REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE YAHIA FARES DE MEDEA



Faculté des **S**ciences

Département du **M**athématique et **I**nformatique

Mémoire de fin d'études de Master

Filière : Informatique

Spécialité : Ingénierie des Systèmes et Technologie du web

Proposition d'une approche pour la génération semi-automatique des requêtes GraphQL pour la composition des services web.

Présentés par :

Benlakehal Yasmine

Belkacem Fethia

Proposé et Dirigé par :

TOUBAL El Bahi

Promotion 2022

Remerciement

Nous remercions tout d'abord, Allah qui nous a donné la force et le courage afin de parvenir à élaborer ce modeste travail.

Pour tous les bienfaits et les faveurs qu'il nous a attribuées et sans lesquels nous ne serons pas arrivés là où l'on est.

Nous tenons aussi à remercier : Mr. Toubal El Bahi, notre encadreur, pour ses conseils, ses orientations, sa disponibilité et son encouragement qui nous ont permis de réaliser ce travail.

Nous remercions tous les enseignants de département MI.

Nous remercions l'honorable jury qui a accepté de juger notre travail pour son temps et son engagement.

Yasmine Benlakehal

Belkacem Fethia

Dédicace

Avant tout, Je remercie ALLAH

Tout puissant qui m'a donné toute la volonté et la

Patience pour aboutir à la réalisation de ce travail.

Je dédie mon travail à :

Mes chers parents, pour tous leurs sacrifices, leur amour,

Leur tendresse, leur soutien tout au long de mes études.

Toutes Mes sœurs.

Et ces enfants.

Tout Mes frères.

Toute ma famille.

Tous mes amis.

Ma Binôme YASMINE BENLAKEHAL

Tous ceux qui me sont chers.



Fethia

Dédicace

Avant tout, Je remercie ALLAH

Tout puissant qui m'a donné toute la volonté et la

Patience pour aboutir à la réalisation de ce travail.

Je dédie mon travail à :

Mes chers parents, pour tous leurs sacrifices, leur amour,

Leur tendresse, leur soutien tout au long de mes études.

Mon frère Walid.

Toutes Mes sœurs : khalida, Zahra, Soumia et Fatiha.

Et ces enfants.

Toute ma famille.

Tous mes amis.

Ma Binôme FETHIA BELKACEM

Tous ceux qui me sont chers.



Yasmine

Résumé

Github est l'une des premières grandes marques technologiques à adopter GraphQL. Et d'après l'étude de Github, GraphQL était l'une des tendances à suivre de 2018. GraphQL est défini par une spécification indépendante des langages de programmation et des protocoles de transport, dans le but de s'inscrire comme un nouveau standard dans le développement d'API.

Dans ce travail, nous proposons une approche de spécification et d'implémentation semi-automatique des requêtes GraphQL pour générer des services composites. Cette approche est basée sur la démarche MDA.

Mot clés : GraphQL, service web, Composition de services web, MDA, WSDL.

Abstract

Github is among the first major technology brands to adopt GraphQL. According to the Github study, GraphQL was one of the trends to follow in 2018. The GraphQL is defined as a specification independent of programming languages and transfer protocols, with the aim of marking as a new standard in the development of APIs.

In this work, we suggest a way to identify and implement GraphQL queries almost automatically to create composite services. This approach is based on the MDA approach.

Key words: GraphQL, web service, web service composition, MDA, WSDL.

ملخص

Github هي واحدة من أوائل العلامات التجارية التقنية الكبرى التي تتبنى GraphQL.وفقًا لدراسة Github، كان GraphQL أحد لاتجاهات التي وجب إتباعها في عام 2018. يتم تعريف GraphQL بمواصفات مستقلة عن لغات البرمجة وبروتوكولات النقل، بهدف التسجيل كمعيار جديد في تطوير واجهة برمجة التطبيقات.

في هذا العمل، نقترح طريقة لتحديد وتنفيذ استفسارات GraphQL بشكل شبه تلقائي لإنشاء خدمات مركبة. يستند هذا النهج إلى نهج MDA.

الكلمات المفتاحية: GraphQL, خدمة ويب, تكوين خدمةمركبة,MDA, WSDL.

Table des matières

Liste des figures VIII

Liste des tableaux X

Liste des acronymes XI

Introduction generale 1

Chapitre 01 Service web

I. Introduction 3

II. Service web 3

II.1. DEFINITION 3
II.2. ARCHITECTURE ORIENTEE SERVICE WEB 3
II.3. DEFINITION DE SOA 3
II.4. COMPOSANTS DE SOA 4
II.5. LES COUCHES UTILISEES PAR LES SERVICES WEB 4
II.5.1. Xml (extensible markup language) 5
II.5.2. Soap (simple object access protocol) 5
II.5.3. Wsdl (web services description langage) 6
II.5.4. Uddi (universal description, discovery and integration) 7
II.6. FONCTIONNEMENT DU SERVICE 7

III. Composition de services web 8

III.1. DEFINITIONS DE COMPOSITION DE SERVICES WEB 8
III.2. TYPES DE COMPOSITIONS 9
III.2.1. Composition statique des services web 9
III.2.2. Composition dynamique des services web 10
III.3. COMPOSITIONS MANUELLE, SEMI-AUTOMATIQUE ET AUTOMATIQUE 11
II.4. LES LANGAGES UTILISES POUR LES COMPOSITIONS[3] 11

IV. Conclusion 14

Chapitre 02 GraphQL

I. Introduction 15

II. Historique 15

III. Definition de graphql 15

IV. Architecture graphql 16

IV.1. FRONTEND 17
IV.1.1. Le flux de communication client-serveur dans le graphql 17
IV.2. BACKEND 18

IV.2.1. Serveur graphql avec une base de données connectée 18
IV.2.2. Serveur graphql avec application api 18
IV.2.3. Le serveur graphql hybride 19
V. Schema graphql 19
V.1. DEFINITION 19
V.2. LES ELEMENTS DE SCHEMA 20
V.3. SCHEMA STITCHING 25
VI. La composition par graphql 26
VI.1. COMPOSITION EN « SERIE »« PARALLELE » 28
VII. Meta model graphql 29
VIII. Graphql vs api rest 29
IX. Avantages et inconvenients de graphql 30
X. Conclusion 31

Chapitre 03 Ingénierie dirigée par les modèles « IDM»

I. Introduction 32
II. Definition 32
III. Les concepts de l'ingenierie dirigee par les modeles 32
III.1. MODELE 32
III.2. META MODELE 33
III.3. META META-MODELE 33
III.4. POINT DE VUE 34
III.4.1. Point de vue entreprise (le pourquoi) 34
III.4.2. Point de vue information (le quoi) 34
III.4.3. Point de vue traitement (le comment) 34
III.4.4. Point de vue ingénierie (le où) 34
III.4.5. Point de vue technologie (avec quoi) 34
IV. Transformation de modele 35
IV.1. CLASSIFICATION DES APPROCHES DE TRANSFORMATION DE MODELES 35
IV.1.1. Transformations modèle vers modèle (m2m : model-to-model) 35
IV.1.2. Transformation modèle vers code (m2t : model-to-text) 36
IV.2. LANGAGE DE TRANSFORMATION 36
IV.2.1. Emf/gef/gmf 36
IV.2.1. Acceleo 36
IV.2.2. Atl (atlas transformation langage) 37
V. L'architecture dirigee par les modeles (mda) 37
V.1. LE PRINCIPE DE BASE D'MDA 38
V.2. L'OBJECTIF DU MDA 39

V.3. LES COUCHES DU MDA 39
VI. Conclusion 40

Chapitre 04 Conception

I. Introduction 41
II. L'approche proposee 41
III. Methode de l'approche proposee 41
IV. Demarche de transformation 42
IV.1. DESCRIPTION LES ETAPES DE LA TRANSFORMATION 43
V. Les regles transformations (wsdl2graphql) 44
VI. La generation d'un fichier graphql de composition 46
VII. Conclusion 46

Chapitre 05 Implémentation et étude de cas

I. Introduction 47
II. Les langages, les outils et les bibliotheques 47
II.1. PYTHON ; SPYDER 47
II.2. ZEEP 48
II.3. NODE.JS 48
II.4. VISUAL STUDIO CODE 49
II.5. GRAPHQL-YOGA 49
III. Etude de cas 50
III.1. PRESENTATION 50
III.1.1 description des services web utilisés 50
III.1.2. Scenario de la composition 51
III.2. REPRESENTATION DE SCENARIO A L'AIDE DE L'APPROCHE 52
III.2.1. Premier étape (la transformation wsdl2graphql) 52
III.2.2. Seconde étape (assemblage des fichiers) 54
III.2.3. Dernier étape (génération la requête graphql) 56
IV. Conclusion 57
Conclusion generale 57
Bibliographie 39

Liste des figures

Figure I. 1. Architecture de base des services web - 4

Figure I.2. La pile protocolaire simplifie des services web - - - - - - - - - - - - - - - - - - - 5

Figure I. 3. Structure des messages SOAP - 6

Figure I. 4. WSDL - 7

Figure I. 5. L'annuaire UDDI - 7

Figure I. 6. Le fonctionnement des services web - 8

Figure I. 7. Composition des services web - 9

Figure I. 8. L'orchestration - 10

Figure I. 9. La chorégraphie - 10

Figure II. 1. GraphQL - 16

Figure II.2. Backend & Frontend - 16

Figure II. 3. Communication client-serveur dans le GraphQL - - - - - - - - - - - - - - - - - 17

Figure II. 4. Serveur GraphQL connecté à une base de données - - - - - - - - - - - - - - - 18

Figure II. 5. Serveur GraphQL intégré à un système de base de données existant - - - -19

Figure II. 6. Serveur GraphQL utilisé pour connecter à une base de données connectée à un système intégré existant - 19

Figure II. 7. Exemple de requête GraphQL - 21

Figure II. 8. Une comparaison entre l'équivalent SQL d'une mutation et une mutation GraphQL - 22

Figure II. 9. La forme de base de l'objet mutation dans le schéma - - - - - - - - - - - - 23

Figure II. 10. Le test dans le tableau de bord GraphiQL - 24

Figure II. 11. Exemple de l'abonnement - 25

Figure II. 12. Schema stitching - 26

Figure II. 13. Exemple d'une application web - 26

Figure II. 14. Diagramme de corrélation - 27

Figure III. 1. Conception d'IDM - 33

Figure III. 2. Diagramme générale de transformation - 35

Figure III. 3. Transformation Acceleo - 37

Figure III. 4. Transformation ATL - 37

Figure III. 5. L'architecture de MDA - 38

Figure III. 6. Les principes de l'approche MDA - 38

Figure IV. 1. Les étapes de l'approche proposée - 41

Figure IV. 2. Les étapes de la transformation - 43

Figure IV. 3. Organigramme de composition -46

Figure V. 1. Python; Spyder - 47

Figure V. 2. Zeep - 48

Figure V. 3. Node.js -48

Figure V. 4. Visual studio code - 49

Figure V. 5. GraphQL-yoga - 49

Figure V. 6. Composition des services - 51

Figure V. 7. WSDL de service Etat civil - 52

Figure V. 8. XSD de service Etat civil - 52

Figure V. 9. index.js - 53

Figure V. 10. Le résultat de l'exécution de fichier GraphQL de la figure V.9 - - - - - - 54

Figure V. 11. Schéma Stitching - 55

Figure V. 12. Fichier de composition - 57

Liste des tableaux

Tableau I.1. Langages utilisés pour les compositions - 13

Tableau II.1. Comparaison entre le REST et GraphQL - 30

Tableau II.2. Les avantages et les inconvénients de GraphQL - - - - - - - - - - - - - - - - - 31

Tableau IV.1. Les règles de transformation WSDL2GraphQL - - - - - - - - - - - - - - - - - 45

Liste des acronymes

API: Application Programming Interface

ATL: Atlas Transformation Langage

BPEL: Business Process Execution Language

CIM: Computation Independent Model

EMF: Eclipse Modeling Framework

GEF: Graphical Editing Framework

GMF: Graphical Modeling Framework

HTTP: Hypertext Transfer Protocol

IDM : L'ingénierie dirigée par les modèles

J2EE: Java 2 Entreprise Edition

LLC: Logical Link Control

M2M: Model-to-Model

M2T: Model-to-Text

MDA: Model Driven Architecture

MDE: Model Driven Engineering

MOF: Meta-Object Facility

OCL: Object Constraint Language

OMG: Object Management Group

PDM: Platform Description Model

PHP: Hypertext Preprocessor

PIM: Platform Independent Model

PSM: Platform Specific Model

UDDI: Universal Description, Discovery and Integration

UML: Unified Modeling Language

URL: Uniform Resource Locator

REST: representational state transfer

RPC: Remote Procedure Call

SDL: Schema Définition Langages

SOA: Service Oriented Architecture

SOAP: Simple Object Access Protocol

SQL: Structured Query Language

TCP: Transmission Control Protocol

WSCI: Services Web Chorégraphie Interface

WSCL: Web Services Conversation Language

WSDL: Web Services Description Langage

WSFL: Web Service Flow Language

XLANG: Web Services for Business ProcessDesign

XMI: XML Metadata Interchange

XML: eXtensible Markup Language

XPath: XML Path Langage

XSLT: Extensible Stylesheet Langage Transformations

XSD: XML Schema

Introduction générale

Les API « Application Programming Interfaces » soit « interface de programmation d'application », sont apparues à l'aube de l'informatique, avant même les ordinateurs personnels. À cette époque, elles étaient surtout utilisées en tant que bibliothèques pour les systèmes d'exploitation. Au début des années 2000, elles sont devenues importantes pour l'intégration des données à distance. Est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications. Elle est parfois considérée comme un contrat entre un fournisseur d'informations et un utilisateur d'informations.

Pour standardiser l'échange des informations entre les API toujours plus nombreuses, il a fallu développer un protocole : le « Simple Object Access Protocol », plus connu sous le nom de SOAP, a pour objectif de simplifier l'échange des informations entre les applications. Le « Representational State Transfer », ou REST, est une autre tentative de normalisation.

Maintenant, REST (Representational State Transfer) est le standard le plus utilisé quand il s'agit de mettre en place une API (Application Programming Interface). Néanmoins une nouvelle approche a le vent en poupe, il s'agit de GraphQL, un nouveau format qui est venu pour remédier à certaines limites de REST. Les API GraphQL peuvent être une nouvelle technologie passionnante.

Les dernières décennies ont été marquées par le développement rapide des systèmes d'information distribués, et la diffusion de l'accès à Internet. Cette évolution du monde informatique a entraîné le développement de nouveaux paradigmes d'interaction entre les applications.

Les services web individuels restant limités par leurs capacités, l'entreprise est amenée à composer un ensemble de services afin de créer des services plus complexes, On parle ainsi de la composition des services web. Cette composition se présente comme un paradigme fondamental de la technologie des services web. Elle permet de résoudre des problèmes complexes en combinant des services de base disponibles pour satisfaire un mais initial. Ce paradigme demeure l'un des axes de recherche les plus actifs des services web, durant ces dix dernières années, vu la complexité du processus de composition et l'évolution rapide des normes et des standards de cette technologie.

Cette complexité est due principalement à l'hétérogénéité des services web puisqu'ils sont créés par des organisations différentes.

Le bzz que GraphQL a fait ces dernières années attire les concepteurs du web à l'utiliser, mais son utilisation pose une question, comment écrire une requête GraphQL pour composer les services web?

Dans ce travail nous avons proposé une approche pour générer des requêtes GraphQL pour des micros services. Cette approche est basée sur l'ingénierie dirigée par les modèles qui permettre de créer des requêtes GraphQL semi-automatique à partir de requête d'un utilisateur.

Ce mémoire est organisé en cinq chapitres. Nous exposons ci-dessous le contenu de ces chapitres :

Chapitre I: Dans ce chapitre, nous avons introduit le concept de service Web et nous avons exposée en détail l'architecture ainsi que les langages et les protocoles utilisés pour leur déploiement. Et le concept de la composition des services (type et les langages de composition).

Chapitre II: Dans ce chapitre, nous présenterons les concepts de base de GraphQL. Tels que les notions de structure, des types et de composition dans GraphQL. Et enfin nous présenterons quelques avantages et inconvénients de GraphQL.

Chapitre III : Dans ce chapitre nous présentons les concepts de base de l'Ingénierie Dirigée par les Modèles (IDM) avec ses différentes variantes, Nous avons survolé également les techniques de transformation de modèle. Avec une présentation d'architecture MDA et ses principes (CIM, PIM, PSM et PDM).

Chapitre IV: Dans ce chapitre, nous présentons notre approche pour générer une requête GraphQL d'une composition des services web en suivent la démarche MDA. Basée sur des règles de transformation.

Chapitre V: Ce chapitre est consacré à la mise en œuvre de notre approche.

Enfin, nous terminons par une conclusion générale décrivant ce que nous avons accompli et nos perspectives.

Chapitre 01
Service web

I. Introduction

Au début des années 2000, le paradigme SOA (Service Oriented Architecture) est né dans le but de résoudre le problème d'interopérabilité entre les différentes technologies informatiques distribuées utilisées par les entreprises.

Ce paradigme est l'un des modèles les plus prometteurs, il reprend les avantages d'autres paradigmes (orientés objet et orientés composants), tels que la modularité et l'encapsulation.

Dans ce chapitre nous allons présenter de quelques concepts de base entourent la technologie des services web.

II. Service web

II.1. Définition

Les services Web représentent un mécanisme de communication entre des applications distantes sur un réseau Internet, indépendant de tout langage de programmation et de toute plate-forme d'exécution. En utilisant le protocole HTTP (Hypertext Transfer Protocol) comme méthode de transport, la communication s'effectue sur un support commun, contrôlé et généralement non filtré par des pares-feux **[1]**.

