (r), Check_Storage_Capacity (w)
	Container (u)	Container_ID (r), Container_Destination (r), Container_Status (w)
a'_4	Event (c)	Event_ID (w), Event_Content (w)
	Customer (u)	Customer_ID (r)
a'_5	Invoice (u)	Invoice_Validity (w)
	Customer (u)	Customer_ID (r), Customer_Notification (w)
a'_6	Payment (u)	Check_Payment_ID (r)
a'_7	Container (u)	Container_ID (r), Container_Destination (r), Container_Status (r)
a'_8	Port (u)	Port_ID (r), Port_Destination (r)
	Container (c)	Container_ID (r), Container_Status (w)
a'_9	Vessel (u)	Vessel_ID (r)
	Container (c)	Container_ID(r), Container_Destination (r)
a'_{10}	Container (u)	Container_ID (r)
	Port (u)	Port_ID (r), Location (r)
	Event (c)	Event_ID (w), Location (w)
a'_{11}	Vessel (u)	Vessel_ID (r), Vessel_Destination (r)
a'_{12}	Control (u)	Control_ID (r), Update_Control (w)
	Customer (u)	Customer_ID (r)
a'_{13}	Event (c)	Event_ID (w), Event_Type (w)
a'_{14}	Trip (u)	Trip_ID (r), Customer_Destination(r)
	Storage (u)	Storage_ID (r), Storage_UseRate (w), Storage_Destination (r)
a'_{15}	Trip (u)	Trip_ID (r), Customer_ID (r)
a'_{15}	Event (c)	Event_ID (w), Container (w), Event_Type (w)
	Truck (u)	Truck_ID (r)

TABLE 5.13: Criticité des artefacts / attributs pour le cas de suivi de cargaisons.

Artefact	Attributes	DC^F
Container	Container_Destination	M (k'$_1$)
	Container_Status	H (k"$_1$)
Customer	Customer_Credit	H (k"$_2$)
	Container_Status	L (k$_2$)
Port	Location	H (k"$_3$)
Vessel	Vessel_Destination	M (k'$_4$)

Artefact	Attributes	DC^{NF}
Event	Location	H (k"$_1$)
Storage	Storage_UseRate	H (k"$_2$)
	Storage_Destination	M (k'$_2$)
Payment	Check_Payment	H (k"$_4$)

Les tables 5.13, 5.14 et 5.15 décrivent respectivement la criticité des artefacts, le poids attribué aux opérations de lecture/écriture, et le poids de la criticité. Enfin, un extrait de la matrice de dépendances entre couple d'activités est illustrée dans la table 5.16. Ces dépendances sont calculées en utilisant l'équation 3.8.

Mode	**Valeur**
R&R	0,25
R&W	0,5
W&W	1
C&R	1
C&W	1
C&D	1

TABLE 5.14: Les valeurs attribuées aux activités pour les opérations de lecture et écriture.

Artefact	**C**	**SC**	**NC**
Container	1	0,5	0,25
Customer	1	0,5	0,25
Storage	1	0,5	0,25
Event	1	0,5	0,25
Port	1	0,5	0,25
Control	1	0,5	0,25

TABLE 5.15: Les valeurs de la criticité des artifacts.

TABLE 5.16: Extrait des *dépendances de données* - suivi de cargaisons.

Activité / Activité	a'_1	a'_2	a'_3	a'_4	a'_5	a'_6
a'_1	0	0.167	0.243	0.031	0.0342	0.218
a'_2	0.167	0	0	0.187	0.437	0
a'_3	0.243	0	0	0	0	0.218
a'_4	0.031	0.187	0	0	0.187	0
a'_5	0.342	0.437	0	0.187	0	0
a'_6	0.218	0	0.218	0	0	0

5.4.3 Dépendances sémantiques

Les tables 5.17, 5.19 et 5.21 décrivent un extrait de dépendances sémantiques pour le suivi de la cargaison à l'aide d'équations 4.3, 4.5 et 4.7, respectivement.

Matrice de dépendances activités-activités

TABLE 5.17: Dépendances sémantiques entre activités en utilisant les annotation orientées termes - cas du suivi de cargaisons.

Activité / Activité	a'_1	a'_2	a'_3	a'_4	a'_5	a'_6	a'_7	a'_8	a'_9	a'_{10}	a'_{11}	a'_{12}	a'_{13}	a'_{14}	a'_{15}
a'_1	1	0.021	0.0040	0.021	0.021	0.025	0.013	0.029	0.023	0.004	0.029	0.014	0.029	0.013	0.029
a'_2	0.021	1	0.015	0.999	0.055	0.020	0.084	0.034	0.04	0.061	0.034	0.37	0.034	0.046	0.034
a'_3	0.0040	0.015	1	0.015	0.0041	0.0057	0.029	0.002	0.48	0.004	0.002	0.04	0.0027	0.0053	0.002
a'_4	0.021	0.999	0.015	1	0.055	0.020	0.084	0.034	0.040	0.061	0.37	0.37	0.034	0.046	0.034
a'_5	0.021	0.055	0.0041	0.055	1	0.029	0.0036	0.024	0.027	0.0076	0.024	0.024	0.0243	0.018	0.024
a'_6	0.025	0.020	0.0057	0.020	0.029	1	0.0037	0.022	0.026	0.0061	0.022	0.0047	0.0225	0.0142	0.022