II.2. Architecture orientée service web

L'architecture orientée services (Service Oriented Architecture ou SOA en anglais) est un cadre technologique stratégique qui permet à tous les systèmes pertinents à l'intérieur et à l'extérieur d'une organisation d'exposer et d'accéder à des services bien définis et à des informations liées à ces services, qui peuvent être ensuite résumées en couches de processus et applications développées par solutions composites. Essentiellement, la SOA ajoute de l'agilité à l'architecture, nous permettant d'utiliser la couche de configuration pour gérer les modifications de nos systèmes sans avoir à redévelopper constamment ces systèmes **[1]**.

II.3. Définition de SOA

SOA est un paradigme pour l'organisation et l'utilisation de connaissances distribuées provenant de divers domaines. Il fournit un moyen unifié de fournir, de

découvrir, d'interagir et d'utiliser des compétences pour produire les résultats souhaités avec des conditions préalables et des objectifs mesurables **[2]**.

II.4. Composants de SOA

Dans ce titre, nous allons présenter l'architecture orienté service est base sur les trois composants suivants : le fournisseur de service, l'annuaire de services et le client (figure I.1).

- **Le fournisseur de service :** Il met en œuvre le service web et le met à la disposition de tous sur internet.
- **L'annuaire de service :** L'annuaire de service est un catalogue contenant les services disponibles.
- **Client :** est un demandeur ou un consommateur qui demande un service web spécifique pour répondre à ses besoins.**[3].**

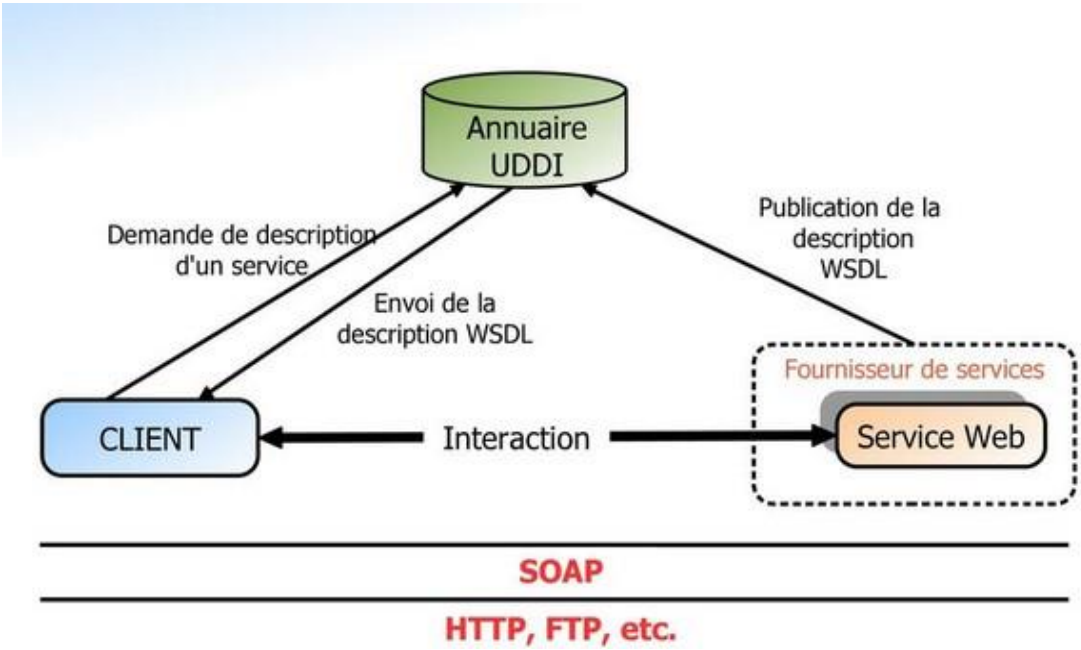


Figure I. 1. Architecture de base des services web

II.5. Les couches utilisées par les services web

Selon le World Wide Web Consortium (W3C), les services Web sont conçus pour prendre en charge les interactions interopérables de machine à machine sur le Web. Il a une interface décrite dans un format lisible par machine. D'autres systèmes utilisent des messages SOAP pour interagir avec les services Web comme spécifié par leur

description, en utilisant généralement le protocole HTTP avec sérialisation XML ainsi que d'autres normes Web (figure I.2**[2]**).

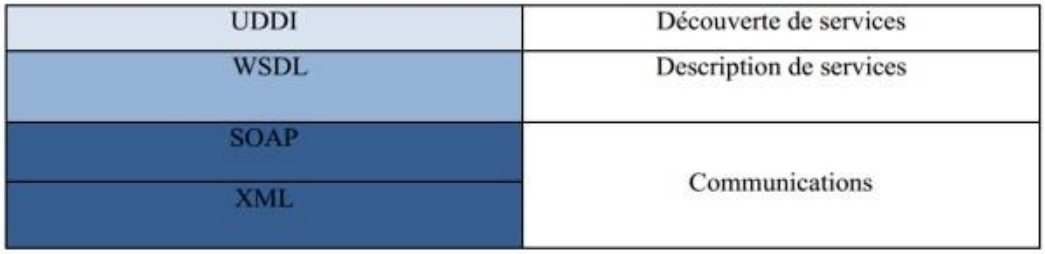


Figure I. 2. La pile Protocolaire simplifiée des services web

II.5.1. XML (eXtensible Markup Language)

XML constitue la technologie de base des architectures Web services ; c'est un facteur important pour contourner les barrières techniques. XML est un standard qui permet de décrire des documents structurés transportables sur les protocoles d'Internet. En effet, il apporte à l'architecture des services web l'extensibilité et la neutralité vis à vis des plateformes et des langages de développement. De plus, grâce à la structuration, XML permet la distinction entre les données des applications et les données des protocoles.

Parmi les spécifications XML, nous soulignons :

- XSD (XML Schema) : c'est un langage qui sert à décrire formellement un vocabulaire.
- XSLT (Extensible Stylesheet Langage Transformations) : est utilisé pour transformer un document XML basé sur un certain schéma en un autre document XML qui peut être un document lui-même basé sur un autre schéma.
- XPath (XML Path Langage) : fournit une syntaxe d'expressions utilisées pour créer des chemins de localisation**[4]**.

II.5.2. SOAP (Simple Object Access Protocol)

SOAP est un protocole de messagerie basé sur XML. il définit un ensemble de règles pour les messages structurés, principalement pour effectuer des conversations demande-réponse de type RPC (Remote Procedure Call) **[1]** (figure I.3).

Un message SOAP respecte les règles de XML et il est composé de trois éléments essentiels :

- **L'entête « *Header* » :** est optionnel, il peut contenir des informations de sécurité (telles que les signatures électroniques), des informations transactionnelles, des informations de traçabilités, etc.
- **L'élément « *Body* »:** est obligatoire, il contient les éléments suivants :
  - Le nom de méthode, avec les données correspondantes, ou un simple document XML, pour le cas d'une requête.
  - Les valeurs de retour pour le cas d'une réponse.
- **L'élément « *Fault* »:**est optionnel, il fournit des informations sur d'éventuelles erreurs survenues lors de l'analyse du message**[3]**.

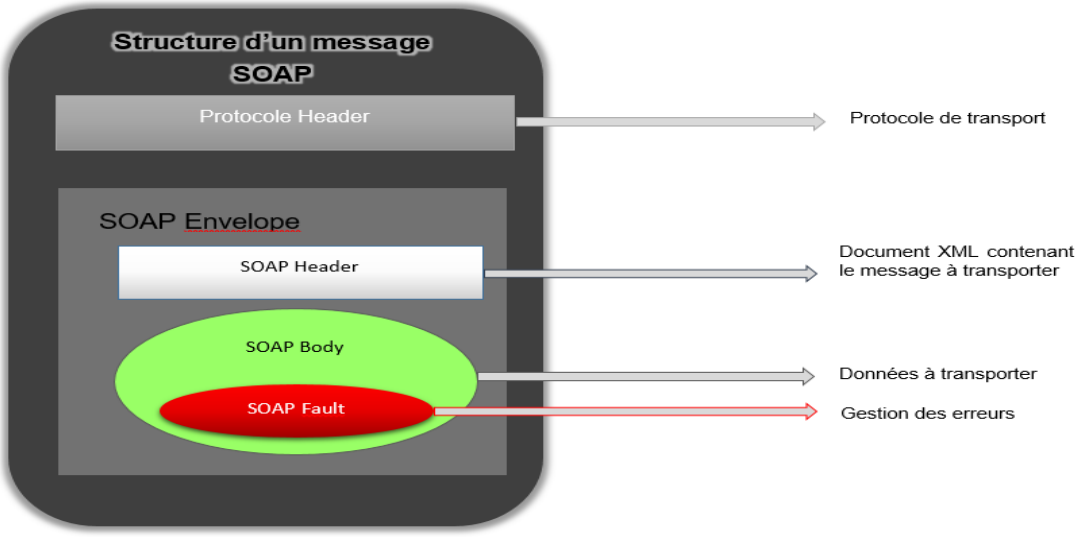


Figure I.3. Structure des messages SOAP

II.5.3. WSDL (Web Services Description Langage)

WSDL est une norme W3C qui définit la structure abstraite et concrète des services. C'est un document XML qui décrit la signature des opérations fournies par le service (nom de l'opération, noms et types de paramètres d'entrée/sortie), et il définit également les liens spécifiques à ces opérations, tels que le protocole de transport, l'URI du service, le service style, et Règles de codage pour les paramètres d'entrée/sortie **[3].**

L'objectif de WSDL est de décrire un service Web, quel que soit le langage d'implémentation du service et la plate-forme de compensation. Afin que WSDL soit compris par différents langages d'implémentation et compatible avec les différentes plateformes utilisées, il est entièrement écrit en XML **[1]** (figure I.4).

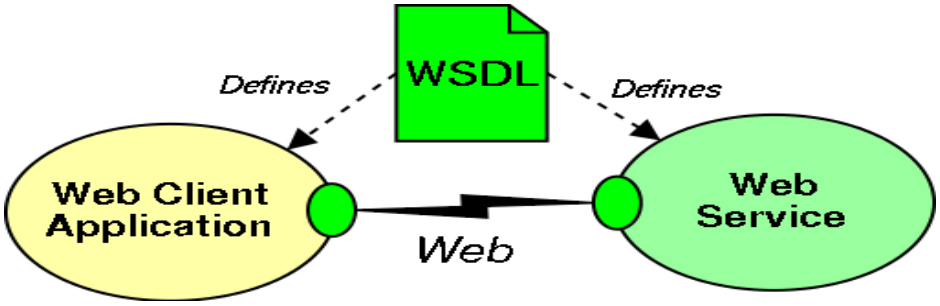


Figure I.4. WSDL

II.5.4. UDDI (Universal Description, Discovery and Integration)

Une fois le service Web décrit, les clients ne peuvent pas l'utiliser car ils ne savent pas que le service existe. C'est pourquoi nous avons besoin d'un support comme une publicité annonçant la création d'un nouveau produit. Les chercheurs ont défini un répertoire appelé UDDI (Universal Description, Discovery, and Integration) à utiliser comme un ensemble de normes pour la publication et la localisation de services Web **[1]**.

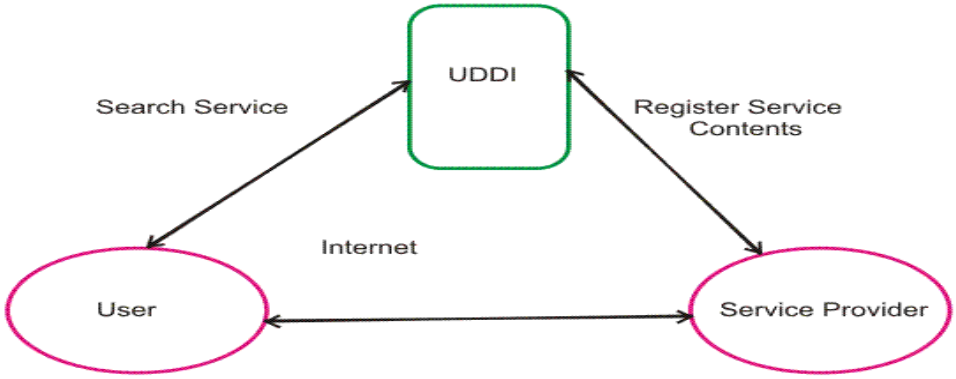


Figure I.5. L'annuaire UDDI

II.6. Fonctionnement du service

Le scénario classique du processus d'invocation d'un service web est décrit comme suit (figure I.6) :

1. Un fournisseur de services Web déploie son service et le rend accessible sur Internet. Il le publie ensuite au catalogue en décrivant ses caractéristiques dans un fichier WSDL.

2. Ensuite, le client de service Web envoie une demande SOAP au catalogue de services, en spécifiant ses propres exigences de recherche. L'annuaire recherche le

service demandé en fonction des critères spécifiés et renvoie les URL de tous les services disponibles au client.

3. Une fois que le client a sélectionné le service souhaité, il établit un contrat avec le fournisseur pour invoquer le service Web.

4. Le client interagit avec le service Web depuis son application via des requêtes SOAP en spécifiant les méthodes à invoquer et à échanger (envoyer et recevoir des messages). De plus, il s'agit d'une agrégation de services Web de base résultant en de nouveaux services Web à valeur ajoutée.**[3]**.

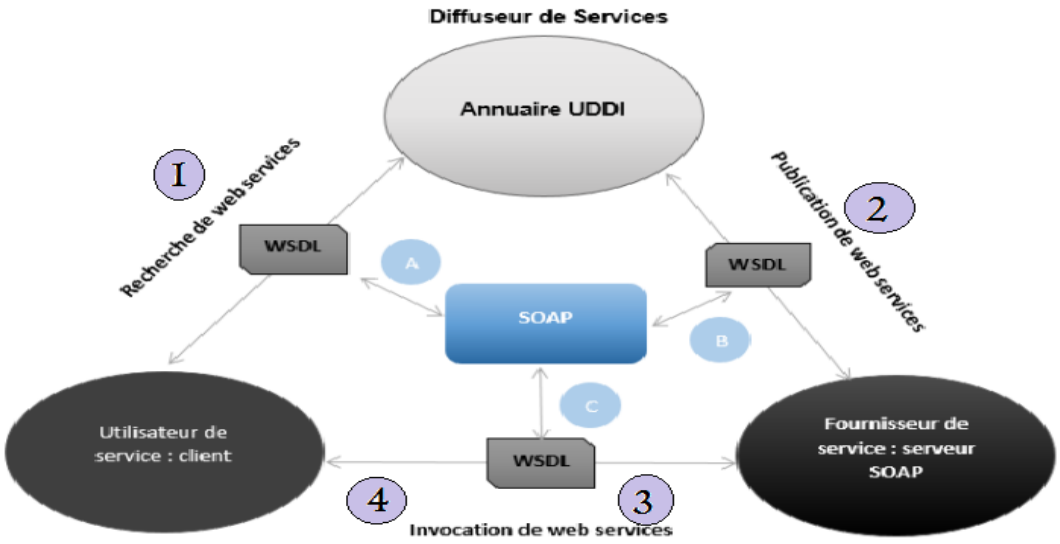


Figure I.6. Le Fonctionnement de service web

III. Composition de services web

III.1. Définitions de composition de services web

- **Définition 1 :** La composition des services Web peut être définie comme : "le processus de sélection, de composition et d'exécution des services pour atteindre un objectif donné" **[4]** (figure I.7**[5]**).

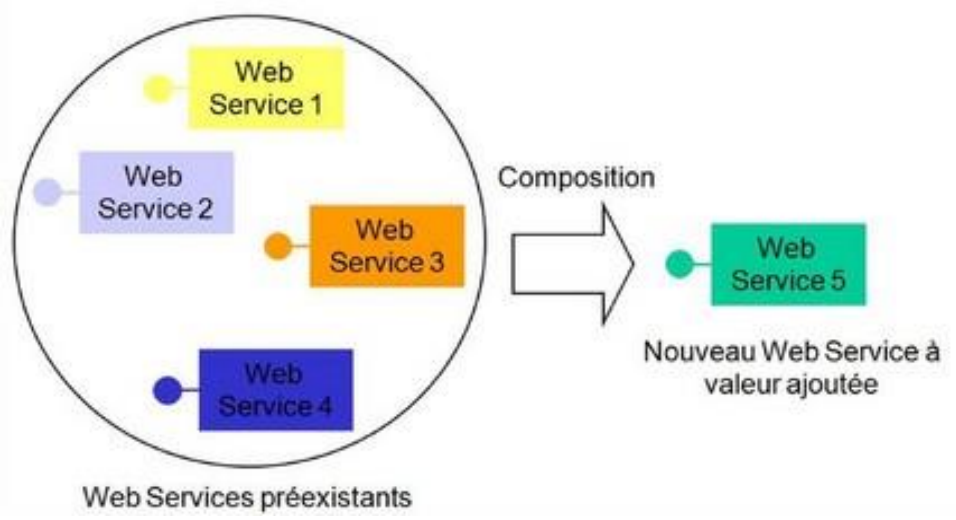


Figure I.7. Compositions des services web

- **Définition 2 :** Un service web est dit composé ou composite lorsque son exécution implique des interactions avec d'autres services web afin de faire appel à leurs fonctionnalités[2].

III.2. Types de compositions

III.2.1. Composition statique des services web

Dans cette approche, les services Web à composer sont choisis au moment de l'architecture et de la conception. Les composants sont sélectionnés et câblés ensemble avant la compilation et le déploiement. Cette approche fonctionne tant qu'il n'y a que peu ou pas de changements dans l'environnement du service Web. Si les fournisseurs de services proposent d'autres services ou modifient d'anciens services, des incohérences peuvent en résulter, ce qui nécessitera des modifications de l'architecture logicielle**[6]**.

La composition statique des services web peut être devisée en deux angles : l'orchestration ou la chorégraphie.

- **L'orchestration**

Pour les solutions Web, la logique métier doit être répartie entre différents services. Ces services doivent être organisés et composés, ce qui est souvent appelé orchestration. Les processus d'orchestration fournissent des services distincts qui peuvent être efficacement combinés entre les flux pour exécuter les processus métier. Les différents services concernés sont coordonnés par le service du contrôleur. Le service généré peut être intégré hiérarchiquement dans une autre composition grâce à son interface WSDL. Dans l'orchestration, les services Web associés sont tous sous un contrôle unique Core

Process (un autre service Web). Ce processus coordonne l'exécution des différentes opérations des services Web participant au processus **[1]** (figure I.8 **[5]**).

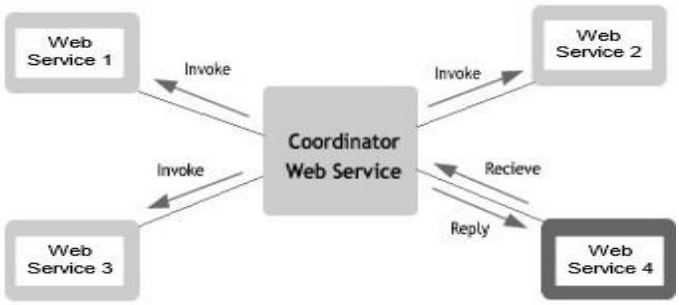


Figure I.8. L'orchestration

- **La chorégraphie**

La chorégraphie des services est la description globale des services participants, définie par les échanges de messages, les règles d'interaction et les protocoles entre deux points de terminaison ou plus. La chorégraphie utilise une approche décentralisée de la composition des services, c'est-à-dire qu'il n'y a pas de contrôleur central et que chaque service participant à l'orchestration doit fonctionner de manière autonome.

Au moment de la conception, La chorégraphie garantit l'interopérabilité entre un ensemble de services d'un point de vue global, ce qui signifie que tous les services participants sont traités de la même manière d'égal à égal **[1]** (figure I.9**[5]**).

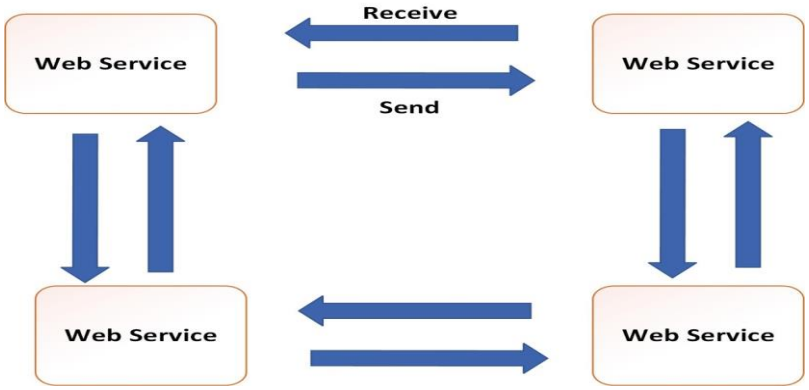


Figure I. 9. La chorégraphie

III.2.2. Composition dynamique des services web

La composition dynamique de services Web est la technologie clé de l'implémentation de l'architecture SOA. Cependant la sélection de services représente

un réel problème dans ce type de composition. La complexité de la sélection des services inclue trois facteurs principaux. Le premier concerne le grand nombre de changement dynamique des instances des services ayant des fonctionnalités similaires et qui pourraient être disponibles pour composer un service. Le deuxième facteur concerne les différentes possibilités d'intégration, des composants d'une instance de service, au processus d'un service complexe. Tandis que le troisième facteur correspond aux diverses exigences de performance d'un service complexe.**[6]**.

III.3. Compositions manuelle, semi-automatique et automatique

Dans ce titre nous allons présenter les différents types de la composition des services web. Cette composition peut être devisée sur trois catégories :

- ***Manuelle :*** La composition manuelle est basée sur l'intervention de l'utilisateur qui gère la composition sans l'aide d'outils dédiés. Donc c'est à l'utilisateur de programmer et d'implémenter la composition.
- ***Semi-automatique :*** La composition semi-automatique est constituée comme un pas en avant en comparaison avec la composition manuelle. Elle fournit des suggestions sémantiques pour aider à la sélection des services web dans le processus de composition.
- ***Automatique :*** La composition automatique prend en charge tout le processus de composition sans qu'aucune intervention de l'utilisateur ne soit requise. Cependant, la composition automatique a été et sera toujours une tâche difficile à réaliser. Nous pouvons citer différentes sources de complexité :
    - La difficulté de la composition en fonction de l'expressivité du modèle de services et l'objectif de la composition.
    - Le grand nombre de services web sur le Web.
    - La diversité des modèles de conception de services en raison des besoins de modélisation et/ou de la vision des développeurs**[4]**.

III.4. Les langages utilisés pour les compositions[3]

Langage	Organisme	Avantages	Inconvénient
XLANG (Web Services for Business	➢ Microsoft (01/2001)	-Xlang est un langage XML de description	- Le langage XLANG est propriétaire et

ProcessDesign)		de processus métier destiné à fournir les moyens d'orchestrer les applications et les services web. -Corrélations des messages entre eux. -Découverte dynamique des services. -Contrats multipartites	n'est pas entièrement documenté.
WSFL (Web Service Flow Language)	➢ IBM (03/2001)	-Utilisation dans les processus de type Workflow (travail collaboratif)	- WSFL ne sert pas à définir le processus mais sa mise en œuvre.
WSCL (Web Services Conversation Language)	➢ W3C (03/2002)	-WSCL (comme BPML) utilise également le concept de «corrélation» pour identifier l'instance de collaboration d'un message donné. - Le principal avantage de l'approche WSCL est en collaboration multipartite.	- WSCL ne spécifie pas comment les interactions sont liées à un protocole de communication spécifique car les liaisons de protocole sortent du cadre de WSCL.
WSCI (Services Web Chorégraphie Interface)	➢ W3C (08/2002)	-L'utilisation de WSCI pour détailler les services référencés dans un annuaire UDDI - WSCI permet ainsi d'utiliser les services web au sein de	- WSCI ne traite pas de la définition du processus pilotant l'échange de messages ou de la définition du comportement interne de chaque

		processus complexes.	service Web.
BPEL ou BPEL4 WS(Business Process Execution Language)	➢ Microsoft IBM+ BEA (08/2002) ➢ Standard d'OASIS (04/2007)	- BPEL organise justement le dialogue entre les différentes applications de l'architecture SOA. - Sans BPEL, les services Web ne pourraient pas dialoguer entre eux. -Il organise donc le déroulement des processus.	-L'inconvénient principal de BPEL4WS est que la définition du processus est rigide. Si une activité du processus échoue, le processus dans son intégralité échoue. Aucun retour en arrière et aucune alternative au processus ne sont possibles. De même, lors de la description du processus exécutable, la définition des flots de données ne permet pas de connecter des services dont les entrées/sorties ne correspondent pas exactement. BPEL4WS ne prévoit pas de mécanisme de transformation de données.

Tableau I.1. Langages utilisés pour les compositions

IV. Conclusion

Dans ce chapitre, nous avons introduit plusieurs notions sur le concept de service web et on a exposée en détail l'architecture, les couches, les opérations et le concept de la composition des services (type et les langages de composition).

Dans le chapitre suivant, nous énumérerons les différents principes du GraphQL, nous vous donnerons donc un aperçu du GraphQL en employons l'ensemble des éléments dont nous aurons besoin dans notre projet.

Chapitre 02
GraphQL

I. Introduction

Traditionnellement, il existe deux approches différentes pour l'implémentation des services web : SOAP et REST. Le protocole SOAP nécessite de spécifier explicitement toutes les méthodes accessibles, y compris les structures de données d'entrée et de sortie dans un schéma, Le principal inconvénient de SOAP réside dans sa complexité et la surcharge imposée par le schéma explicite.

REST est actuellement le protocole API le plus répandu dans l'industrie. Les services Web compatibles REST mettent leurs ressources à disposition via des points de terminaison HTTP spécifiés. Chaque ressource nécessite sa propre requête. L'une des principales limitations de REST est que les utilisateurs d'API ont souvent besoin de plusieurs ressources associées à la fois, ce qui entraîne plusieurs requêtes aller-retour pour récupérer chaque ressource.

Pour ces inconvénients, GraphQL est présenté comme une bonne alternative aux API REST, qui cherchent à combiner les avantages des deux protocoles. Dans ce chapitre on va introduire les concepts de base de GraphQL.

II. Historique

GraphQL a été développé en interne par Facebook en 2012. À partir de ce temps-là, le concepteur de GraphQL, Lee Byron, a travaillé pour rendre GraphQL omniprésent sur les plateformes Web. Et il a rendu public en 2015.

La Fondation GraphQL a été créée en 2019 en tant que point focal neutre pour les organisations qui soutiennent l'écosystème GraphQL, et le projet de spécification GraphQL a également été créé en 2019 sous le nom de Joint Développent Fondation Projects, LLC, GraphQL Séries [7].

III. Définition de GraphQL

GraphQL est un langage de requête API qui peut être utilisé pour décrire une sélection arbitraire d'objets et leurs champs qu'un serveur GraphQL doit renvoyer. La requête est analysée et résolue par un serveur GraphQL, et les données sont renvoyées au client qui a effectué la requête. La spécification GraphQL définit le langage de requête, mais ne définit pas comment la requête doit être résolue. La résolution de la requête peut se produire de n'importe quelle manière et elle diffère entre les

implémentations de GraphQL. Une implémentation GraphQL est généralement une petite couche dans le serveur qui analyse uniquement la requête et appelle les résolveurs de données en conséquence[8].

GraphQL fournit une description complète et facile à comprendre de vos données API, permettant aux clients de demander exactement ce dont ils ont besoin et rien de plus. Un service GraphQL est créé en définissant des types et des champs, puis en fournissant des fonctions pour chaque champ sur chaque type [7](figure II.1).

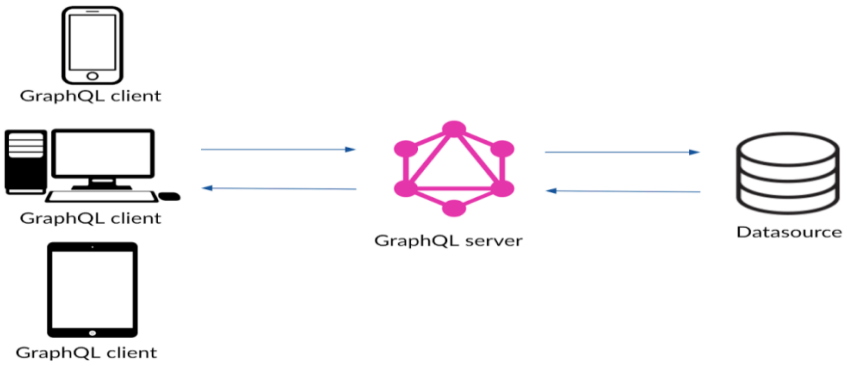


Figure II. 1. GraphQL

IV. Architecture GraphQL

GraphQL est constitué de deux parties « Frontend » qui fait la gestion des demandes de requête côté client et « backend » qui gère également la réponse sur les demandes de client côté serveur (figure II.2).

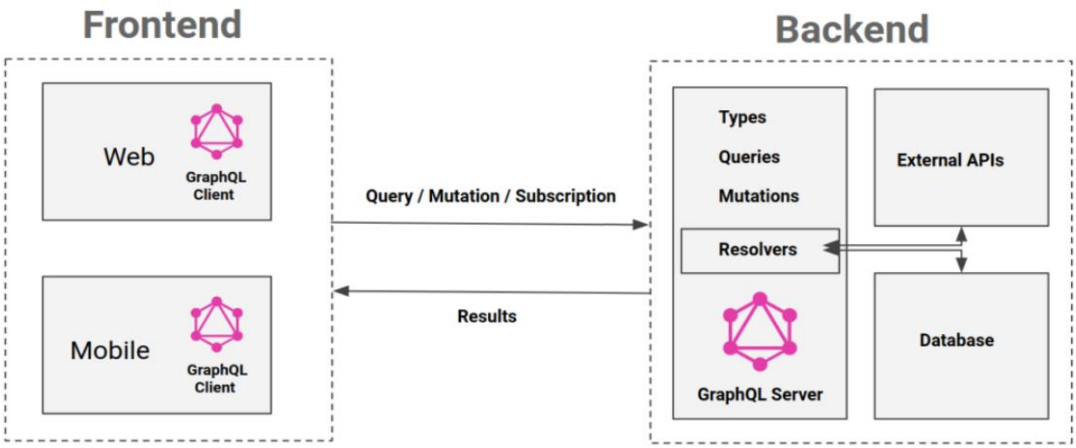


Figure II. 2. Backend & Frontend

IV.1. Frontend

Un client GraphQL pourra envoyer des requêtes HTTP, y compris des requêtes, des mutations ou des abonnements, et le serveur GraphQL traitera ces requêtes à l'aide d'un résolveur. Le résolveur récupérera les données d'une base de données ou d'une API REST par exemple[9].

IV.1.1. Le flux de communication client-serveur dans le GraphQL

- Tout d'abord, chaque fois qu'un client fait une requête 'POST' pour envoyer la requête au serveur GraphQL, la requête est envoyée sous la forme d'une chaîne.
- Le serveur reçoit la chaîne puis extrait la requête du client de la chaîne. Il commence à traiter et à valider la requête du client selon les syntaxes du schéma GraphQL.
- Le serveur API de GraphQL effectue un appel à la base de données selon la requête et récupère les données de la base de données, que le client a demandées dans sa requête.
- Après cela, lorsque GraphQL récupère les données de la base de données, le serveur GraphQL prend les données et les renvoie au client sous la forme d'un objet JSON[9](figure II.3).

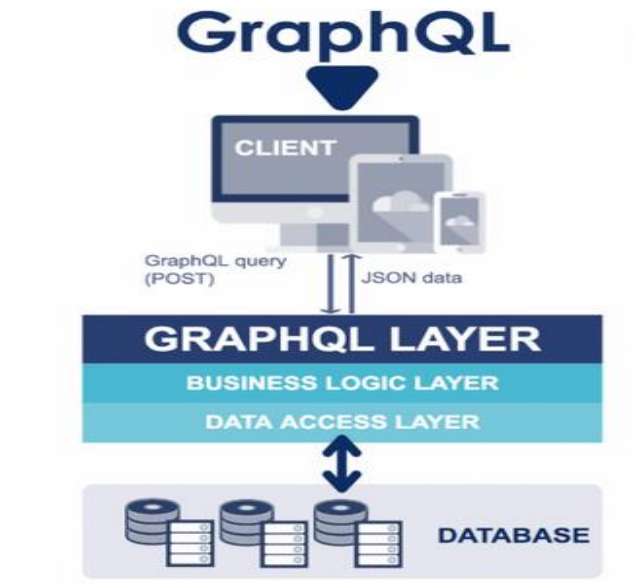


Figure II. 3. Communication client-serveur dans le GraphQL

IV.2. Backend

Pour utiliser GraphQL dans une application ou des projets, ils doivent d'abord créer un serveur GraphQL. Il existe 3 types d'architectures différentes qui incluent un serveur GraphQL. Les trois architectures représentent les principaux cas d'utilisation de GraphQL et démontrent la flexibilité en termes de contexte dans lequel il peut être utilisé :

IV.2.1. Serveur GraphQL avec une base de données connectée

Dans la configuration de cette architecture il a un seul serveur (web) qui implémente la spécification GraphQL. Lorsqu'une requête arrive sur le serveur GraphQL, le serveur lit la charge utile de la requête et récupère les informations requises dans la base de données. Il construit ensuite l'objet de réponse et le renvoie au client. Il est important de noter que GraphQL est en fait indépendant de la couche de transport. Cela signifie qu'il peut potentiellement être utilisé avec n'importe quel protocole réseau disponible. Ainsi, il est éventuellement possible d'implémenter un serveur GraphQL basé sur TCP, WebSockets, etc.[9](figure II.4).

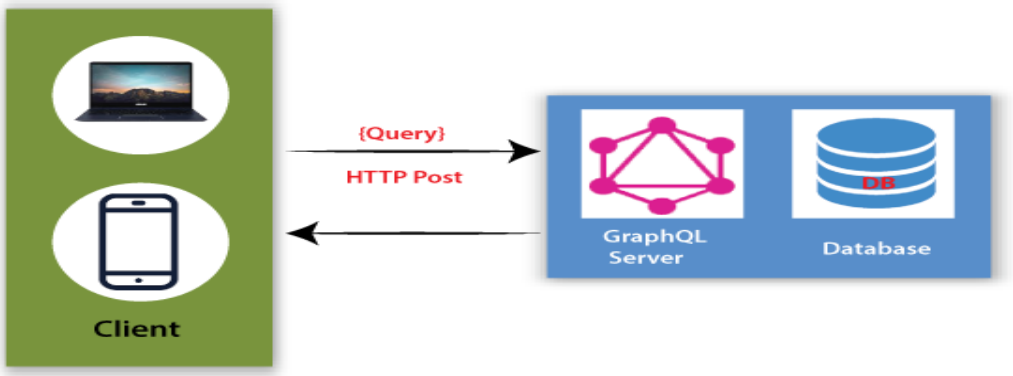


Figure II. 4. Serveur GraphQL connecté à une base de données

IV.2.2. Serveur GraphQL avec application API

Un autre cas d'utilisation de GraphQL est l'intégration de plusieurs systèmes existants derrière une API GraphQL unique et cohérente, GraphQL peut être utilisé pour cacher leur complexité derrière une API GraphQL. De cette façon, des nouvelles applications clientes peuvent être développées qui communiquent simplement avec le serveur GraphQL pour récupérer les données dont elles ont besoin. Le serveur GraphQL

est ensuite chargé de récupérer les données des systèmes existants et de les conditionner au format de réponse GraphQL[9](figure II.5).