Matrice de dépendances Concepts-activités

TABLE 5.18: Extrait de similarité Concept-activité en utilisant $Disco_2$

Activité / Concept	C'_1	C'_2	C'_3	C'_4	C'_5	C'_6
a'_1	-	0.394576	0.397204	0.493221	0.493818	0.322241
a'_2	0.394576	-	0.326376	0.392990	0.393466	0.272318
a'_3	0.397204	0.326376	-	0.395607	0.396086	0.274131
a'_5	0.493818	0.393466	0.396086	0.491833	-	0.321335
a'_6	0.322241	0.272318	0.274131	0.320940	0.321335	-

TABLE 5.19: Dépendances sémantiques entre activités en utilisant les annotations orientées concepts - cas du suivi de cargaisons

Activité / Activité	a'_1	a'_2	a'_3	a'_4	a'_5	a'_6	a'_7	a'_8	a'_9	a'_{10}	a'_{11}	a'_{12}	a'_{13}	a'_{14}	a'_{15}
a'_1		0,466	0,498	0,466	0,495	0,497	0,48	0,489	0,49	0,48	0,489	0,484	0,489	0,496	0,4890
a'_2	0,466		0,399	0,369	0,386	0,392	0,392	0,381	0,381	0,394	0,381	0,375	0,381	0,395	0,381
a'_3	0,498	0,399		0,306	0,394	0	0,264	0,388	0,393	0,323	0,388	0,218	0,388	0,376	0,388
a'_4	0,466	0,369	0,306		0,488	0,497	0,43	0,484	0,484	0,452	0,484	0	0,483	0,470	0,484
a'_5	0,495	0,386	0,394	0,488		0,398	0,229	0,382	0,387	0,324	0,382	0,191	0,382	0,378	0,382
a'_6	0,497	0,392	0	0,497	0,398		0,358	0,394	0,398	0,343	0,394	0,366	0,394	0,390	0,389

Matrice de dépendances Fragments-activités

La figure 5.8 représente un extrait de l'ontologie de domaine pour le suivi des cargaisons avec l'ensembles des fragments de concepts.

TABLE 5.20: Extrait de similarité Fragment-activité en utilisant $Disco_2$

Fragment / Activité	F'_1	F'_2	F'_3	F'_4	F'_5	F'_6
a'_1	-	0.267	0.205	0.3	0.25	0.288
a'_2	0.363	-	0.32	0.33	0.35	0.27
a'_3	0.398	0.33	-	0.307	0.308	0.21
a'_5	0.218	0.266	0.308	0.233	-	0.335
a'_6	0.224	0.218	0.231	0.340	0.321	-

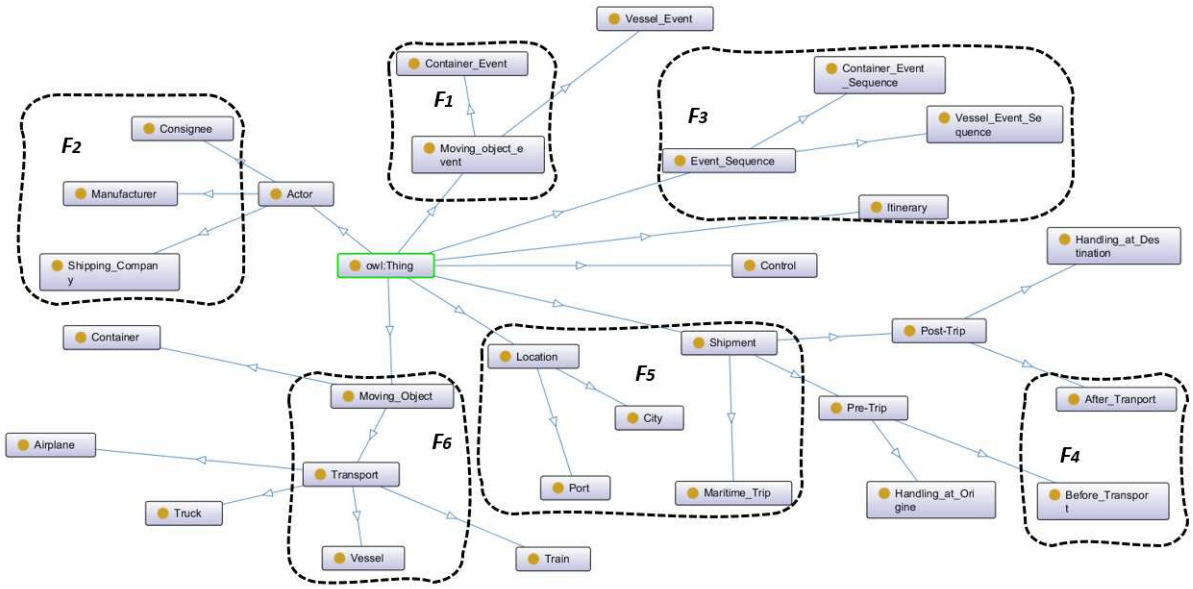


FIGURE 5.8: Extrait de l'ontologie du domaine de l'application Cargo du suivi de cargaisons adapté de (83)

5.4.4 Indicateurs de performance

Dans les applications à grande échelle, les µServices présentent de nombreux avantages par rapport aux architectures monolithiques. Cependant, la transformation apporte également de multiples défis. Ainsi, en vue de comparer les différentes approches d'identification automatiques de µServices, les critères de couplage et cohésion sont utilisés. Le couplage mesure le degré d'interdépendance des µServices. Les µServices doivent être autant que possible moins interdépendants les uns des autres. La cohésion mesure la sémantique de chaque unité constituant le système et dans quelle mesure les fonctionnalités d'un µService sont liées les unes aux autres.

Ainsi, nous nous appuyons sur le travail de (44) qui quantifie les aspects couplage et cohésion à l'aide de 4 métriques : Afférent Couplage (*AC*), Efferent Couplage (*EC*), Instabilité (*I*) et Relationnelle Cohésion (*RC*). La table 5.22 définit ces 4 métriques. Les métriques définies dans (44) font référence à des classes et à des packages de classes, nous les avons alors mappées sur des concepts pertinents et applicables à notre travail. Par conséquent, les termes

TABLE 5.21: Dépendances sémantiques entre les activités en utilisant les annotations orientées fragments cas du suivi de cargaisons

Activité / Activité	a'_1	a'_2	a'_3	a'_4	a'_5	a'_6	a'_7	a'_8	a'_9	a'_{10}	a'_{11}	a'_{12}	a'_{13}	a'_{14}	a'_{15}
a'_1		0.421	0.440	0.321	0.381	0.325	0.413	0.329	0.323	0.204	0.229	0.214	0.329	0.313	0.229
a'_2	0.321		0.315	0.299	0.355	0.220	0.284	0.334	0.204	0.261	0.234	0.307	0.234	0.246	0.234
a'_3	0.140	0.215		0.125	0.141	0.257	0.129	0.202	0.248	0.204	0.2	0.34	0.207	0.180	0.102
a'_4	0.21	0.99	0.25		0.55	0.22	0.23	0.227	0.45	0.268	0.73	0.27	0.23	0.32	0.25
a'_5	0.31	0.21	0.32	0.25		0.72	0.24	0.43	0.21	0.54	0.27	0.34	0.43	0.34	0.34
a'_6	0.34	0.20	0.37	0.50	0.39		0.37	0.42	0.34	0.21	0.52	0.57	0.36	0.36	0.29

TABLE 5.22: Mesures de performance d'identification de microservices extraites de (44)

Aspect	Metrics	Definition
Couplage	Couplage afférent (*AC*)	Mesure le nombre de classes dans d'autres packages (services) qui dépendent des classes dans le package (service) lui-même, en tant que tel, il indique la responsabilité du package (service)
((61))	Couplage efférent (*EC*)	Mesure le nombre de classes dans d'autres packages (services), dont dépendent les classes d'un package (service), indique ainsi sa dépendance vis-à-vis des autres
	Instabilité (*I*)	Mesure la résilience d'un package (service) au changement et est calculé comme suit : $\frac{EC}{EC+AC}$. $I = 0$ indique un package (service) complètement stable alors que $I = 1$ un package (service) complètement instable
Cohésion	Cohésion relationnelle (*RC*)	Mesure le rapport entre le nombre de relations internes et le nombre de types dans un package (service). Les relations internes incluent l'héritage entre les classes, l'appel de méthodes, l'accès aux attributs de classe et des références explicites comme la création d'une instance de classe. Un nombre plus élevé de RC indique une plus grande cohésion d'un package (service).
((58))		

classe et *package* correspondent respectivement à *activité* et *cluster*, *service* fait référence à *microservice*, et *relations internes* entre les classes dans le même paquet correspond soit à *type connecteur*, soit à *transfert de données*, ou encore à *similarité sémantique* entre les activités d'un même cluster.

5.4.5 Discussions

Les critères définis précédemment sont donc utilisés pour comparer les performances de notre approche aux performances de certaines approches de l'état de l'art qui utilisent la même étude de cas (suivi de cargaisons).

La table 5.24 présente les résultats obtenus pour le calcul des 4 métriques pour notre approche. Les résultats obtenus pour le calcul des 4 métriques pour les approches de l'état de

l'art sont présentés dans la table 5.23.

TABLE 5.23: Métriques des μServices candidats par approche de l'état de l'art - cas du suivi de cargaisons.

	Métrique / Microservice	*AC*	*EC*	*I*	*RC*	
(48)	*Location*	14	1	0.1	21.5	
	Tracking	11	3	0.2	16.7	
	Voyage&Planning	15	4	0.2	16.7	
		13.3	2.7	0.2	14.2	*Avg*
(9)	*Planning*	16	2	0.1	20.3	
	Product	13	4	0.2	1.8	
	Tracking	15	5	0.3	14.9	
	Trip	8	2	0.2	0	
		13	3.3	0.2	9.3	*Avg*
(60)	*Cargo*	13	4	0.2	1.8	
	Planning	10	3	0.2	11.5	
	Location	15	1	0.1	21.5	
	Tracking	16	5	0.2	14.1	
		13.5	3.3	0.2	12.2	*Avg*

TABLE 5.24: Métriques des μServices candidats en utilisant notre approche - Cas de suivi de cargaison.