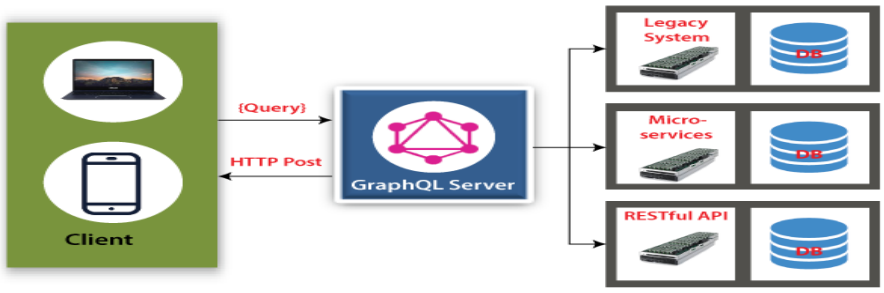


Figure II. 5. Serveur GraphQL intégré à un système de base de données existant

IV.2.3. Le serveur GraphQL hybride

Enfin, il est possible de combiner les deux approches et de créer un serveur GraphQL qui dispose d'une base de données connectée mais qui communique toujours avec des systèmes hérités ou tiers. Lorsqu'une requête est reçue par le serveur, il la résout et récupère les données requises à partir de la base de données connectée ou de certaines des API intégrées[9](figure II.6).

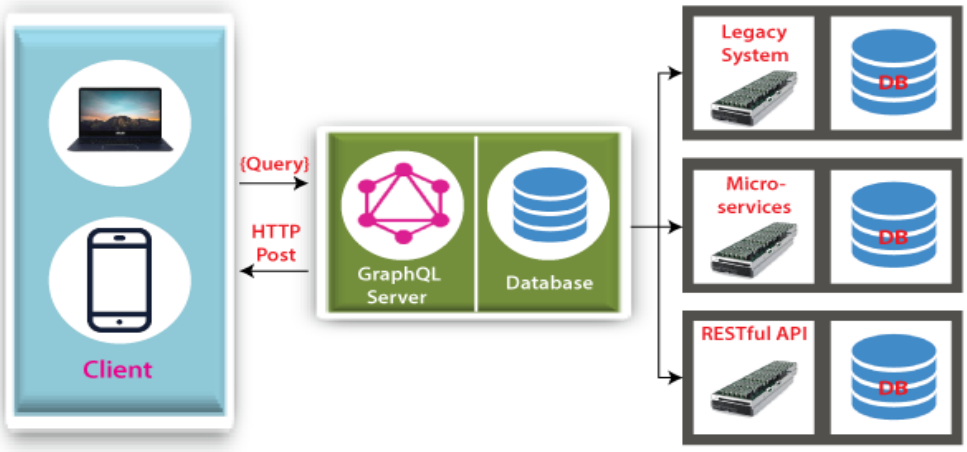


Figure II. 6. Serveur GraphQL utilisé pour connecter à une base de données connectée à un système intégré existant

V. Schéma GraphQL

V.1. Définition

Les schémas GraphQL ne sont pas une forme de schémas dans le sens traditionnel. C'est-à-dire qu'un tel schéma ne précise pas ce que les instances d'une base de données

particulière peuvent ressembler et quelles contraintes doivent être appliquées pour ces instances de base de données. Un schéma GraphQL représente les données que le client peut demander au serveur, le schéma d'une telle API spécifie les types d'objets pour lesquels l'API possède des données et le type de données dont elle dispose pour ces objets. De plus, il implémente des fonctions telles que des requêtes et des mutations qui, à leur tour, récupèrent et publient des données vers/depuis des sources de données. La spécification GraphQL définit un langage appelé "Schema Définition Langages (SDL)" pour écrire ces schémas[10].

V.2. Les éléments de schéma

Pour implémenter les schémas, on va utiliser le "Schema Definition Language (SDL), qui permet de rédiger des schémas GraphQL d'une manière indépendante du langage [11].

Et on peut résumer les éléments de schéma dans ces types comme suit [8]:

- **Types d'objets et champs**

Représentent simplement un type d'objet que vous pouvez récupérer à partir de votre service.

- **Types scalaires**

Ce type définie les données concrètes.

- **Types d'énumération**

Aussi appelés Enums, c'est un type spécial de scalaires limités à un ensemble particulier de valeurs autorisées (NEWHOPE, EMPIRE ou JEDI).

- **Type d'opération (requête, mutation et d'abonnement)**

Il existe trois types d'opérations modélisées par GraphQL :

- **Requêtes (Queries)**

Une requête GraphQL est utilisée pour lire ou récupérer des valeurs. Lors de l'exécution de la demande, toutes les validations par rapport au schéma GraphQL du serveur doivent réussir. Cela signifie que tous les champs demandés et leurs types sont définis dans le document de demande de la même manière que ceux définis dans le schéma.

Les requêtes doivent sélectionner et renvoyer les données du serveur en fonction des champs définis dans le document de requête[8].

L'exemple suivant expliqué comment travailler la requête (figure II.7)[12].

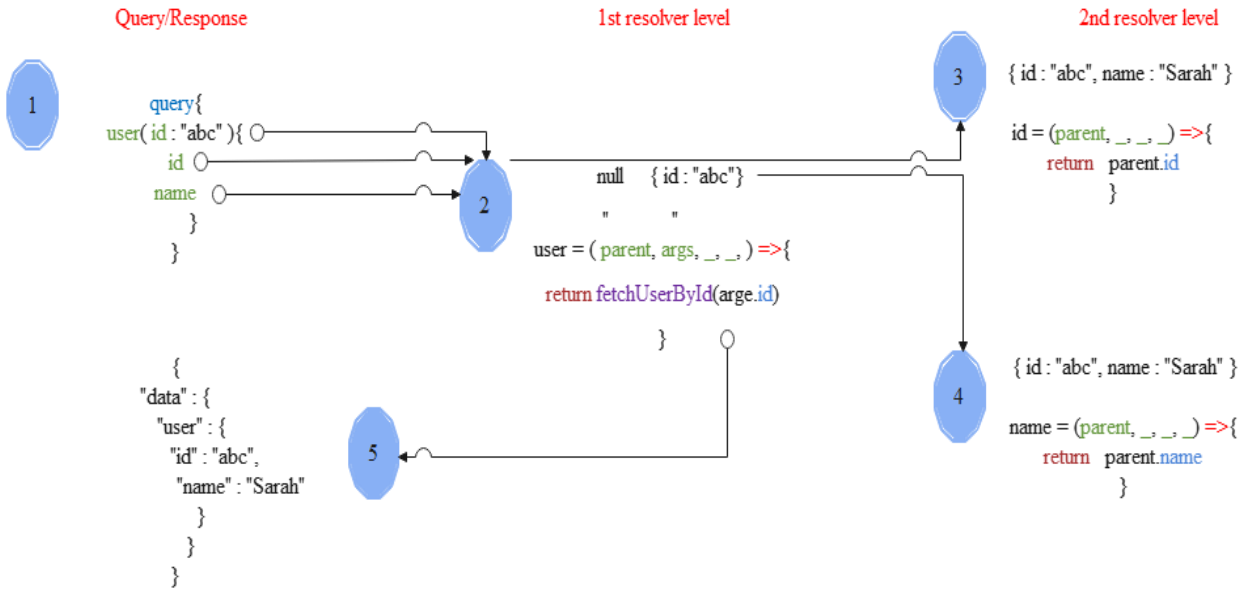


Figure II. 7. Exemple de requête GraphQL

L'explication :

1. La requête arrive sur le serveur.
2. Le serveur invoque le résolveur pour l'utilisateur du champ racine nous supposons qu'il renvoie cet objet : {"id": "abc", "name": "Sarah"}.
3. Le serveur appelle le résolveur pour le champ id sur le type d'utilisateur. L'argument d'entrée racine pour ce résolveur est la valeur de retour de l'invocation précédente, il peut donc simplement renvoyer root.id.
4. Analogue à 3, mais renvoie root.name à la fin. (Notez que 3 et 4 peuvent se produire en parallèle).
5. Le processus de résolution est terminé - enfin, le résultat est enveloppé avec un champ de données pour adhérer à la spécification GraphQL [12]:

{"data": {"user": {"id": "abc", "name": "Sarah" }}}

- **Mutations (Mutations)**

La plupart des discussions sur GraphQL se concentrent sur la récupération de données, mais toute plate-forme de données complète a également besoin d'un

moyen de modifier les données côté serveur. GraphQL définit le concept de mutations pour écrire ou publier des valeurs[13].

L'une des principales différences entre une mutation GraphQL et une requête d'action SQL est que la mutation gère à la fois l'action et le retour des données résultantes en un seul aller-retour[9] (figure II.8).



Figure II. 8. Une comparaison entre l'équivalent SQL d'une mutation et une mutation GraphQL

- Nous allons présenter cet exemple pour en savoir plus sur les mutations. Nous devons d'abord ajouter la forme de base de l'objet mutation dans le schéma comme suit :

```
Const graphql = require ('graphql')
Const fetch = require ('node-fetch')
Const {
GraphQLObjectType,
GraphQLString,
GraphQLInt,
} = graphql
Const PostType= require ('. / PostType')
Const mutation = newGraphQLObjectType ({
Name: 'Mutation',
Fields: {
Create: {
Type: PostType,
Args :{
Title: {type: GraphQLString},
Body: {type: GraphQLString},
User ID: {type: GraphQLString},
```

Figure II. 9. La forme de base de l'objet mutation dans le schéma

Ensuite, nous le testons dans le tableau de bord GraphiQL (figure II.10) Notre première mutation s'appellera (create) et vous devez spécifier le type de retour, ce sont les arguments et la fonction de la solution.

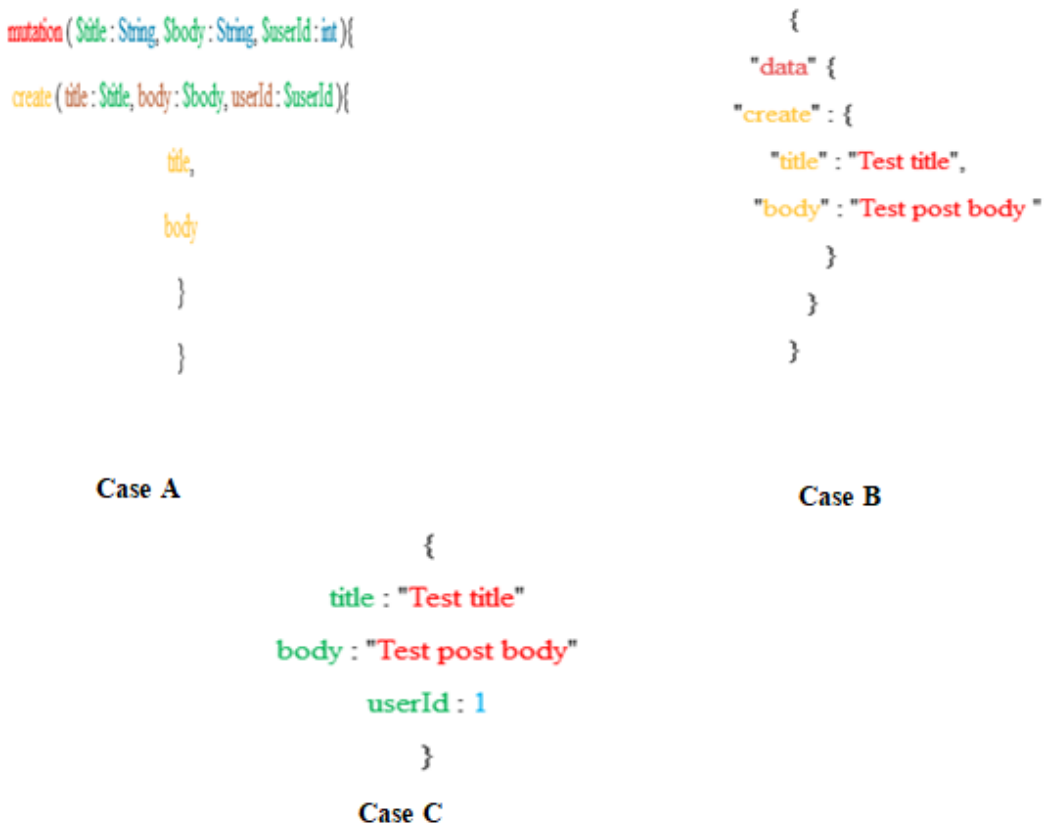


Figure II. 10. Le test dans le tableau de bord GraphiQL

L'explication :

- La **case A** est l'endroit où nous appelons la mutation et sélectionnons les variantes que nous utiliserons. Dans la **case B**, nous spécifions les valeurs des variables elles-mêmes qui doivent correspondre au type que nous leur avons donné. Ensuite, lorsque nous appelons la mutation réelle à la ligne 2 dans la **case A**, nous la transmettons en tant que variables qui l'affecteront automatiquement.
- Lorsque nous avons identifié le type de retour pour les mutations, nous l'avons fait afin de pouvoir interroger son résultat. La valeur renvoyée peut être utilisée comme une requête GraphQL normale comme montrer dans la **case C**[14].

➢ **Abonnement (Subscription)**

Comme les requêtes, les abonnements vous permettent de récupérer des données. Contrairement aux requêtes, les abonnements sont des opérations de longue durée qui peuvent modifier leur résultat dans le temps. Ils peuvent maintenir une connexion active à votre serveur GraphQL, permettant au serveur de pousser les mises à jour du résultat de l'abonnement [9].

Une requête est une réponse unique tandis qu'un abonnement est une écoute continue des réponses du serveur (figure II.11).

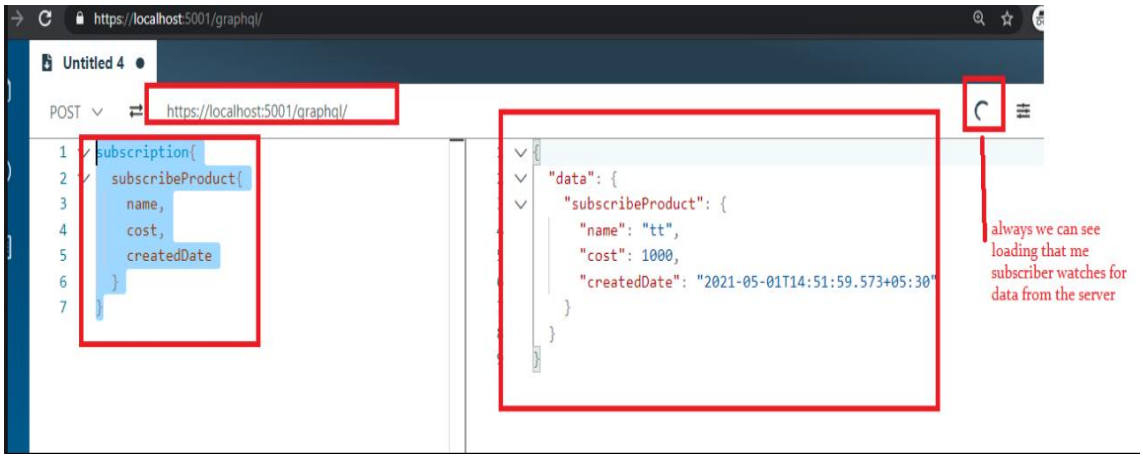


Figure II. 11. Exemple de l'abonnement

V.3. Schema stitching

Si les services web exposent leurs fonctionnalités via une API GraphQL, une fonctionnalité appelée Schema Stitching peut être utilisée pour combiner les différentes API GraphQL en une seule API.

L'assemblage de schémas peut être utilisé pour combiner des schémas GraphQL locaux et des schémas distants et également étendre les types des schémas combinés avec de nouveaux champs, ou ajouter des champs ou des entités à partir d'autres schémas[8].

C'est une opération pour fusionner les schémas de telle sorte que nous puissions simplement envoyer ces requêtes côte à côte.

Les étapes de cette opération sont :

- Le serveur obtient les URL vers les serveurs, utilise l'introspection GraphQL pour découvrir les schémas et fusionne tous les types ensemble.
- Lorsque vous exécutez une requête, chaque champ du type de requête initial est associé à un schéma particulier. Ainsi, le serveur sait que le premier objet doit aller à la première API et que le deuxième objet va à la deuxième API.
- Le serveur proxy qui créé à l'aide de l'assemblage de schémas finit par effectuer exactement de nombreuses requêtes GraphQL : une pour le premier champ initial et une pour le second[15](figure II.12).