Metric / Microservice	*AC*	*EC*	*I*	*RC*	
Preparation	23	0	0	14	
Handling	18	3	0.14	11	
Planning&Tracking	16	2	0.11	23	
Delivery	12	8	0.4	5	
Return	7	5	0.4	-	
	15.2	3.6	0.21	13.25	*Avg*

Le tableau 5.24 montre que notre approche donne cinq μServices candidats qui sont par conséquents plus fins par rapport aux approches de l'état de l'art qui en identifient soit trois soit quatre μServices. De plus, nous observons que les μServices de notre approche sont plus fortement cohésifs par rapport aux approches de Baresi et al. et Li et al. et moins faiblement couplés par rapport l'approche de Gysel et al. Cela démontre que notre approche donne de meilleurs résultats en considérant à la fois les métriques *AC* et *RC* par rapport aux trois autres approches. De plus, il n'y a pas de réelle distinction pour la métrique *I* en moyenne dans

TABLE 5.25: Métriques du candidat μServices utilisant notre approche - Cas de Bicing

	Microservice	Metric / Activities	*AC*	*EC*	*I*	*RC*	
word-driven	*RequestHandling*	a_1, a_2	13	0	0	2	
	Revision&Validation	a_3, a_4, a_5	18	1	0.05	5	
	BikeAbandon	a_{12}, a_{13}, a_{14}	11	5	0.31	6	
	BikeReturn	a_{10}, a_{11}	12	5	0.29	3	
	BikeReparation	a_6, a_7, a_8, a_9	23	3	0.12	5	
			15.4	2.8	0.154	4	*Avg*
concept-driven	*ms*#1	a_1, a_2, a_3, a_4, a_5	25	0	0	14	
	ms#2	a_{13}, a_{14}	14	1	0.07	2	
	ms#3	a_{11}, a_{12}	14	1	0.07	2	
	ms#4	a_9, a_{10}	14	1	0.07	2	
	ms#5	a_8, a_7, a_6	21	1	0.05	3	
			17.6	0.8	0.052	5	*Avg*

toutes les approches. En ce qui concerne la métrique *RC*, notre approche fonctionne mieux que les approches de Li et et de Baresi et al, tout en étant moins attrayante que l'approche de Gysel et al. Ceci peut être expliqué comme suit. L'approche de Gysel et al. repose beaucoup sur les critères prédéfinis pour déterminer les caractéristiques de chaque entité, et elle utilise également un graphe pondéré non orienté pour la décomposition. Cependant, le poids des arcs du graphe est également généré de manière subjective, ce qui peut conduire à une description inexacte des relations entre les services.

Nous avons aussi calculé les mêmes métriques *AC*, *EC*, *IC* et *RC* pour le cas de la location de vélos (première expérimentation). Elles sont illustrées dans la table 5.25. Ces résultent confortent ceux obtenus dans le cas du suivi de cargaisons. Nous pouvons aussi noter que l'approche de Li et al minimise la dépendance à la décision du concepteur car elle est basée sur un diagramme de flux de données qui décrit strictement les logiques réelles et le flux de données des systèmes. L'approche de Gysel et al. nécessite une spécification détaillée et exhaustive du système, ainsi que des artefacts de spécification ad-hoc et subjectifs associés aux critères de couplage.

5.5 Conclusion

Nous avons présenté dans ce chapitre nos résultats expérimentaux menés dans le cadre de deux applications. Le prototype développé démontre la faisabilité technique de l'approche proposée. Celle-ci étant multi-modèles, l'algorithme du clustering collaboratif trouve tout son sens.

Les résultats expérimentaux sont très encourageants, ils sont à la hauteur des approches de l'état de l'art. Les expérimentations réalisées ont concerné un et un seul processus métier à la fois. Il serait certainement important de trouver des études de cas concrètes basées sur un ensemble de processus métiers. Ceci permettra de renforcer les différentes dépendances entre activités et donnerait encore probablement plus d'importance à l'algorithme du clustering collaboratif.

6

Conclusion et Perspectives

6.1 Conclusion

- **Résumé des contributions**. L'identification de μServices reste un obstacle important lors de la migration des applications existantes vers ce style architectural. À l'inverse des approches académiques et industrielles existantes, nous avons adopté les processus métier comme source principale d'information offrant une vue complète de ce qui se passe dans les organisations en termes de qui fait quoi, quand, où et pourquoi. Grâce à cette vue globale, nous avons conçu et démontré une approche multi-modèles pour l'identification de μServices. Trois modèles ont été identifiés. Le premier, appelé modèle de dépendances de contrôle, représente les liens d'exécution entre les activités. Un tel modèle permet d'identifier les activités à regrouper ensemble pour renforcer le couplage faible des μServices. Le second modèle, appelé modèle de dépendance de données, représente les liens entre les activités par rapport à la dimension partage de données. Il permet de regrouper les activités qui nécessitent l'accès à des données communes. Le troisième modèle, appelé modèle de dépendances sémantiques, représente les activités dont les noms présentent des similarités du point de vue métier. Il permet de regrouper les activités qui relèvent d'un même champ d'application (domaine). Ces différents modèles sont utilisés sur un ou plusieurs processus métiers pour générer des matrices de dépendances entre couples d'activités. Ces dernières sont ensuite utilisées par un algorithme de clustering collaboratif pour calculer les différents clusters d'activités. Chaque cluster d'activités correspond à un μService. Les différents modèles que nous avons proposés sont indépendants des algorithmes de clustring. Ils peuvent aussi être utilisés avec des méthodes de clustering classique.