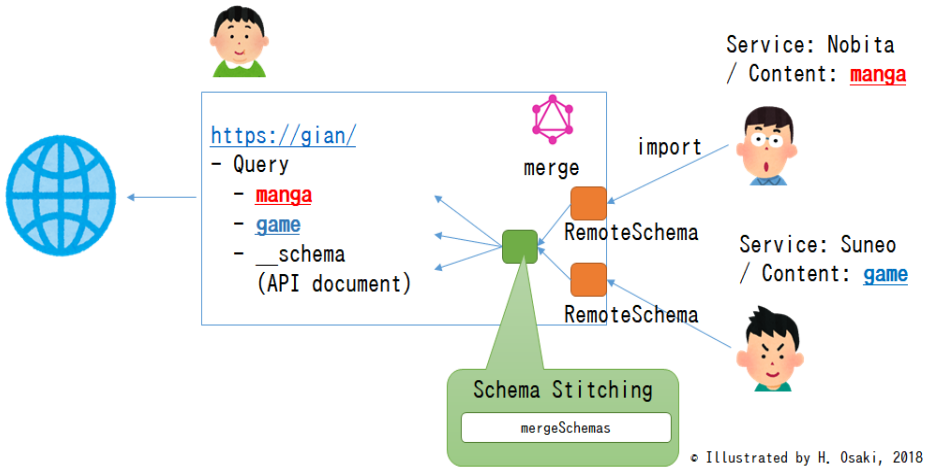


Figure II. 12. Schema stitching

VI. La composition par GraphQL

L'un des avantages de GraphQL a été de récupérer des données à partir de plusieurs sources de données, en combinant ce qui serait plusieurs appels d'API en un seul, puis en présentant les données au client, cela est identique à la définition de la composition des services web[8].

On va présenter un exemple d'une application web qui réalise la gestion des clients et des commandes qui utilise 3 services (profile service, produits services et ordres service) pour expliquer comment déroule l'exécution de la composition [16] (figure II.13).

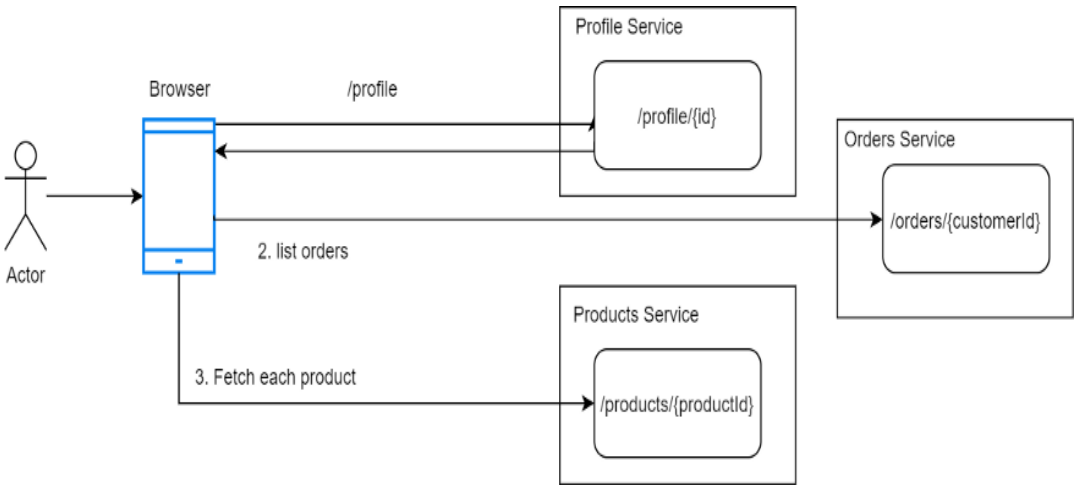


Figure II. 13. Exemple d'une application web

Ensuite, nous allons créer le schéma qui montre la création de graphql (diagramme de corrélation) (figure II.14) [17]:

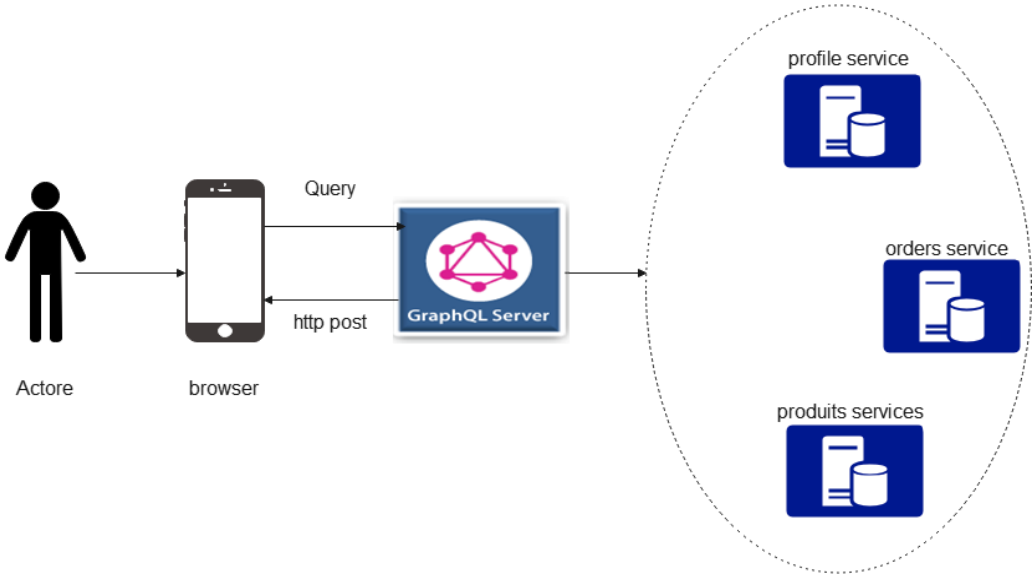


Figure II. 14. Diagramme de corrélation

Cette architecture est représentée dans un schéma de GraphQL comme suit (figure II.15) :

```
1  {
2    profile{                    Customer Service
3      title
4      firstName
5      lastName
6    orders{                     Order Service
7      total
8      items{
9        quantity
10       unitCost
11       productId
12       product{                Product Service
13         name
14       }
15     }
16   }
17 }
18 }
```

Figure II. 15. Schéma des services

VI.1. Composition en « série »« Parallèle »

Les stratégies d'exécution des requêtes GraphQL peuvent être classées en fonction de leurs caractéristiques. L'une de ces caractéristiques est liée au parallélisme d'exécution des requêtes. Cela regroupe les stratégies en série ou en parallèle.

En série : La stratégie d'exécution en série est une stratégie qui renvoie des plans d'exécution avec un seul chemin possible du sommet racine à la feuille et contenant tous les sommets (champs). Notez que tous les sommets d'un tel plan d'exécution doivent avoir un degré inférieur ou égal à 1.

Ex : décrire les données :

Type User {

Name: string

Age: Int

Messages:

Title: string

Date: date

}

En Parallèle : La stratégie d'exécution parallèle est une stratégie qui renvoie plans d'exécution, dont au moins un contient plusieurs chemins différents de la racine, et aucun de ces chemins ne contient tous les sommets. Une telle stratégie d'exécution est la méthode par défaut d'exécution des requêtes selon la spécification GraphQL[16].

Et Les outre stratégies d'exécution seront abordées dans les futures recherches.

Ex : décrire les données :

Type User {

Name: string

Age: Int

Messages: [message]

}

Type Message {

Title: string

Date: date

}

VII. Méta model GraphQL

Après une longue recherche du métamodèle GraphQL, nous avons remarqué qu'aucun métamodèle précis n'a encore été unifie normaliser par les organisations qualifiées. Mais il existe des travaux de recherches ont proposé des méta-modèles selon le domaine de recherche comme le papier [18], [19].

VIII. GraphQL vs API REST

GraphQL est une alternative aux applications basées sur REST. pour Comprendre les différences entre GraphQL et REST, On va présenter dans le tableau suivent la comparaison entre le REST et GraphQL[20].

GraphQL	**REST**
GraphQL exploite une architecture axée sur le client.	REST utilise une architecture pilotée par le serveur.
GraphQL ne prend en charge JSON que comme moyen de représenter les données.	REST prend en charge divers formats de données pour le rendre pratique pour ses utilisateurs.
GraphQL ne dispose pas d'un mécanisme de mise en cache automatique.	REST, contrairement à GraphQL, implémente automatiquement la mise en cache lors de ses opérations.
La gestion des versions d'API n'est pas disponible dans GraphQL.	REST prend en charge différentes versions d'API.
GraphQL est défini comme un langage de requête qui offre flexibilité et efficacité	REST est défini comme un style architectural largement considéré

dans la résolution de problèmes courants lors de l'intégration d'API.	comme une norme conventionnelle pour le développement d'API.
GraphQL est déployé sur HTTP via un point de terminaison unique qui offre toutes les fonctionnalités du service exposé.	REST est déployé sur un ensemble d'URL où chacune d'entre elles peut exposer une seule ressource.
L'identification des erreurs dans GraphQL est complexe et implique la gestion des codes d'état HTTP.	REST, d'autre part, peut utiliser les codes d'état HTTP pour identifier facilement les erreurs.
GraphQL utilise uniquement GraphiQL pour la documentation.	REST offre une large gamme d'options pour la documentation automatisée telles que l'API et le plan OpenAPI.

Tableau II.1. Comparaison entre le REST et GraphQL

IX. Avantages et inconvénients de GraphQL

Dans le tableau suivant on va lister quelques avantages et inconvénients da GraphQL :

Avantages	**Inconvénients**
❖ Un schéma GraphQL définit une source unique de vérité dans une application GraphQL. Il permet à une entreprise de fédérer l'ensemble de son API. ❖ Les appels GraphQL sont gérés par un seul aller-retour. Ce qui limite les problèmes de communication entre le client et le serveur. ❖GraphQL est introspectif. Un	❖ Les développeurs qui travaillent habituellement avec des API REST devront apprendre à utiliser GraphQL. ❖ GraphQL déplace l'essentiel de la charge de travail d'une requête de données du côté serveur, ce qui complique la tâche des développeurs serveur. ❖ L'un des inconvénients de l'API GraphQL des outils de commerce

client peut demander une liste des types de données disponibles. ❖ GraphQL permet de faire évoluer l'API d'une application sans dégrader les requêtes existantes.	est qu'elle ne prend pas en charge toutes les entités de la plate-forme contrairement à son API REST (elle prend en charge les produits, les catégories, les remises… mais il n'y a pas de support pour les paniers, les commandes et les clients).

Tableau II.2. Les avantages et les inconvénients de GraphQL

X. Conclusion

Dans ce chapitre nous avons présenté les concepts de bases du GraphQL tels que la structure et les types, la notion de la composition dans graphql et à la fin on va présenter quelques avantages et les inconvénients du GraphQL.

Le chapitre suivant est consacré à une introduction complète au MDE (Model Driven Engineering). Nous analysons la situation existante sur le terrain pour déterminer dans quelle mesure nous pouvons intégrer une combinaison de modèles dans un cycle de développement guidé par un modèle, et comment cette opération peut pleinement bénéficier des techniques et des résultats obtenus dans le domaine de la transformation.

Chapitre 03

Ingénierie Dirigée par les Modèles

« IDM »

I. Introduction

L'ingénierie dirigée par les modèles IDM (ou aussi Model Driven Engineering MDE) devient de plus en plus populaire, ou du moins attire de plus en plus l'attention, tant dans la recherche que dans les communautés industrielles. Il ne fait aucun doute que des standards comme UML ont joué un rôle important dans la diffusion du concept de modèle dans l'industrie. Fin 2000, l'OMG a lancé une nouvelle norme baptisée Model Driven Architecture (MDA). En fait, MDA offre une vision globale de l'avenir du développement logiciel. MDA est un ensemble de normes industrielles plutôt qu'une norme unique[21].

II. Définition

L'IDM est une pratique d'ingénierie des systèmes utilisant les capacités des technologies informatiques pour décrire à travers des modèles, concepts, et langages, à la fois le problème posé (besoin) et sa solution. Cette pratique est construite autour de deux notions fondamentales : les modèles et les transformations. En focalisant le raisonnement sur les modèles, l'IDM permet de travailler à un niveau d'abstraction supérieur et de vérifier sur une maquette numérique (ensemble de modèles qui traitent divers aspects d'un système) un ensemble de propriétés que l'on ne pouvait aborder auparavant qu'en fin de cycle de développement. L'un des apports supplémentaires du travail sur maquette numérique, en lieu et place du traditionnel code logiciel, est de fournir un référentiel central pour l'étude des différentes problématiques d'un système permettant à différents acteurs de s'intéresser à différents aspects du système à développer[22].

III. Les concepts de l'ingénierie dirigée par les modèles

Dans cette section, nous proposons de définir les concepts de base de l'IDM afin d'appréhender au mieux cette approche(figure III.1 [23]):

III.1. Modèle

Un modèle est une simplification d'un système construit dans une intention particulière. Le modèle doit pouvoir répondre à des questions en lieux et place du système modélisé et écrite dans un langage bien défini[23].

III.2. Méta Modèle

Un méta-modèle est un modèle qui définit le langage d'expression d'un modèle, c'est-à-dire le langage de modélisation. La notion de méta-modèle conduit à l'identification d'une relation, liant le modèle et le langage utilisé pour le construire. Un modèle est dit conforme à un méta-modèle si chacun de ses éléments (objets et relations) correspond à un élément du méta-modèle, et s'il respecte l'ensemble des propriétés (par exemple, contraintes d'Invariant) exprimées sur le méta-modèle. C'est sur ces principes de base que s'appuie l'OMG (Object Management Group) pour définir l'ensemble de ses standards, en particulier UML (Unified Modeling Language) dont le succès industriel est largement reconnu[24].

III.3. Méta Méta-Modèle

Le méta-modèle MOF (Meta-Object Facility) est un langage standard de définition de méta-modèles afin d'éviter l'augmentation démesurée des méta-modèles divers et incompatibles. En tant que modèle, le méta-modèle doit être défini à partir d'un langage de modélisation appelé méta méta-modèle. Pour limiter le nombre de niveaux d'abstraction, le méta méta-modèle doit alors avoir la propriété de méta-circularité, c'est-à-dire la capacité de se décrire lui-même[24].

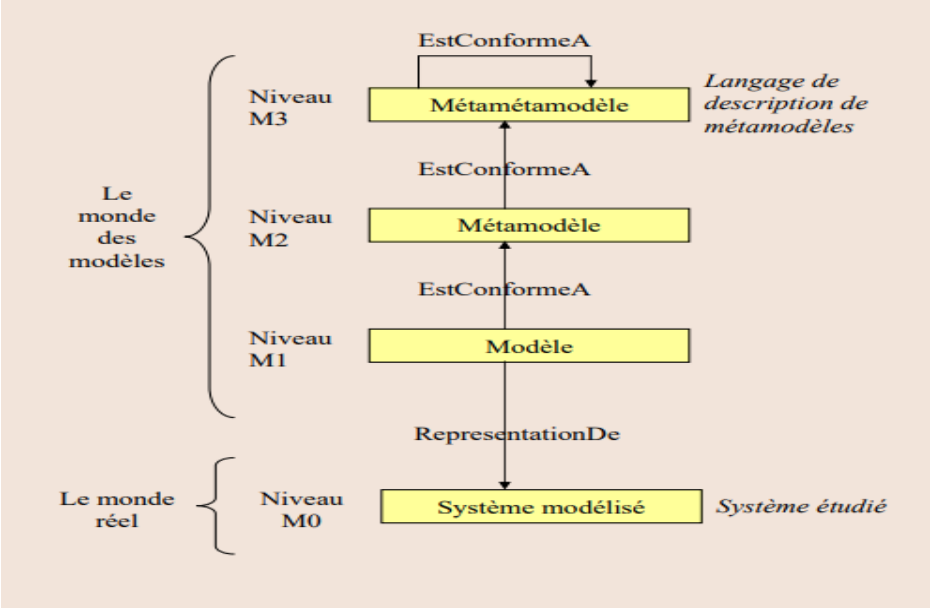


Figure III. 1. Conception d'IDM

III.4. Point de vue

Un point de vue est un concept général qui permet d'appliquer le principe de la construction, la sélection et la présentation d'une partie d'un modèle, répondant aux préoccupations particulières des parties prenantes.

La norme RM-ODP (Reference Model of Open Distributed Processing) définit cinq points de vue :

III.4.1. Point de vue Entreprise (le Pourquoi)

Il se concentre sur l'objectif, le domaine et les politiques du système. Il décrit les besoins de l'entreprise et la manière de les satisfaire.

III.4.2. Point de vue Information (le Quoi)

Il se concentre sur les informations et leur traitement. Il exprime la sémantique de l'application à l'aide de structures de données, de fonctions les manipulant et de propriétés d'intégrité sur ces données.

III.4.3. Point de vue Traitement (le Comment)

Il se concentre sur la conception fonctionnelle de l'application. Il décrit les fonctionnalités fournies par le système et sa décomposition fonctionnelle.

III.4.4. Point de vue Ingénierie (le Où)

Il représente les plateformes support de l'application. Il se concentre sur les mécanismes et les fonctions nécessaires pour soutenir les interactions entre les objets distribués dans le système. Il décrit les fonctions et services de base que doit fournir l'infrastructure d'exécution pour que l'on puisse construire les mécanismes assurant l'interopérabilité en environnement hétérogène.

III.4.5. Point de vue Technologie (Avec Quoi)

Il se concentre sur le choix de la technologie du système. Il décrit les technologies choisies pour fournir le traitement, les différentes fonctionnalités du système et la présentation des informations[4].

IV. Transformation de modèle

La transformation de modèles est une opération qui consiste à générer un ou plusieurs modèles cibles conformément à leur méta-modèle à partir d'un ou de plusieurs modèles sources conformément à leur méta-modèle.

La transformation est dite raffinement (Transformation endogène) si les modèles sources et cibles sont conformes au même méta-modèle (source et cible sont dans le même espace technologique). La transformation est dite projection(Transformation exogène) si les modèles sources et cibles ne sont pas conformes au même méta-modèle et elle se fait entre deux méta-modèles différents (source et cible sont dans deux espaces technologiques différents)(figure III.2 [24]).