  Notre approche d'identification de μServices favorise le découplage et la cohésion des futurs μServices et prend également en compte dans une certaine mesure les exigences non fonctionnelles via le modèle de dépendances de données. Les résultats sont prometteurs et montrent que dans certains cas les résultats obtenus surclassent certains résultats

de l'état de l'art.

Étant donné que notre approche repose assez fortement sur l'ontologie de domaine via l'utilisation du modèle de dépendances sémantiques, il est essentiel de garantir la qualité de l'ontologie (par exemple, l'exhaustivité et la précision de son contenu) pour une meilleur qualité des résultats de l'identification des μServices. Pour faire face à un éventuel manque de qualité de l'ontologie, des facteurs supplémentaires tels que le contexte d'utilisation des données peuvent être incorporés dans notre analyse pour une meilleure identification des μServices.

- **Limites de notre approche**. En termes de limites, les analystes doivent examiner attentivement et minutieusement chaque activité soit de manière manuelle, soit à l'aide d'outils automatiques, afin de s'assurer qu'ils aient un point de vue cohérent et que la décomposition soit correctement équilibrée. Outre les relations entre le flux de contrôle et les flux de données, d'autres facteurs peuvent avoir une incidence sur la question de savoir si les activités doivent être fusionnées en un seul μService, notamment la fréquence de communication entre les activités et le temps d'exécution des activités. Cependant, ces types de facteurs ne peuvent être collectés qu'en exécutant le système pendant un certain temps. En outre, peu d'exigences non fonctionnelles sont clairement revendiquées dans cette étude, à l'exception de la réutilisabilité. Pour répondre à cette limite, un modèle purement basé sur les dépendances non fonctionnelles pourrait être rajouté aux trois premiers modèles pour intégrer des facteurs tels que la performance et la sécurité.
- **Limites non spécifiques à notre approche**. Premièrement, il n'existe pas de métriques et de méthodes (de simulation) complètes et bien connues pour soutenir l'évaluation des résultats de la méthode d'identification des μService. À notre connaissance, Sonargraph Architect est un outil prometteur à cet effet. Cependant, il y a aussi des aspects qui peuvent affecter la validité de l'évaluation. Par exemple, les dépendances entre les clus-

ters peuvent ne pas être en corrélation 1 : 1 avec les interfaces à définir entre les services. Deuxièmement, nous manquons de cas de μServices appropriés comme points de référence pour exécuter des expériences et comparer les résultats. Bien qu'une étude de cas industrielle dans une grande organisation soit importante pour la validation d'une approche, la décomposition et l'évaluation de ces systèmes prennent du temps et sont coûteuses. Dans ces circonstances, un vaste ensemble de données open source de μServices peut servir pour la recherche actuelle et future dans le domaine.

6.2 Perspectives

En termes de travaux futurs, nous aimerions explorer quelques idées de recherche qui pourraient enrichir la solution que nous avons proposée.

- **Définition de nouveaux modèles**. L'architecture multi-modèles proposée est flexible étant capable d'intégrer de nouveaux modèles de dépendances. Ainsi, un nouveau modèle orienté exigences fonctionnelles permettrait d'identifier des μServices en tenant en compte des aspects non fonctionnels de la future application telles que la sécurité et les performances.
- **Gestion des préférences par rapport à l'importance des modèles**. Il serait intéressant d'accorder aux différents modèles de dépendances des poids différents de sorte à tenir compte des préférences des utilisateurs. En effet, certains utilisateurs pourraient insister sur l'importance des aspects non fonctionnels quand d'autres pourraient plutôt préférer maximiser le partage de données et donc favoriser le découplage des μServices. La gestion de telles préférences pourrait alors être prise en compte soit directement dans le matrices de dépendances soit indirectement dans les algorithmes de clustering.
- **Utilisation des techniques de traitement des langages naturels**. Le modèle sémantique proposé consiste à comparer les noms d'activités et à mesurer leurs degrés de

similarité. Un tel modèle pourrait être enrichi en permettant aux utilisateurs d'annoter leurs processus métiers sous forme de phrases exprimées en langage naturel. Il sera ensuite question d'explorer et d'utiliser les techniques d'apprentissage automatique et de traitement des langues naturelles pour déduire des relations de dépendance entre les activités.

- **Exploration et exploitation du code source des activités**. Notre approche est basée sur les activités étant définies uniquement par leurs noms, simples ou composés. La définition des activités peut être enrichie en explorant les codes sources auxquels elles s'y réfèrent. En effet, les différents codes sources peuvent révéler des dépendances entre entités difficiles à identifier à partir des noms d'entités. Les nouvelles dépendances ne pourront qu'enrichir les différents modèles proposés.

Bibliographie

[1] Ahmadvand, M. and Ibrahim, A. (2016). Requirements reconciliation for scalable and secure microservice (de)composition. In *IEEE International Requirements Engineering Conference (RE)*, pages 68–73. IEEE Computer Society.

[Amazon] Amazon, I. Services et produits de cloud amazon | AWS. `https://aws.amazon.com/fr/`.

[3] Amiri, M. J. (2018). Object-aware identification of microservices. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 253–256. IEEE.

[4] Andritsos, P. and Tzerpos, V. (2005). Information-theoretic software clustering. *IEEE Transactions on Software Engineering*, 31(2) :150–165.

[Atlassian] Atlassian. Bitbucket | the git solution for professional teams. `https://bitbucket.org/product`.

[Azure] Azure, M. Services de cloud computing | microsoft azure. `https://azure.microsoft.com/fr-fr/`.

[7] Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2016). Microservices architecture enables devops : Migration to a cloud-native architecture. *Ieee Software*, 33(3) :42–52.

[8] Baresi, L. and Garriga, M. (2020). Microservices : The evolution and extinction of web services ? In *Microservices*, pages 3–28. Springer.

[9] Baresi, L., Garriga, M., and De Renzis, A. (2017). Microservices identification through interface analysis. In *European Conference on Service-Oriented and Cloud Computing*, pages 19–33. Springer.

[10] Bucchiarone, A., Soysal, K., and Guidi, C. (2019). A model-driven approach towards automatic migration to microservices. In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 15–36. Springer.

[11] Butzin, B., Golatowski, F., and Timmermann, D. (2016). Microservices approach for the internet of things. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–6. IEEE.

[Center] Center, S. Netflix open source software center. `https://netflix.github.io/`.

[13] Cerny, T., Donahoo, M. J., and Pechanec, J. (2017). Disambiguation and comparison of soa, microservices and self-contained systems. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pages 228–235.

[14] Chatterjee, M., Das, S. K., and Turgut, D. (2002). Wca : A weighted clustering algorithm for mobile ad hoc networks. *Cluster computing*, 5(2) :193–204.

[15] Chen, L. (2018). Microservices : Architecting for continuous delivery and devops. In *2018 IEEE International Conference on Software Architecture (ICSA)*, pages 39–397. IEEE.

[16] Chen, R., Li, S., and Li, Z. (2017). From monolith to microservices : a dataflow-driven approach. In *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pages 466–475. IEEE.

[17] Chris, R. (2018). Pattern : Microservice architecture. *Internet].[Citerad 7 febr., 2018]. Tillgänglig från : http ://microservices. io/patterns/microservices. html*.

[18] Cojocaru, M.-D., Oprescu, A., and Uta, A. (2019). Attributes assessing the quality of microservices automatically decomposed from monolithic applications. In *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 84–93. IEEE.

[19] Cornuéjols, A., Wemmert, C., Gançarski, P., and Bennani, Y. (2018). Collaborative clustering : Why, when, what and how. *Information Fusion*, 39 :81–95.

[20] Daoud, M., El Mezouari, A., Faci, N., Benslimane, D., Maamar, Z., and Elfazziki, A. (2021). A multi-model based microservices identiftcation approach. *Journal of Systems Architecture*.

[21] Davenport, T. H. and Short, J. E. (1990). The new industrial engineering : information technology and business process redesign.

[22] Desai, P. R. (2016). A survey of performance comparison between virtual machines and containers. *Int. J. Comput. Sci. Eng*, 4(7) :55–59.

[Design] Design, G. . `https://github.com/goadesign/goa/`.

[24] Diepenbrock, A., Rademacher, F., and Sachweh, S. (2017). An ontology-based approach for domain-driven design of microservice architectures. *INFORMATIK 2017*.

[25] Djogic, E., Ribic, S., and Donko, D. (2018a). Monolithic to microservices redesign of event driven integration platform. In *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1411–1414. IEEE.

[26] Djogic, E., Ribic, S., and Donko, D. (2018b). Monolithic to microservices redesign of event driven integration platform. In *41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018, Opatija, Croatia, May 21-25, 2018*, pages 1411–1414.

[27] DOCKER, D. (2017). Disponível :< https ://www. docker. com>. *Acesso em*, 11.

[28] Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., and Safina, L. (2017). Microservices : How to make your application scale. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 95–104. Springer.

[Drone.io] Drone.io. Automate software testing and delivery. `https://www.drone.io/`.

[Dropwizard] Dropwizard. Java framework. `https://www.dropwizard.io/en/latest/`.

[Ellis] Ellis, A. Serverless functions made simple with kubernetess. `https://www.openfaas.com/`.

[Enterprise] Enterprise, A. Spinnaker. `https://spinnaker.io/`.

[33] Escobar, D., Cárdenas, D., Amarillo, R., Castro, E., Garcés, K., Parra, C., and Casallas, R. (2016a). Towards the understanding and evolution of monolithic applications as microservices. In *2016 XLII Latin American Computing Conference (CLEI)*, pages 1–11. IEEE.