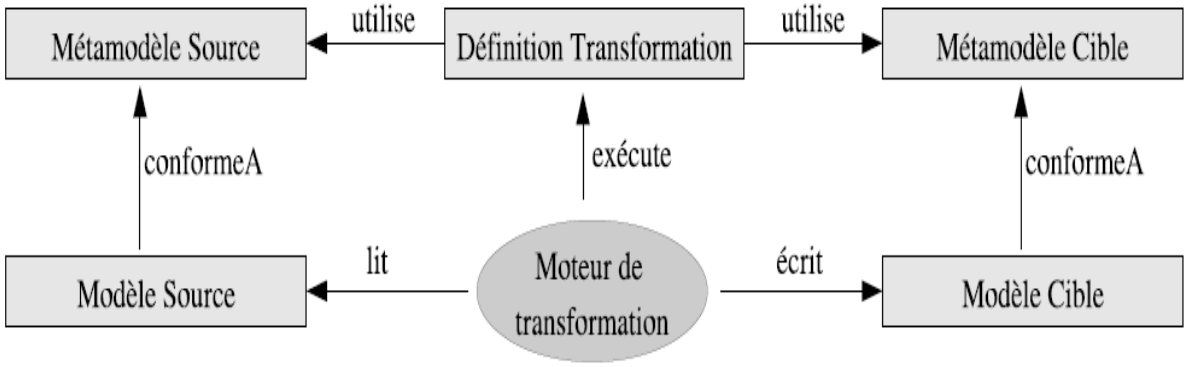


Figure III. 2. Diagramme générale de transformation

IV.1. Classification des approches de transformation de modèles

Les transformations de modèles peuvent être partagées en deux grandes classes : les transformations « Modèle vers Modèle » et les transformations « Modèle vers Code ».

IV.1.1. Transformations Modèle vers Modèle (M2M : Model-to-Model)

Les conversions de type M2M incluent la conversion de modèles sources en modèles cible, ces modèles peuvent être des instances de différents méta modèles. Ils fournissent Transformations plus modulaires et maintenables, et permet de générer plusieurs Modèle intermédiaire avant d'atteindre le modèle final. Ces modèles intermédiaires peuvent Permet d'étudier différentes vues du système, optimisation, validation propriétés et leur validation[3].

IV.1.2. Transformation Modèle vers Code (M2T : Model-to-Text)

Il existe deux types de traductions « modèle vers code » : la première est basée sur le principe du visiteur, qui consiste à fournir un mécanisme d'accès pour parcourir la représentation interne du modèle et créer du code. Et le second est le modèle source du principe d'accès à l'information basé sur un modèle (basé sur un modèle), qui utilise des fragments de méta-code dans le code objet [3].

IV.2. Langage de transformation

IV.2.1. EMF/GEF/GMF

Les outils de méta-modélisation EMF (Eclipse Modeling Framework), GEF (Graphical Editing Framework) et GMF (Graphical Modeling Framework) peuvent être considérés comme étant des solutions génériques. Il s'agit d'environnements de développement qui permettent à l'utilisateur de programmer tout le comportement graphique et toutes les notations graphiques spécifiques qu'il souhaite. Cela laisse beaucoup de liberté pour construire des éditeurs graphiques bien adaptés aux besoins des experts du domaine. Toutefois, cette généricité requiert une grande expertise en programmation et au final, la quantité de code nécessaire peut décourager les développeurs[22].

IV.2.1. Acceleo

Acceleo est une implémentation de la norme OMG Model to Text Language (M2T) et une implémentation de la norme MOF Models to Text Transformation (MOFM2T) [25]. L'Acceleo peut être interprété comme un langage orienté objet où les modules sont les classes, et les modèles sont les méthodes à l'intérieur d'un module. Acceleo est un générateur de code de langage qui permet de générer un fichier structuré à partir d'un modèle EMF, la sortie est un texte qui peut être un langage de programmation ou un autre formalisme. Acceleo impose de définir une EMF métamodèle et un modèle conforme au métamodèle qui résultera en texte [26] (figure III.3).



Figure III. 3. Transformation Acceleo

IV.2.2. ATL (Atlas Transformation Langage)

ATL est un langage hybride (déclaratif + impératif), exprimant les contraintes de transformation de modèle, les ensembles de règle et l'application des règles sont exprimés dans OCL (Object Constraint Language), un plug-in éclipse développé pour ATL.

La transformation de modèle via des règles de transformations écrites au niveau méta-modèle, peut être appliquée à notre problématique en utilisant ATL comme langage de transformation et les outils développent autour de ce langage. Cela nécessite alors l'exploitation d'un ou plusieurs méta-modèles adéquats (figure III.4).

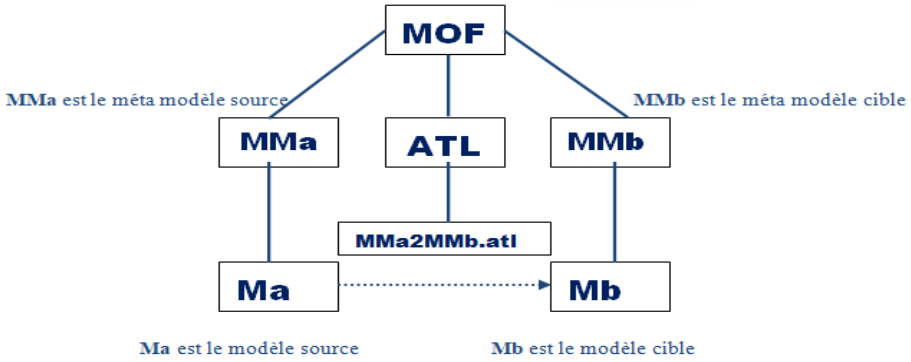


Figure III. 4. Transformation ATL

V. L'architecture dirigée par les modèles (MDA)

L'ingénierie dirigée par les modèles est un domaine basé sur les modèles et les technologies liées à leurs manipulations. Pour la pratique, il existe plusieurs manières d'utiliser les modèles dans le processus de développement d'un système. L'approche la

plus développée et la plus utilisée est L'Architecture Dirigée par les Modèles (MDA : Model Driven Architecture)[4].

L'OMG a défini l'architecture MDA en 2000 pour promulguer de bonnes pratiques de modélisation et exploiter pleinement les avantages des modèles. Cette approche vise à mettre en valeur les qualités intrinsèques des modèles, telles que la pérennité, la productivité et la prise en compte des plateformes d'exécution. MDA inclut pour cela la définition de plusieurs standards, notamment UML, MOF et XMI[24] (figure III.5).

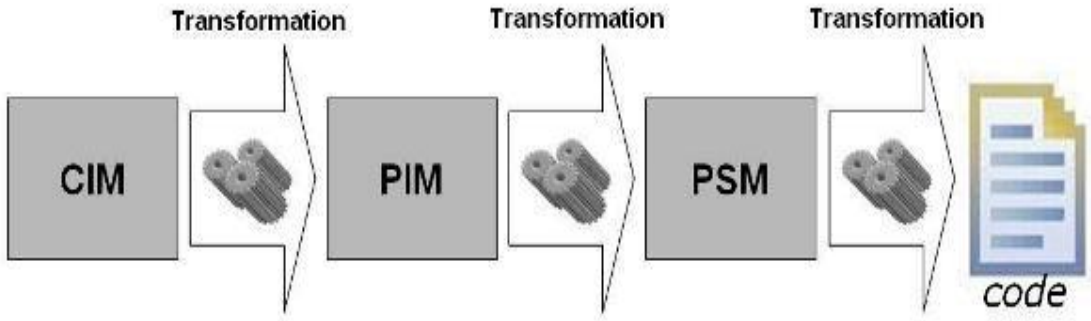


Figure III. 5. L'architecture de MDA.

V.1. Le principe de base d'MDA

Le principe clé et initial de MDA consiste à s'appuyer sur le standard UML pour Décrire séparément des modèles pour les différentes phases du cycle de développement d'une application. Plus précisément, le MDA préconise l'élaboration de modèles (figure III.6[27]).

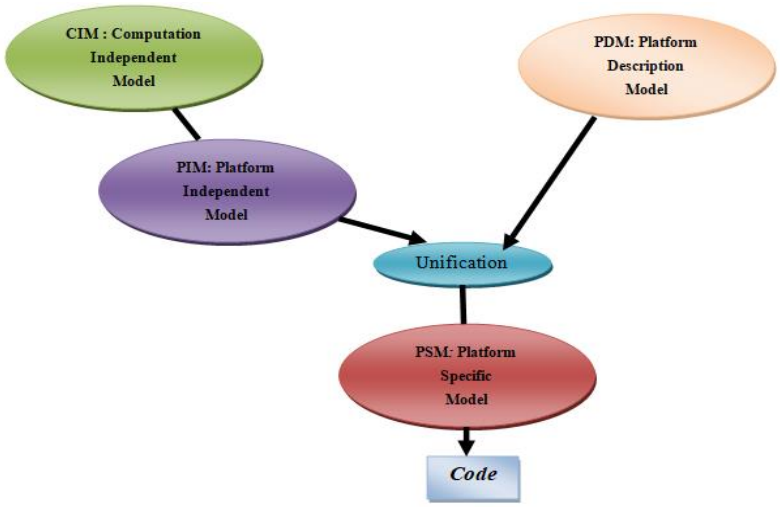


Figure III. 6. Les principes de l'approche MDA

V.2. L'objectif du MDA

Est l'élaboration de modèles pérennes (PIM), indépendants des détails techniques des plates-formes d'exécution (J2EE, .Net, PHP, etc.), afin de permettre la génération automatique de la totalité des modèles de code (PSM) et d'obtenir un gain significatif de productivité.

Le passage de PIM à PSM fait intervenir des mécanismes de composition et de transformation de modèle avec un modèle de description de la plate-forme (Platform Description Model – PDM), un modèle de description de la qualité de service (Quality of Service – QoS) ou autres modèles comme les non-fonctionnels (par exemple, la sécurité) [24].

V.3. Les couches du MDA

- **CIM:**(Computation Independent Model) est le modèle d'analyse de base du métier ou du domaine d'application. Ce modèle est indépendant de tout système informatique, il décrit les concepts de l'activité métier, le savoir-faire, les processus, la terminologie et les règles de gestion (de haut niveau), il décrit aussi la situation dans laquelle le système est utilisé. C'est un modèle de très longue durée de vie, il n'est modifié uniquement que si les connaissances ou les besoins métier changent[22].
- **PIM:**(Platform Independent Model) est un modèle de conception, il décrit le système informatique indépendamment de toute plateforme technique et de toute technologie utilisée pour déployer l'application. Il représente la logique métier spécifique au système (fonctionnement des entités et des services)[22].
- **PSM:**(Platform Specific Model) le modèle PSM combine les spécifications dans le modèle PIM avec les détails qui décrivent comment le système utilise un type particulier de plateforme. Notons qu'un modèle PSM peut être dérivé à partir d'un ou plusieurs modèles PIM (chaque modèle PIM a un but conceptuel différent)[24].
- **PDM :** (Platform Description Model) est un modèle de transformation pour permettre le passage du PIM vers le PSM, spécifié une plateforme. Contient des informations pour la transformation des modèles vers une plateforme[27].

VI. Conclusion

Dans ce chapitre nous sommes intéressés à l'ingénierie dirigée par les modèles et ses différents concepts. Nous avons survolé également les techniques de transformation de modèle. Avec une présentation d'architecture MDA et ses principes (CIM, PIM, PSM et PDM).

Dans le prochain chapitre on va présenter notre approche proposée pour générer les requêtes GraphQL pour les compositions des services web.

Chapitre 04
Conception

I. Introduction

Dans les chapitres précédents, nous avons parlé en détail de la composition des services Web, de GraphQL et de l'ingénierie dirigée par les modèles. Dans ce chapitre, nous proposons une approche pour générer des requêtes GraphQL semi-automatique pour réaliser une composition des services en basant sur la démarche IDM.

II. L'approche proposée

Dans notre démarche nous restreignons par les phases de l'approche MDA (Model driven architecture). Il s'agit d'une implémentation de la norme de l'Object Management Group (OMG) pour les différentes transformations (M2M ou M2T) présenté dans le chapitre 3. Et comme nous l'avons clarifié dans le chapitre 2, il n'existe actuellement aucun métamodèle GraphQL standardisé validé selon les normes, c'est ce qui nous a fait suivre le chemin de M2T.

III. Méthode de l'approche proposée

Dans cette sous-section, nous Proposons la démarche suivie pour mettre en œuvre de l'approche proposée. Ces étapes sont détaillées ci-dessous, comme le montre la figure IV.1.

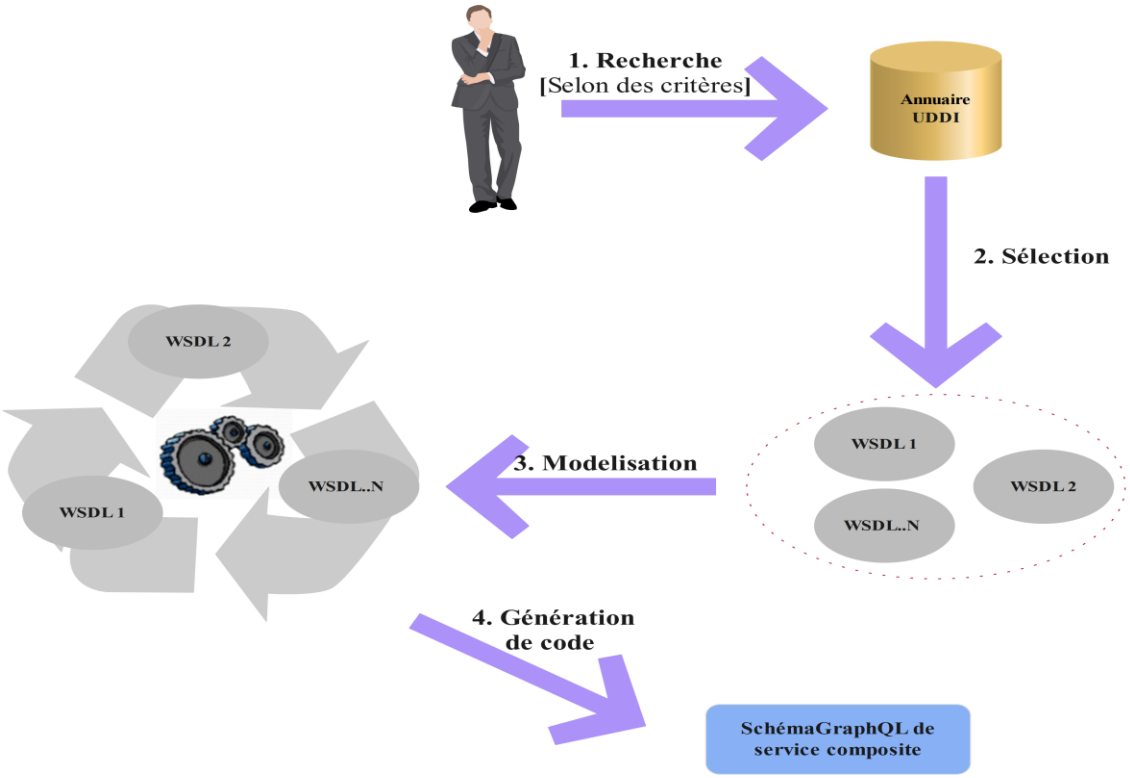


Figure IV. 1. Les étapes de l'approche proposée

- **Recherche**

Le concepteur commence cette phase par une recherche dans un annuaire sur des services web selon des critères de recherche dans UDDI cela contient la spécification WSDL de chaque service.

- **Sélection**

Apres la recherche, le concepteur sélectionne les services utiles.

- **Modélisation (composition)**

Cette étape consiste, à composer les services sélectionnes à l'étape précédente, selon un schéma de composition proposée.

- **Génération de code**

Dans cette phase, on va générer automatiquement le fichier GraphQL à partir de notre composition.

IV. Démarche de transformation

Dans cette section, nous décrirons en détail les étapes de notre approche illustrée dans la figure IV.3.