[34] Escobar, D., Cardenas, D., Amarillo, R., Castro, E., Garcés, K., Parra, C., and Casallas, R. (2016b). Towards the understanding and evolution of monolithic applications as microservices. In *XLII Latin American Computing Conference, CLEI 2016, Valparaíso, Chile, October 10-14, 2016*, pages 1–11.

[35] Estañol, M. et al. (2016). *Artifact-centric business process models in UML : specification and reasoning*. PhD thesis, Universitat Politècnica de Catalunya.

[36] Forsgren, N. (2015). 2015 state of devops report.

[Fowler] Fowler, M. refactoring. com, 2015. *URl : https ://refactoring. com*.

[38] Fowler, M. (2015a). How to break monolith into microservices. *línea]. Available : https ://martinfowler. com/articles/break-monolith-into-microservices. html.[Último acceso : 2 March 2019]*.

[39] Fowler, M. (2015b). Microservice trade-offs. *Dosegljivo : https ://martinfowler. com/articles/microservice-trade-offs. html*.

[40] Fowler, M. and Lewis, J. (2014). Microservices a definition of this new architectural term. *URL : http ://martinfowler. com/articles/microservices. html*, page 22.

[Fox] Fox, T. Jvm framework. `https://vertx.io/`.

[framework] framework, S. Kubeless. `https://kubeless.io/`.

[Framework] Framework, S. Seneca, a microservices toolkit for node.js. `https://senecajs.org/`.

[44] Fritzsch, J., Bogner, J., Zimmermann, A., and Wagner, S. (2018). From monolith to microservices : a classification of refactoring approaches. In *International Workshop on Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 128–141. Springer.

[45] Furda, A., Fidge, C., Zimmermann, O., Kelly, W., and Barros, A. (2017). Migrating enterprise legacy source code to microservices : on multitenancy, statefulness, and data consistency. *IEEE Software*, 35(3) :63–72.

[Google] Google. Production-grade container orchestration. `https://kubernetes.io/`.

[47] Götz, B., Schel, D., Bauer, D., Henkel, C., Einberger, P., and Bauernhansl, T. (2018). Challenges of production microservices. *Procedia CIRP*, 67 :167–172.

[48] Gysel, M., Kölbener, L., Giersche, W., and Zimmermann, O. (2016). Service cutter : A systematic approach to service decomposition. In *European Conference on Service-Oriented and Cloud Computing*, pages 185–200. Springer.

[49] Hammouda, K. and Kamel, M. (2006). Collaborative document clustering. In *SIAM International Conference on Data Mining*.

[50] Heinrich, R., van Hoorn, A., Knoche, H., Li, F., Lwakatare, L. E., Pahl, C., Schulte, S., and Wettinger, J. (2017). Performance engineering for microservices : research challenges and directions. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, pages 223–226.

[51] Jin, W., Liu, T., Cai, Y., Kazman, R., Mo, R., and Zheng, Q. (2019). Service candidate identification from monolithic systems based on execution traces. *IEEE Transactions on Software Engineering*.

[52] Jin, W., Liu, T., Zheng, Q., Cui, D., and Cai, Y. (2018). Functionality-oriented microservice extraction based on execution trace clustering. In *2018 IEEE International Conference on Web Services (ICWS)*, pages 211–218. IEEE.

[53] Kalske, M., Mäkitalo, N., and Mikkonen, T. (2017). Challenges when moving from monolith to microservice architecture. In *International Conference on Web Engineering*, pages 32–47. Springer.

[54] Knoche, H. and Hasselbring, W. (2018a). Using microservices for legacy software modernization. *IEEE Software*, 35(3) :44–49.

[55] Knoche, H. and Hasselbring, W. (2018b). Using microservices for legacy software modernization. *IEEE Softw.*, 35(3) :44–49.

[56] Kolb, P. (2008). Disco : A multilingual database of distributionally similar words. *Proceedings of KONVENS-2008, Berlin*, 156.

[57] Koschmider, A. (2017). Microservices-based business process model execution. In *RADAR+ EMISA@ CAiSE*, pages 158–161.

[58] Larman, C. (2012). *Applying UML and patterns : an introduction to object oriented analysis and design and interative development*. Pearson Education India.

[59] Levcovitz, A., Terra, R., and Valente, M. T. (2016). Towards a technique for extracting microservices from monolithic enterprise systems. *arXiv preprint arXiv :1605.03175*.

[60] Li, S., Zhang, H., Jia, Z., Li, Z., Zhang, C., Li, J., Gao, Q., Ge, J., and Shan, Z. (2019). A dataflow-driven approach to identifying microservices from monolithic applications. *Journal of Systems and Software*, 157.

[61] Martin, R. C. (2002). *Agile software development : principles, patterns, and practices*. Prentice Hall.

[Martin Fowler] Martin Fowler. GOTO 2014 • microservices. `https://www.youtube.com/watch?v=wgdBVIX9ifA&ab_channel=GOTOConferences`.

[63] Mazlami, G., Cito, J., and Leitner, P. (2017). Extraction of microservices from monolithic software architectures. In *2017 IEEE International Conference on Web Services (ICWS)*, pages 524–531. IEEE.