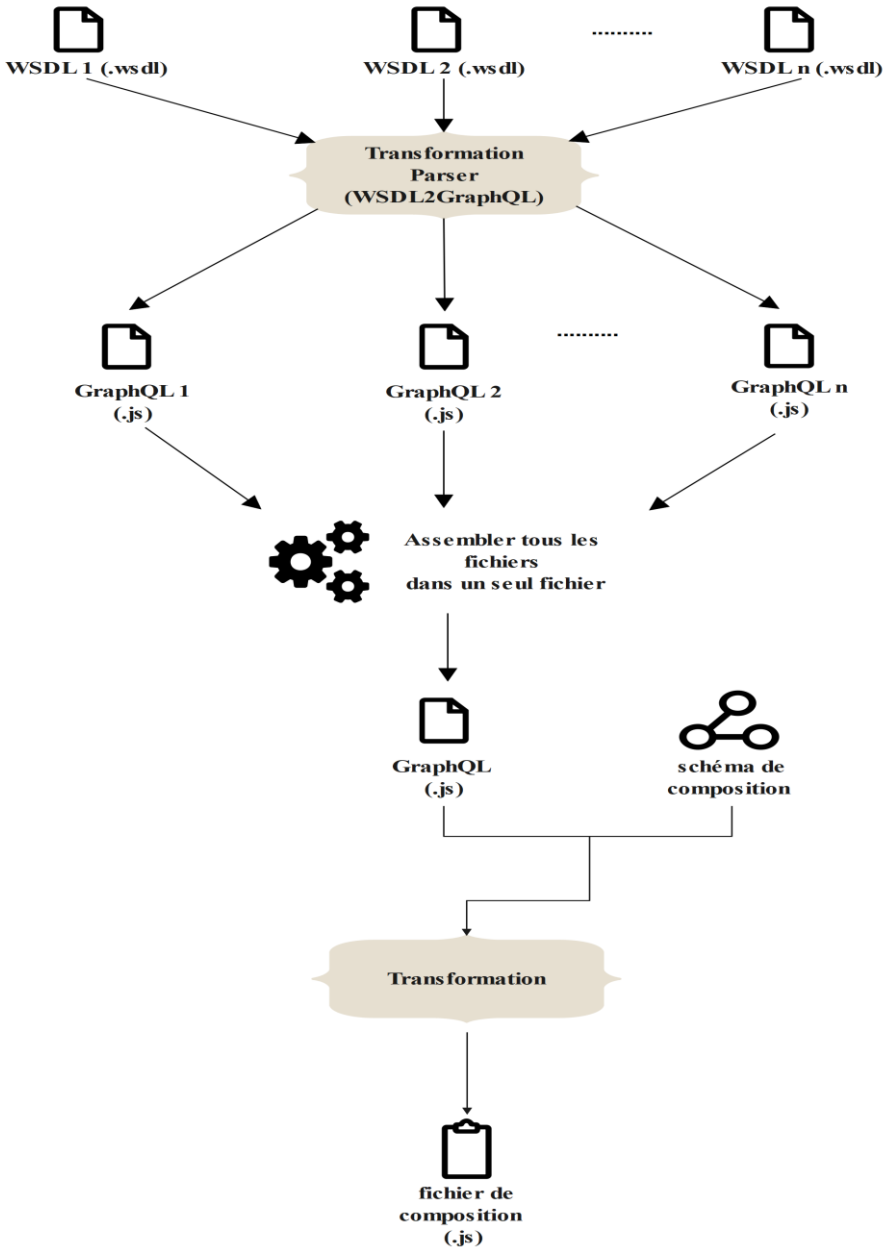


Figure. 2. Les étapes de la transformation

IV.1. Description les étapes de la transformation

- **Etape 01 (Transformation (WSDL2GraphQL))**

Dans cette étape, nous convertissons le fichier WSDL propre a chaque service a une requête GraphQL (Schema + server). En utilisant une méthode basée sur une parser, qui inspecte le document WSDL et génère le code correspondant pour utiliser pour cela on a définie des règles de transformation (définies dans la section V).

➢ **Etape 02 (Assemblage des fichiers GraphQL)**

Cette étape consiste à assembler toutes les fichiers GraphQL produise dans l'étape précédente par le biais la technique d'assemblage de schémas "Schema stitching".

➢ **Etape 03 (Génération de la requête de composition)**

Dans cette étape, nous utilisons le fichier GraphQL créé pour réaliser un fichier graphql avec son serveur d'un service composite selon un schéma de composition. Nous parlerons de cette étape dans la section VI.

V. Les règles de transformations (WSDL2GraphQL)

Dans cette section, on va présenter les règles de transformation WSDL à GraphQL.

Classe WSDL	Code GraphQL	Signification dans GraphQL
DEFINITION	Const typeDefs { Type Query {} Type mutation {} Type Subscription {} Type scalaire {} Type objet {} Type input {} Type enum {} }	# API Name: name_definition **« Schema »**
ELEMENT	Name_element : type	**« Field »**
Type Schema	Type name_complexe_type { Name_element: type }	**« Type scalaire »**
MESSAGE	Type name_message { Name_part: name_complexe_type (t	**« ObjectType »**

	}	
PART	Name_part: name_complexe_type (type)	**« Field»**
PORTTYPE	Const resolvers{ Operation:{ Field () =>{return }; }, };	**« resolvers »**
OPERATION	Type Query { Field: type Field: object } Type Mutation { Field (input: var_input): object } Type Subscription { Field : object }	**«Input (||or &&) ObjectType »**
PORT	server. start(() => console.log('server is running on localhost :4000'))	**« Field»**
SERVICE	Import{ GraphQLServer}from 'graphql-yoga' Const server= new GraphQLServer({Name_definition, name_porttype}) server. start(() => console.log('server is running on localhost :4000'))	**« server »**
IMPORT	Const {}	**« Const »**

Tableau IV.1. Les règles de transformation WSDL2GraphQL

VI. La génération d'un fichier GraphQL de composition

Dans cette étape, on va ajouter les operateurs de la composition au schéma assemblé GraphQL selon le schéma de composition introduit par le concepteur. le résultat de cette étape est le fichier GraphQL de service composite (figure IV.3).

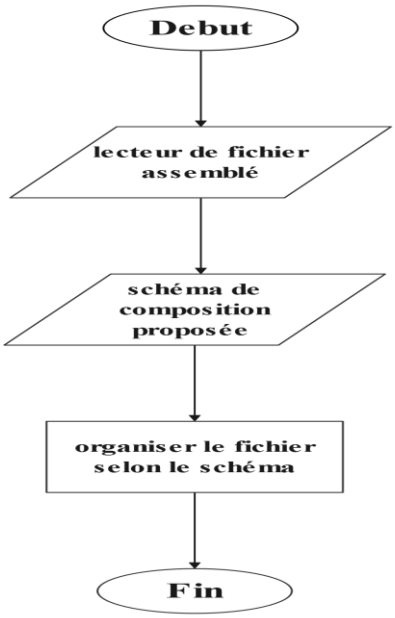


Figure IV.3. Organigramme de composition

VII. Conclusion

Dans ce chapitre, nous avons présenté une approche semi-automatique pour générer une requête GraphQL d'une composition des services web en suivent la démarche MDA. Pour faire cette transformation on a défini des règles de transformation.

Dans le chapitre suivant, on va présenter l'implémentation de cette approche avec un cas d'étude.

Chapitre 05
Implémentation et Étude de cas

I. Introduction

Après avoir exposé notre approche de modélisation, nous allons maintenant décrire dans la phase de mise en œuvre de cette approche, Afin d'illustrer les différentes idées et concepts impliqués dans notre approche, nous donnerons ensuite un exemple sous forme d'étude de cas.

II. Les langages, les outils et les bibliothèques

Pour illustrer la solution proposée dans notre travail; nous avons appliqué notre approche par l'utilisation de ces outils :

II.1.Python ; Spyder :

Python est un langage de programmation haut niveau, Python est un excellent langage de programmation orienté objet, interprété et interactif. Il est souvent comparé (favorablement bien sûr) à Ruby, Visual Basic, Schème ou Java ... etc. Python combine un pouvoir remarquable avec une syntaxe très claire. Il comporte des modules, des classes, des exceptions, des types de données dynamiques de très haut niveau et le typage dynamique. Il existe des interfaces vers de nombreux appels systèmes et bibliothèques, ainsi que vers différents systèmes de fenêtrage[28].

Nous avons utilisé Python en particulier Spyder : pour écrire notre programme de transformation.



Figure V. 1. Python ; Spyder

II.2. Zeep

Zeep est un module en python, Zeep inspecte le document WSDL et génère le code correspondant pour utiliser les services et les types du document. Cela fournit une interface de programmation facile à utiliser pour un serveur SOAP[29].

Nous utilisons Zeep pour inspecter le fichier WSDL et générer le code GraphQL correspondant.



Figure V. 2. Zeep

II.3. Node.js

Node.js est une plate-forme logicielle pour les applications réseau qui vise à simplifier la mise en œuvre de services réseau rapides et évolutifs, y compris les applications Web. Le projet a été lancé en 2009 et a depuis gagné du terrain dans les communautés de développeurs.

L'architecture de la plateforme Node.js se distingue par son modèle d'exécution basé sur les événements. Il utilise un seul thread, le thread d'événement, pour exécuter le code de l'application. L'exécution dans un seul thread signifie qu'il n'y a pas de simultanéité au niveau de l'application et que toutes les demandes entrantes sont gérées par le même thread. Sans simultanéité au niveau de l'application. D'une part, cette approche simplifie le développement d'applications [30].

Nous utilisons Node.js Pour créer un nouveau projet et installer le fichier GraphQL.js dans notre répertoire.



Figure V. 3. Node.js

II.4. Visual studio code

Visual Studio Code est un éditeur de code simplifié, qui est gratuit et développé en open source par Microsoft. Il fonctionne sous Windows, mac OS et Linux. Il fournit aux développeurs à la fois un environnement de développement intégré avec des outils permettant de faire avancer les projets techniques, de l'édition, à la construction, jusqu'au débogage.

Visual Studio Code permet également aux développeurs de créer et d'utiliser des extensions grâce à son API, afin de personnaliser leur utilisation de l'outil. Il est livré avec un support pour JavaScript, Type Script et Node.js[31].

Nous avons utilisé Visual Studio Code pour exécuter notre projet Node.js;



Figure V. 4. Visual studio code

II.5. GraphQL-yoga

GraphQL-yoga est un serveur GraphQL complet mettant l'accent sur la facilité d'utilisation, les performances et une excellente expérience de développement. De plus, il prend en charge tous les clients GraphQL comme Apollo[32].

Nous avons créé un serveur GraphQL-yoga pour répondre à nos requêtes obtenu par la transformation de wsdl vers les fichiers graphql.js.



Figure V. 5. GraphQL-yoga

III. Etude de cas

III.1. Présentation

III.1.1 Description des services web utilisés

Dans cette partie, nous donnerons une description complète de chaque service Web et quels sont ses paramètres d'entrée et de sortie.

- **Service «Etat civil »**

Service basé sur numéro d'identification national qui renvoie l'état civil d'un citoyen donné (prénom, nom de famille, date de naissance, sexe, âge, situation familiale).

- **Service «Diplôme universitaire»**

Ce service permet de restituer le diplôme universitaire obtenu par un citoyen donné. Le service suit les recherches en se basant sur le nom de famille, le prénom et la date de naissance du citoyen.

- **Service «Expérience de travail»**

Ce service permet de restituer toutes les expériences professionnelles d'un citoyen donné. Le service consiste en des recherches basées sur l'identifiant du diplôme. La recherche est faite pour un diplôme et un travail donnés.

- **Service « Santé »**

Ce service permet de renvoyer les informations médicales d'un citoyen donné. Le service consiste en des recherches basées sur le nom de famille, le prénom et la date de naissance.

- **Service « Casier judiciaire »**

Ce service restitue, selon le nom et prénom du citoyen et sa date de naissance, un casier judiciaire selon le type de casier judiciaire demandé.

III.1.2. Scenario de la composition

Pour illustrer la solution proposée dans ce travail, nous proposons une étude qu'une institution qui souhaite précéder à un recrutement d'une personne. Pour le processus de recrutement d'un personnel, nous devons récupérer les informations du candidat (état civil, diplômes, casier judiciaire, expérience de travail, etc.…). Ces informations peuvent être transmises par d'autres institutions (ministère de l'Intérieur, ministère de l'Éducation, ministère de la Justice, …), chaque institution est un moyen de Service Web (figure V. 6).

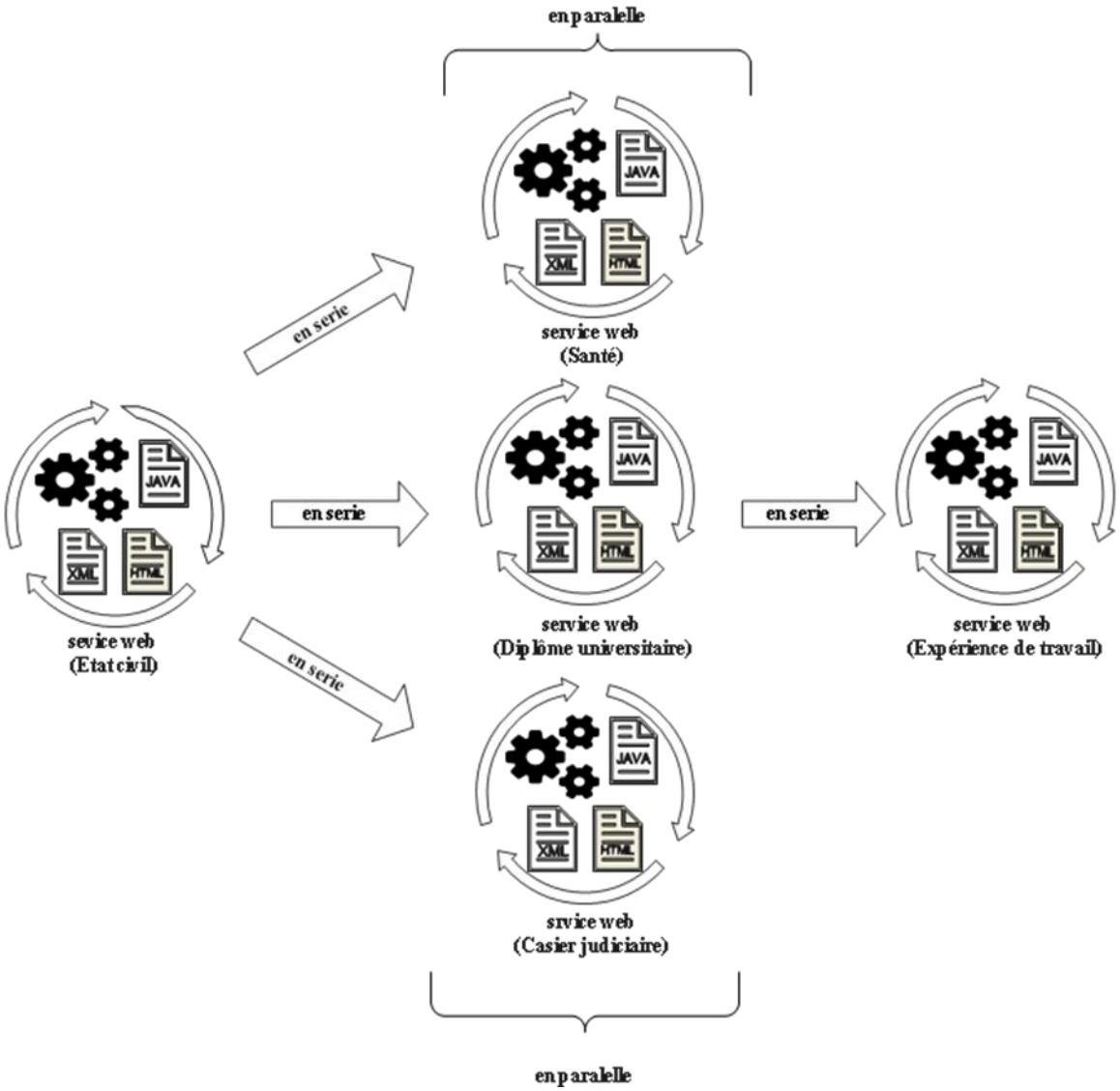


Figure V. 6. Composition des services

III.2. Représentation de scénario à l'aide de L'approche

Dans cette section, nous expliquerons comment appliquer notre approche à l'application présentée dans la section précédente.

L'explication déterminée dans les étapes suivantes.

III.2.1. Premier étape (la transformation WSDL2GraphQL)

Cette étape consiste à convertir WSDL en GraphQL. Nous extrayons donc le fichier wsdl pour chaque service (figure V.7).

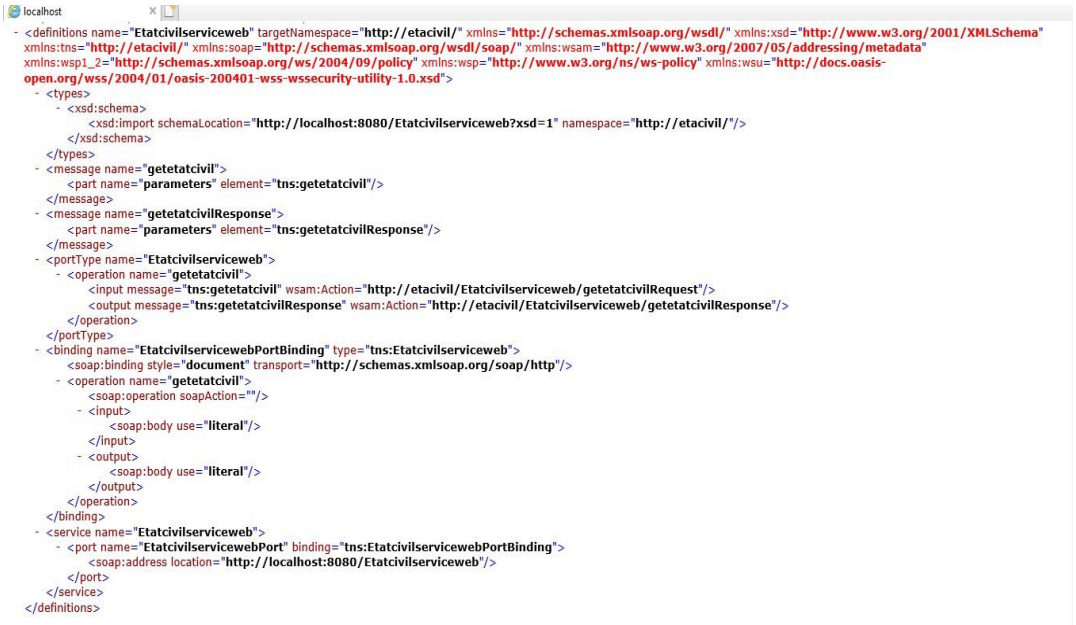


Figure V. 7. WSDL de service Etat civil



Figure V. 8. XSD de service Etat civil

Ensuite, nous créons deux programmes, le premier pour extraire l'index.js (schéma, resolvers et server) (en utilisant le module Zeep) (figure V. 9).

```
// Create your server
const server = createServer({
  schema: {
    typeDefs: /* GraphQL */ `

      type personne  {
        nom: String, prenom: String, dateNaissance: String, lieuNaissance: String, sexe: String, situationFamiliale: String, adresse: String
      }
      type getetatcivil {

        Num_iden_nationnale: Int
      }
      type getetatcivilResponse {

        return: personne
      }
      type Query {
        getgetetatcivil:[getetatcivil],
        getgetetatcivilResponse:[personne]
      }
```

```
      }

    `,
    resolvers: {

      Query: {
        getgetetatcivil: () => {return getetatcivil},
        getgetetatcivilResponse: () => {return personne },

      },
    },
  },
})

// start the server and explore http://localhost:4000/graphql
server.start()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER
C:\Program Files\nodejs\node.exe .\src\index.js
Yoga - Running GraphQL Server at http://127.0.0.1:4000/graphql
```

Figure V. 9. index.js

Apres l'exécution de ce fichier, le résultat dans la (figure V.10).