[microservice shell] microservice shell, T. Fuge, an execution environment for micro service development with node.js. `https://fuge.io/`.

[65] Miell, I. and Sayers, A. H. (2016). *Docker in practice*. Manning Publications.

[66] Murtagh, F. and Legendre, P. (2011). Ward's hierarchical clustering method : clustering criterion and agglomerative algorithm. *arXiv preprint arXiv :1111.6285*.

[67] Nadareishvili, I., Mitra, R., McLarty, M., and Amundsen, M. (2016). *Microservice architecture : aligning principles, practices, and culture*. " O'Reilly Media, Inc.".

[68] Newman, S. (2015). *Building microservices : designing fine-grained systems*. " O'Reilly Media, Inc.".

[69] O'Connor, R. V., Elger, P., and Clarke, P. M. (2017). Continuous software engineering—a microservices architecture perspective. *Journal of Software : Evolution and Process*, 29(11) :e1866.

[Odersky] Odersky, M. Lagom - microservices framework. `https://www.lagomframework.com/`.

[71] Paulsen, C., J.M., B., Bartol, and Winkler, N. (2018). Criticality analysis process model. Technical report.

[72] Perez de Prado, R., García-Galán, S., Muñoz-Expósito, J. E., Marchewka, A., and Ruiz-Reyes, N. (2020). Smart containers schedulers for microservices provision in cloud-fog-iot networks. challenges and opportunities. *Sensors*, 20(6) :1714.

[73] Richards, M. (2015). Microservices vs. service-oriented architecture.

[74] Richardson, C. (2017). Pattern : microservice architecture. *URL : http ://microservices. io/patterns/microservices. html*.

[75] Salvadori, I., Huf, A., Mello, R. d. S., and Siqueira, F. (2016). Publishing linked data through semantic microservices composition. In *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, pages 443–452. ACM.

[76] Santis, S. D., Florez, L., Nguyen, D. V., and Rosa, E. (2016). *Evolve the Monolith to Microservices with Java and Node*. IBM RedBooks.

[77] Savchenko, D. I., Radchenko, G. I., and Taipale, O. (2015). Microservices validation : Mjolnirr platform case study. In *2015 38th International convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 235–240. IEEE.

[78] Scott, J. (2017). A practical guide to microservices and containers mastering the cloud, data, and digital transformation.

[79] Selim, M. (2016). *An Empirical Analysis on the Microservices Architecture Pattern and Service-Oriented Architecture*. PhD thesis.

[80] Selmadji, A. (2019). *From monolithic architectural style to microservice one : structure-based and task-based approaches*. PhD thesis, Université Montpellier.

[81] Singh, V. and Peddoju, S. K. (2017). Container-based microservice architecture for cloud applications. In *2017 International Conference on Computing, Communication and Automation (ICCCA)*, pages 847–852. IEEE.

[82] Taibi, D., Lenarduzzi, V., and Pahl, C. (2017). Processes, motivations, and issues for migrating to microservices architectures : An empirical investigation. *IEEE Cloud Computing*, 4(5) :22–32.

[83] Villa, P., Camossi, E.", e. C., Levashkin, S., and Bertolotto, M. (2011). A description logic approach to discover suspicious itineraries from maritime container trajectories. In *GeoSpatial Semantics*, pages 182–199. Springer.

[VMware] VMware, I. Spring | microservices. `https://spring.io/microservices`.

[85] Weske, M. (2019). *Business Process Management : Concepts, Languages, Architectures*. Springer.

[86] Wilkin, G. and H., X. (2008). A practical comparison of two k-means clustering algorithms. *BMC bioinformatics*, 9.

[87] Wizenty, P., Sorgalla, J., Rademacher, F., and Sachweh, S. (2017). Magma : Build management-based generation of microservice infrastructures. In *Proceedings of the 11th European Conference on Software Architecture : Companion Proceedings*, pages 61–65.

[88] Wolff, E. (2016). *Microservices : flexible software architecture*. Addison-Wesley Professional.

[89] Wu, Z. and Palmer, M. (1994). Verb semantics and lexical selection. *arXiv preprint cmp-lg/9406033*.

[90] Zdun, U., Navarro, E., and Leymann, F. (2017). Ensuring and assessing architecture conformance to microservice decomposition patterns. In *International Conference on Service-Oriented Computing*, pages 411–429. Springer.

Liste des publications

Journal

1. Daoud, M., El Mezouari, A., Faci, N., Benslimane, D., Maamar, Z., and Fazziki, A. (2021). A multi-model based microservices identiftcation approach. *Journal of Systems Architecture.*

Conférence

1. Daoud, M., El Mezouari, A., Faci, N., Benslimane, D., Maamar, Z., and Fazziki, A. (2020). Automatic Microservices Identification from a Set of Business Processes. In : *International Conference on Smart Applications and Data Analysis. Springer, Cham, pages 299-315.*
2. Daoud, M., El Mezouari, A., Faci, N., Benslimane, D., Maamar, Z., and Fazziki, A. (2020). Towards an Automatic Identification of Microservices from Business Processes. In : *2020 IEEE 29th International Conference on Enabling Technologies : Infrastructure for Collaborative Enterprises (WETICE), pages 42-47.*