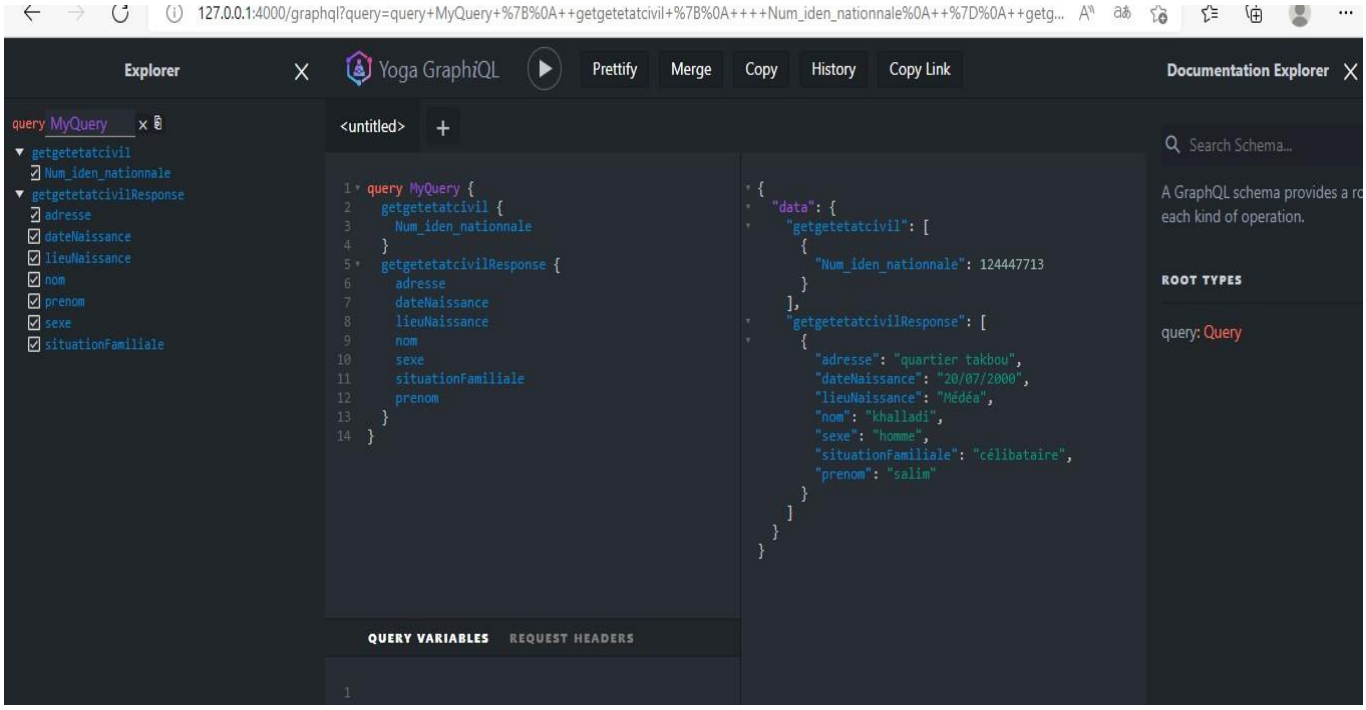
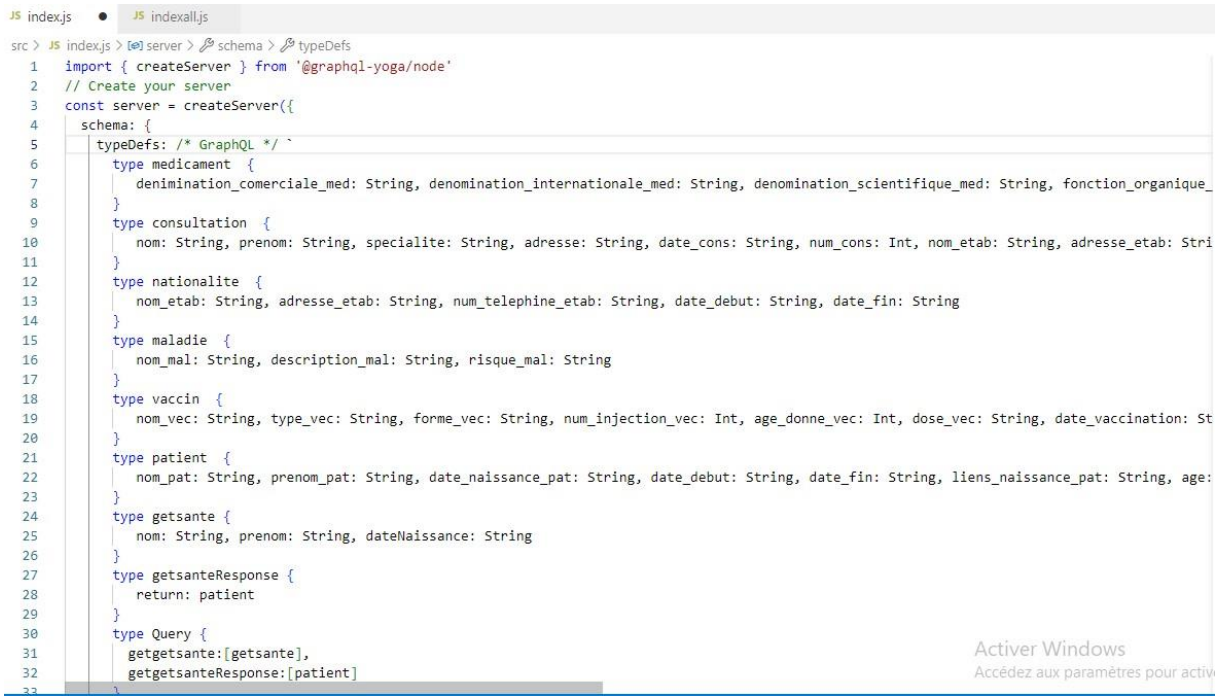


Figure V. 10. Le résultat de l'exécution de fichier GraphQL de la figure V.9

III.2.2. Seconde étape (assemblage des fichiers)

Apres l'exécution de toutes les fichiers GraphQL des services (pour valider la transformation) on va assembles ces derniers dans un seul fichier.

Ce fichier contient les schémas des services et un resolver de toutes les opérations (figure V.11).



index.js ● indexall.js

src > index.js > server > schema > typeDefs

```
type travail  {
   dateDebutPer: String, dateFinPer: String, nomEntreprise: String, activite: String, nomProffition: String, salaireProffition: String, dur
}
type personnet {
   nomP: String, prenomP: String, dateNissance: String, lieuNissance: String, adresse: String, id_dep: Int, institution: String, date_debut
}
type gettravail {

   id_dep: Int
}
type gettravailResponse {

   return: personnet
}
type Query {
  getgettravail:[gettravail],
  getgettravailResponse:[personnet]
}
type personne  {
   nom: String, prenom: String, dateNaissance: String, lieuNaissance: String, sexe: String, situationFamiliale: String, adresse: String
}
type getetatcivil {

  Num_iden_nationnale: Int
}
type getetatcivilResponse {

  return: personne
}
type Query {
  getgetetatcivil:[getetatcivil],
  getgetetatcivilResponse:[personne]
}
```

Activer Windows
Accédez aux paramètres pour active

ex.js > server > schema > typeDefs

```
type emprisonnement  {
   nomPrison: String, duree: String
}
type amende  {
   montant: String
}
type infraction  {
   amende: amende, emprisonnement: emprisonnement, numJugement: String, dateJugement: String, nomCour: String, adressCour: String
}
type accusation  {
   sanction: infraction, description: String, nom: String
}
type personnec  {
   nom: String, prenom: String, lieuNissance: String, dateNissance: String, adress: String, accusation: accusation, infraction: accusation,
}
type getcasier {

   nomc: String, prenomc: String, dataNaissance: String, casier: Int
}
type getcasierResponse {

   return: personnec
}
type Query {
  getgetcasier:[getcasier],
  getgetcasierResponse:[personnec]
}
```

```
type etudiant  {
   nom_etudiant: String, prenom_etudiant: String, date_naissance: String, diplome: String, id_dip: Int, universite: String
}
type getdiplome {

   nom: String, prenom: String, dateNaissance: String, diplome: String
}
type getdiplomeResponse {

   return: etudiant
}
type Query {
  getgetdiplome:[getdiplome],
  getgetdiplomeResponse:[etudiant]
}
```

ndex.js ● JS indexall.js

> JS index.js > [@] server

```
    resolvers: {


      Query: {
        getgetsante: () => {return getsante},
        getgetsanteResponse: () => {return patient },
      },
      Query: {
        getgettravail: () => {return gettravail},
        getgettravailResponse: () => {return personnet },
      },
      Query: {
         getgetetatcivil: () => {return getetatcivil},
         getgetetatcivilResponse: () => {return personne },
      },
      Query: {
         getgetcasier: () => {return getcasier},
         getgetcasierResponse: () => {return personnec },
      },
      Query: {
         getgetdiplome: () => {return getdiplome},
         getgetdiplomeResponse: () => {return etudiant },
      },

    },
  },
})

// start the server and explore http://localhost:4000/graphql
server.start()
```

Figure V. 11. Schéma Stitching

III.2.3. Dernier étape (génération la requête GraphQL)

Dans cette étape, on va étudier le schéma de composition et on va aménager le fichier assemblé selon le schéma de composition (figure V.12).

Cette figure V.12 est similaire à la figure V.11 précédentes, mais nous avons ajouté ces modifications

```
type getcasier {
   nomc: personne.nom, prenomc: personne.prenom, dataNaissance: personne.dateNaissance, casier: Int
}

type getdiplome {
   nom: personne.nom, prenom: personne.prenom, dateNaissance: personne.dateNaissance, diplome: String
}

type getsante {
   nom: personne.nom, prenom: personne.prenom, dateNaissance: personne.dateNaissance
}

   type gettravail {
      id_dep: etudiant.id_dip
   }
```

```
    resolvers: {

      Query: {
        getgetetatcivil: () => {return getetatcivil},
        getgetetatcivilResponse: () => {return personne },
      },
      Query: {
        getgetcasier: () => {return getcasier},
        getgetcasierResponse: () => {return personnec },
      },
      Query: {
        getgetdiplome: () => {return getdiplome},
        getgetdiplomeResponse: () => {return etudiant },
      },
      Query: {
        getgetsante: () => {return getsante},
        getgetsanteResponse: () => {return patient },
      },
      Query: {
        getgettravail: () => {return gettravail},
        getgettravailResponse: () => {return personnet },
      },

    },
  },
})

// start the server and explore http://localhost:4000/graphql
server.start()
```

Figure V. 12. Schéma de composition

IV. Conclusion

Dans ce chapitre, Nous avons commencé par présentation des différentes technologies utilisées pour le développement de notre approche. Ensuite, nous avons expliqué le fonctionnement détaillé de notre application sur une étude de cas.

Conclusion générale

Dans ce mémoire nous sommes intéressés à la transformation les WSDL vers des spécifications GraphQL. Nous avons commencé avec une étude détaillée sur les services web et sur le plus important point de notre travail GraphQL suivis d'une représentation d'une technique permettant la génération des requêtes GraphQL d'un service composite à partir des fichiers WSDL des services web sélectionnée à partir des critères de recherche. Nous avons proposé une approche pour la modélisation de la composition des services web. Le travail que nous avons présenté se situe à l'intersection de plusieurs axes de recherche dont nous pouvons citer : la modélisation de la composition des services web, la méta-modélisation de WSDL et la transformation Des modèles.

La technique proposée permet de créer un fichier GraphQL lisible après exploration des composants qui intègrent le fichier WSDL. En effet, une série des règles de transformation sont établies pour traiter les composants du fichier WSDL. Pour cela, nous avons implémenté notre méthode proposée dans un outil python qui permet la conversion automatique des fichiers WSDL en fichiers GraphQL.

L'approche proposée permet de traiter une composition des services web avec des stratégies en série et en parallèle. Dans un travail futur nous comptons étendre l'approche pour couvrir un sous-ensemble plus large des stratégies de la composition comme les tests, les boucles ….etc.

Bibliographie

Bibliographie

[1] B. Abdelbasset, “Composition de service web dans le cloud computing,” Université au Québec, 2018.

[2] M. Academique, “Thème Une approche formelle pour la composition des services web,” pp. 2014–2015, 2015.

[3] A. Klalib, “proposition d’une approche pour la génération automatique d’un fichier BPEL,” université de medea, 2016.

[4] C. Jury, “Une approche de composition de services Web à l ’ aide des Réseaux de Petri orienté s objet,” 2014.

[5] D. Benmerzoug, “Composition de Services Web Composition de Services Web Plan :”

[6] C. E. Partielle and A. Alkamari, “Composition de services web par appariement de signatures,” 2008.

[7] “GraphQL | A query language for your API.” https://graphql.org/ (accessed Mar. 03, 2022).

[8] V. Touronen, “Microservice architecture patterns with GraphQL,” UNIVERSITY OF HELSINKI, 2019.

[9] S. Bruna, *GraphQL in Action*. 2021.

[10] O. Hartig and J. Hidders, “Defining schemas for property graphs by using the GraphQL schema definition language,” *Proc. ACM SIGACT-SIGMOD-SIGART Symp. Princ. Database Syst.*, 2019, doi: 10.1145/3327964.3328495.

[11] “Schemas and Types | GraphQL.” https://graphql.org/learn/schema/ (accessed May 19, 2022).

[12] “GraphQL Server Basics: GraphQL Schemas, TypeDefs & Resolvers | Prisma.” https://www.prisma.io/blog/graphql-server-basics-the-schema-ac5e2950214e (accessed Apr. 07, 2022).

[13] “GraphQL.” https://spec.graphql.org/draft/#OperationDefinition (accessed May 19, 2022).

[14] “Mutations in GraphQL | HackerNoon.” https://hackernoon.com/mutations-in-graphql-9ac6a28202a2 (accessed Apr. 08, 2022).

[15] “GraphQL schema stitching - Apollo GraphQL Blog.” https://www.apollographql.com/blog/backend/graphql-schema-stitching/ (accessed Jun. 01, 2022).

[16] P. Roksela, M. Konieczny, and S. Zielinski, “Evaluating execution strategies of GraphQL queries,” *2020 43rd Int. Conf. Telecommun. Signal Process. TSP 2020*, pp. 640–644, 2020, doi: 10.1109/TSP49548.2020.9163501.

[17] “GraphQL Microservices — C#. GraphQL API in a microservices… | by Raphael Yoshiga | ITNEXT.” https://itnext.io/graphql-schema-stitching-c-2ccfd209d3ce (accessed Apr. 10, 2022).

Bibliographie

[18] S. Y. Shin and C. K. Suh, *Discovering Platform Government Research Trends Using Topic Modeling*, vol. 11815 LNCS. 2019.

[19] P. Stünkel, O. von Bargen, A. Rutle, and Y. Lamo, "GraphQL federation: A model-based approach," *J. Object Technol.*, vol. 19, no. 2, pp. 1–21, 2020, doi: 10.5381/JOT.2020.19.2.A18.

[20] G. Brito and M. T. Valente, "REST vs GraphQL: A controlled experiment," *Proc. - IEEE 17th Int. Conf. Softw. Archit. ICSA 2020*, no. February, pp. 81–91, 2020, doi: 10.1109/ICSA47634.2020.00016.

[21] J.-M. Favre, "{F}oundations of {M}eta-{P}yramids: {L}anguages vs. {M}etamodels -- {E}pisode II: {S}tory of {T}hotus the {B}aboon1," *Lang. Eng. Model. Softw. Dev.*, no. 04101, 2005.

[22] A. Déplanche, "Présentée et soutenue par : Remerciements," vol. 9, 2017.

[23] I. Omg and S. Factories, "Ingénierie Dirigée par les Modèles," pp. 1–22.

[24] D. De Recherche, "Une approche basée sur l ' Ingénierie Dirigée par les modèles pour identifier , concevoir et évaluer des aspects sécurité Denisse Yessica Muñante Arzapalo," 2014.

[25] A. Abdulazeez, "Combining SysML and SystemC to Simulate and Verify Complex Systems," p. 136, 2017.

[26] "Acceleo | Home." https://www.eclipse.org/acceleo/ (accessed Apr. 16, 2022).

[27] M. A. I. Nfe, L. Cim, P. D. M. Psm, and F. Villemin, "L ' architecture dirigée par les modèles (MDA) Le CIM (Computation Independent Model)," 2014.

[28] "Welcome to Python.org." https://www.python.org/ (accessed Jun. 19, 2022).

[29] M. Van Tellingen, "Zeep Documentation," 2021.

[30] A. Ojamaa and K. Düüna, "Assessing the security of Node.js platform," *2012 Int. Conf. Internet Technol. Secur. Trans. ICITST 2012*, pp. 348–355, 2012.

[31] "Mon utilisation de Visual Studio Code — Lilian Besson @ perso.crans.org." https://perso.crans.org/besson/visualstudiocode.fr.html (accessed Jun. 19, 2022).

[32] "GraphQL Code Libraries, Tools and Services." https://graphql.org/code/ (accessed Jun. 18, 2022).